Učinkovitost algoritama sortiranja

Božiček, Nikola

SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
V A R A Ž D I N

Nikola Božiček

UČINKOVITOST ALGORITAMA SORTIRANJA

ZAVRŠNI RAD

Sisak, 2021.

SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
V A R A Ž D I N

Nikola Božiček
Matični broj: 0016135062
Studij: Primjena informacijske tehnologije u poslovanju

UČINKOVITOST ALGORITAMA SORTIRANJA

ZAVRŠNI RAD

Mentor:
Prof. dr. sc. Lovrenčić Alen

Sisak, prosinac 2021.

Nikola Božiček

Izjava o izvornosti

Izjavljujem da je moj završni rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor/Autorica potvrdio/potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

---

Sažetak

Ovaj rad objašnjava najpoznatije algoritme sortiranja kako se izvode i daje usporedbu učinkovitosti češće korištenih algoritama sortiranja u različitim uvjetima. Uvodno se objašnjava što je to algoritam i koje su njegove karakteristike, te što je sortiranje. Daje se uvid što je algoritam sortiranja, navode se i tumače karakteristike algoritma sortiranja. Svaki algoritama sortiranja ima dvije glavne složenosti po kojima se zna koji ima bolju učinkovitost, tako se ovdje definiraju te složenosti, a one su vremenska i prostorna složenost. Detaljno je objašnjena notacija veliko O jer se ona najčešće koristi kod analize učinkovitosti algoritma sortiranja jer je najkorisnije znati najbrže vrijeme izvršavanja u najgorem slučaju. U ovom radu su implementirani algoritmi sortiranja: Mjehuričasto sortiranje (eng. Bubble Sort), Sortiranje izborom (eng. Selection Sort), Sortiranje umetanjem (eng. Insertion Sort), Shellovo sortiranje, Sortiranje spajanjem (eng. Merge Sort), Quicksort, Sortiranje prebrojavanjem (eng. Counting Sort). Prikazuje se usporedba složenosti implementiranih algoritama te daje zaključak koji od implementiranih su najbolji za primjenu u poslovnom okruženju. Zaključuje se da su jednostavni algoritama sortiranja najbolji za manje raspone brojeva, dok su napredni algoritmi sortiranja najprikladniji za velike količine podataka. Za razliku od prethodno navedena dva tipa algoritama sortiranja, algoritmi za distribucijska sortiranja su vrlo učinkoviti kod male i velike količine podataka za sortiranje, jedino što zauzimaju više radne (RAM eng. Random Access Memory) ili vanjske (stalne) memorije pri njihovom izvršavanju.

Ključne riječi: algoritam; sortiranje; struktura podataka; algoritam sortiranja; karakteristike algoritama sortiranja; vremenska složenost; prostorna složenost; usporedba algoritama sortiranja;

Sadržaj

1. Uvod

Kako bi se riješio neki problem, potreban je neki postupak kojim bi se to postiglo. Tako se kod programiranja pojavljuje pojam algoritam. Algoritam je niz instrukcija koji od unesenih vrijednosti daje nama željeni rezultat. Postoje razne vrste algoritama, a u ovom radu će se obraditi algoritmi sortiranja. Kako bi se ljudi mogli lakše snaći u informacijama kojima raspolažu, i lakše raspolagali s njima, potrebno ih je sortirati uz pomoć nekog algoritma sortiranja. Za odabir odgovarajućeg algoritma za sortiranje, treba se znati koji je prikladan za koju količinu podataka, te koliko je koji brz. Sa znanjem o svrsi pojedinih algoritama sortiranja, njihovoj složenosti i vremenu izvršavanja može se odabrati odgovarajući algoritam za određeni problem. U ovome radu bit će objašnjeno što je algoritam, navedene njegove glavne karakteristike i gdje se oni koriste. Zatim će biti objašnjena potreba za sortiranjem. U nastavku se navodi što je algoritam sortiranja i koje su njegove karakteristike. Dalje će biti objašnjeno kako se oni analiziraju, koji se pojmovi koriste za iskazivanje njihove složenosti. Detaljno će se proći kroz vremensku složenost. Ovaj rad objašnjava sljedeće algoritme sortiranja i načine njihove izvedbe: Mjehuričasto sortiranje (eng. Bubble Sort), Sortiranje izborom (eng. Selection Sort), Sortiranje umetanjem (eng. Insertion Sort), Shellovo sortiranje (eng. Shellsort), Dvosmjerno mjehuričasto sortiranje (eng. Cocktail Sort), Sortiranje spajanjem (eng. Merge Sort), Sortiranje pomoću hrpe (eng. Heap Sort), Quicksort, Sortiranje prebrojavanjem (eng. Counting Sort), Sortiranje pomoću pretinaca (eng. Bucket Sort), Radix sortiranje (eng. Radix Sort). Implementirat će se najčešće korišteni algoritmi sortiranja: Selection sort, Bubble sort, Cocktail sort, Insertion sort, Shell sort, Merge sort, Quicksort i Counting sort. Zatim će se izvršiti usporedba svih algoritama sortiranja pod istim uvjetima na različite količine elemenata i izvršiti njihova analiza.

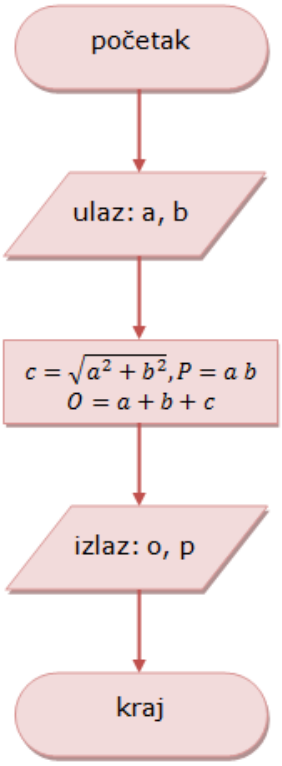
2. Algoritam i sortiranje

2.1. Definicija i pojam algoritma

Navedeno je nekoliko definicija algoritma kako bi se bolje dobio uvid u taj pojam:

- „Algoritam je konačni slijed definiranih naredbi za rješavanje zadatka. U računarstvu, algoritam je set naredbi ili uputa računalu za rješavanje nekog problema, korak po korak“ [Mooc, bez dat.].
- „Riječ algoritam znači postupak ili skup pravila kojih se treba pridržavati u izračunima ili drugim operacijama rješavanja problema“ [RishabhPrabhu, 2020].
- „Algoritam je slijed jednoznačnih uputa za rješavanje problema, tj. za dobivanje potrebnih rezultata za bilo koji legitimni ulaz u ograničeno vrijeme“ [Anany Levitin, 2012].

Sljedeća slika prikazuje primjer dijagrama tijeka jednostavnog algoritma:



Slika 1. Primjer dijagrama tijeka algoritma (Loomen, bez dat.)

Dijagram tijeka sa slike 1. se dijeli na 3 dijela, a to su unos podataka, obrada podataka i prikaz rezultata. Ovaj algoritam će izračunati treću stranicu pravokutnog trokuta, njegov opseg i površinu. Na početku se traže dva ulazna podatka, broj a i broj b. Zatim se obrađuju ti podaci kroz zadane formule. Na posljetku dobivamo izlaz, odnosno vrijednosti zbog kojeg je i stvoren algoritam. Za rezultat dobivamo površinu i opseg pravokutnog trokuta [Loomen, bez dat.]. Ovo dovodi do zaključka da se računalni program može promatrati kao razrađeni algoritam [TechTarget Contributor, 2019].

2.2. Karakteristike algoritma

Svaki algoritam bi trebao imati sljedeća svojstva:

- Konačnost što znači da algoritam mora završiti kroz određeni broj koraka. Računalni program koji ima beskonačan broj koraka se ne smatra algoritmom.
- Definitnost koja označava da je svaki korak algoritma jednoznačno definiran. Ne smije se dogoditi da je neki korak dvosmislen ili da se može izvesti na različite načine.
- Determiniranost kojom se nakon izvršenja svakog koraka jednoznačno određuje koji je sljedeći korak koji se treba izvršiti u algoritmu.
- Algoritam može imati ulazne podatke, no oni nisu obavezni. Postoje algoritmi koji ne primaju ulazne podatke, već sami generiraju podatke koje kasnije koriste.
- Algoritam mora imati minimalno jedan izlazni podatak ili rezultat, a može ih imati i više. Algoritam nakon završetka izvršavanja mora dati neki izlaz, jer inače od njega ne bi bilo koristi. Njegova svrha je davanje rješenja zadanog problema.
- Efektivnost se postiže tako da sve operacije u algoritmu postanu što jednostavnije kako bi ih čovjek mogao uraditi u što kraćem roku samo uz pomoć olovke i papira. Ona se odnosi na to da svaka operacija u algoritmu mora biti algoritam.
- Efikasnost nije nužna za algoritam, no svakako je poželjna kako bi došli što brže do željenog rezultata [Lovrenčić, 2018, str. 75,76].

Prednosti algoritama:

- Lako ih je razumjeti
- Algoritam je korak po korak prikaz rješenja datog problema
- U algoritmu se problem raščlanjuje na manje dijelove ili korake, pa ga je programeru lakše pretvoriti u stvarni program [RishabhPrabhu, 2020].

Mane algoritama:

- Pisanje algoritma traje dugo
- Grananje i petlje je teško prikazati u algoritmima [RishabhPrabhu, 2020].

2.3. Sortiranje

Sortiranje je aktivnost koja se koristi svaki dan. Sortira se novac, datoteke na računalu, prikaz proizvoda na online trgovini i slično. Lakše se pronalaze i koriste stvari ili podaci kada su sortirani, te je to razlog zašto je sortiranje bitno. Sortiranje je proces preuređivanja niza podataka. Podatke je moguće sortirati prema broju, abecedno, na temelju određenog atributa, uzlazno i silazno i na mnoge druge načine ovisno o kakvom se tipu podataka radi. Sortiranje je osnovna aktivnost u računalnoj znanosti. Ono je važno jer olakšava svakodnevne aktivnosti. Tako na primjer operativni sustavi mogu sortirati datoteke prema nazivu, memoriji, datumu, tipu podatka i slično. [Đuranović, 2017].

3. Algoritmi sortiranja

3.1. Pojam Algoritam sortiranja

Algoritmi razvrstavanja ili sortiranja se danas koriste često jer se podaci svuda nalaze, a podaci su pregledniji i lakši za korištenje kad su organizirani. Ručno sortiranje podataka može biti komplicirano i neefikasno, pogotovo kad se radi o iznimno velikim količinama podataka. Sortiranje podataka putem računala uvelike pojednostavljuje i automatizira takve zadatke. Definicija algoritama za sortiranje bi bila „Raspoređivanje podataka određenim redoslijedom“ [Maghsoudi, 2019]. Uz to algoritam sortiranja je metoda koja može poslužiti za organiziranje velikog broja podataka u određeni redoslijed, poput numeričkog ili abecednog reda, najveće do najmanje vrijednosti ili najkraće do najduže duljine. Oni uzimaju za ulazne podatke niz vrijednosti, izvode određene operacije nad tim nizom i isporučuju uređene nizove kao izlaz [Wigmore, bez dat.]. Ovakvi algoritmi se koriste za organiziranje neurednih podataka kako bi se olakšala upotreba tih podataka. S algoritmima sortiranja se olakšava i ažuriranje podataka, te su zato oni jedno od najosnovnijih istraživanja iz područja informatike i računalnog programiranja [Anwar Naser Frak, 2016]. Sortiranje često koristimo za smanjivanje složenosti problema. Ti algoritmi imaju izravnu primjenu u algoritmima pretraživanja, algoritmima baza podataka, algoritmima strukture podataka i mnogim drugim [FreeCodeCamp, 2020]. Postoji nekoliko algoritama za sortiranje koje možemo izabrati, svaki sa svojim prednostima, ovisno o situaciji. U nastavku su navedeni popularni algoritmi sortiranja u tri podjele.

Jednostavni algoritmi za sortiranje:

- Mjehuričasto sortiranje (eng. Bubble Sort)
- Sortiranje izborom (eng. Selection Sort)
- Sortiranje umetanjem (eng. Insertion Sort)
- Shellovo sortiranje (eng. Shellsort)
- Dvosmjerno mjehuričasto sortiranje (eng. Cocktail Sort) [Lovrenčić, 2018, str. 82-114]

Napredni algoritmi za sortiranje:

- Sortiranje spajanjem (eng. Merge Sort)
- Sortiranje pomoću hrpe (eng. Heap Sort)
- Quicksort [Lovrenčić, 2018, str. 82-114]

Distribucijska sortiranja:

- Sortiranje prebrojavanjem (eng. Counting Sort)
- Sortiranje pomoću pretinaca (eng. Bucket Sort)
- Radix sortiranje (eng. Radix Sort) [Lovrenčić, 2018, str. 335, 338, 353]

Klasifikacija algoritama po njihovoj unutarnjoj strukturi:

- Algoritmi sortiranja temeljeni na zamjeni konceptualno započinju s cijelim nesortiranim nizom i razmjenjuju određene parove elemenata krećući se prema više sortiranom dijelu niza. Tako se zamjenjuju susjedni elementi kao kod Mjehuričastog sortiranja ili elementi u određenom razmaku kod Shell sortiranja.
- Algoritmi razvrstavanja temeljeni na spajanju stvaraju početne "prirodne" ili "neprirodne" razvrstane sekvenca. Zatim dodaju sekvenci bilo element po element čuvajući redoslijed sortiranja kao kod sortiranja umetanjem ili spajaju dva već razvrstana niza dok ne ostane samo jedan segment sortiranja u kojem se nalaze svi podaci. Ovim algoritmima obično treba dodatni memorijski prostor za privremene nizove.
- Algoritmi sortiranja zasnovani na principu stabla pohranjuju podatke, barem konceptualno, u binarno stablo. Postoje dva različita pristupa, jedan koji se temelji na hrpama, a drugi na stablima pretraživanja.
- Sortiranje prema distribucijskim algoritmima koristi dodatne informacije o indeksu i vrijednosti za spremanje znamenki i nizova u računalo. Primjeri ove vrste algoritama su Radix sortiranje, Sortiranje pomoću pretinaca i Sortiranje prebrojavanjem [Softpanorama Society, 2020].

4. Karakteristike algoritama sortiranja

Ne postoji algoritam koji bi bio najbolje rješenje u svim situacijama iako su neki značajno bolji od drugih. Neki algoritmi su jednostavni, ali relativno spori, dok su drugi brži, ali složeniji. Postoje pak oni koji bolje rade na nasumično uređenim ulazima, dok ostali bolje rade na gotovo razvrstanim popisima. Neki su prikladni samo za popise koji se nalaze u brzoj memoriji, dok se drugi mogu prilagoditi za razvrstavanje velikih datoteka pohranjenih na disku [Anany Levitin, 2012].

Metode sortiranja razlikuju se po ovim karakteristikama:

- Brzina ili vremenska složenost - odnosi se na vrijeme potrebno da se izvrši algoritam, a ona ovisi o broju elemenata koji se trebaju sortirati
- Složenost prostora - govori koliko dodatnog prostora treba algoritam ovisno o broju elemenata koji trebaju biti sortirani
- Stabilnost - održava relativni slijed elemenata koji imaju istu vrijednost, ne mijenja mjesta kod usporedbe istih vrijednosti
- Rekurzivno / ne-rekurzivno - rekurzija je metoda unutar programskog koda koja poziva samu sebe te stoga rekurzivan algoritam sortiranja zahtijeva dodatnu memoriju na stogu. Korištenjem rekurzivnih algoritama sortiranja postoji mogućnost prekoračenja dostupne memorije (prelijevanje stoga, eng. stack overflow) ako je rekurzija preduboka.

Od navedenih karakteristika o učinkovitosti algoritma najviše ovise dva parametra:

1. Vremenska složenost i
2. Prostorna složenost [GeeksforGeeks, 2021 a]

Dva osnovna resursa koji algoritmi koriste su vrijeme i memorijski prostor. Kod mjerenja se oni nazivaju vremenska i prostorna složenost. Mjera složenosti algoritma označava koliko resursa algoritam koristi. [Lovrenčić, 2018, str. 143].
Vremenska složenost označava vrijeme koje je potrebno algoritmu da izračuna rješenje zadanog problema. Brzina izračunavanja rezultata će ovisiti i o brzini računala koje izvršava algoritam, no pri određivanju složenosti se želi mjeriti kvaliteta algoritma, a ne brzina računala. U većini slučajeva vremenska složenost predstavlja mjeru koja određuje kvalitetu

određenog algoritma, no prostorna složenost se ne smije zanemariti. „Prostorna složenost algoritma mjeri se količinom memorijskog prostora koji algoritam treba za izračunavanje rezultata.“ [Lovrenčić, 2018, str. 143].

4.1. Vremenska složenost

„Najvažniji kriterij pri odabiru metode sortiranja je njezina brzina. Glavna točka interesa ovdje je kako se brzina mijenja ovisno o broju elemenata za sortiranje“. Jedan algoritam može biti dvostruko brži od drugog na stotinu elemenata, ali na tisuću elemenata može biti pet puta sporiji ili čak puno sporiji [Woltmann, 2020 a]. Učinkovitost algoritama sortiranja se temelji na broju elemenata koji se obrađuju. Za male zbirke podataka, složena metoda razvrstavanja vjerojatno neće biti toliko korisna u usporedbi s jednostavnijom metodom sortiranja [Anwar Naser Frak, 2016]. Veliki utjecaj na učinkovitost sortiranja će imati i redoslijed podataka koji se trebaju sortirati. Redoslijed podataka može bi biti nasumičan, djelomično ili potpuno sortiran, ili obrnut od željenog redoslijeda. Kada se uzmu u obzir ovi elementi povećavanjem količine ulaza se povećava i vrijeme potrebno da se algoritma sortiranja izvrši, a to se naziva vremenska složenost. Pomoću vremenske složenosti se odabire algoritam s optimalnim vremenom izvođenja za određeni problem koji treba riješiti. [Maghsoudi, 2019]. Vremenska složenost se definira kao broj izvršavanja određenog skupa naredbi umjesto ukupnog vremena. To je zato što ukupno potrebno vrijeme ovisi i o nekim vanjskim čimbenicima kao brzine procesora, upotrijebljenog prevoditelja itd. [GeeksforGeeks, 2021]. Kod odabira optimalnog algoritma za sortiranje podataka potrebno je uzeti u obzir nekoliko čimbenika. U matematici postoji Asimptotska analiza koja omogućuje da se pronađe najbolji algoritam s obzirom na vrstu unosa i vrstu stroja koji će se koristiti. Ova analiza nije savršena jer koristi velike veličine unosa i zanemaruje konstante koje su manje od najveće vrijednosti niza n. Tako kod formule $T(n)=c*n^2+k$ se zanemaruju konstante c i k jer imaju vrlo mali značaj pri velikim vrijednostima n-a. Zbog toga algoritam može biti asimptotski sporiji, ali brži za potrebe određenog problema. Asimptotska analiza je još uvijek najbolji način analize algoritama [Maghsoudi, 2019]. Pomoću asimptotske analize možemo vrlo dobro zaključiti najbolji slučaj, prosječan slučaj i najgori mogući slučaj potrebnog vremena sortiranja algoritma. Ova analiza je vezana za ulaz, tj. ako nema unosa u algoritam, zaključuje se da radi u konstantnom vremenu. Ovdje se osim ulaza svi ostali čimbenici smatraju konstantnima. Asimptotska analiza odnosi se na računanje vremena izvođenja bilo koje operacije u matematičkim jedinicama izračuna. Tako vrijeme izvođenja jedne operacije

računa se kao f (n), a za drugu operaciju koja se nalazi unutar prve se računa kao g(n^2). To znači da će se vrijeme izvođenja prve operacije linearno povećavati s povećanjem n, a vrijeme rada druge operacije će se eksponencijalno povećavati kada se n poveća. Vrijeme izvođenja obje operacije bit će gotovo isto ako je n značajno mali [Tutorialspoint, bez dat.]. Kada je u pitanju analiza složenosti nekog algoritma u smislu vremena i prostora, nikada ne možemo dati točan broj za definiranje potrebnog vremena i prostora potrebnog algoritmu. Zbog toga to vrijeme izražavamo pomoću standardnih zapisa, također poznatih kao asimptotske notacije [Studytonight Technologies, bez dat.].

Postoje 3 slučaja vremenske složenosti algoritma (asimptotskih notacija):

- O notacija označava najgori slučaj - maksimalno vrijeme potrebno za izvršavanje algoritma
- Θ (theta) notacija označava prosječan slučaj - prosječno vrijeme potrebno za izvršavanje algoritma
- Ω (omega) notacija označava najbolji slučaj - minimalno vrijeme potrebno za izvođenje algoritma [Tutorialspoint, bez dat.]

Ova procjena vremena izvršavanja algoritma se naziva „a priori“ analiza složenosti, a ona se obavlja prije stvarnog mjerenja. A priori analize se mogu definirati notacijama u najgorem slučaju složenosti koje mogu biti redova veličine: $O(1) < O(\log n) < O(n) < O(n \log n) < O(n^2) < O(n^3) < \ldots < O(2^n) < O(n!)$ [Đuranović, 2017].
Najčešće se koristiti velika O notacija pri odabiru optimalnog algoritma jer se traži algoritam koji najbrže radi u najgorem slučaju [Maghsoudi, 2019].

4.1.1. Notacija veliko O

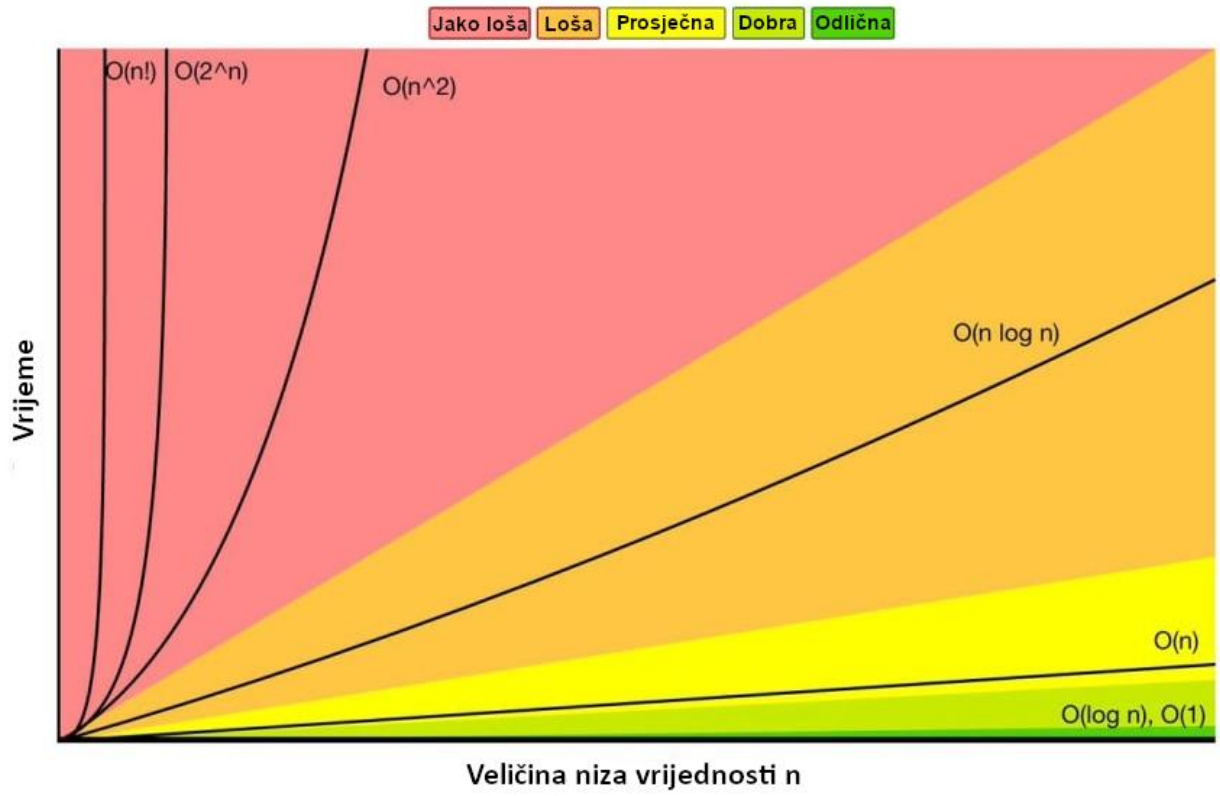
Notacija veliko O je poznata kao gornja granica algoritma ili najgori slučaj algoritma. Ona nam govori da određena funkcija nikada neće premašiti navedeno vrijeme za bilo koju vrijednost ulaza n [Studytonight Technologies, bez dat.].
Ako imamo dva algoritma za sortiranje, jedan s kvadratnim vremenom izvođenja, a drugi s logaritamskim vremenom izvođenja tada će logaritamski algoritam uvijek biti brži od kvadratnog kada je skup podataka prikladno velik. To vrijedi čak i ako se prvi pokreće na stroju koji je daleko brži od drugog. Do toga dolazi jer notacija O izolira ključni faktor u

analizi algoritama: rast. Algoritam koji se pokreće s kvadratnim vremenom raste brže od onog s logaritamskim trajanjem. [Barnett i Tongo 2008].
Veliko O osim što se koristi za označavanje vremenske složenosti algoritma, ono se koristi i za izračun koliko će prostora zauzimati. To se može izračunati identificirajući najgori slučaj za ciljani algoritam i formulirajući funkciju njegove izvedbe s obzirom na n količinu elemenata. Ako postoji algoritam koji traži broj 2 u nizu, onda bi najgori slučaj bio da je 2 na samom kraju niza. Stoga bi veliki O zapis bio O (n) jer bi morao proći kroz cijeli niz n elemenata prije nego što pronađe broj 2 [Galler i Kimura, 2019]. Performanse odnosno vrijeme izvođenja jedne metode unosa u stvarnim okolnostima oslanjaju se na n koja predstavlja veličinu unosa ili broj operacija za svaku stavku unosa [Ashwani, 2021].

Klase kompleksnosti notacije O su:

- O(1) Konstantno vrijeme - Najbrže je moguće vrijeme izvođenja, bez obzira na veličinu unosa, algoritmu je potrebno uvijek isto vrijeme za izvršavanje. Ovo je idealno za algoritam, ali rijetko je moguće [Ashwani, 2021].
- O(log n) Logaritamsko vrijeme - Vrijeme izvođenja se povećava približno za konstantan iznos kada se broj ulaznih elemenata udvostruči.
- O(n) Linearno vrijeme - Vrijeme linearno raste s brojem ulaznih elemenata n: Ako se n udvostruči, tada se i vrijeme približno udvostručuje. Vrijeme se približno udvostručuje jer izvršavanje algoritma može uključivati i komponente s nižim razredima složenosti. Oni postaju beznačajni ako je n dovoljno veliko pa su izostavljeni u zapisu. Za pronalaženje određenog elementa u nizu se moraju svi elementi niza ispitati, te ako ima dvostruko više elemenata, potrebno je dvostruko više vremena. Klasa složenosti ne daje izjavu o apsolutnom potrebnom vremenu, nego samo o promjeni potrebnog vremena ovisno o promjeni veličine unosa.
- O(n log n) Superlinearni algoritam - Vrijeme izvođenja raste nešto brže od linearnog jer se linearna komponenta množi s logaritamskom O (n × log n)
- (n^2) Kvadratno vrijeme - Vrijeme linearno raste do kvadrata broja ulaznih elemenata. Ako se broj ulaznih elemenata n udvostruči, vrijeme se otprilike učetverostručuje. Ukoliko se broj elemenata poveća deset puta, vrijeme izvođenja se povećava za sto puta [Woltmann, 2020 b].

- O(n^3) Kubno vrijeme - Vrijeme izvršavanja će biti kubno ako imamo 3 petlje jednu unutar druge. Broj ugniježđenih petlji jednako je broju petlji koje su ugniježđene tako da koliko ima petlji toliki će biti eksponent.
- O(2^n) Eksponencijalno vrijeme - Ovdje i blagi ulazi postaju vrlo brzo preveliki jer se za svaki element u nizu udvostruči vrijeme izvršavanja. Ovo vrijeme je složenije od O(n99). Obično se odabire 2 kao osnova za logaritamske i eksponencijalne vrijednosti budući da u računalnoj znanosti sve teži biti binarno. Eksponenti se mogu mijenjati mijenjanjem njihovih koeficijenata. Ako nije navedena pretpostavlja se da je baza logaritma 2.
- O(n!) Faktorijelski algoritam - Vrijeme izvođenja raste najbrže i postaje brzo neupotrebljivo čak i za male vrijednosti n. [Ashwani, 2021].

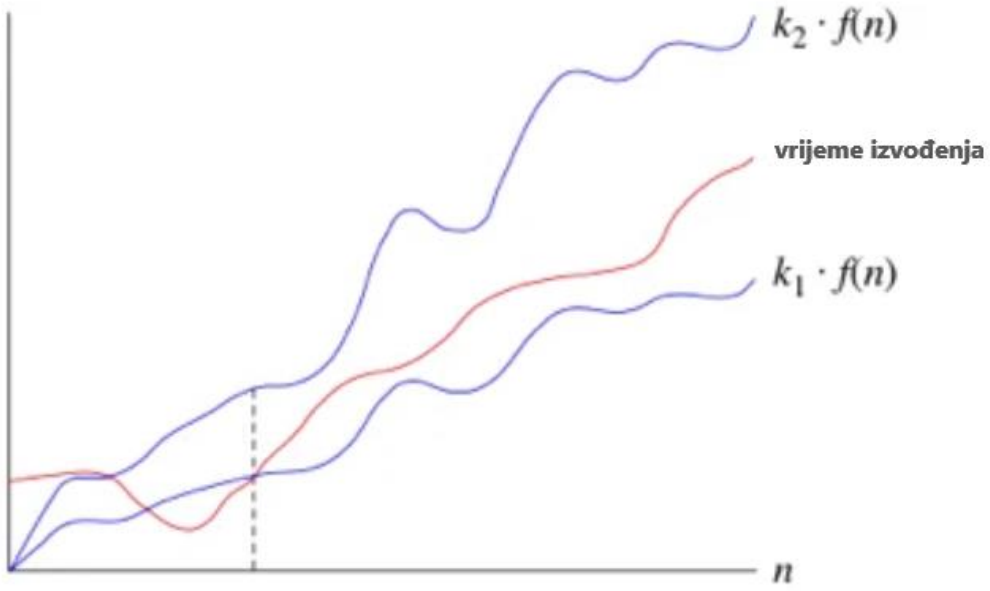


Slika 2. . Klase složenosti s notacijom veliko O (Trettevik, 2020)

Klase kompleksnosti O(n^m) ,O(2^n), O(n!) su toliko loše da je najbolje izbjegavati algoritme s tim kompleksnostima, ako je moguće [Woltmann, 2020 b].

4.1.2. Notacija theta (Θ)

Ova notacija opisuje i gornju i donju granicu algoritma pa se može reći da definira točno asimptotičko ponašanje. U stvarnom slučaju algoritam ne radi uvijek na najboljim i najgorim slučajevima. Prosječno vrijeme rada leži između najboljeg i najgoreg i može se predstaviti oznakom veliko Θ (theta). U slučaju da algoritam ima vremensku složenost predstavljenu izrazom $3n^2 + 5n$, a za prikaz toga koristimo oznaku Θ, tada bi vremenska složenost bila Θ (n^2), zanemarujući konstantni koeficijent i uklanjajući beznačajniji dio 5n. Ovdje složenost Θ (n^2) znači da će prosječno vrijeme za bilo koji ulaz n ostati između, $k1*n^2$ i $k2*n^2$, gdje su k1, k2 dvije konstante, koje čvrsto vezuju izraz koji predstavlja rast algoritma [Studytonight Technologies, bez dat.].



Slika 3. Graf Θ (theta) notacije (Studytonight Technologies, bez dat.)

4.1.3. Notacija omega (Ω)

Velika oznaka Ω (omega) koristi se za definiranje donje granice bilo kojeg algoritma odnosno najboljeg slučaja bilo kojeg algoritma. Kada predstavljamo vremensku složenost za bilo koji algoritam u obliku veliko Ω, mislimo da će algoritmu biti potrebno najmanje ovoliko vremena da se dovrši njegovo izvršavanje, a uvijek može potrajati i dulje od ove granice [Studytonight Technologies, bez dat.].

4.2. Prostorna složenost

Radni prostor ili pohrana potrebni su bilo kojem algoritmu. On je izravno ovisan ili proporcionalan količini unosa koji algoritam uzima. Da bi se izračunala složenost prostora, treba se izračunati prostor koji zauzimaju varijable u algoritmu. Što je manje prostora, brže se izvršava algoritam. Također je bitno znati da vremenska i prostorna složenost nisu međusobno povezane [Great Learning Team, 2020]. „Prostorna složenost opisuje koliko je algoritmu potrebno dodatne memorije ovisno o veličini ulaznih podataka.“ Ovo ne znači da će za memoriju potrebnu za same ulazne podatke biti potrebno dvostruko više prostora za ulazni niz dvostruko veći nego se odnosi na dodatnu memoriju potrebnu algoritmu za varijable petlje, pomoćne varijable, privremene nizove itd. [Woltmann, 2020 b]. Složenost prostora navedena je istim klasama kao i složenost vremena. Ovdje nam klasa O(1) predstavlja konstantno vrijeme kod kojeg algoritam za sortiranje dodatni memorijski prostor neće mijenjati ovisno o broju elemenata za sortiranje [Woltmann, 2020 a].
Važno je znati koliki prostor koji program zauzima za izvršavanje zadatka. Složenost prostora povezana je s količinom memorije koju će program koristiti, pa je stoga on važan faktor za analizu. Neki algoritmi, poput Bucket sort-a, imaju prostornu složenost O (n), ali mogu smanjiti vremensku složenost na O (1). Bucket sort sortira niz stvaranjem sortiranog popisa svih mogućih elemenata u nizu, a zatim povećava broj kad god se naiđe na element. [Huang, 2020]. Značajka algoritma za sortiranje koja je povezana s prostornom složenošću je da li se niz sortira na mjestu odnosno unutar originalnog niza ili su mu potrebni i dodatni, sporedni nizovi za pohranu elemenata. Kada se koristi unutarnje sortiranje ono se izvršava unutar samog niza bez potrebe za dodatnim nizovima, odnosno dodatnim poljima gdje bi se privremeno pohranjivali elementi niza. No kada se elementi niza pohranjuju u privremenim nizovima, do neke granice će se moći pohranjivati i dodatni nizovi u radnoj memoriji. Ukoliko više nema mjesta u RAM-u (eng. Random Access Memory), elementi niza će se trebati pohranjivati u sporiju, vanjsku memoriju odnosno na tvrdi disk [GeeksforGeeks, 2021. b]. „U fazi sortiranja, dijelovi podataka koji su dovoljno mali da stanu u glavnu memoriju čitaju se, sortiraju i zapisuju u privremenu datoteku. U fazi spajanja, sortirane pod-datoteke se kombiniraju u jednu veću datoteku.“ [GeeksforGeeks, 2021. b].

4.3. Svojstvo stabilnosti

Algoritam sortiranja će biti stabilan ako u ulazu čuva redoslijed bilo koja dva jednaka elementa. Ovdje neće doći do zamjene elemenata koji su istih vrijednosti. Ovo svojstvo može biti poželjno u slučaju kada imamo popis učenika razvrstanih po abecedi i želimo ga razvrstati prema ocjenama iz nekog predmeta. Stabilan algoritam će dati popis u kojem će se studenti s istim ocjenama i dalje poredati po abecedi. [Anany Levitin, 2012].
Algoritmi sortiranja koji su stabilni: Sortiranje umetanjem, sortiranje spajanjem, mjehuričasto sortiranje, sortiranje prebrojavanjem, dvosmjerno mjehuričasto sortiranje, sortiranje pomoću pretinaca, Radix sortiranje. Nestabilni algoritmi sortiranja su: sortiranje izborom, Quicksort, višestruko sortiranje umetanjem, sortiranje pomoću hrpe [FreeCodeCamp, 2019].

5. Vrste algoritama sortiranja

5.1. Jednostavni algoritmi za sortiranje

5.2.1. Mjehuričasto sortiranje (eng. Bubble Sort)

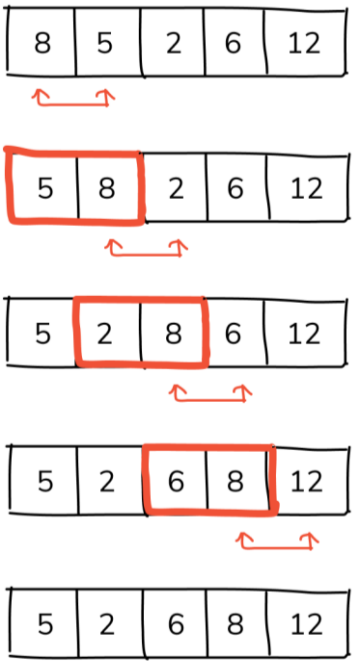
Mjehuričasto sortiranje je jedan od jednostavnih algoritama sortiranja koji prolazi kroz popis koji se sortira, uspoređuje svaki par susjednih elemenata i zamjenjuje ih ako su u pogrešnom redoslijedu. Prolaz kroz popis niza podataka se ponavlja sve dok nisu potrebne zamjene, što znači da je popis razvrstan. Svakim prolazom kroz niz se najveći element pomakne na kraj niza [Scott Bronder, 2019]. Algoritam je nazvan po načinu na koji se veći elementi prenose do vrha popisa kada se sortiraju vrijednosti od najmanje do najveće npr. kod brojeva. Iako je algoritam jednostavan, prespor je i nepraktičan za većinu problema, posebno u usporedbi sa sortiranjem umetanjem (eng. Insertion Sort) [Everything Computer Science, bez dat.].

Ovaj algoritam ima veliku složenost O (n^2), te je vrlo spor u usporedbi s drugim algoritmima za sortiranje poput Quicksort-a. Prednost mu je to što je jedan od najjednostavnijih algoritama za sortiranje za razumijevanje i kodiranje kod početnika. [FreeCodeCamp, 2019 b]. Učinkovitost može varirati o više faktora. Tako kod ovog algoritma je najbolja učinkovitost kada je polje već većim dijelom sortirano te je potreban samo jedan prolazak kroz polje, a vremenska složenost u tom slučaju je O(n). Učinkovitost će biti najgora u slučaju gdje su podaci obrnuto sortirani od našeg željenog redoslijeda te je tad vremenska složenost O(n^2). U slučaju kada je nasumično poredan niz podataka, prosječna vremenska složenost isto će iznositi O(n^2) [Klen, 2019]. Položaj elemenata u mjehuričastom sortiranju ima bitnu ulogu u određivanju njegovih performansi. Veliki elementi na početku popisa brzo se mijenjaju, dok se mali elementi na početku izuzetno sporo kreću prema vrhu. To je dovelo do toga da se ove vrste elemenata zovu zečevi i kornjače. Ovaj algoritam u prvom prolazu kroz niz napravi n usporedbi, a zatim napravi n-1 usporedbi i tako dalje. To daje n + (n-1) + (n-2) + ... + 2 + 1 što je jednako n (n + 1)/2 što je O (n^2). [Alnihoud i Mansi, 2010]. S tehničke perspektive, ovakvo sortiranje je razumno za razvrstavanje niza malih dimenzija ili za izvođenje algoritama sortiranja na računalima s izrazito ograničenim memorijskim resursima [FreeCodeCamp, 2019 b].

Ovaj algoritam se koristi za:

- Upoznavanje studenata s konceptom algoritma sortiranja.
- Otkrivanje vrlo male pogreške poput zamjene samo dva elementa u gotovo razvrstanim nizovima, a to je potrebno u računalnoj grafici [Sandipan Das, 2021].

U sljedećem primjeru vidimo jedan prolaz kroz niz uz pomoć mjehuričastog sortiranja.
U svakom prolazu kroz elemente u nizu se uspoređuju međusobna dva broja; prvi s drugim, drugi s trećim itd. Ovdje trebaju podaci biti sortirani od najmanjeg do najvećeg. Brojevi se zamjenjuju za svoja mjesta ako je lijevi broj veći od desnog. Prolazi kroz niz se ponavljaju dok god niz nije potpuno sortiran [Galler i Kimura, 2019].

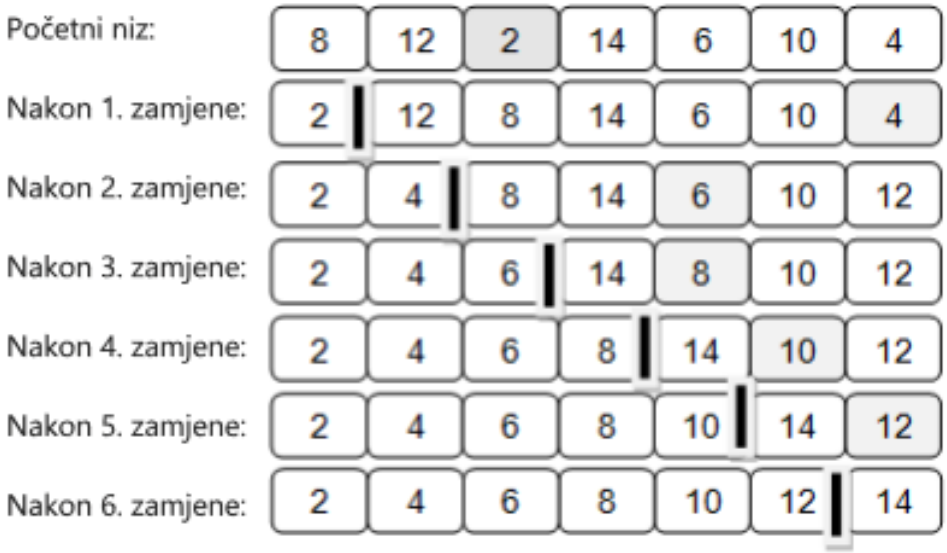


Slika 4.Primjer mjehuričastog sortiranja promjenom mjesta elemenata niza (faceprep, 2020)

5.2.2. Sortiranje izborom (eng. Selection Sort)

Sortiranje izborom je prilično jednostavno i često nadilazi mjehuričasto sortiranje. U ovom algoritmu se dijeli ulazni popis ili niz podataka na dva dijela: podskup već sortiranih podataka i podskup preostalih podataka za sortiranje koje čine ostatak popisa. Na početku je razvrstani podskup prazan, a nerazvrstani je cijeli popis podataka. Najprije se pronalazi najmanji element na nerazvrstanoj strani niza i stavlja se na kraj razvrstanog podskupa. Tako se zamjenjuju mjesta najmanjeg elementa s nesvrstane strane s onim na početku nesvrstane

strane niza. Sortiranje se ponavlja za preostale N-1 elemente. Tako se neprestano uzima najmanji nerazvrstani element i stavlja uz redoslijedom poredani sortirani podskup. Ovaj se proces ponavlja sve dok se popis potpuno ne sortira [Seif, 2018].



Slika 5. Primjer rada algoritma sortiranje izborom (Ivančić, 2018)

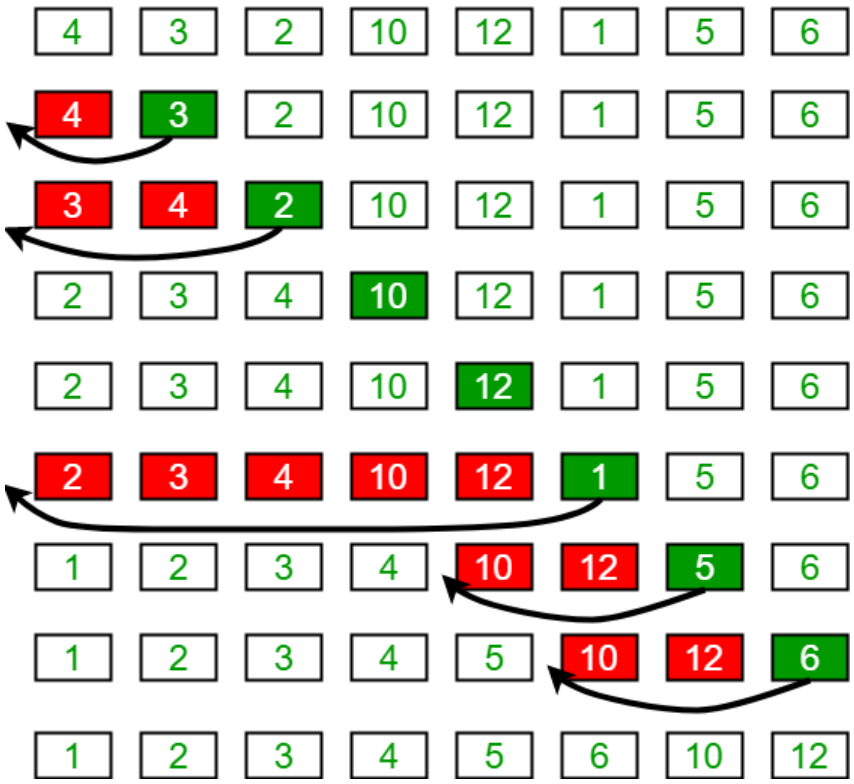
Ovaj se algoritam koristi:

- Kada je veličina popisa podataka mala zbog velike vremenske složenosti sortiranja $O(n^2)$ i svojstva nestabilnosti što ga čini neučinkovitim za veliki popis [Woltmann, 2020 a].
- Kada je memorijski prostor ograničen jer čini minimalni mogući broj zamjena tijekom sortiranja [Sandipan Das, 2021].

5.2.3. Jednostavno sortiranje umetanjem (eng. Insertion Sort)

Sortiranje umetanjem je jednostavan algoritam sortiranja koji gradi konačni sortirani niz jedan po jedan element. On ubacuje element na pravo mjesto već sortiranog dijela niza [Galler i Kimura, 2019]. U ovom algoritmu se za svaki korak pomiče element iz nesortiranog odjeljka u sortirani odjeljak sve dok se svi elementi ne sortiraju na popisu.
Ovaj algoritam radi tako da prvi element na popisu elemenata stavi u odjeljak sortiranog popisa, a sve preostale elemente u nesortiran odjeljak. Uspoređuje se prvi element s nesortiranog popisa s onim iz sortiranog i umetne se ispred prvog sortiranog ako je manji, a

inače se stavlja na kraj sortiranog odjeljka. Ukoliko se traži silazan niz vrijednosti, tada se umetne broj u sortirani niz iza manjeg broja, a inače ostavlja na desnom kraju niza. Umetanje elemenata se ponavlja dok se svi elementi s nerazvrstanog popisa ne premjeste na sortirani popis. [Makhija, bez dat.]. Mnogo je manje učinkovit s velikim nizovima podataka od naprednijih algoritama kao što su Quicksort, sortiranje hrpom ili sortiranje spajanjem. Kada ljudi ručno razvrstavaju karte u ruci, većina koristi metodu sličnu sortiranju umetanja. [Everything Computer Science, bez dat.].



Slika 6. Primjer postupka Algoritma umetanja (geeksforgeeks, 2021 c)

Unatoč slaboj učinkovitosti za velike nizove podataka sortiranje umetanjem ima sljedeće prednosti:

- Jednostavnu implementaciju
- Učinkovitiji je u praksi za male skupove podataka od većine drugih jednostavnih kvadratnih algoritama (O (n^2)), kao što su sortiranje izborom ili mjehuričasto sortiranje
- Učinkovit je za skupove podataka koji su većim dijelom sortirani
- Stabilan je jer ne mijenja relativni redoslijed elemenata s jednakim vrijednostima

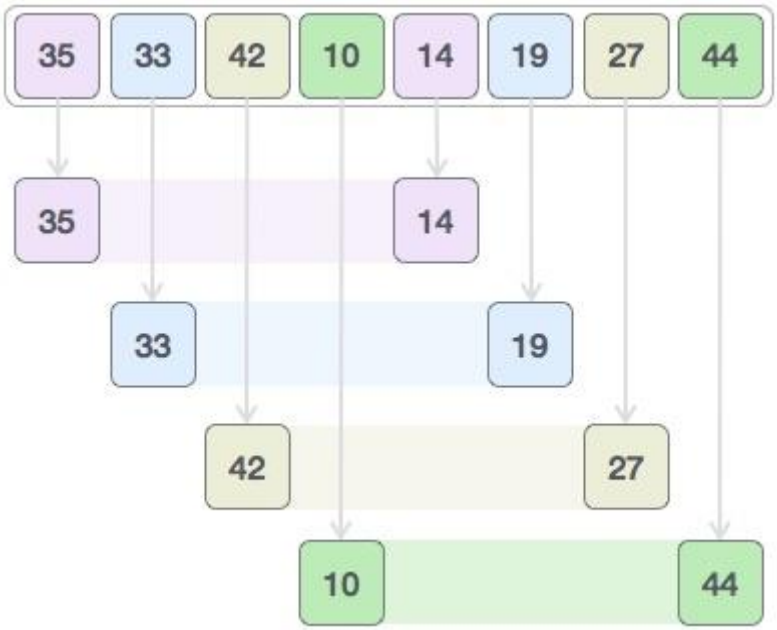
- zahtijeva samo konstantnu količinu O (1) dodatnog memorijskog prostora [Everything Computer Science, bez dat.]

5.2.4. Višestruko sortiranje umetanjem (eng. Shell Sort)

Algoritam umetanjem je bio najbrži od jednostavnih algoritama sortiranja do 1957. godine kada je Donald Shell predložio inačicu sortiranja koja je još brža, a naziva se Shellovo sortiranje (eng. Shell Sort). Ovaj algoritam sortiranja definira razmak d i svi elementi koji su udaljeni za udaljenost d se smatraju jednim nizom koji se sortira. Početna vrijednost razmaka d se dobiva dijeljenjem veličine niza n s dva. Kod svakog sljedećeg prolaza kroz niz, razmak d se dijeli sa dva. Kada ne dobijemo cijeli broj takvim dijeljenjem, odbacujemo decimalni ostatak. Ukoliko je d=4, tada će se sortirati elementi a0, a4, a8, a12,…, a zatim a1, a5, a9, a13,... i tako svaki četvrti element do kraja niza. Sortiraju se nizovi koji počinju s nultim elementom, prvim pa sve do d-1-og elementa. U sljedećem prolazu kroz niz razmak d dobivamo dijeljenjem s dva i u ovom slučaju dobijemo d=2. Kada je d=2 uspoređuje se svaki drugi element, odnosno između elemenata koji se uspoređuju je jedan drugi elementa niza. U zadnjem prolazu je d=1 i onda se uspoređuju elementi niza kao kod sortiranja umetanjem. Tako se razmak d dijeli s 2 od prvog prolaza kroz niz na kraju svakog sljedećeg prolaza dok se ne dođe da je d=1. Ovisno o izboru udaljenosti niza će biti drugačija brzina rada ovog algoritma [Lovrenčić, 2018, str. 88-90]. Shell sort ima vremensku složenost u najgorem slučaju $O(n(\log(n))^2)$ i složenost prostora O(n) [Galler i Kimura, 2019].

Shell sort se koristi:

- Razvrstavanje umetanjem ne radi dobro kada su bliski elementi daleko jedan od drugog. Kod tog problema Shell sort pomaže smanjiti razmak između bliskih elemenata.
- Također se koristi kada rekurzija prijeđe ograničenje. Bzip2 kompresor ga komprimira [Sandipan Das, 2021].

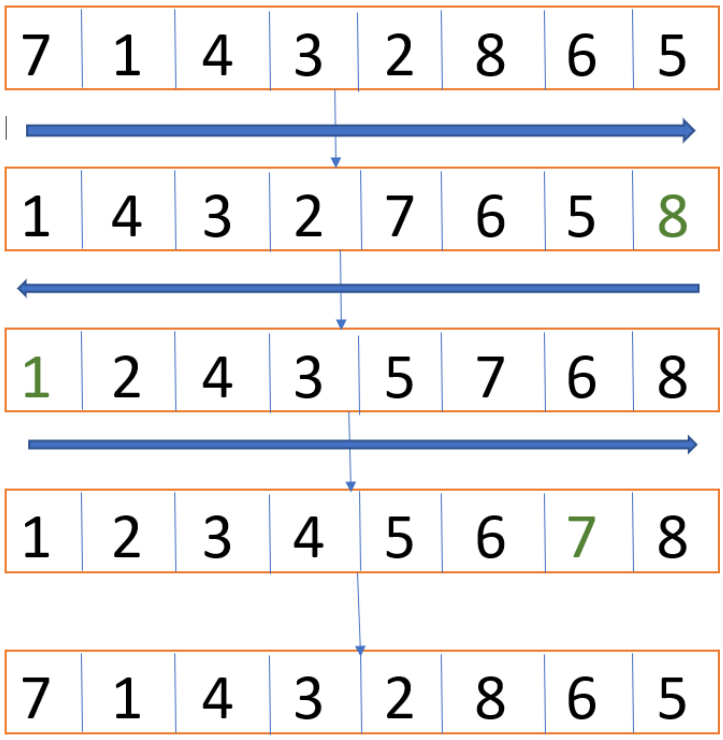


Slika 7. Primjer uspoređivanja elemenata Shell sort-a s razmakom d=4 (Tutorialspoint, bez dat.)

5.2.5. Dvosmjerno mjehuričasto sortiranje (eng. Cocktail Sort)

Dvosmjerno mjehuričasto sortiranje je varijacija mjehuričastog sortiranja (eng. Bubble sort) i sortiranja izborom (eng. Selection Sort) koji su oba stabilni algoritmi sortiranja i oba sortiranje izvršavaju usporedbom. Ovaj algoritam sortiranja se razlikuje od mjehuričastog sortiranja po tome što sortira u oba smjera kroz svaki prolaz kroz popis elemenata niza. Ovaj je algoritam sortiranja samo neznatno kompliciraniji od mjehuričastog sortiranja, a rješava problem s takozvanim kornjačama u mjehuričastom sortiranju [Mansi, 2010]. Te kornjače označavaju elemente niza koji vrlo sporo mijenjanju položaj ka njegovom odgovarajućem položaju, a to su mali elementi na početku niza. Suprotno, zečevima se nazivaju elementi koji su veliki i brzo se kreću te dolaze na potrebno mjesto. Cocktail Sort rješava taj problem takozvanih kornjača i zečeva [Alnihoud i Mansi, 2010]. U prvoj fazi prvog prolaza dvosmjernog mjehuričastog sortiranja, petlja kroz niz prolazi kao kod mjehuričastog sortiranja. Tijekom petlje uspoređuju se susjedni elementi. Ako je u bilo kojem trenutku vrijednost s lijeve strane veća od vrijednosti s desne strane, elementi se zamjenjuju. Na kraju prve faze prvog prolaza najveći broj će se nalaziti na kraju skupa. U drugoj fazi prvog prolaza petlja prolazi kroz niz u suprotnom smjeru tako da počinje od stavke neposredno prije posljednje razvrstane stavke, pa se vraćajući prema početku popisa.

Ponovno se zamjenjuju susjedni elementi ako je potrebno [Mansi, 2010]. Sada se ide prema početku niza i zamjenjuju se susjedni elementi tako da se manji premjeste s desne strane na lijevu. Tako na kraju prvog prolaza drugog dijela dobivamo najmanji element na prvom mjestu. [Dhruv Punetha, 2021]. Prostorna složenost ovog algoritma za sortiranje je O (1). Vremenska složenost Cocktail Sort-a u najgorem slučaju je O (n^2). [Mansi, 2010]. Za slučajeve u kojima je niz većim dijelom sortiran, vremenska složenost se približava prema O (n). Najbolji slučaj će biti kada je niz već sortiran, a tada je složenost vremena O (n) . [Dhruv Punetha, 2021].



Slika 8. Primjer dvosmjernog mjehuričastog sortiranja (Dhruv Punetha, 2021)

U gornjem primjeru u prvom dijelu prolaza kroz elemente broj 8 dosegne svoje točno mjesto na kraju niza. Nakon toga, najmanji broj 1 doseže svoj ispravan položaj na početku niza. Sljedeći prolaz 7 dosegne svoje mjesto na drugom posljednjem mjestu. Vidimo da su elementi razvrstani tako da se sljedeći put položaj ne mijenja. Kako nema promjene u zadnjem prolazu, program smatra niz sortiranim i izlazi iz petlje [Dhruv Punetha, 2021].

5.2. Napredni algoritmi za sortiranje

5.2.1. Sortiranje spajanjem (eng. Merge Sort)

Sortiranje spajanjem je algoritam sortiranja koji se temelji na metodi podijeli pa ovladaj (eng. divide and conquer method). Kod ove metode se problem dijeli na manje dijelove i zatim se rješavaju ti manji problemi. „Ovo je jedan od prvih predloženih algoritama za sortiranje na računalu uopće i predložio ga je John von Neumann 1945. godine." [Lovrenčić, 2018, str. 96].

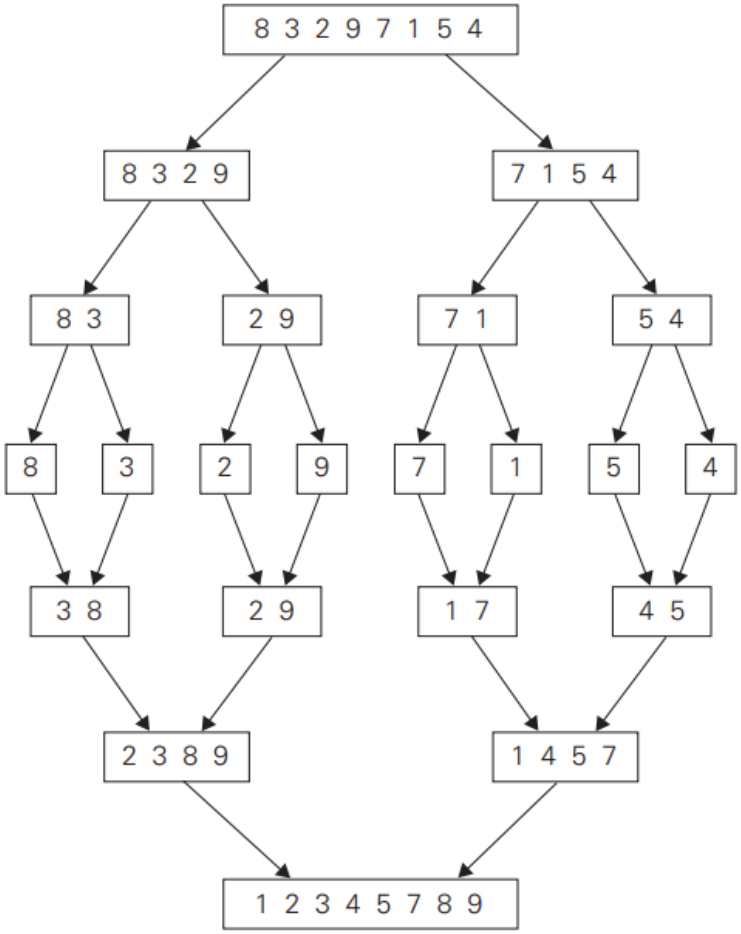
Ovaj algoritam se sortira tako da se najprije niz dijeli na dva jednaka dijela i zatim svaki manji niz na sve manje dijelove dok se ne dobiju samostalni elementi. Kada je svaki element zasebno poredan, tada se uspoređuje sa susjednim, dva po dva i slaže s lijeve strane manji, a s desne veći u slučaju kada se niz sortira od najmanje do najveće vrijednosti. Ovo uspoređivanje dva elementa se ponavlja dok se svi ne sortiraju. Nakon toga se uspoređuju dva sortirana elementa sa susjedna dva elementa i tako se svaki puta uspoređuju sve veći sortirani nizovi sa susjednim dok se ne dobije jedan potpuno sortirani niz. Kod usporedbe dva veća sortirana niza koji čine isti niz n, promatra se prvi element u prvom i prvi u drugom nizu uspoređuje i manji se stavlja na početak sortiranog niza, i tako se promatra svaki sljedeći nesortirani element. Kod postupka spajanja se koristi pomoćno polje jer kada se spoje dva sortirana dijela u jedan sortirani dio, taj novi sortirani dio niza zamjenjuje dva dijela koja su se spajala. U slučaju da se ne koristi dodatno polje, vrijednosti drugog polja bi se prepisale samo kao nastavak na dio prvog polja i tada taj dio niza ne bi bio sasvim sortiran. Kada se spajanje završi, tada se vrijednosti iz pomoćnog polja prepisuju u glavno. [Lovrenčić, 2018, str. 96-98].

Sortiranje spajanjem ima vremensku složenost za najbolji, prosječan i najgori slučaj O(n log(n)) jer se uvijek izvršava isti broj koraka neovisno da li je niz potpuno sortiran, djelomično ili ima nasumične elemente. Ovaj algoritam sortiranja koristi prostornu složenost O(n) i stabilan je [Woltmann, 2020 a].

Merge sort se koristi:

- kada se radi s povezanim listama jer dodjeljuje memoriju uzastopno dopuštajući algoritmu da brzo razbije strukturu podataka

- tamo gdje se nasumični pristup u radnoj memoriji brzo popuni zbog velike količine elemenata niza pa je potrebna pohrana u vanjskoj memoriji (primjer je baza podataka) [Sandipan Das, 2021].



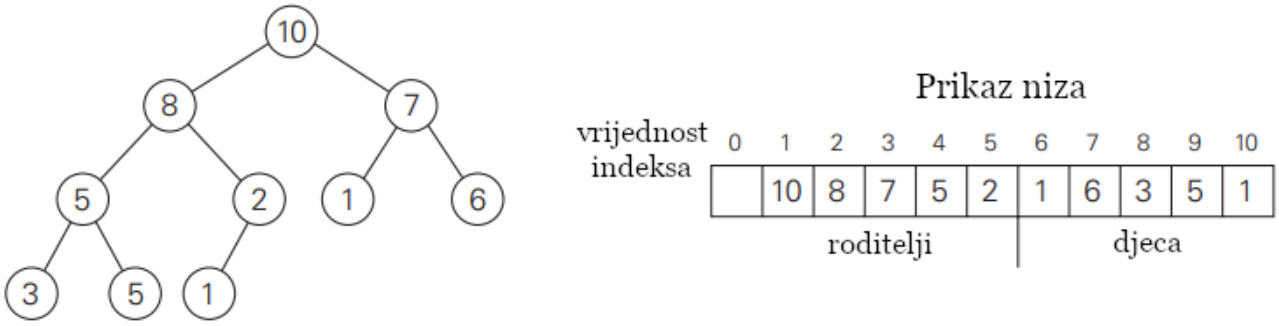
Slika 9. Primjer sortiranja spajanjem (Anany Levitin, 2012)

5.2.2. Sortiranje pomoću hrpe (eng. Heap Sort)

Algoritam sortiranja pomoću hrpe se temelji na apstraktnom tipu podataka koji se naziva hrpa. Osmislio ga je John Joseph Willams 1964. godine. Hrpa je binarno stablo u kojem je vrijednost svakog čvora manja od vrijednosti njegove djece. Binarno stablo će biti potpuno ako je svaki njegov čvor na pretposljednjoj i posljednjoj razini ima dvoje djece. [Lovrenčić, 2018, str. 105]. U prvoj fazi algoritma stvara se stablo s vrijednostima koje treba sortirati. Počinje se od lijeve strane i stvara se korijenski čvor s prvom vrijednošću. Zatim se stvara lijevi podređeni čvor i ubacuje se sljedeća vrijednost. Procjenjuje se je li vrijednost postavljena na podređeni čvor veća od vrijednosti u korijenskom čvoru te ako je veća, zamjenjujemo mjesta tih vrijednosti. To činimo u cijelom stablu. Roditeljski čvorovi uvijek

će imati veće vrijednosti od čvorova podređenih odnosno njihove djece. Na kraju prvog koraka se stvara stablo koji počinje s korijenskom vrijednošću i kreće se s lijeva na desno ispunjavajući stablo.

Sada se uspoređuju vrijednosti roditeljskog i podređenog čvora tražeći najveću vrijednost među njima, a kad se pronađe, mijenjaju im se mjesta preuređujući vrijednosti. U prvom koraku uspoređujemo korijenski čvor s posljednjim elementom na stablu. Ako je korijenski čvor veći, tada mijenjamo vrijednosti i nastavljamo ponavljati postupak sve dok posljednji element ne bude veće vrijednosti. Kad nema više vrijednosti za preuređivanje, stablu dodajemo zadnji element i ponovno pokrećemo proces. [Galler i Kimura, 2019].



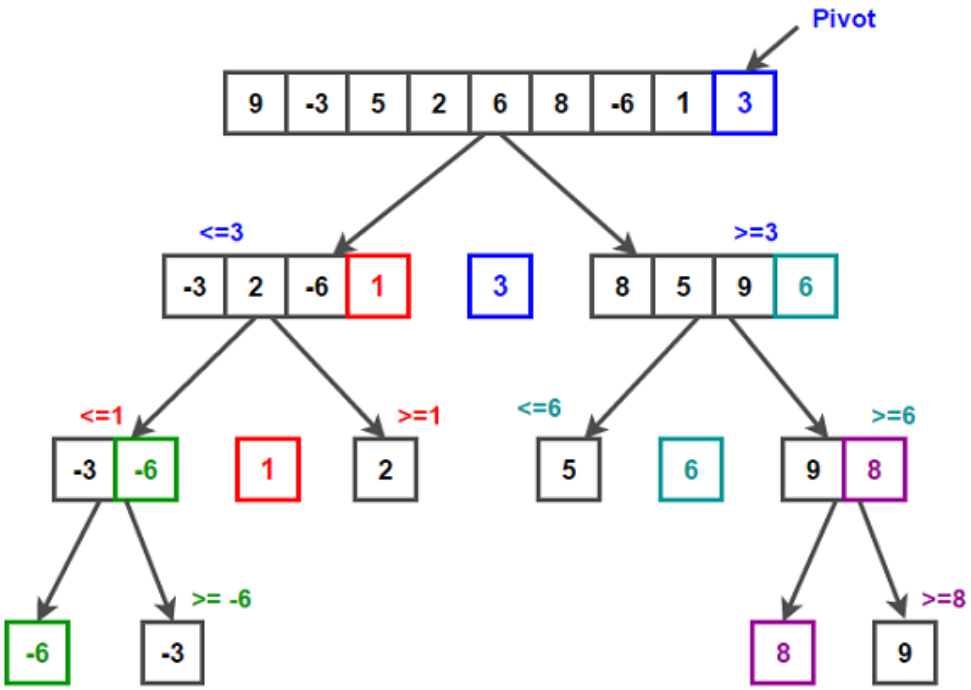
Slika 10. Hrpa i niz vrijednosti koji ona predstavlja (Anany Levitin, 2012)

Heap Sort ima vremensku složenost O (n log n) za najbolji, prosječni i najgori slučaj, te prostornu složenost O (1) [Galler i Kimura, 2019]. Algoritam je nestabilan jer prilikom usporedbe objekata s istim vrijednostima izvorni redoslijed ne bi ostao isti. Ovaj algoritam je zasnovan na usporedbi elemenata pa se može koristiti za ne-numeričke skupove podataka koliko neki odnos (svojstvo hrpe) može definirati nad elementima. [FreeCodeCamp, 2019 b].

5.2.3. Quicksort

Quicksort se često prikazuje kao najbrži algoritam sortiranja. On se kao i Merge sort temelji na metodi podijeli i ovladaj. Ovaj algoritam sortiranja je predložio Tony Hoare 1959. godine. Prilikom sortiranja niza se bira stožerni (eng. pivot) element. Zatim se na lijevu stranu stavljaju elementi manji od vrijednosti stožernog elementa, a s desne strane elementi koji su veći ili jednaki vrijednosti stožernog elementa. Nakon prve podjele niza na dva dijela, posebno se sortira dio polja s desne i s lijeve strane od prvog stožernog elementa, odnosno pivota. Tako se svaki manji dio niza sortira na način da se odabere stožerni broj i s lijeve

strane stavlja svaki manji broj od stožernog, a s desne svaki veći broj. To se ponavlja sa svakim sve manjim dijelom niza dok se cijeli niz ne sortira. Element u nizu se nalazi na točnom položaju ako su svi elementi s lijeve strane manji od njega i svi s desne veći od njega. Kod Quicksort-a velik utjecaj na brzinu izvršavanja ima odabir stožernog elementa. Tako odabirom prvog ili posljednjeg elementa u dijelu polja koji se sortira dovodi do nejednakih dijelova koji se dalje trebaju sortirati što puno usporava rad algoritma. Sortiranje će biti značajno sporije ako se odabere prvi element niza za stožerni broj, nego posljednji. Da bi sortiranje bilo što brže, za stožerni broj je najbolje uzeti neki element iz niza slučajnim odabirom ili uzeti medijan prvog, srednjeg i posljednjeg elementa. Upravo zbog svojstva nestabilnosti kod Quicksort-a se treba paziti kako će se algoritam implementirati. Zato ga je najbolje implementirati da se za stožerni broj odabire srednji element u nizu kako bi se on najbrže izvršavao [Lovrenčić, 2018, str. 111-114]. U najgorem slučaju kada su elementi poredani silaznim redoslijedom, vremenska složenost Quicksort-a je $O(n^2)$ [Woltmann, 2020 a]



Slika 11. Primjer sortiranja niza s Quicksort-om (Techie delight, bez dat.)

Na gornjoj slici (Slika 11.) se vidi primjer sortiranja s Quicksort-om. Na početku sortiranja se za stožerni broj (eng. pivot) uzeo broj tri. Zatim se niz podijelio na pola tako da su svi brojevi manji od tri prebačeni na lijevu, a veći od tri na desnu stranu. Zatim je lijevi dio niza dobio

stožerni broj jedan i opet je napravljena podjela. Kod desnog dijela niza je izabran stožerni broj šest i poredani su manji brojevi s lijeve i veći s desne. Takvo sortiranje je nastavljeno dok nije svaki element imao sa lijeve strane manje i s desne strane veće brojeve. Konačan sortirani niz u ovom slučaju je -6, -3, 1, 2, 5, 6, 8, 9.

Quicksort se koristi:

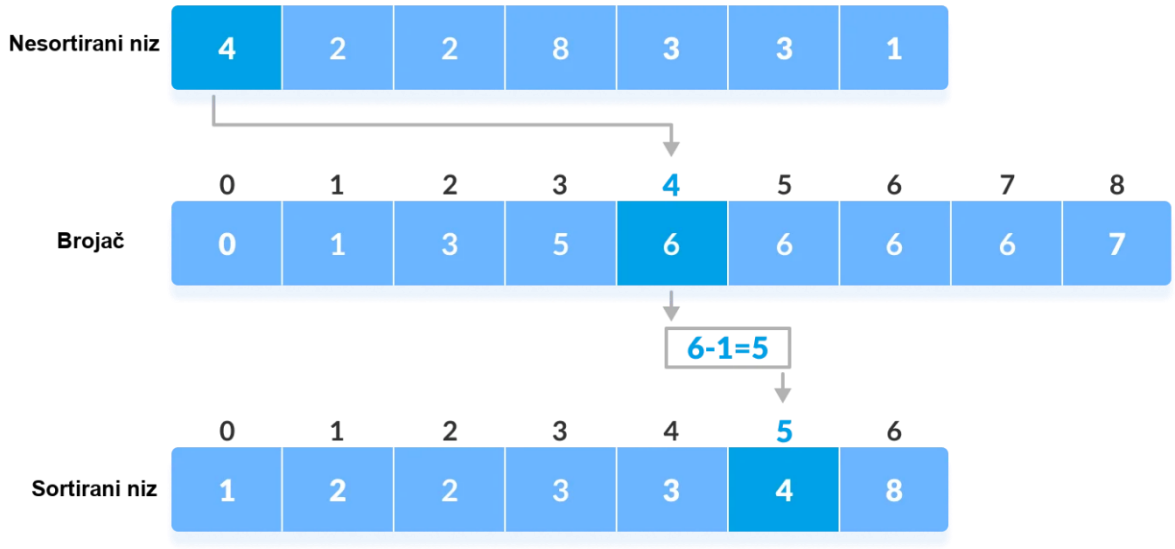
- Tamo gdje nije potrebno svojstvo stabilnosti kod sortiranja
- Za operativna istraživanja i simulaciju potaknutu događajima.
- Za numerička izračunavanja i znanstvena istraživanja,
- za skupove podataka koji stanu u memoriju, pa ne trebaju dodatni memorijski prostor [GeeksforGeeks, 2021. c]

5.3. Distribucijska sortiranja

5.3.1. Sortiranje prebrojavanjem (eng. Counting Sort)

„Sortiranje prebrojavanjem je algoritam za razvrstavanje koji razvrstava elemente niza brojeći broj pojavljivanja svakog jedinstvenog elementa u nizu. Brojač je pohranjen u pomoćni niz, a sortiranje se vrši preslikavanjem broja kao indeksa pomoćnog niza.“ [Parewa Labs Pvt, bez dat. a]. Ovaj algoritam sortiranja radi tako što broji broj elemenata niza koji imaju različitu vrijednost indeksa i pomoću aritmetike na tim brojevima određuje položaje svake vrijednosti ključa u izlaznom slijedu. Važno svojstvo sortiranja prebrojavanjem je da je stabilno tako da se brojevi s istom vrijednošću pojavljuju u izlaznom nizu istim redoslijedom kao u ulaznom nizu. Ovdje se broj koji se prvi pojavi u ulaznom nizu pojavljuje prvi i u izlaznom nizu [Ahmad Elkahlout1, Ashraf Maghari, 2017]. Sortiranje se izvršava tako da se najprije pronađe maksimalni element max iz zadanog niza. Zatim se ispuni novi niz duljine max+1 sa svim elementima 0. Ovaj niz se koristi za pohranjivanje broja elemenata u nizu. Broj svakog elementa se sprema u njihov odgovarajući indeks u nizu brojača. Tako ako je prebrojeno dva elementa broja 4, broj dva se sprema u niz pod indeksom 4. Ako npr. nema broja 6 u nizu koji se treba sortirati, tada se u novi niz pod indeksom šest, piše 0. Nakon toga se pohranjuje zbroj susjednih elemenata niza brojača na način da se prethodni broj od indeksa 1 zbraja sa sljedećim koji je indeks 2 i zapisuje zbroj u indeks 2, te se tako svaka dva broja sa 2 susjedna indeksa zbroje i rezultat zapisuje na mjesto desnog indeksa. Zatim se gleda redom svaki element od prvog nesortiranog elementa i za svaki element se traži indeks u novom

praznom nizu koji je jednak vrijednosti brojača tog broja oduzetog za jedan i pod tim indeksom se pohranjuje broj iz nesortiranog niza [Parewa Labs Pvt, bez dat. a].



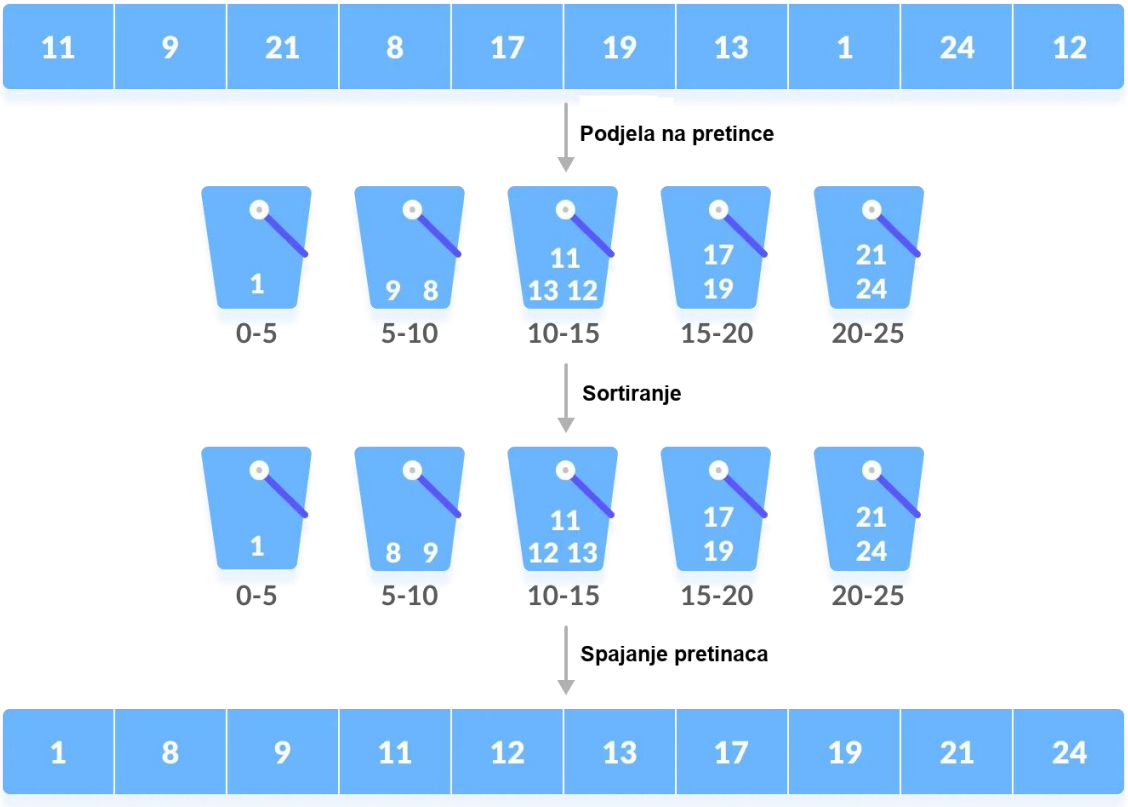
Slika 12. Primjer sortiranja pomoću Counting sort-a (Parewa Labs Pvt, bez dat. a)

Vremenska složenost ovog algoritma u svim slučajevima je O(max)+O(n)+O(max)+O(n)= O(max+n) gdje je n veličina niza, a max najveći broj u nizu koji se kod vremenske i prostorne složenosti još označava sa k [Parewa Labs Pvt, bez dat. a]. Vremenska složenost je uvijek ista O(n+k) jer bez obzira na to kako su elementi postavljeni u niz, algoritam prolazi kroz n+k puta. Kod ovog algoritma sortiranja nema usporedbe između elemenata, pa je to bolje od tehnika razvrstavanja temeljenih na usporedbi. Opet s druge strane nedostatak ovog algoritma je ako su cijeli brojevi jako veliki što tada treba napraviti niz te veličine. Prostorna složenost Counting Sort će uvijek biti O(max), što znači da će veći raspon elemenata rezultirati većom prostornom složenošću [Parewa Labs Pvt, bez dat. a].

5.3.2. Sortiranje pomoću pretinaca (eng. Bucket Sort)

Sortiranje pomoću pretinaca je tehnika sortiranja koja radi na elementima niza dijeleći ih u više kantica ili pretinaca rekurzivno, a zatim ih razvrstava pojedinačno koristeći poseban algoritam za sortiranje. Najčešće se koristi algoritam sortiranja umetanjem. Nakon što je sortiran svaki pretinac, oni se spajaju redom kako su sortirani od prvog pretinca kako bi se dobio sortirani niz [Sandipan Das, 2021]. Ovaj algoritam sortiranja se može koristiti za cijele brojeve i za brojeve sa pomičnim zarezom. Niz elemenata se podijeli na k podsegmenata koji

se zovu pretinci. Povećavanjem pretinaca se ubrzava sortiranje, no povećava se veličina prostora koji se zauzima. No ukoliko ima manje pretinaca, manje će se prostora zauzeti, a više će trebati vremena da se algoritam sortiranja izvrši. Algoritmi sortiranja koji se najčešće koriste za sortiranje elemenata unutar pretinaca su Insertion sort i Quicksort [Lovrenčić, 2018, str. 338-340].



Slika 13. Primjer Bucket sort-a (Parewa Labs Pvt, bez dat. b)

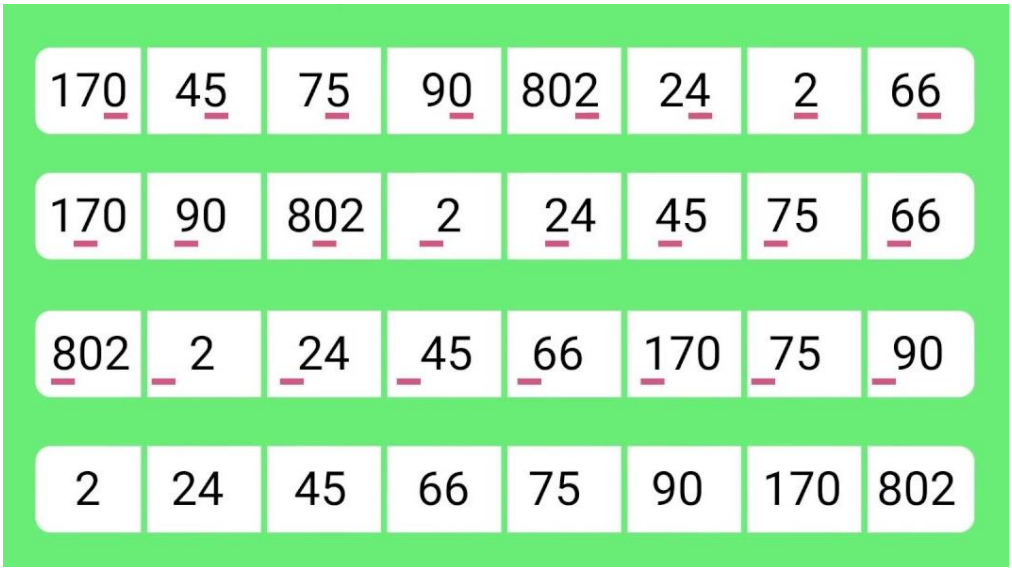
Bucket sort ima vremensku složenost $O(n^2)$, a prostornu složenost $O(n+k)$ gdje k označava broj pretinaca koji su kreirani, a n broj elemenata u nizu [Sandipan Das, 2021].

Bucket sort se koristi:

- kada je unos jednoliko raspoređen u rasponu
- za vrijednosti s pomičnim zarezom [Sandipan Das, 2021]

5.3.3. Radix sortiranje (eng. Radix Sort)

Radix sort se temelji na raspoređivanju vrijednosti ulaznog polja u pretince kao i Bucket sort. Kod ovog algoritma sortiranja se vrijednosti gledaju kao pozicijski sustav s fiksno definiranom minimalnom pozicijom. Ovakav sustav ima prirodne brojeve kod kojeg su minimalne pozicije jedinice, a zatim slijede desetice, stotice itd. Ovaj se algoritam ne koristi za sortiranje brojeva s pomičnim zarezom, a uspješno izvodi sortiranje nizova znakova. Dobro funkcionira kod sortiranja prirodnih brojeva, no kod negativnih se treba posebno definirati. Kod ovog algoritma sortiranja se vrijednosti niza sortiraju tako da se one raspoređuju u pretince po iznosu jedne od njihovih znamenaka. Elementi niza se mogu stavljati u neki od deset pretinaca, a ti pretinci su brojevi od nula do devet. Svaki se element stavlja u onaj pretinac koji odgovara broju znamenke koja se trenutno promatra. Tako se na početku gleda prva znamenka svakog elementa niza, odnosno jedinca. Tako će se u prvom prolazu kroz niz redom stavljati brojevi sa prvom najmanjom znamenkom koja je nula u pretinac „nula“ neovisno o tome koliko taj broj ima znamenaka. Nakon što se svaki element u prvom prolazu stavi u odgovarajući pretinac gledajući prvu znamenku, redom će se od pretinca „nula“ prepisivati brojevi u niz. Zatim se gleda druga znamenka i svaki se element stavlja u odgovarajući pretinac ovisno o znamenki desetice tog broja. Postupak završava kada je zadnja znamenka sa lijeve strane najveće težine stavljena u odgovarajući pretinac. Tada se zadnji put prepisuju redom elementi iz pretinaca od pretinca „nula“ do pretinca „devet“ u prazan niz kojim se dobiva sortirani niz [Lovrenčić, 2018, str. 353-354].
Radix Sort obrađuje elemente niza jednu po jednu znamenku, počevši od najviše značajne znamenke (MSD) ili najmanje značajne znamenke (LSD). MSD Radix sort koristi leksikografski redoslijed, što je posebno prikladno za razvrstavanje nizova, poput riječi, ili prikaza cijelih brojeva fiksne duljine. Radix sortiranje od najmanje značajne znamenke (LSD) prikladno je samo za elemente malog broja znamenki kao što su cijeli brojevi ili IP adrese i kada svi elementi niza imaju istu količinu znamenka ili kod koje nije prevelika razlika u broju znamenaka jer svakom dodatnom znamenkom će Radix sort trebati još jednom proći kroz cijeli niz [Softpanorama Society, 2020]. Counting sort se često koristi kao potprogram Radix sort-u jer Counting sort stabilan algoritam, a to je nužno svojstvo za Radix sort da bi dobro funkcionirao [Ahmad Elkahlout1, Ashraf Maghari, 2017].



Slika 14. Primjer sortiranja niza Radix sort-om (GeeksforGeeks, 2021)

Vremenska složenost Radix sort-a je za svaki slučaj O(n*k) gdje je n broj ulaznih podataka, a k je najveći element u nizu, a prostorna složenost mu je O(n+k). Ovdje prostorna složenost ovisi o najvećem elementu u nizu, odnosno elementu koji ima najveći broj znamenki [Kavita Bisht, bez dat.].

6. Usporedba efikasnosti algoritama sortiranja „a priori“

Razlike kod algoritma sortiranja koje su najbitnije kod odabira algoritma sortiranja su vremenska i prostorna složenost. Ove dvije složenosti se najviše razmatraju kako bi se bolje moglo zaključiti koji će algoritam sortiranja najbolje odgovarati za određenu situaciju i veličinu ulaznih vrijednosti.

Algoritam	Vremenska složenost			Prostorna složenost
	Najbolja	Prosječna	Najgora	Najgora
Quicksort	Ω(n log(n))	θ(n log(n))	O(n^2)	O(log(n))
Mergesort	Ω(n log(n))	θ(n log(n))	O(n log(n))	O(n)
Heapsort	Ω(n log(n))	θ(n log(n))	O(n log(n))	O(1)
Bubble Sort	Ω(n)	θ(n^2)	O(n^2)	O(1)
Cocktail Sort	Ω(n)	θ(n^2)	O(n^2)	O(1)
Insertion Sort	Ω(n)	θ(n^2)	O(n^2)	O(1)
Selection Sort	Ω(n^2)	θ(n^2)	O(n^2)	O(1)
Shell Sort	Ω(n log(n))	θ(n(log(n))^2)	O(n(log(n))^2)	O(1)
Bucket Sort	Ω(n+k)	θ(n+k)	O(n^2)	O(n)
Radix Sort	Ω(nk)	θ(nk)	O(nk)	O(n+k)
Counting Sort	Ω(n+k)	θ(n+k)	O(n+k)	O(k)

Slika 15. Usporedba vremenske i prostorne kompleksnosti algoritama sortiranja (Galler i Kimura, 2019)

Na slici 15. vidimo vremensku složenost i prostornu složenost za najpoznatije algoritme sortiranja. Najveći mogući broj usporedbi elemenata u nizu je označen crvenom bojom, nešto lošiji od prosjeka narančastom bojom, a prosječni broj je žutom bojom. Svjetlo zelenom je označena složenost malo bolja od prosječne, a tamno zelenom bojom odlična složenost odnosno najbrže vrijeme izvršavanja algoritma sortiranja. Ova slika pokazuje da algoritmi sortiranja sa većom prostornom složenošću često imaju i manju vremensku složenost odnosno da će se brže izvršavati. Tako algoritmi sortiranja dobiju na brzini jer koriste više dodatne memorije. Po ovoj slici možemo vidjeti koji algoritmi su poželjniji za sortiranje, a koji nisu. Svaki će od njih dati isti rezultat, no u većini slučajeva želimo taj rezultat dobiti što

brže. Zbog sporog izvršavanja algoritama koji imaju složenost $O(n^2)$ bi ih trebali izbjegavati kada je god to moguće [Woltmann, 2020 a].

Uvijek postoji bolji algoritam sortiranja za sortiranje određenog niza, no kako bi se znalo preciznije odrediti za koji algoritam sortiranja je najbolji za koje uvjete i kakvu veličinu niza potrebno je izvršiti adekvatno testiranje algoritama sortiranja. Prostorna složenost kod jednostavnih algoritama sortiranja je konstantna O(1) što znači da se ne mijenja promjenom veličine niza koji se treba sortirati. Kod naprednih i distribucijskih sortiranja će veličina niza imati značajan utjecaj na količinu dodatne memorije.

7. Implementacija algoritama sortiranja

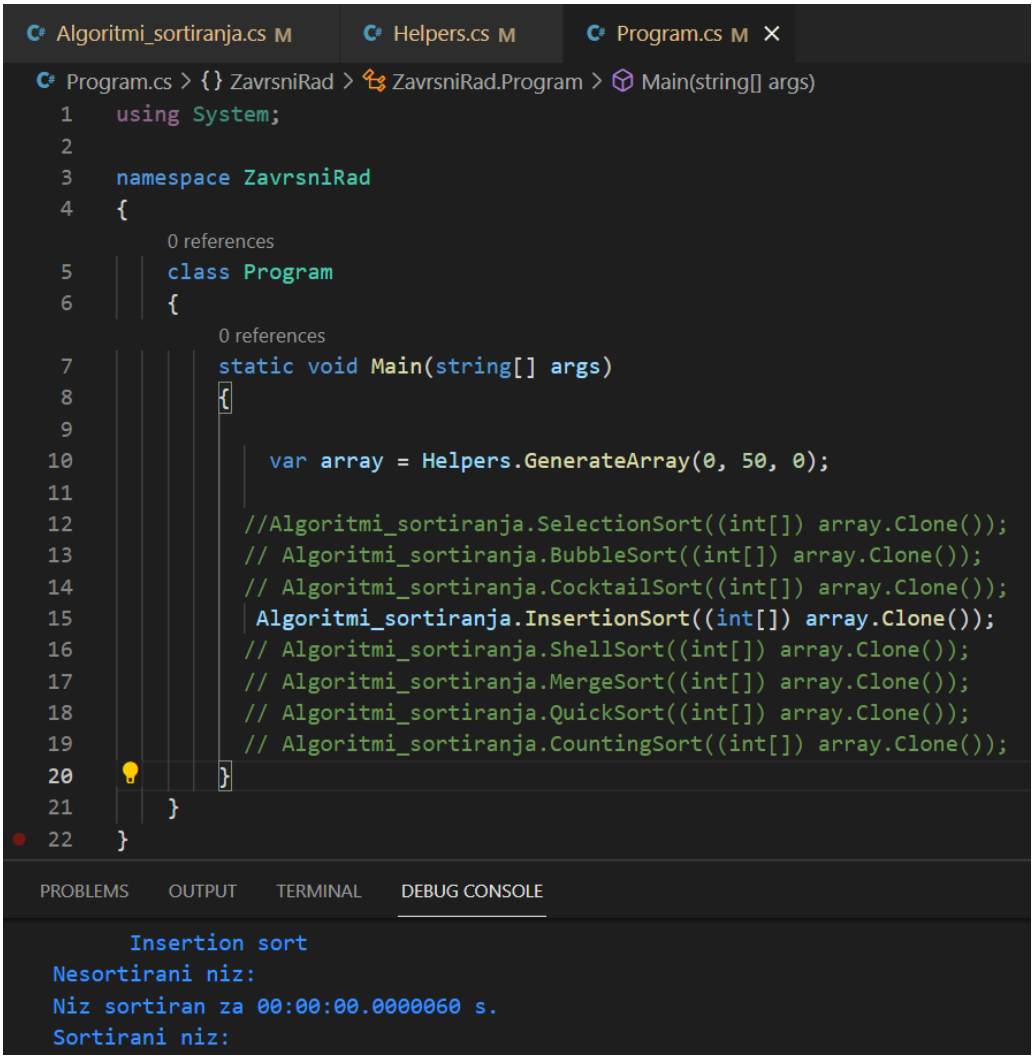
„A posteriori“ analiza daje precizniju procjenu stvarne vjerojatnosti od a priori analize. Ova analiza daje manje odstupanje od rezultata za razliku od „a priori“ analize.“ [Đuranović, 2017]. Prilikom testiranja korišten je programski jezik C# 9.0 (eng. C sharp) , a programski kod algoritama sortiranja je napisan u programu Visual Studio Code 1.61.
Kod pisanja programskog koda algoritama sortiranja pomogla je knjiga Alena Lovrenčića; Apstraktni tipovi podataka i algoritmi iz 2018 godine i internetske stranice: https://www.geeksforgeeks.org/selection-sort/, https://www.geeksforgeeks.org/bubble-sort/, https://www.geeksforgeeks.org/insertion-sort/, https://www.geeksforgeeks.org/shellsort/, https://www.geeksforgeeks.org/cocktail-sort/, https://www.geeksforgeeks.org/merge-sort/, https://exceptionnotfound.net/quick-sort-csharp-the-sorting-algorithm-family-reunion/ i https://www.alphacodingskills.com/cs/pages/cs-program-for-counting-sort.php

Performanse računala na kojem je provedeno testiranje algoritama sortiranja je:

- Procesor: Intel Core i5-6300HQ 2.30GHz, 2304 Mhz, 4 Core(s),
- RAM: 8 GB,
- SSD: 256 GB
- Operacijski sustav: Windows 10 Education 64-bit

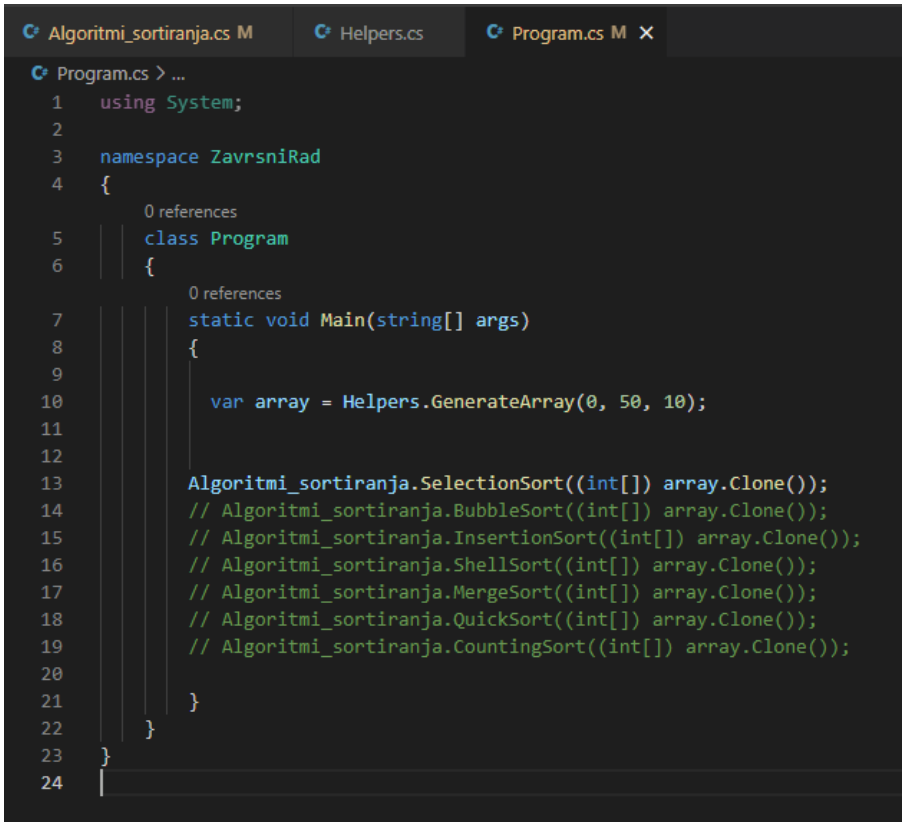
Kod testiranja algoritama sortiranja najviše se obraćala pažnja na njihovu brzinu izvršavanja, odnosno vremensku složenost. Algoritmi sortiranja koji su bili testirani su: Selection sort, Bubble sort, Cocktail sort, Insertion sort, Shell sort, Merge sort, Quicksort i Counting sort. Testiranje se vršilo tako da se pokrene programirani program u Visual Studio Code-u koji se sastoji od prethodno navedenih sedam algoritama sortiranja. U ovom programu se generirala određena količina brojeva u određenom rasponu, a za to je korištena funkcija Random. Ti generirani brojevi su se koristili za ulaz kao niz elemenata za svaki algoritam. Pokretanjem programa svaki puta se generiraju različiti brojevi, na različitim mjestima u nizu u definiranom rasponu brojeva. Zbog toga tijekom testiranja program prolazi kroz svaki algoritam i tako svaki algoritam dobiva isti problem za riješiti. Svaki algoritam sortiranja testira se sa istim nizom brojeva koji se nalaze na istim pozicijama u određenom rasponu brojeva. Na taj način se mogla dobiti vrlo precizna usporedba algoritama.

Prije svakog programskog koda algoritma sortiranja se nalazila naredba koja će ispisati njegov naziv. Za mjerenje vremena izvršavanja algoritma se prije samog njegovog koda stavljala funkcija System.Diagnostics.Stopwatch.StartNew() i stopwatch.Start() gdje „stopwatch“ označava ime varijable koja znači štoperica. Postavljeno je da se počne brojati vrijeme izvršavanja prije samog koda algoritma sortiranja, te da se završi brojanje odmah poslije zadnje linije programskog koda koji čini pojedini algoritam. Za završetak brojanja vremena je korištena funkcija stopwatch.Stop() kod koje je prvi dio sam naziv varijable kojom je počelo brojanje vremena. Liniju prije i liniju poslije programskog koda algoritma sortiranja je bila postavljena varijabla „stopwatch“ zato da se ne bi uzelo u obzir i ispisivanje naziva algoritma ili nešto drugo kako bi dobiveni rezultati bili što precizniji.
Prije testiranja brzine izvršavanja algoritama svaki algoritam je najprije bio testiran da li se ispravno generiraju vrijednosti, te da li se dobije kod svakog algoritma točan rezultat. To se provjerilo tako da program prije početka izvršavanja samog algoritma ispiše niz brojeva koji će se sortirati. Zatim da se ispiše koliko je bilo potrebno vremena u sekundama da se izvrši algoritam. Te ispod toga konačan sortirani niz da se vidi da li je algoritam sortiranja zaista učinio svoj zadatak kako treba. Drugi način testiranja ispravnosti algoritma je bio da svaki algoritam sortiranja za unos dobije niz bez ijednog elementa. Rezultat ovog testiranja je bio samo ispisano vrijeme izvršavanja svakog algoritma. Normalno je da će nekoliko vremena trebati algoritmu da se izvrši čak i ako nema nijednog elementa u nizu za sortiranje, a to vrijeme će ovisiti o njegovoj kompleksnosti, odnosno nizu programskih naredbi koje se svejedno treba odraditi.



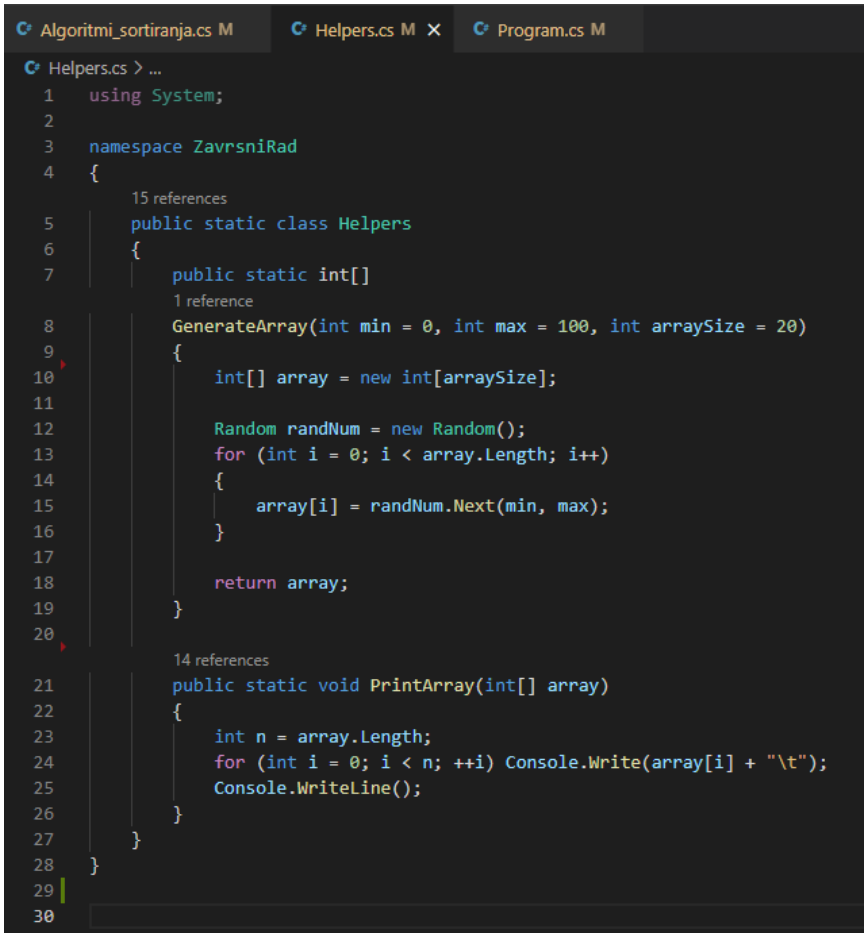
Slika 16. Testiranje ispravnosti algoritma sortiranja umetanjem sa unosom niza bez vrijednosti

Na slici 16. se izvršava funkcija GenerateArray kod koje prva dva broja označavaju raspon u kojem će se generirati brojevi, a treći broj koliko će se brojeva generirati za niz array. U ovom slučaju se nije generirao ni jedan element, no svejedno je trebalo algoritmu nešto vremena da shvati da nema ništa za sortirati. Ovdje se izvršavao algoritam sortiranja umetanjem preko funkcije InsertionSort.



Slika 17. Skripta Program koja generira brojeve i pokreće izvršavanje algoritama sortiranja

Slika 17. prikazuje skriptu Program koja pomaže kod generiranja niza brojeva tako što se definira određeni raspon i koliko će niz sadržavati elemenata. U ovom slučaju se samo izvršavao algoritam Selection sort kako bi se provjerilo da li on funkcionira. Na slici vidimo da je postavljeno da će se generirati deset elemenata za niz.



Slika 18. Skripta Helpers koja generira niz i ispisuje vrijednosti niza

Slika 18. prikazuje skriptu Helpers koja služi za generiranje niza brojeva i ispis brojeva tako da se brojevi u nizu ispisuju jedan do drugog no s razmakom.

Slika 19.Dio Skripte Algoritmi_sortiranja koja prikazuje kod algoritma sortiranja Selection sort

Slika 19. prikazuje programski kod algoritma sortiranja Selection sort. Ovim testom vidimo koje su bile generirane vrijednosti za niz, te ispis rezultata izvršavanja algoritma. Uz to je i ispisano vrijeme izvršavanja tog algoritma koji je u ovom slučaju 7,5 mikro sekundi.

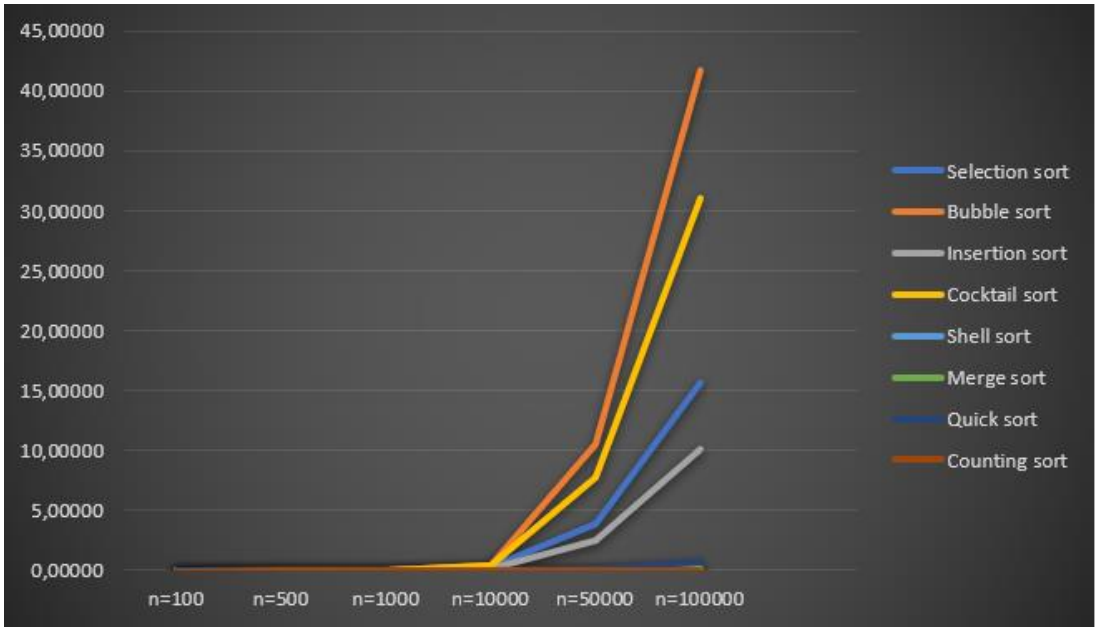
7.1. Usporedba algoritama sortiranja „a posteriori“

Napravljena je usporedba 8 algoritama sortiranja pod istim uvjetima. Ti uvjeti su bili niz iste veličine i istih brojeva na istim pozicijama. Raspon brojeva je ostao isti, a povećavana je samo količina generiranih brojeva. Tako u prvoj tablici (Tablica 1.) se vide rezultati vremenske složenosti algoritama sortiranja u rasponu brojeva od 0 do 50 sa brojem elemenata 100, 500, 1000, 10000, 50000 i 100000. Vrijednosti u tablici su prikazane u sekundama.

Raspon brojeva od 0 do 50						
	Ulaz					
Algoritam	n=100	n=500	n=1000	n=10000	n=50000	n=100000
Selection sort	0,00003	0,00042	0,00174	0,17508	3,91075	15,56216
Bubble sort	0,00004	0,00082	0,00367	0,40330	10,45784	41,70189
Cocktail sort	0,00025	0,00092	0,00286	0,31179	7,72527	31,00582
Insertion sort	0,00001	0,00032	0,00105	0,10244	2,46374	10,01516
Shell sort	0,00015	0,00027	0,00040	0,00267	0,01040	0,02228
Merge sort	0,00035	0,00057	0,00124	0,00328	0,01856	0,03146
Quick sort	0,00023	0,00032	0,00045	0,00781	0,16436	0,66544
Counting sort	0,00020	0,00030	0,00027	0,00041	0,00162	0,00164

Tablica 1. Tablica vremenske složenosti algoritama sortiranja u sekundama u rasponu brojeva 0-50

Iz ove tablice se vidi značajna razlika između jednostavnih algoritama sortiranja (Selection sort, Insertion sort, Shell sort, Bubble sort i Cocktail sort) i složenih algoritama sortiranja (Merge sort, Quicksort i Counting sort). Tako da su ovdje svi jednostavni algoritmi osim Shell sort-a značajno sporiji kako broj elemenata u nizu raste. Kod složenih algoritama je vremenska složenost vrlo malo promijenjena promjenom količine elemenata u nizu. Bitno je napomenuti da je Quicksort za stožerni broj imao zadnji element u nizu, jer da je bio neki drugi, to bi značajno utjecalo na brzinu njegovog izvršavanja.
U slučaju da je neki od algoritama bio nestabilan, to bi rezultiralo sporijim sortiranjem zbog ponavljanja istih vrijednosti kojima se svejedno mijenja položaj kod uspoređivanja. Ovdje je svojstvo nestabilnosti kod Selection sort-a i Quicksort-a imalo značajan utjecaj na brzinu sortiranja, a znamo da je bilo puno ponavljanja istih vrijednosti što je više brojeva bilo generirano u vrlo malom rasponu, posebno kod generiranja sto tisuća brojeva. Zanimljivo je primijetiti da svojstvo nestabilnosti nije značajno usporilo algoritam sortiranja Shell sort.



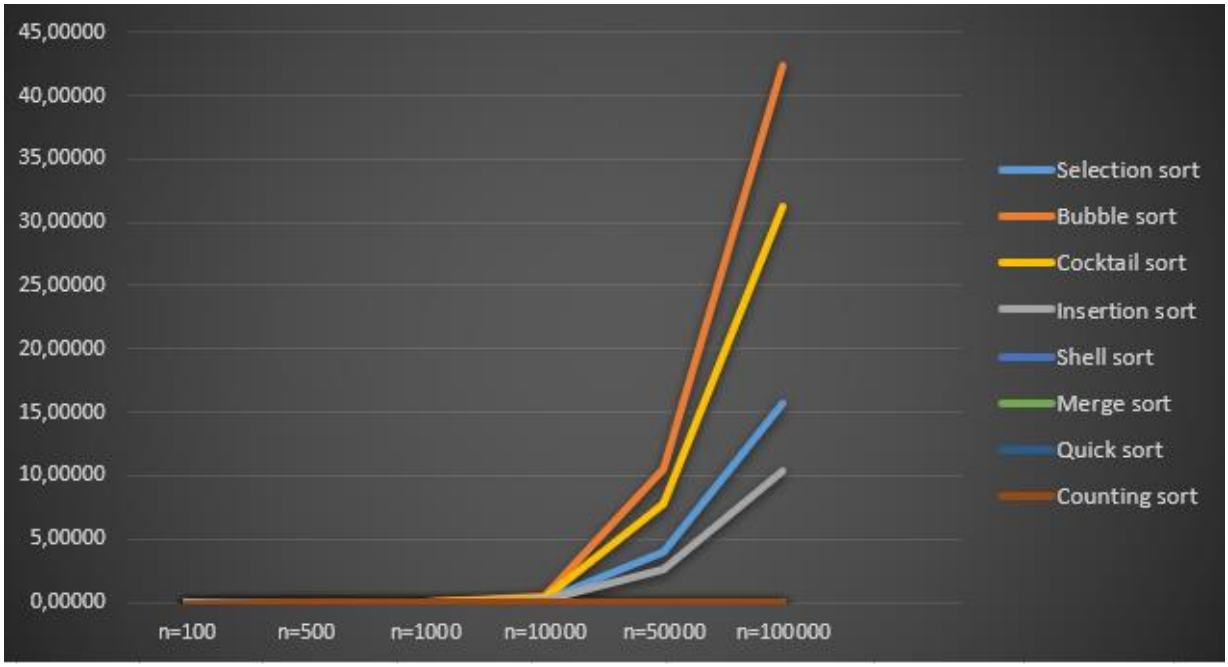
Slika 20. Grafikon vremenske složenosti algoritama sortiranja u sekundama u rasponu brojeva 0-50

Na grafikonu 17. se vidi da najveću vremensku složenost ima Bubble sort. Kod sortiranja 100 i 500 elemenata, Cocktail sort je bio sporiji od Bubble sort-a, no sa većim brojem elemenata se sve više raste vremenska složenost Bubble sort-u. Selection sort i Insertion sort su isto bolje izraženi kod sortiranja sto tisuća elemenata, no opet značajno brži. Cocktail sort je napravljen kao zamjena za Bubble sort, no i dalje nije puno brži, tako da nije primjenjiv za velike količine elemenata. Na ovom grafikonu se napredni algoritmi sortiranja jedva zapažaju zbog svoje nevjerojatne brzine koja čak ni kod sto tisuća elemenata nije došla do jedne sekunde izvršavanja. Shell sort je iznenađujuće brz unatoč tome što spada pod jednostavne algoritme, te se on također ne vidi na grafikonu zbog svoje velike brzine izvršavanja.

	Raspon brojeva od 0 do 100000					
	Ulaz					
Algoritam	n=100	n=500	n=1000	n=10000	n=50000	n=100000
Selection sort	0,00003	0,00047	0,00181	0,16370	3,99317	15,62884
Bubble sort	0,00004	0,00083	0,00356	0,40832	10,58836	42,33277
Cocktail sort	0,00024	0,00088	0,00306	0,32010	7,84334	31,16926
Insertion sort	0,00001	0,00026	0,00147	0,10515	2,55104	10,31821
Shell sort	0,00016	0,00035	0,00039	0,00401	0,01855	0,03959
Merge sort	0,00035	0,00051	0,00071	0,00380	0,02929	0,04176
Quick sort	0,00023	0,00029	0,00040	0,00207	0,01115	0,02322
Counting sort	0,00106	0,00111	0,00229	0,00149	0,00297	0,00393

Tablica 2. Tablica vremenske složenosti algoritama sortiranja u sekundama u rasponu brojeva 0-100000

Druga tablica (Tablica 2.) prikazuje vremensku složenost istih algoritama, no u rasponu brojeva od 0 do 100000 sa brojem elemenata 100, 500, 1000, 10000, 50000 i 100000. Ovim testiranjem se željelo vidjeti do kakve će promjene doći u brzini algoritama za sortiranje kada je raspon brojeva relativno mali i kada je raspon brojeva dovoljno velik da se značajno smanjuje vjerojatnost generiranja istih brojeva. U prethodne dvije tablice se prikazuje prosječno vrijeme izvršavanja algoritama. Vidi se velika razlika u brzini jednostavnih i složenih algoritama sortiranja. Shell sort nije puno kompleksniji od ostalih testiranih jednostavnih algoritma sortiranja, no vidi se da je odličan za sortiranje od 100 do 100 000 tisuća elemenata te bi bio najbolji odabir jer nije jako kompleksan, a dovoljno brz za sortiranje. Kada se usporede prethodne dvije tablice može se vidjeti da je Shell sort-u trebalo više vremena za sortiranje u drugom slučaju kada je bio veći raspon brojeva, dok je Quicksort-u trebalo značajno manje vremena. Quicksort nije stabilan algoritam pa se može zaključiti da mu ponavljanje velike količine istih brojeva može primjetno usporiti izvršavanje. Tako kada se usporedi sortiranje sto tisuća elemenata u rasponu od nula do pedeset i od nula do sto tisuća, ovaj algoritam sortiranja je bio skoro tri puta brži kod sortiranja brojeva u nizu gdje se brojevi slabo ponavljaju. S druge strane Counting sort je bio mnogo brži kod sortiranja gdje se isti brojevi puno ponavljaju jer je tada imao manji k, a vremenska složenost mu je u svakom slučaju O(n+k). Tako da je Counting sort sa manjim rasponom brojeva dobio još veću brzinu izvršavanja od deset tisuća elemenata na dalje.



Slika 21. Grafikon vremenske složenosti algoritama sortiranja u sekundama u rasponu brojeva 0-100000

Kod usporedbe prethodna dva grafikona (Slika 20. i Slika 21.) se vidi da ponavljanje istih brojeva u prvom slučaju i rijetko ponavljanje istih u drugom nije dovelo do vidljive promjene u vremenskoj složenosti Selection sort-a, Bubble sort-a, Cocktail sort-a i Insertion sort-a. Grafikoni su gotovo isti za te četiri vrste algoritama sortiranja, pa se može zaključiti da na njih ima najviše utjecaj što se više veličina niza povećava, a to i dokazuje njihovu vremensku složenost u prosječnom slučaju $\Theta(n^2)$ i najgorem slučaju $O(n^2)$.

7.2. Usporedba naprednih algoritama sortiranja „a posteriori“

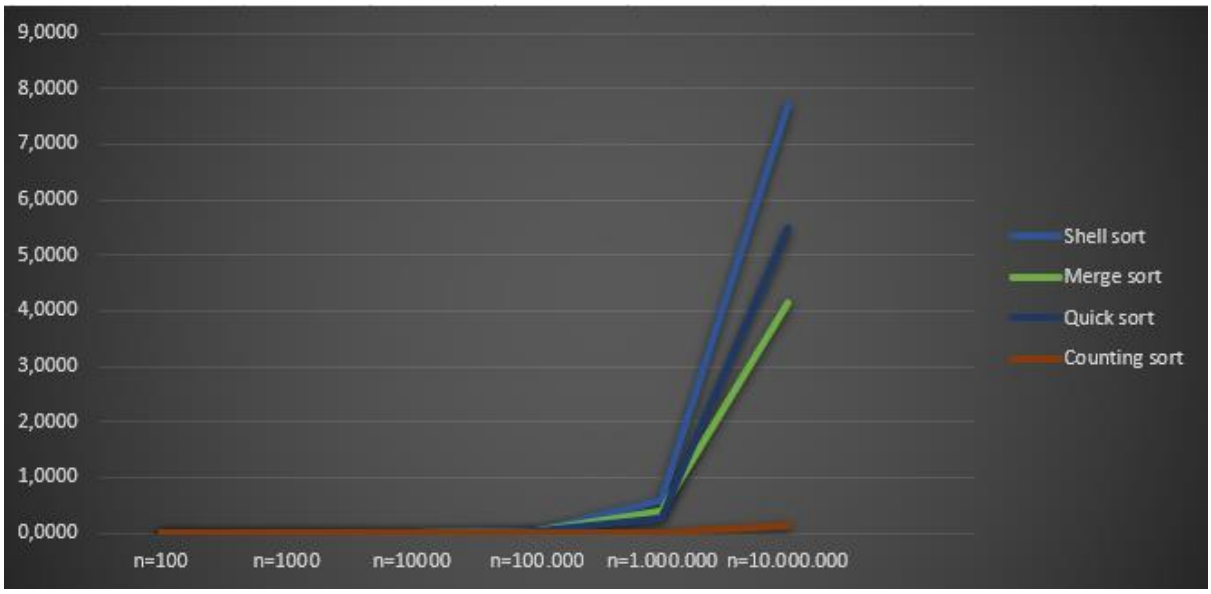
Sada kada se zna koje su mogućnosti kojih algoritama testirat će se složeniji algoritmi s većom količinom elemenata u nizu. Uz dva napredna algoritma sortiranja, u testiranje će se dodati distribucijsko sortiranje Counting sort i jednostavni algoritam sortiranja Shell sort jer su se pokazali vrlo dobri na prijašnjim testiranjima. Sljedeća tablica prikazuje vrijednosti vremenske složenosti u sekundama u prosječnom slučaju.

Raspon brojeva od 0 do 100.000

	Ulaz					
Algoritam	n=100	n=1000	n=10000	n=100.000	n=1.000.000	n=10.000.000
Shell sort	0,0002	0,0004	0,0031	0,0461	0,5776	7,6864
Merge sort	0,0004	0,0006	0,0039	0,0406	0,3795	4,1510
Quick sort	0,0003	0,0004	0,0020	0,0225	0,2507	5,4943
Counting sort	0,0010	0,0011	0,0028	0,0039	0,0153	0,1286

Tablica 3. Vremenska složenost bržih algoritama sortiranja u rasponu brojeva 0-100000

Iz tablice 3. vidimo da je Shell sort približno brz kao i Merge sort, Quicksort i Counting sort. Tek kod deset milijuna elemenata je počeo pokazivati nešto veću vremensku složenost. Counting sort se ovdje pokazao najbolji jer čak ni kod deset milijuna elemenata niza nije mu vremenska složenost došla ni do polovice sekunde. Quicksort je kod zadnjeg niza brojeva imao 5,4943 sekundi vremensku složenost, a za to je zaslužno ponavljanje istih elemenata jer ima svojstvo nestabilnosti kao i Shell sort kod kojeg je to isto zasigurno utjecalo na povećanje vremenske složenosti. U ovom testu je Quicksort također imao vrijednost za stožerni broj zadnji element niza.



Slika 22. Grafikon vremenske složenosti jednih od najbržih algoritama sortiranja u rasponu brojeva 0-100000

Na grafikonu sa slike 19. se može najbolje vidjeti koliko je ustvari Counting sort bio nevjerojatno brži od ostala tri algoritma sortiranja. On ima vremensku složenost u najgorem, najboljem i prosječnom slučaju O(n+k) što se samo može dodatno potvrditi ovom analizom.

8. Zaključak

Potrebno je znati koje su prednosti i mane svakog algoritma sortiranja kako bi se u programiranju moglo odabrati onaj koji najviše odgovara s obzirom na hardver s kojim se raspolaže, vrijeme na raspolaganju za sortiranje podataka i kolika je količina podataka. Bez znanja o bitnim pojedinostima algoritama sortiranja lako se može odabrati algoritam koji neće dovoljno brzo sortirati podatke kao npr. Bubble sort za sortiranje velike količine podataka. S druge strane Bubble sort bi bio dobra solucija za vrlo staro računalo kada se ima dovoljno vremena za sortiranje nekog niza. Zato i postoji više vrsta algoritama sortiranja kako bi se mogao izabrati onaj koji najviše odgovara za određeni slučaj kod programiranja, jer jedan algoritam sortiranja nikada ne može biti odličan odabir za svaki slučaj ili problem. Iz ovog rada se može zaključiti da je najbolje koristiti jednostavne algoritme sortiranja za relativno male nizove do tisuću elemenata. Za velike količine elemenata za sortiranje najbolje je koristiti napredene algoritme sortiranja kada nije na raspolaganju puno radne ili vanjske memorije. Kada se želi iskoristiti najveći potencijal brzine sortiranja nekih podataka, najbolji izbor su distribucijska sortiranja. Implementacija, odnosno njihovo samo programiranje naprednih algoritama sortiranja je zahtjevnija no s njima se dobiva odlična brzina u stvarnom vremenu do milijun elemenata, no za male količine elemenata je njihova implementacija prezahtjevna. Kod ogromnih količina elemenata su najprikladnija distribucijska sortiranja. Kada se treba sortirati niz do otprilike milijun elemenata može se koristiti i Shell sort ukoliko nije brzina izvršavanja izričito bitna.

Literatura

Internetski izvori

[1] Anany Levitin (2012). Introduction to the design and analysis of algorithms. Preuzeto 5.7.2021. s https://doc.lagout.org/science/0_Computer%20Science/2_Algorithms/Introduction%20to%20the%20Design%20and%20Analysis%20of%20Algorithms%20%283rd%20ed.%29%20%5BLevitin%202011-10-09%5D.pdf

[2] Mooc (bez dat.). Pojam algoritma. Preuzeto 5.7.2021. s https://mooc.carnet.hr/mod/book/view.php?id=26661&chapterid=7823

[3] RishabhPrabhu (2020). Introduction to Algorithms. Preuzeto 7.7.2021. s https://www.geeksforgeeks.org/introduction-to-algorithms/

[4] Wigmore, I. (bez dat.). sorting algorithm. Preuzeto 16.8.2021. s https://whatis.techtarget.com/definition/sorting-algorithm

[5] TechTarget Contributor (2019). algorithm. Preuzeto 7.7.2021. s https://whatis.techtarget.com/definition/algorithm

[6] Galler, L. i Kimura, M. (2019). Sorting Algorithms. Preuzeto 24.7.21. s https://lamfo-unb.github.io/2019/04/21/Sorting-algorithms/

[7] FreeCodeCamp (2020). Sorting Algorithms Explained. Preuzeto 24.7.2021. s https://www.freecodecamp.org/news/sorting-algorithms-explained/

[8] GeeksforGeeks (2018). Sorting Algorithms. Preuzeto 27.7.2021. s https://www.geeksforgeeks.org/sorting-algorithms/

[9] Barnett, G. i Tongo, L. D. (2008). Data Structures and Algorithms. Preuzeto 26.8.2021. s https://apps2.mdp.ac.id/perpustakaan/ebook/Karya%20Umum/Dsa.pdf

[10] Woltmann, S. (2020 a). Sorting Algorithms [Ultimate Guide]. Preuzeto 21.8.2021. s https://www.happycoders.eu/algorithms/sorting-algorithms/

[11] Woltmann, S. (2020 b). Big O Notation and Time Complexity – Easily Explained. Preuzeto 31.8.2021. s https://www.happycoders.eu/algorithms/big-o-notation-time-complexity/

[12] GeeksforGeeks (2021 a). Time Complexities of all Sorting Algorithms. Preuzeto 1.9.2021. s https://www.geeksforgeeks.org/time-complexities-of-all-sorting-algorithms/

[13] Maghsoudi, R. (2019). Sort It Out With Algorithms!. Preuzeto 1.9.2021. s https://towardsdatascience.com/sort-it-out-with-algorithms-6509d811781

[14] Anwar Naser Frak (2016). COMPARISON STUDY OF SORTING TECHNIQUES IN STATIC DATA STRUCTURE. Preuzeto 5.9.2021. s https://core.ac.uk/download/pdf/78469299.pdf

[15] FreeCodeCamp (2019 b). Sorting Algorithms Explained with Examples in Python, Java, and C++ . Preuzeto 5.9.2021. s https://www.freecodecamp.org/news/sorting-algorithms-explained-with-examples-in-python-java-and-c/

[16] Everything Computer Science (bez dat.). Sorting Algorithms. Preuzeto 5.9.2021. s https://everythingcomputerscience.com/algorithms/CSSorting_Algorithms.html

[17] Sandipan Das (2021). 10 Best Sorting Algorithms - Explained with Simple Examples. Preuzeto 5.9.2021. s https://www.crio.do/blog/top-10-sorting-algorithms/

[18] Klen, V. (2019). Qt GUI aplikacija za testiranje algoritama sortiranja. Preuzeto 8.9.2021. s https://repozitorij.etfos.hr/islandora/object/etfos%3A2390/datastream/PDF/view

[19] Seif, G. (2018). A tour of the top 5 sorting algorithms with Python code. Preuzeto 9.9.2021. s https://medium.com/@george.seif94/a-tour-of-the-top-5-sorting-algorithms-with-python-code-43ea9aa02889

[20] Makhija, N. (bez dat.). 6 Basic Different Types of Sorting Algorithms Explained in Detail. Preuzeto 11.9.2021. s https://www.csestack.org/different-types-sorting-algorithms/

[21] Scott Bronder (2019). Preuzeto 13.9.2021. s https://levelup.gitconnected.com/a-sort-of-all-sorting-algorithms-506cbc76d47

[22] Betterexplained (2006). Sorting Algorithms. Preuzeto 13.9.2021. s https://betterexplained.com/articles/sorting-algorithms/

[23] Mansi, R. (2010). A New Algorithm for Sorting Small Integers. Preuzeto 13.9.2021. s http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.184.9563&rep=rep1&type=pdf

[24] Alnihoud, J. i Mansi, R. (2010)An Enhancement of Major Sorting Algorithms. Preuzeto 13.9.2021. s https://ccis2k.org/iajit/PDF/vol.7,no.1/9.pdf

[25] Dhruv Punetha (2021). Cocktail Shaker Sort in C++ . Preuzeto 13.9.2021. s https://helloml.org/cocktail-shaker-sort-in-c/

[26] Ahmad H. Elkahlout1, Ashraf Y. A. Maghari (2017). A comparative Study of Sorting Algorithms Comb, Cocktail and Counting Sorting. Preuzeto 13.9.2021. s https://iugspace.iugaza.edu.ps/bitstream/handle/20.500.12358/25153/Maghari,%20Ashraf%20Y.%20A._23.pdf?sequence=1

[27] Đuranović, T. (2017). Usporedba algoritama sortiranja. Preuzeto 15.9.2021. s https://repozitorij.unipu.hr/islandora/object/unipu%3A2331/datastream/PDF/view

[28] Softpanorama Society (2020). Slightly Skeptical View on Sorting Algorithms. Preuzeto 26.9.2021. s http://www.softpanorama.org/Algorithms/sorting.shtml#Classification_of_sorting_algorithms

[29] Tutorialspoint (bez dat.). Data Structures - Asymptotic Analysis. Preuzeto 1.10.2021. s https://www.tutorialspoint.com/data_structures_algorithms/asymptotic_analysis.htm

[30] Ashwani, K. (2021). Complete Tutorial on big O (big oh) notation. Preuzeto 7.10.2021. s https://www.devopsschool.com/blog/complete-tutorial-on-big-o-big-oh-notation/

[31] Studytonight Technologies Pvt. Ltd. (bez dat.). Asymptotic Notations. Preuzeto 7.10.2021. s https://www.studytonight.com/data-structures/aysmptotic-notations

[32] Huang, S. (2020). What is Big O Notation Explained: Space and Time Complexity. Preuzeto 8.10.2021. s https://www.freecodecamp.org/news/big-o-notation-why-it-matters-and-why-it-doesnt-1674cfa8a23c/

[33] Great Learning Team (2020). Why is Time Complexity Essential and What is Time Complexity?. Preuzeto 10.10.2021. s https://www.mygreatlearning.com/blog/why-is-time-complexity-essential/#t7

[34] GeeksforGeeks (2021. b). External Sorting. Preuzeto 31.10.2021. s https://www.geeksforgeeks.org/external-sorting/

[35] GeeksforGeeks (2021. c). Application and uses of Quicksort. Preuzeto 6.11.2021. s https://www.geeksforgeeks.org/application-and-uses-of-quicksort/

[36] Parewa Labs Pvt. Ltd. (bez dat. a). Counting Sort Algorithm. Preuzeto 6.11.2021. s https://www.programiz.com/dsa/counting-sort

[37] Kavita Bisht (bez dat.)Radix Sort. Preuzeto 8.11.2021. s https://iq.opengenus.org/radix-sort/

Knjige

[38] Lovrenčić, A. (2018). Apstraktni tipovi podataka i algoritmi: Dio 1 Uvod u složenost algoritama i struktura podataka s primjerima pretraživanja i sortiranja. Varaždin: Sveučilište u Zagrebu, Fakultet organizacije i informatike

Popis Slika

Popis tablica