SVEUČILIŠTE U SPLITU
PRIRODOSLOVNO MATEMATIČKI FAKULTET

ZAVRŠNI RAD

VIZUALIZACIJA OSNOVNIH ALGORITAMA ZA SORTIRANJA

Mate Oljica

Mentor:

Doc.dr.sc. Ani Grubišić

Split, rujan 2014.

SADRŽAJ:

Uvod

Kao uvod u algoritme sortiranja proučit ćemo nekoliko osnovnih metoda koje su prikladne za sortiranje manjih nizova ili datoteka sa specijalnom strukturom. Postoji nekoliko razloga zbog kojih je dobro analizirati ove jednostavnije metode, npr., predstavljaju nam relativno jednostavan način da naučimo terminologiju i jednostavnije mehanizme sortiranja. Na taj način dobivamo dobru pozadinu za daljnje proučavanje složenijih algoritama.

Metode sortiranja koje ćemo obraditi u ovom završnom radu su :

- Selection sort,
- Insertion sort,
- Bubble sort,
- Merge sort i
- Quicksort .

Ovaj rad prikazuje mnoštvo rješenja što se tiče problema sortiranja podataka u jednodimenzionalnom polju (nizu) elemenata. U praksi se često pojavljuje potreba za sortiranjem podataka te se zbog toga traži što efikasniji i brži algoritam. U ovom radu detaljno je prikazano pet algoritama. Napravljene su i praktične izvedbe algoritama za sortiranje koje su testirane na različitim skupovima podataka.

U znanosti je problem sortiranja još uvijek aktualan i traže se bilo kakve mogućnosti ubrzanja. Međutim neki algoritmi sortiranja koji nas zanimaju spadaju u osnovnu vrstu algoritama, te su najjednostavniji i najsporiji algoritmi, kao što su primjerice bubble sort i selection sort, dok neki algoritmi spadaju u kompliciraniju vrstu algoritama poput quick sorta i merge sorta. Cilj ovog rada je vizualizacija tih algoritama sortiranja i usporedba dvije vrste, one najbrže i najsporije.

Osim vizualizacije algoritama slijedeća stvar koju moramo uzeti u obzir je prostorna složenost. Teoretski se algoritmi sortiranja dijele u tri skupine: algoritmi koji ne zahtijevaju dodatnu memoriju, algoritmi koji koriste vezane liste („linked list“) i zahtijevaju N dodatnih mjesta u memoriji za pokazivače i algoritmi kojima je potrebna dodatna memorija da čuvaju još jedan niz.

U sljedećem poglavlju objasniti ćemo neke osnovne pojmove kao što je algoritam i što je složenost algoritma, a nakon toga ćemo u još daljnijim poglavljima objasniti osnovne algoritme sortiranja.

Vizualizacija algoritama sortiranja je napravljena u Java programskom okruženju. Jasno je vidljivo kako pojedini algoritmi sortiranja rade i opcijom korak po korak možemo uz komentare pratiti svaki korak rada algoritama sortiranja.

1. Algoritam i složenost algoritma

Riječ "*algoritam*" dolazi od latinskog prijevoda imena iranskog matematičara Al‑Hvarizmija koji se bavio trigonometrijom, astronomijom, zemljopisom, kartografijom, a smatra se ocem algebre jer je definirao osnovna pravila rješavanja linearnih i kvadratnih jednadžbi. Njegovi radovi su osnova razvoja mnogih matematičkih i prirodnih disciplina, među njima i računarstva.(Daffa,1977)

Algoritam kao postupak za dobivanje rješenja, sastoji se od konačnog niza koraka – tzv. instrukcija, naredbi ili operacija, koje treba izvršiti (izvesti) da bi se dobilo rješenje.

Svaki algoritam ima slijedećih pet bitnih osnovnih svojstava:

- Ulaz,
- Izlaz,
- Konačnost,
- Definiranost i nedvosmislenost (određenost),
- Efikasnost (efektivnost).

Svaki od ovih svojstava ima direktne posljedice za formulaciju pojma složenosti algortima.

Složenost algoritma definirati ćemo neformalno kao maksimalni broj operacija potrebnih za izvršavanje pojedinog algoritma sortiranja.

Prvo uvodimo prepostavku da su sve (osnovne) operacije iste složenosti, s obzirom da nam je potreban samo njihov broj. Sa druge strane, broj operacija će svakako zavisiti od samog ulaza, odnosno koliko elemenata neki niz ima,u našem slučaju taj broj je 5. Iz tog razloga, kada se ispituje složenost nekog algoritma treba razmatrati “najgori mogući slučaj“, a najgori mogući slučaj je kad je neki niz elemenata sortiran uzlazno „ascending“ a mi tražimo silazno „descending“ sortirani niz.

Jednostavne ili osnovne operacije predstavljaju niz operacija kod kojih se vrijeme izvršavanja može ograničiti nekom konstantom koja zavisi samo od konkretne realizacije te operacije. Drugačije rečeno, pretpostavljamo da se svaka jednostavna operacija izvršava za jedinično vrijeme.

Tipične osnovne operacije su:

- dodijela vrijednosti varijabli
- uspoređivanje dvije varijable
- aritmetičke i logičke operacije
- ulazno / izlazne operacije

Kako je i sama gore zadana definicija složenosti još uvijek komplicirana i nije je lako razumjeti, moramo ignorirati još neke faktore. Radi tog razloga uvodimo novo pravilo po kojem zanemarujemo konstante. Složenost će zavisiti isključivo od ulaznih veličina, dok ćemo konstante koje su “mnogo” manje od ulaznih ograničenja zanemariti.(Singer,2005.)

2. Algoritmi za sortiranje

Algoritmi sortiranja su algoritmi koji postavljaju elemente niza u određeni redoslijed. Najviše korištena sortiranja su brojevna sortiranja i leksikografska sortiranja. Učinkoviti algoritam sortiranja je vrlo važno za optimizaciju gdje se zahtijeva da ulazna lista bude u sortiranom redu. (Demuth, 1956). Izlaz algoritma sortiranja mora zadovoljiti dva uvjeta:

- Izlaz je permutacija ulaza
- Izlaz je u „nondecreasing“ pravilu

2.1. Selection sort

U računalnoj znanosti, selection sort je vrsta algoritma za sortiranje, posebno je bitan u sortiranju uspoređivanjem. Selection sort svojevrsno je poznat po svojoj jednostavnosti, i to ima svoje prednosti nad složenijim algoritmima u određenim situacijama, osobito ondje gdje je pomoćna memorija ograničena.

Selection sort algoritam dijeli uneseni niz u dva dijela: podniz predmeta koji su već sortirani, koji je izgrađen s lijeva na desno iz niza, a podniz preostalih predmeta će biti riješen tako da zauzme ostatak niza. U početku, sortirani podniz je prazan, a nesortiran niz je zapravo cijeli niz kojeg dovodimo na ulaz.

Algoritam se izvršava na principu pronalaženja najmanjeg elementa u nerazvrstanom nizu kojeg smo doveli na ulaz, te kad ga pronađe zamjeni ga s elementom koji je skroz lijevo u nesortiranom nizu (stavljajući ga u položaj koji je fiksiran i već sortiran), nakon toga pomiće se granica podniza za jedan element u desno jer je onaj prvi već sortiran. (Donald Knuth,1997)

Primjer: Sljedeća Slika 1. prikazuje korake za sortiranje niza pomoću selection sorta. Zadani niz je (3, 7, 4, 9, 5, 2, 6, 1). U svakom koraku, element u razmatranju je podvučen. Dio koji je pomaknut (ili ostavljen na mjestu zato što je najveći dotad razmatrani) u prethodnom koraku biti će podebljan.

3 7 4 9 5 2 6 **1**

1 7 4 9 5 **2** 6 3

1 2 4 9 5 7 6 **3**

1 2 3 9 5 7 6 **4**

1 2 3 4 **5** 7 6 9

1 2 3 4 5 7 **6** 9

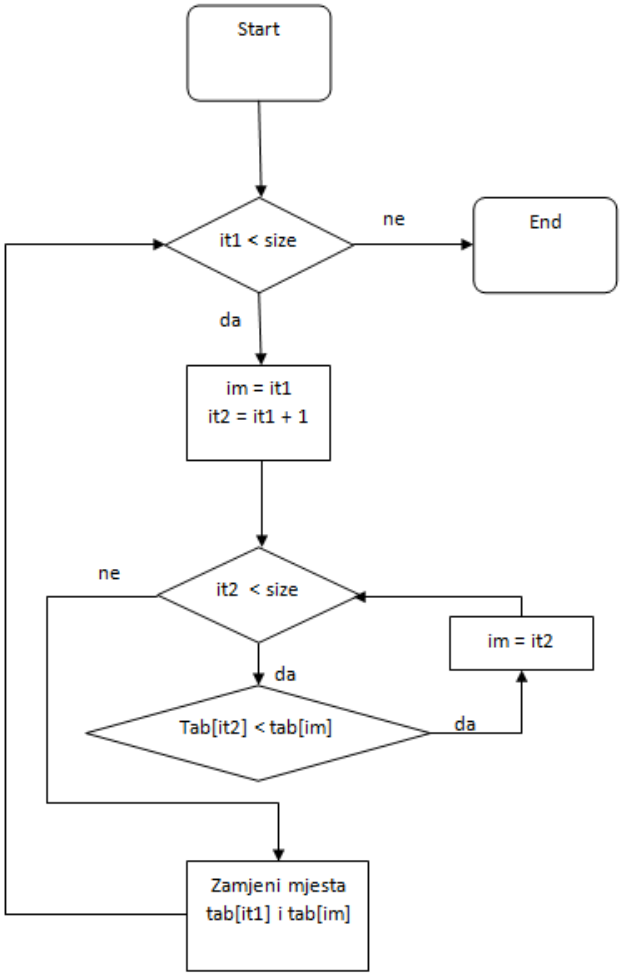
1 2 3 4 5 6 7 9

1 2 3 4 5 6 7 **9**

1 2 3 4 5 6 7 9

Slika 1. Princip rada Selection sorta

Nakon objašnjenog principa rada Selection sorta Slika 2. prikazuje dijagram toka podataka istog algoritma.



Slika 2. Dijagram toka podataka Selection sorta

2.1.1. Složenost Selection sorta

Selection sort ima $O(n^2)$ vremensku složenost, što ga čini neučinkovitim na sortiranje s velikim nizovima, i općenito je čak lošiji i od sličnog njemu insertion sort-a. Selection sort nije teško analizirati u odnosu na druge algoritme sortiranja. Odabere se najmanji element niza koji zahtijeva skeniranje svih n elemenata niza (to traje n - 1 usporedbe), a zatim ga postavi na prvu poziciju. Pronalaženje sljedećeg najmanjeg elementa koji zahtijeva skeniranje preostalih n - 1 elementa i tako dalje, za $(n - 1) + (n - 2) + ... + 2 + 1 = n (n - 1) / 2 \in \Theta (n^2)$ usporedbe. Svaka od ovih usporedbi zahtijeva jednu zamjenu za n - 1 elemenata (konačni element je već na mjestu).

2.2. Insertion sort

Sortiranje umetanjem (engl. Insertion sort) je jednostavan algoritam za sortiranje, koji gradi završni sortirani niz jednu po jednu stavku. Mnogo je manje efikasan na većim listama od mnogo složenijih algoritama kao što su quicksort, heapsort ili mergesort. (Sedgewick, 1983).Međutim sortiranje umetanjem ima svoje prednosti:

- Jednostavna primjena
- Efikasan na malim skupovima podataka
- Prilagodljiviji za skupove podataka koji su vec značajno sortirani: Vrijeme kompleksnosti je O(n+d), gdje je d broj inverzija
- Efikasniji u praksi od većine drugih kvadratnih (tj. O (n^2)) algoritama, kao što su selection sort ili bubble sort
- Stabilan tj. ne mijenja relativni redoslijed elemenata sa jednakim vrijednostima
- U mjestu (tj. zahtijeva samo konstantan iznos O (1) dodatnog memorijskog prostora)
- Trenutan (online, može sortirati listu, odmah pri primanju)

Primjer: Sljedeća Slika 3. prikazuje korake za sortiranje niza (3, 7, 4, 9, 5, 2, 6, 1). U svakom koraku, element u razmatranju je podvučen. Dio koji je pomaknut (ili ostavljen na mjestu zato što je najveći dotad razmatrani) u prethodnom koraku biti će podebljan.

3 7 4 9 5 2 6 1

3 7 4 9 5 2 6 1

3 **7** 4 9 5 2 6 1

3 **4** 7 9 5 2 6 1

3 4 7 **9** 5 2 6 1

3 4 **5** 7 9 2 6 1

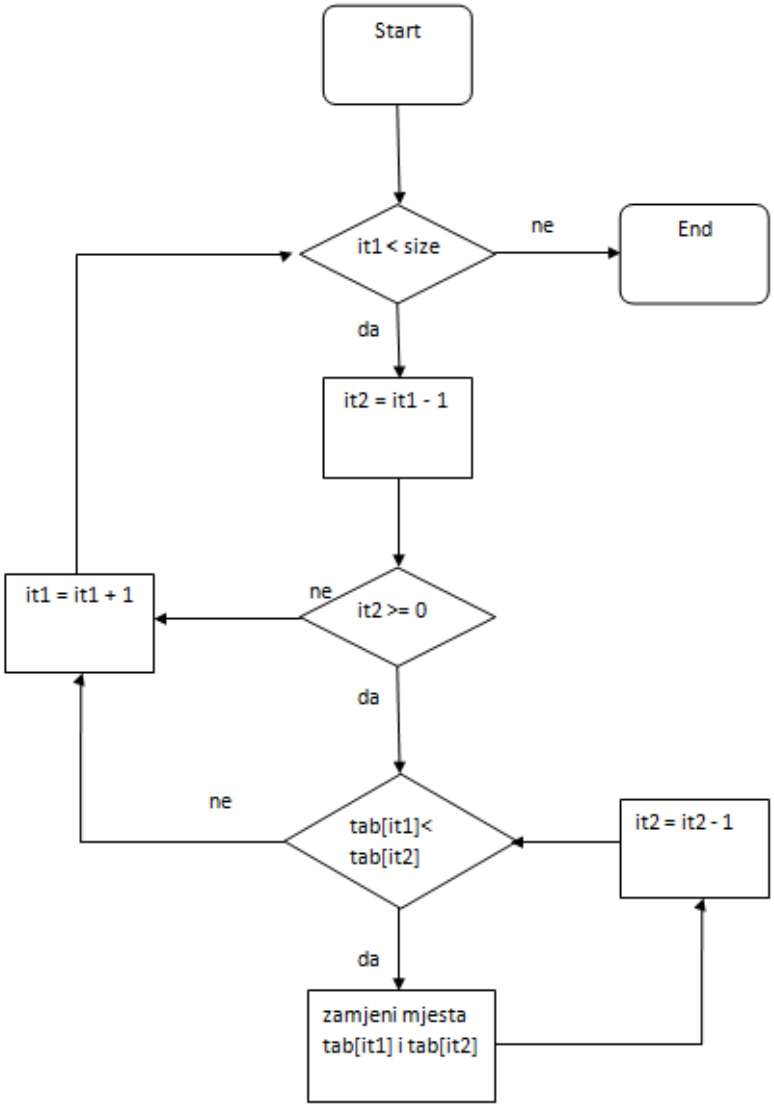
2 3 4 5 7 9 6 1

2 3 4 5 **6** 7 9 1

1 2 3 4 5 6 7 9

Slika 3. Princip rada Insertion sorta

Nakon objašnjenog principa rada Insertion sorta Slika 4. prikazuje dijagram toka podataka istog algoritma.



Slika 4. Dijagram toka podataka Insertion sorta

2.2.1. Složenost Insertion sorta

Sortiranje umetanjem uzima jedan ulazni element pri svakom prolasku i povećava sortiranu izlaznu listu. Pri prolasku, sortiranje umetanjem uklanja jedan element iz ulaznih podataka, pronalazi mjesto gdje pripada taj element u sortiranoj listi i stavlja ga tamo. Ponavlja prolaske sve dok ne ostane nijedan ulazni element.

Sortiranje se obično obavlja u mjestu, prolazeći kroz niz, povećavanjem sortirane liste. Na svakom prolasku kroz niz, provjerava vrijednost ulaznog podatka, uspoređujući ga sa najvećom vrijednošću niza u posljednjoj provjeri. Ako je veći ostavlja element na mjestu i prelazi na sljedeći, ako je manji nađe odgovarajuću poziciju u nizu, pomiče sve veće vrijednosti da napravi prostor i ubacuje element na ispravno mjesto.

Rezultirajući niz nakon k iteracija predstavlja sortirane prvih k+1 („+1“ zato što se prvi unos preskače) ulaznih elemenata. U svakoj iteraciji, prvi preostali unos iz ulaznih podataka je uklonjen i ubačen u rezultirajući niz na ispravno mjesto, čime produžava rezultat.

Najčešće varijante sortiranja umetanjem, koji radi sa nizovima, može se opisati kao:

1. Pretpostavimo da postoji funkcija Insert koja ubacuje vrijednost u sortiranu sekvencu na početku niza. Počinje rad sa kraja sekvence i pomiče elemente jedno mjesto udesno, dok ne nađe odgovarajuće mjesto za novi element. Popratna pojava ove funkcije je uništavanje prvog sljedećeg člana iz nesortiranih ulaznih elemenata.
2. Da bi se izvršilo sortiranje umetanjem, potrebno je uzimati „najlijeviji“ element ulaznog niza i pozivati funkciju Insert. Ovim je osigurano da funkcija uvijek upisuje preko „najlijevijeg elementa“ što je u stvari element koji se sortira.

Najbolji slučaj je kada je unos već sortiran niz, u ovom slučaju sortiranje umetanjem ima linearno vrijeme rada (tj. O(n)). Tokom svake iteracije, prvi preostali element ulaznog niza se samo uspoređuje sa zadnjim elementom sortiranog niza.

Najjednostavniji najgori slučaj je kada je niz u obrnutom redosljedu. Skup svih najgorih slučajeva ulaza sastoji se od nizova gdje je svaki element najmanji ili drugi najmanji od elemenata prije njega. U ovim slučajevima svaka iteracija unutrašnje petlje će proći kroz svaki sortirani element i pomaknuti ga, prije nego što ubaci sljedeći element.

To daje sortiranju umetanjem kvadratno vrijeme izvršavanja (O (n^2)). Prosječan slučaj ima također kvadratno vrijeme izvršavanja, što čini sortiranje umetanjem nepraktičnim za sortiranje većih nizova.

Međutim, sortiranje umetanjem je jedan od najbržih algoritama za sortiranje vrlo malih nizova, čak brži od quicksort-a.

2.2.2. Varijante Insertion sorta

D. L. Shell je napravio značajna unapređenja algoritma. Unaprijeđena verzija se zove Shell sort. Ovaj algoritam usporedi elemente na rastojanju koje se povećava sa svakim novim prolazom. Shell sort je primjetno poboljšao vremena izvršavanja, sa dvije jednostavne varijante koje zahtijevaju O($n^{3/2}$) i O ($n^{4/3}$) vremena za izvršavanje. Ukoliko troškovi uspoređivanja premašuju troškove zamjene, kao što je slučaj sa stringovima sačuvanim po referenci ili kada je potrebna ljudska interakcija, onda primjena binarnog sortiranja umetanjem može dati bolje rezultate. Binarno sortiranje umetanjem upotrebljava binarnu pretragu da odredi lokaciju za ubacivanje novog elementa i zbog toga izvršava O $\lceil \log_2(n) \rceil$ uspoređivanja, a u najgorem slučaju O(n log(n)). Cijeli algoritam i dalje ima prosječno O(n^2) vrijeme izvršavanja, jer su serije zamjena neophodne za svako pojedinačno ubacivanje.

Broj zamjena može biti smanjen i računanjem pozicija više elemenata prije njihovog pomicanja. Npr. ako se pozicija dva elementa proračuna prije pomicanja, broj zamjena može biti smanjen do 25% za proizvoljne podatke. U ekstremnim slučajevima ova varijanta radi slično kao mergesort. Da bi se izbjeglo pravljenje serija zamjena, ulazni podaci mogu biti tipa ulančane liste, što bi omogućilo elementima da budu spojeni u listu u konstantnom vremenu, kada je pozicija liste poznata. Ipak, pretraga uvezane liste zahtjeva praćenje veza do tražene pozicije, pa se ne mogu koristiti metode poput binarne pretrage. Zbog toga vrijeme potrebno za pretragu je O (n) i O (n^2) za sortiranje. Pri upotrebi sofisticiranijih struktura podataka (npr. heap (gomila) - ili binarna stabla), vrijeme potrebno za pretragu i ubacivanje može biti značajno smanjeno. Ovo predstavlja osnovnu heap sort –a i binary tree sort – a.

Dvije tisuće i četvrte Bender, Farah-Colton i Mosteiro objavljuju novu varijantu sortiranja umetanjem nazvanu library sort. Ova metoda ostavlja mali broj neiskorisćenih mjesta kroz čitav niz. Ovim se postiže da se pri umetanju elementi pomiću samo do praznine. To omogućuje da vrijeme izvršenja bude O (n log n) sa velikom verovatnoćom.

Pri upotrebi uvezanih listi vrijeme umetanja je svedeno na O(log n), i zamjene nisu potrebne. Konačno vrijeme za čitav algoritam je O (n log n). Sortiranje umetanjem liste je varijanta sortiranja umetanjem koja smanjuje broj poteza u algoritmu.

2.3. Merge sort

Sortiranje spajanjem (engl. merge sort) je algoritam sortiranja baziran na uspoređivanju složenosti O(n log n). Vecina implementacija daje stabilan niz što znači da implementacija čuva redosljed jednakih elemenata sa ulaza. Sortiranje spajanjem je zavadi-pa-vladaj algoritam koji je izmislio John von Neumann 1945-te. (Katajainen and Larsson,1977)

Detaljan opis i analiza sortiranja spajanjem odozdo-nagore su se pojavili u izvještaju koji su napisali Goldstine i Neumann još davne 1948-e.

Konceptualno, sortiranje spajanjem radi po sljedećem principu:

- Podijeliti nesortiran niz na n podnizova, od kojih svaki sadrži 1 element (niz od jednog elementa se smatra sortiranim).
- U više navrata spajati podnizove sve dok se ne dobije novi podniz. Ovo ce biti sortiran niz.

Primjer: Sljedeća Slika 5. prikazuje korake za sortiranje niza (3, 7, 4, 9, 5, 2, 6, 1). U svakom koraku, element u razmatranju je podvučen. Dio koji je pomaknut (ili ostavljen na mjestu zato što je najveći dotad razmatrani) u prethodnom koraku biti će podebljan.

3 | 7 | **4** | 9 | 5 | **2** | 6 | **1**

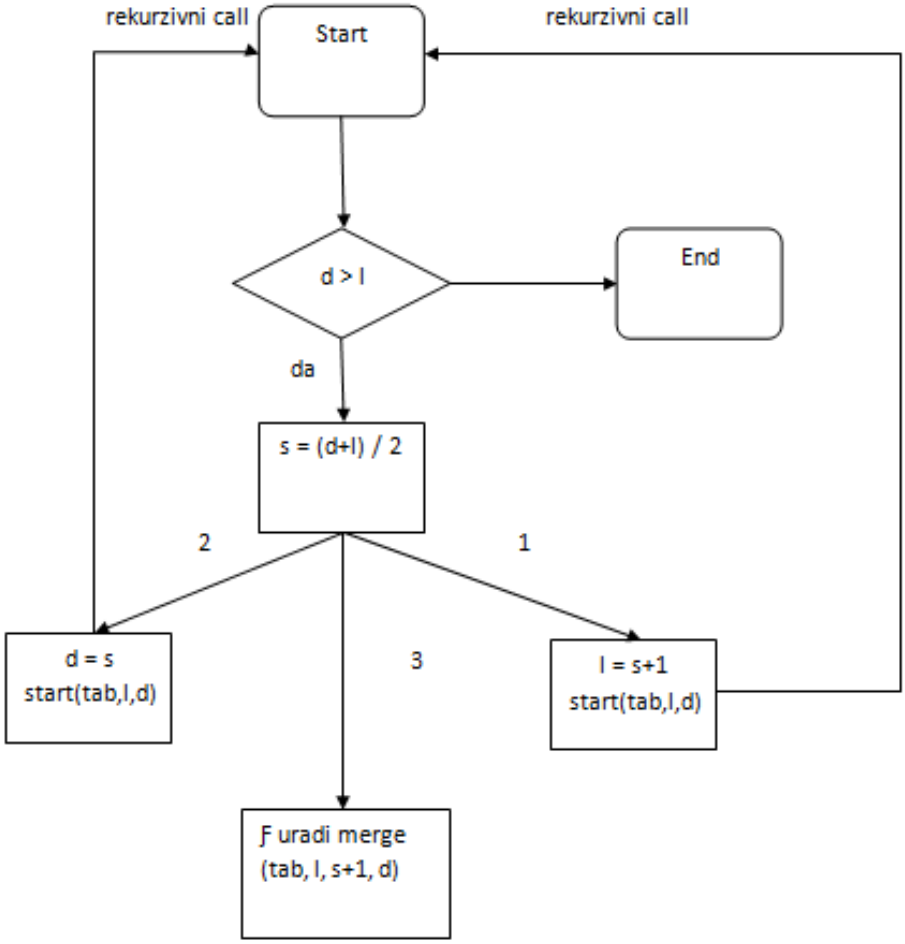
3 7 | 4 9 | 2 **5** | **1** 6

3 **4** 7 **9** | **1** **2** **5** **6**

1 2 3 4 5 6 7 9

Slika 5. Princip rada Merge sorta

Nakon objašnjenog principa rada Merge sorta Slika 6. prikazuje dijagram toka podataka istog algoritma.



Slika 6. Dijagram toka podataka Merge sorta

2.3.1. Složenost Merge sorta

Za sortiranje n elemenata, složenost u prosječnom i najgorem slucaju je $O(n \log n)$. Ako je vrijeme izvršavanja za niz dužine $nT(n)$ onda relacija $T(n) = 2R(n/2) + n$ sljedi iz definicije algoritma (primjeni algoritam na dva niza čija je dužina polovina originalnog niza i dodaj n koraka za spajanje dva dobijena niza). Ovo sljedi iz Master teorema.

U najgorem slučaju, broj uspoređivanja je manji ili jednak $(n \lg n - 2^{\lg n} + 1)$, što je između $(n \lg n - n + 1)$ i $(n \lg n + n + O(\lg n))$. Za veliko n i slučajno sortiran ulazni niz, očekivani broj uspoređivanja je $a \cdot n$ manji nego u najgorem slučaju, gdje je:

$$\alpha = -10 + \sum_{k=0}^{\infty} \frac{1}{2^k + 1} \approx 0.2645 \qquad (1)$$

U najgorem slučaju, kod sortiranja spajanjem imamo 39% manje uspoređivanja nego što imamo kod quicksorta u prosječnom slučaju. U broju koraka, složenost u najgorem slučaju iznosi O (n log n) - ista složenost kao u najboljem slučaju quicksort-a, a najbolji slučaj sortiranja spajanjem ima upola manje uspoređivanja nego njegov najgori slučaj. Najčešća implementacija sortiranja spajanjem koristi pomoćni niz za koji je potrebno alocirati prostor u memoriji.

Stabilno sortiranje u mjestu je moguće ali kompliciranije i obično sporije, čak i ako algoritam također radi u vremenu O(n log n). Jedan način sortiranja u mjestu je da se podnizovi spoje rekurzivno. Kao i standardno sortiranje spajanjem, sortiranje u mjestu je također stabilno sortiranje. Stabilno sortiranje povezanih listi je jednostavnije. U ovom slučaju, algoritam ne koristi više prostora od onoga što se već koristi za predstavljanje liste, ali koristi O(log(k)) prostora za rekurzivne pozive.

Sortiranje spajanjem je efikasnije nego quicksort za neke vrste nizova ako se podacima koje treba sortirati može efikasno prići jedino sekvencijalno i zbog toga je popularan u jezicima kao što je Lisp, gdje su strukture podataka kojima se pristupa sekvencijalno veoma uobičajene. Za razliku od nekih (efikasnih) implementacija quicksorta sortiranje spajanjem je stabilno sortiranje sve dok se operacija spajanjem primjenjuje kako treba. Sortiranje spajanjem ima i neke nedostatke. Jedan od njih je korištenje 2n lokacija, dodatnih n lokacija koje se koriste zato što je spajanje 2 sortirana niza u mjestu kompliciranije i potrebno je više uspoređivanja i pomicanja. Usprkos korištenju ovog

prostora, algoritam i dalje obavlja dosta posla: sadržaj m se prvo kopira u left i right a kasnije u niz result za svaki poziv merge_sort (imena promjenljivih se odnose na pseudokod). Alternativa ovom kopiranju je da se poveže novo polje informacije sa svakim ključem (elementi u m se zovu ključevi). Ovo polje će biti korišteno kako bi se povezali ključevi i informacije u sortiranu listu (ključ i informacije vezane za njega se zovu ploče). Onda sljedi spajanje sortiranih listi koje se odvija mjenjanjem vrijednosti veze, nijedna ploča se ne mora pomicati. Polje koje sadrži samo vezu će generalno biti manje u odnosu na čitavu ploču tako da će i manje prostora biti upotrebljeno.

Druga alternativa za smanjenje prostora na n/2 jeste da se održe left i right kao kombinirana struktura, da se kopira samo left dio niza m u pomoćni niz i da se preusmjeri merge rutina kako bi se rezultat spajanja smjestio u m. Sa ovom verzijom bolje je alocirati pomoćni niz izvan merge rutine, tako da je potrebna samo jedna alokacija. Suvišno kopiranje spomenuto u prethodnom dijelu je također smanjeno.

Sortiranje spajanjem se može izvršiti spajanjem više od dva podniza istovremeno koristeći n-way algoritam spajanja. Ipak, broj operacija je približno isti. Uzmimo na primjer spajanje k podnizova istovremeno gde je zbog jednostavnosti k stupanj dvojke. Relacija postaje $T(n) = k\ T(n/k) + O(n \log k)$. (Posljednji dio proizilazi iz algoritma spajanja, kome je ako se primjeni optimalno koristeći hip ili balansirano binarno stablo potrebno O (log k) vremena po elementu.) Ako promatramo rekurzivnu relaciju za obično sortiranje spajanjem ($T(n) = 2T(n/2) + O(n)$) i proširimo je $\log_2 k$, dobijemo istu relaciju. Ovo je točno i ako k nije konstanta.

2.3.1.1. Upotreba sa trakama

Sortiranje je praktično za rad sa diskovima i trakama kada su podaci koje treba sortirati preveliki da stanu u memoriju. Tipičan sort koristi četiri trake. Sav Ulaz/Izlaz je sekvencijalan. Minimalna implementacija je moguća korišćenjem 2 buffera i nekoliko promenljivih.

Ako četiri trake označimo sa A, B, C, D, sa originalnim podacima na A, koristimo samo 2 buffera, tada je algoritam sličan implementaciji algoritma odozdo-nagore, i koristimo parove traka umjesto nizova u memoriji. Opis algoritma glasi:

- Spojiti podatke sa trake A; pišući podliste sa dva podatka naizmjenično na C i D
- Spojiti podliste sa dva podatka sa C i D u podliste sa četiri podatka; pišući naizmjenično na A i B

- Spojiti podliste sa četiri podatka sa A i B u podliste sa osam podataka; pišući naizmjenično na C i D
- Ponavljati dok se ne dobije jedna lista koja sadrži sve podatke, sortirana za log2(n) prolazaka.

Umjesto čitanja malo podataka, prvi prolaz će učitati dosta podataka u memoriju, napraviti će unutrašnje sortiranje i na taj način omogućiti učitavanje velikog broja podataka i onda ih prosljediti na izlaz. Ovim korakom se izbjegavaju rani prolasci.

Na primjer, unutrašnje sortiranje 1024 podataka će uštediti 9 prolazaka. Unutrašnje sortiranje je često veliko jer ima takvu prednost. Zapravo, postoje tehnike kojima se mogu produžiti početni prolasci u odnosu na dostupnu internu memoriju. Sofisticiranije sortiranje spajanjem koje optimizira prostor traka i diskova je uz pomoć sortiranja polifaznim spajanjem.

2.3.1.2. Optimizacija merge sorta

Na suvremenim računalima, lokalitet reference može biti od najveće važnosti u softverskoj optimizaciji, jer se koristi memorijska hijerarhija sa više nivoa. Verzije temeljene na korištenju hash-a algoritma sortiranja spajanjem, čije su operacije posebno izabrane kako bi smanjile pomicanje stranica u i iz hash memorije računala su bile predložene.

Na primjer, tiled merge sort algoritam prestaje dijeliti podnizove kada se dostignu podnizovi veličine S, gde je S broj elemenata podataka koji staju u hash procesor. Svaki od ovih podnizova se sortira koristeći algoritam za sortiranje koji radi u mjestu kao što je sortiranje umetanjem, da bi se izbjegle razmjene u memoriji, i onda se kompletira normalno sortiranje spajanjem na standardni rekurzivni način. Ovaj algoritam je pokazao bolje performanse na računalima koje imaju korist od optimizacije hash-a.

Kronrod (1969) je predložio alternativnu verziju sortiranja spajanjem koja koristi konstantni dodatni prostor. Ovaj algoritam je kasnije poboljšan. Također, mnoge aplikacije sortiranja koriste formu sortiranja spajanjem pri čemu se ulaz dijeli na veći broj podlista, u idealnom slučaju na onaj broj za koji važi da bi njihovim spajanjem setovi strana koji se trenutno obrađuju stali u glavnu memoriju.

2.4. Bubble sort

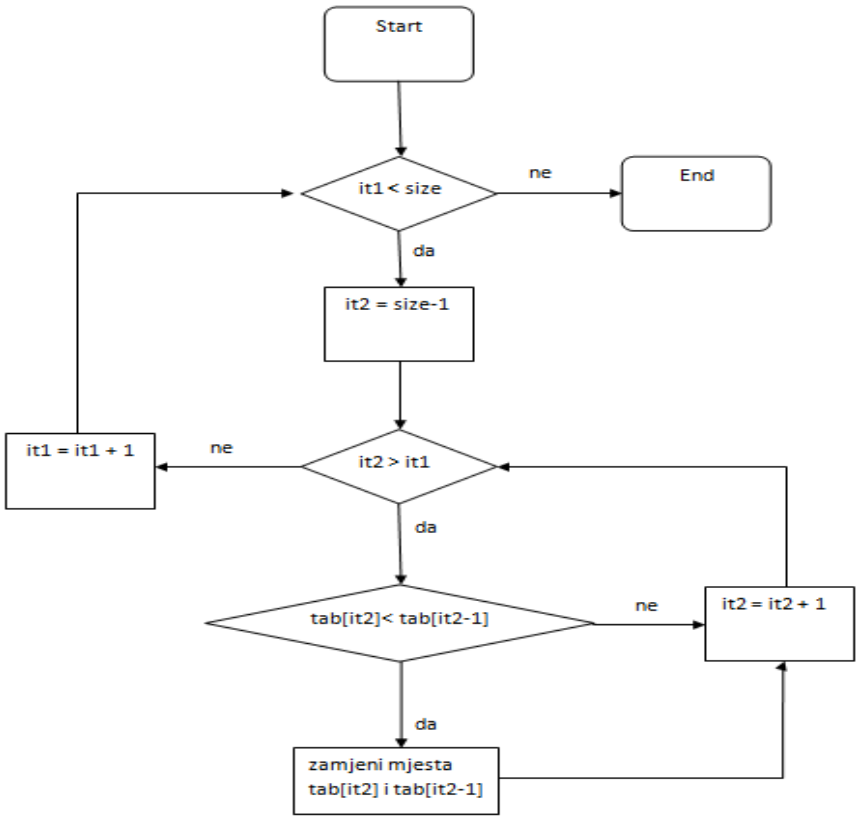
Sortiranje mjehurom (engl. Bubble sort), ponekad pogrešno nazivan sinking sort, je jednostavan algoritam za sortiranje koji radi tako što više puta prolazi kroz niz koji treba biti sortiran i uspoređuje se svaki par susjednih elemenata. Elementi zamjenjuju mjesta ako su napisani pogrešnim redoslijedom. Prolaz kroz niz se ponavlja sve dok se ne izvrše sve potrebne zamjene, što ukazuje da je niz sortiran. Algoritam je dobio ime zbog načina na koji najmanji element „bubble“ dolazi na početak niza. Pošto se samo uspoređuju elementi, ovo je komparativno sortiranje. Iako je ovo jednostavan algoritam, većina drugih algoritama za sortiranje su efikasniji za velike nizove.(Knuth, 1998)

Primjer: Sljedeća Slika 7. prikazuje korake za sortiranje niza (3, 7, 4, 9, 5, 2, 6, 1). U svakom koraku, element u razmatranju je podvučen. Dio koji je pomaknut (ili ostavljen na mjestu zato što je najveći dotad razmatrani) u prethodnom koraku biti će podebljan.

3 7 4 9 5 2 6 1
3 7 **4** 9 5 2 6 1
3 4 **7** 9 5 2 6 1
3 4 7 9 **5** 2 6 1
3 4 7 5 9 **2** 6 1
3 4 7 5 2 9 **6** 1
3 4 7 5 2 6 9 **1**
3 4 7 **5** 2 6 1 9
3 4 5 7 **2** 6 1 9
3 4 5 2 7 **6** 1 9
3 4 5 2 6 7 **1** 9
3 4 5 **2** 6 1 7 9
3 4 2 5 6 **1** 7 9
3 4 **2** 5 1 6 7 9
3 2 4 5 **1** 6 7 9
3 **2** 4 1 5 6 7 9
2 3 4 **1** 5 6 7 9
2 3 **1** 4 5 6 7 9
1 3 **2** 4 5 6 7 9
1 2 3 4 5 6 7 9

Slika 7. Princip rada Bubble sorta

Nakon objašnjenog principa rada Bubble sorta Slika 8. prikazuje dijagram toka podataka istog algoritma.



Slika 8. Dijagram toka podataka Bubble sorta

2.4.1. Složenost Bubble sorta

Bubble sort vrsta sortiranja ima najgoru složenost koja je dana sa $O(n^2)$, gdje je n broj elemenata koji se sortiraju, kao što je vidljivo i na Slika10. Postoji puno drugih algoritama za sortiranje koji imaju znatno bolju složenost O(n log n) od samog Bubble sort-a. Čak i drugi algoritimi sortiranja složenosti $O(n^2)$, kao što je insertion sort, imaju tendenciju da budu što bolji i da imaju što bolje performanse od bubble sort-a. Dakle, bubble sort nije toliko praktičan algoritam za sortiranje ako je n veliki, međutim u našem slučaju taj n je 5 tako da ne predstavlja veliku razliku korisiti bubble ili insertion sort.

Jedina značajna prednost koju ima bubble sort za razliku od drugih vrsta sortiranja, čak i quicksorta, ali ne i insertion sorta, je sposobnost otkirvanja da je sortirani niz efikasno

ugrađen u algoritam. Složenost bubble sorta kad imamo na ulazu sortirani niz (što je ujedno i najbolji slučaj) je O(n) .

Suprotno ovome, većina drugih algoritama sortiranja, čak i oni sa boljom prosječnom složenošću i koji imaju puno bolje performanse od samog buble sorta, obavljaju cijeli proces sortiranja na setu i na taj način su složeniji i zahtjevniji. Međutim, sami insertion sort ne da ima ovaj mehanizam koji smo opisali, već i puno bolje radi na nizu koji je znatno sortiran (mali broj inverzija).

Pozicije elemenata u bubble sortu igraju veliku ulogu u određivanju složenosti samog algoritma. Veliki elementi na početku niza ne predstavljaju problem jer se brzo zamjene. Mali elementi pri kraju niza se kreću na početak veoma sporo. Zbog toga se ove vrste elemenata respektivno nazivaju zečevi i kornjače.

Učinjeni su razni napori da se eliminiraju elementi niza koje zovemo kornjače kako bi se poboljšala brzina bubble sorta jer je to za programere izgledalo jako spor način sortiranja. Cocktail sort je dvosmjerni bubble sort koji ide od početka do kraja, a onda se poništava i ide od kraja do početka.

On može pomicati kornjače prilično brzo, ali je zadržao onu istu složenost $O(n^2)$. Comb sort još jedna vrsta poboljšanog bubble sorta, usporedi elemente razdvojene velikim prazninama, a može pomicati kornjače izuzetno brzo prije nego što pređe na manje praznine.

Sa ovim načinom sortiranja programeri su dobili jako puno na brzini samog algoritma sortiranje nazvanog Comb sort, a ta brzina se može usporediti sa brzinom quick sorta.

2.5. Quicksort

Quicksort (engl. Quicksort, qsort; od engl. quick, brz, brzo; i engl. sort, sortiranje) je poznat algoritam sortiranja koji radi na principu podijeli pa vladaj, a razvio ga je Toni Hor 1960. godine.

U prosjeku, čini O (n log n) usporedbi kako bi sortirao n stavki. U najgorem slučaju, čini O (N2) usporedbe, iako je takvo ponašanje quicksort-a rijetkost. Quicksort je u praksi često brži od drugih O (n log n) algoritama. Osim toga, dobro radi s cach memorijom. Quicksort

je sortiranje uspoređivanjem i, u učinkovitim implementacijama, nije stabilna vrsta sortiranja. ("Data structures and algorithm: Quicksort". Auckland University.)

Sljedi opis rada algoritma koji elemente sortira u rastućem poretku. Osnovni princip rada algoritma se dijeli u tri sljedeće cjeline:

- Izabiranje pivot-elementa na datom intervalu
- Raspored svih elemenata manjih ili jednakih ovom pivot-elementu lijevo od njega, a svih većih desno od njega u nizu
- Rekurzivno ponavljanje ovog postupka na novonastale intervale lijevo i desno od ovog pivot-elementa

Primjer: Sljedeća Slika 9. prikazuje korake za sortiranje niza (3, 7, 4, 9, 5, 2, 6, 1). U svakom koraku, element u razmatranju je podvučen. Dio koji je pomaknut (ili ostavljen na mjestu zato što je najveći dotad razmatrani) u prethodnom koraku biti će podebljan.

Izaberemo pivota = 1

| 3 7 4 9 5 2 6 **1**

Novi pivot = 3

1 | 7 4 9 5 **2** 6 **3**

1 2 | 7 4 9 5 6 **3**

Novi pivot = 6

1 2 3 | 7 **4** 9 5 **6**

1 2 3 4 | 7 9 **5** **6**

1 2 3 4 5 | 7 9 **6**

Novi pivot = 9

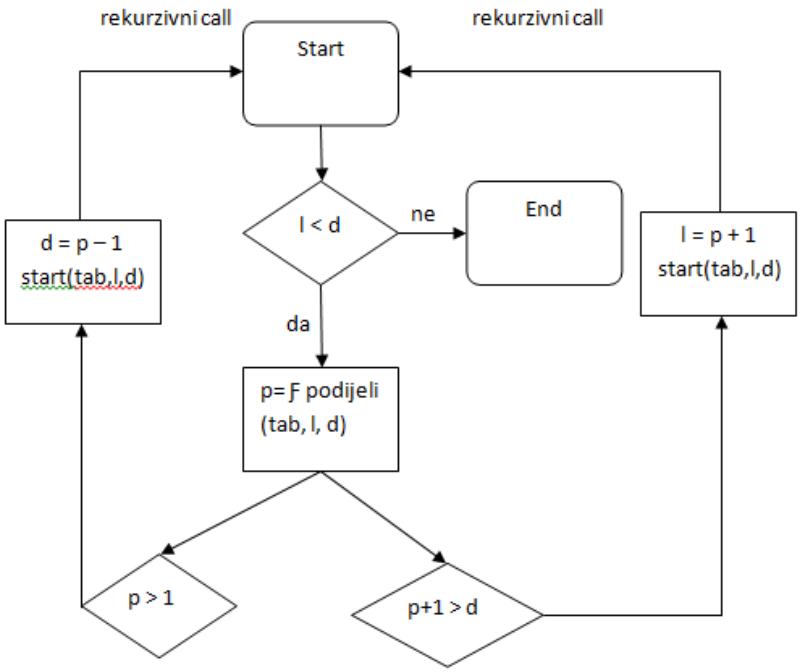
1 2 3 4 5 6 | 7 **9**

1 2 3 4 5 6 7 | **9**

1 2 3 4 5 6 7 9

Slika 9. Princip rada Quicksorta

Nakon objašnjenog principa rada Quicksorta Slika 10. prikazuje dijagram toka podataka istog algoritma.



Slika 10. Dijagram toka podataka Quicksorta

2.5.1. Složenost Quicksorta

Algoritam na početku dobije niz i lijevu i desnu granicu intervala koga treba sortirati. Za ove granice mora važiti da lijeva ima manji indeks od desne, inače se algoritam prekida. Unutar tog intervala, algoritam izabere tzv. pivot-element, koji se obično uzima sa njegove sredine ili njene okoline. Potom, algoritam zamjenjuje mjesta pivot-elementa i posljednjeg elementa u nizu (iz razloga koji će biti spomenut kasnije) i sortiranje može početi.

Algoritam prolazi kroz cijeli zadat interval i sve elemente koji su manji ili jednaki tom pivot-elementu slaže na prvo, drugo, treće itd. mjesto u nizu. Pritom elementi koji su se zatekli na tom prvom, drugom, trećem itd. mjestu zamjenjuju (eng. swap) svoja mjesta za mjesta nađenih elemenata. Elementi koji se već nalaze na nekom od ovih mjesta i ispunjavaju uslov da na njemu ostanu se ne premještaju.

Po završetku ovog procesa, pivot-element sa kraja intervala se stavlja na kraj ovog niza, iza zadnjeg postavljenog elementa. Taj element je sad na svom mjestu i neće biti više potrebe premještati ga. Zbog ovoga je na početku i bio premješten na kraj da se tokom razmještanja ne bi moralo pratiti njegovo mjesto u nizu te da premještanje na kraj bude lakše. Dati

interval je sada ovim pivot-elementom podjeljen na dva podintervala od kojih lijevi sadrži sve elemente njemu manje ili jednake, a desni sve one koji su od njega veći. Algoritam se sada ponavlja za ova dva novonastala intervala.

Proces preraspoređivanja i djeljenja se na daljim podintervalima ponavlja dok se ne dođe do intervala dužine jedan ili nula, u kojima je element već na početku algoritma na svom mjestu.

3. Usporedba algoritama sortiranja

U tablici 1., n je broj elemenata niza koji će biti sortirani. Stupci "Prosječan slučaj" i "Najgori slučaj" dati će nam uvid u vremensku složenost svakog sortiranja posebno (koju smo već opisali u prijašnjim poglavljima a tu ga koristimo samo radi usporedbe), pod pretpostavkom da je duljina svakog niza elemenata jednaka, te da stoga sve usporedbe, zamjene, te druge potrebne operacije može napraviti u konstantnom vremenu.(Internet literatura,10.)

"Memorijski prostor" označava količinu pomoćnog prostora za pohranu i izmjene niza elemenata, pod istim pretpostavkama. Vrijeme izvođenja i memorijski prostor koji su navedeni u nastavku treba shvatiti da se nalaze unutar velikog O zapisa.

Ime	Najbolji slučaj	Prosječan slučaj	Najgori slučaj	Memorijski prostor
Selection	n^2	n^2	n^2	1
Insertion	n	n^2	n^2	1
Merge	n log n	n log n	n log n	n-najgori slučaj
Bubble	n	n^2	n^2	1
Quicksort	n log n	n log n	n^2	log n-prosječni slučaj, n-najgori slučaj

Tablica 1. Usporedba algoritama sortiranja

Iako postoji veliki broj algoritama sortiranja, u praktičnim implementacijama nekoliko algoritama prevladavaju. Insertion sort se naširoko koristi za male skupove podataka, dok se za velike skupove podatka postavlja asimptotski učinkovito sortiranje, prije svega merge sort ili quicksort. Učinkovite implementacije uglavnom koriste hibridni algoritam, kombinirajući asimptotski učinkovite algoritme za ukupno sortiranje s insertion sortom za male liste na dnu rekurzije.

Dva najjednostavnija tipa sortiranja su insertion sort i selection sort, a oba su učinkoviti sa malim brojem podataka, ali ne i učinkoviti na velikim brojem podataka. Insertion sort je općenito brži od selection sort-a u praksi, zbog manjeg broja usporedbi i dobre performanse na gotovo sortiranim podatcima, što ga čini poželjnim, ali selection sort koristi manje zapisa odnosno pisanja, i na taj način se koristi kada su performanse zapisivanja bitnije.

Insertion sort je jednostavan algoritam za sortiranje koji je relativno učinkovit za male liste i uglavnom sortirane liste ili male i sortirane nizove, a često se koristi kao dio sofisticiranijih algoritama. Radi tako da uzima elemente s popisa jedan po jedan, te ih umeta u ispravnom redoslijedu u novi sortirani popis. U poljima, novi popis i preostali elementi mogu dijeliti prostorna polja, ali umetanje je skupo, zahtijeva prebacivanje svih sljedećih elemenata preko jednog. Prednost insertion sorta u odnosu na selection sort je u tome da samo skenira onoliko elemenata koliko je potrebno, da bi postavio k+1. element, dok selection sort mora skenirati sve preostale elemente kako bi pronašao k + 1. element.

Jednostavan izračun stoga pokazuje da će insertion sort obično obavljati oko pola usporedbi od onoga koliko obavi selection sort. To se može promatrati kao prednost za neke real-time aplikacije koje će selection sort obavljati jednako bez obzira na poredak niza, dok vrijeme izvođenja insertion sorta može znatno varirati. Međutim, to je češće prednost za insertion sort da radi puno učinkovitije ako je niz već sortiran ili je blizu tome da je sortiran. Na kraju, Selection sort je uvelike nadmašio očekivanja na većim nizovima O (n log n) kao što su podijeli-pa-vladaj algoritmi - mergesort. Međutim, Insertion sort ili Selection sort su i tipično jako brzi za male nizove (*odnosno manje od 10-20 elemenata*). Korisna optimizacija u praksi za rekurzivne algoritme je da se prebacite na Insertion sort ili Selection sort za "dovoljno male" podliste.

Neki podijeli pa vladaj algoritmi, kao sto su quicksort i mergesort vrše sortiranje tako što rekurzivno dijele liste u manje podliste, koje se onda sortiraju. Upotreba insertion sorta za sortiranje podlista se pokazalo kao korisna optimizacija u praksi. Ovdje sortiranje umetanjem pokazuje bolje performanse od drugih kompleksnijih algoritama. Veličina listi kod kojih je sortiranje umetanjem u prednosti u zavisnosti od okruženja i primjene varira, ali je obično između osam i dvanaest elemenata.

Merge sort zahtjeva samo T(1) pomoćnog prostora za razliku od quicksorta koje zahtjeva T(n), i on je često brži u praktičnim primjenama. Na tipičnim suvremenim arhitekturama, efikasne quicksort implementacije generalno bolje funkcioniraju od merge sorta kada treba

sortirati nizove bazirane na RAM-u. S druge strane, sortiranje spajanjem je stabilno sortiranje, efikasnije je kada treba pristupiti sekvencijalnim medijima kojima se sporo pristupa. Sortiranje spajanjem je često najbolji izbor kada treba sortirati povezanu listu, u ovoj situaciji relativno je lako implementirati merge sort tako da bi ono zahtjevalo samo T(1) dodatnog prostora, a spor nasumični pristup povezane liste čini da neki drugi algoritmi (kao što je quicksort) rade loše.

Bubble sort je jedan od najjednostavnijih algoritama za sortiranje ali ipak potrebno je razumjeti i njegovu primjenu. Njegova složenost $O(n^2)$ znači da se njegova efikasnost dramatično smanjuje na nizovima koji imaju veći broj elemenata, što ujedno znači ako imamo niz od 5 i 10 elemenata, da će niz od 5 elemenata biti znatno prije sortiran jer je potrebno puno manje uspoređivanja. Ako pogledamo i usporedimo ostale algoritme sortiranja s vremenskom složenošću $O(n^2)$ zaključiti ćemo da je od bubble sorta bolji čak i insertion sort.

Zbog svoje jednostavnosti, bubble sort se često koristi da se uvede koncept algoritama ili algoritama za sortiranje na uvodnim predavanjima studentima informatika. Međutim, neki istraživači kao što je Oven Astrachan su otišli predaleko omalovažavajući bubble sort i njegovu popularnost u obrazovanju informatičara preporučujući da se više i ne uči. (Astrachan,2003)

The Jargon file, čiji je poznati naziv bogosort "the archetypical [sic] perversely awful algorithm", također naziva bubble sort generički lošim algoritmom. Donald Knuth u svojoj čuvenoj knjizi Umjetnost računarskog programiranja zaključio je da se Bubble sort nema po čemu preporučiti osim po privlačnom imenu i činjenici nekih zanimljivih teorijskih problema o kojim je on tada govorio. Bubble sort je ekvivalentan insertion sortu po vremenu izvršavanja u najgorem slučaju. Ova dva algoritma se veoma razlikuju po broju potrebnih zamjena. (Knuth, 1998)

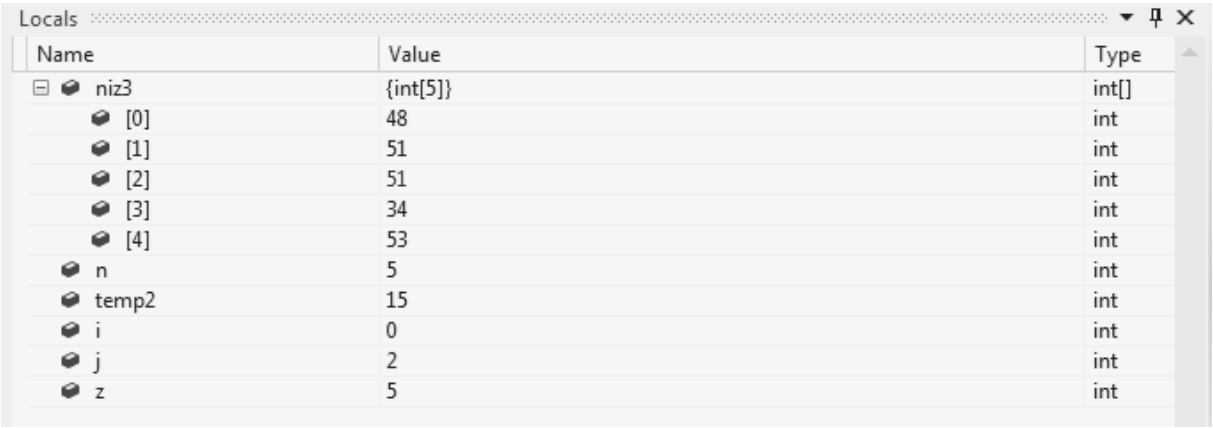
Eksperimentalni rezultati kao što su oni od Astrachana su pokazali da insertion sort radi znatno bolje čak i na slučajnim listama. Iz tih razloga mnogi moderni udžbenici izbjegavaju korištenje algoritma bubble sorta u korist insertion sorta. Eksperimenti Astrachana za sortiranje stringova u Javi pokazuje da bubble sort može biti oko 5 puta sporiji od inserton sorta i 40% sporiji od selection sorta što su jako loši rezultati istraživanja. (Astrachan,2003).

4. Vizualizacija algoritama sortiranja

Vizualizacija algoritama sortiranja napravljena je kao i sam kod algoritama u programskom okruženju C#. C# je objektno orijentirani programski jezik kojeg su razvili Anders Hejlsberg i drugi u tvrtci Microsoft. C# je izumljen s ciljem da .NET platforma dobije programski jezik, koji bi maksimalno iskoristio njezine sposobnosti. Sličan je programskim jezicima Java i C++.

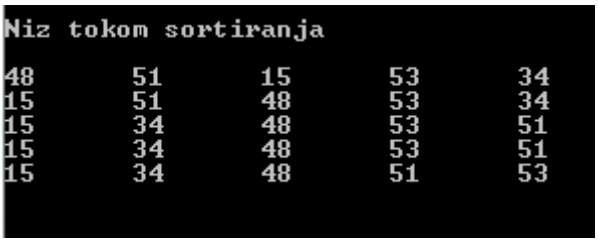
Osnovna zadaća vizualizacije je prikaz promjena elemenata niza korak po korak. Pruža lakše shvaćanje rada algoritama sortiranja. Najveći problem ove naše vizualizacije je bio kako i gdje postaviti petlju koja će ispisivati novo nastali niz, odnosno postupak zamjene mjesta određenih elemenata niza.

Taj postupak je riješen na način da se Debuggerom pratio svaki korak i nakon toga je bilo jasno vidljivo gdje treba postaviti petlju za ispis niza. *Debugger* je računalni program koji se koristi za analiziranje izvršnog koda programa radi otklanjanje grešaka.



Slika 12. Primjer pračenja rada metode preko Debuggera

Nakon pračenja ponašanja varijabli pomoću Debuggera dobili smo ispis niza onako kako smo željeli a to je korak po korak, odnosno u trenutku promjene niza dobijamo ispis trenutnog stanja niza.



Slika 13. Primjer vizualizacije Selection sorta u Console

Dijagram klasa (dio UML-a) jest vrsta strukturnog dijagrama u softverskom inžinjeringu, koji opisuje strukturu sustava objašnjavajući klase unutar sustava, njihove atribute i odnose. Dijagram klasa pokazuje postojanje klasa i njihovih međusobnih odnosa prilikom logičkog oblikovanja sustava. Dijagrami klasa mogu prikazivati cijelu ili samo dio klasne strukture sustava. Prikazuju statičnu strukturu modela, a ne prikazuju privremene informacije.Prilikom modeliranja statičnog pogleda na sustav, dijagrami klasa se obično koriste za modeliranje:

- Rječnika sustava
- Jednostavnih kolaboracija
- Logičke sheme baze podataka

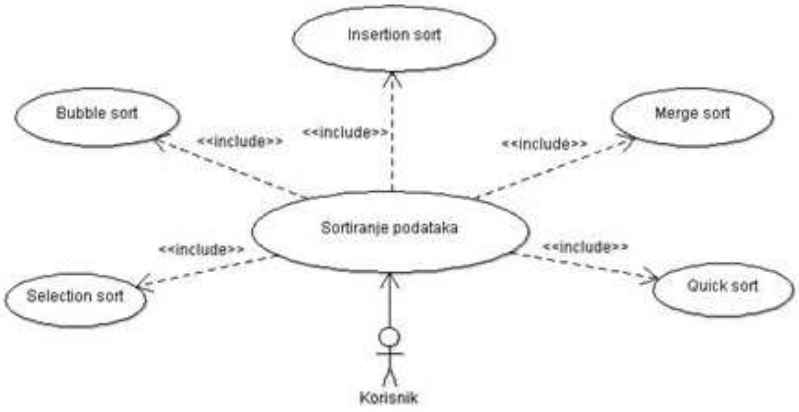
Dijagram klasa našeg programa generirao je sam C#. (Internet literatura,12.)

Slika 14. Dijagram klasa

Dijagram slučajeva korištenja na svojoj najjednostavniji način prikazuje interakciju korisnika sa sustavom i prikazuje specifikaciju dijagrama korištenja. Dijagram može prikazati različite vrste korisnika sustava i različite načine na koje su korisnici u interakciji sa sustavom. Ova vrsta dijagrama obično se koristi u kombinaciji s tekstualnim dijagramom korištenja i često će biti u pratnji drugih vrsta dijagrama kao što je i prmijer u ovom našem poglavlju.

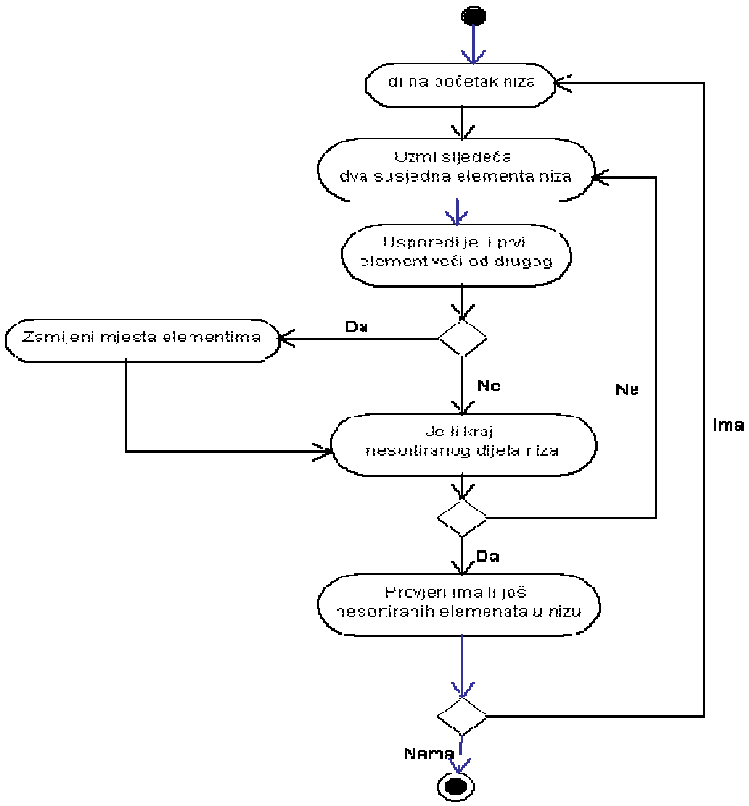
Iako uporaba dijagrama korištenja sama po sebi može pružiti puno detalja o sustavu i svaku njegovu mogućnost, uporaba dijagrama korištenja može pomoći na način da pruži pogled na višoj razini sustava. Davno je rečeno da su dijagrami korištenja nacrt sustava. Oni pružaju pojednostavljen i grafički prikaz onoga što sustav zapravo mora učiniti.

Zbog svoje jednostavnosti, dijagrami korištenja mogu biti dobar komunikacijski. Crteži pokušavaju oponašati stvarni svijet i pružaju pogled promatraču razumjeti kako će sustav biti dizajniran.



Slika 15. Dijagram korištenja za našu console aplikaciju

Nakon dijagrama klasa i dijagrama korištenja još jedan dijagram kojeg ćemo objasniti je *dijagram aktivnosti*. On služi za prikaz događaja i aktivnosti tijekom nekog procesa. Prikazuje dinamiku odvijanja procesa, nadopunjuje dijagram klasa koji je statički. Sastoji se od aktivnosti, događaja, prijelaza i odluka. U ovom poglavlju pokazati ćemo samo dijagram aktivnosti za dvije vrste sorta (bubble i selection sort) dok ostale možete pronaći u poglavlju koje se zove Prilozi.



Slika 16. Dijagram aktivnosti za Bubble sort

Slika 17. Dijagram aktivnosti za Selection sort

Zaključak

Algoritmi sortiranja su važan dio upravljanja podataka. Svaki algoritam ima određene prednosti i nedostatke te u mnogim slučajevima najbolja stvar za učiniti je samo koristiti ugrađene funkcije za sortiranje qsort. Za vrijeme kada to nije opcija i kad samo trebate brzo i neuredno sortiranje pomoću algoritama, postoji mogućnost izbora. Većina algoritama za sortiranje radi se usporedbom podataka koji se sortiraju. U nekim slučajevima, može biti poželjno riješiti veliki komad podataka (primjerice, struct sadrži ime i adresu) na temelju samo jednog dijela tih podataka. Mnogi algoritmi koji imaju istu učinkovitost nemaju istu brzinu na istom ulazu.

Prvi kriterij, algoritmi moraju biti suđeni na temelju njihovog prosječnog slučaja, najboljeg i najgoreg slučaja. Neki algoritmi, kao što je quicksort, će obavljati iznimno dobro za neke ulaze, ali strašno za druge. Ostali algoritmi kao što su merge sort, ne ovise o vrsti ulaznih podataka. Čak modificirana verzija Bubble sorta može završiti O(n) za najpovoljnije ulaze.

Drugi kriterij za prosuđivanje algoritama je njihov zahtjev za prostorom, da li oni zahtijevaju brisanje prostora ili će niz biti razvrstani u mjestu (bez dodatne memorije izvan nekoliko varijabli)? Neki algoritmi nikada ne zahtijevaju dodatni prostor, dok su neki najlakši algoritmi nerazumljivi jer se koriste pomoćnim prostorom (heap sort, na primjer, može biti odrađen na mjestu, ali konceptualno puno je lakše razmišljati o zasebnoj hrpi). Potreban prostor možda čak i ovisi o strukturi podataka koji se koristi (spajanje sortiranje polja u odnosu na spajanje sortiranje povezane liste, na primjer).

Treći kriterij za prosuđivanje algoritama je sama stabilnost algoritama.

U ovom završnom radu mi smo objasnili te kriterije prosuđivanja algoritama i shvatili smo pomoću analiza koji je od algoritama najbrži, dok smo za svaki algoritam napravili i samu usporedbu s drugim. A na samom početku rada napravili smo i neke osnovne informacije o algoritmu i što je to sama složenost algoritma o kome smo pisali kroz cijeli rad.

Literatura

[Daffa, 1977]

Daffa, Ali Abdullah al- (1977), The Muslim contribution to mathematics

[Demuth,1956]

Demuth, H. Electronic Data Sorting. PhD thesis, Stanford University, 1956

[Knuth,1997]

Donald Knuth. The Art of Computer Programming,1997

[Plumbr,2012]

Plumbr.eu: How many Java developers are there in the world? July 18, 2012, by Priit Potter

[Sedgewick, 1983]

Robert Sedgewick, Algorithms, Addison-Wesley 1983

[Katajainen and Larsson,1977]

Jyrki Katajainen and Jesper Larsson Träff (1997). A meticulous analysis of mergesort programs.

[Knuth, 1998]

Donald Knuth. The Art of Computer Programming Second Edition. Addison-Wesley, 1998

["Data structures and algorithm: Quicksort". Auckland University.]

[Singer,2005]

Saša Singer, Složenost algoritama, 2005

[Astrachan,2003]

Owen Astrachan. Bubble Sort: An Archaeological Algorithmic Analysis. SIGCSE 2003 Hannan Akhtar

[Internet literatura korištena u završnom radu]:

1. http://en.wikipedia.org/wiki/Selection_sort
2. http://en.wikipedia.org/wiki/Insertion_sort
3. http://en.wikipedia.org/wiki/Merge_sort

4. http://en.wikipedia.org/wiki/Bubble_sort
5. http://en.wikipedia.org/wiki/Quicksort
6. http://www.cprogramming.com/tutorial/computersciencetheory/sortcomp.html
7. http://alas.matf.bg.ac.rs/~mr09006/algoritmi-sortiranja/?s=grafici
8. http://en.wikipedia.org/wiki/Sorting_algorithm
9. http://web.math.pmf.unizg.hr/~singer/oaa/00.pdf
10. http://en.wikipedia.org/wiki/Sorting_algorithm
11. http://www.cs.toronto.edu/~jepson/csc148/2007F/notes/sorting.html
12. http://hr.wikipedia.org/wiki/Dijagram_klasa

Sažetak

U ovom završnom radu opisana je vizualizacija algoritama sortiranja uz razne usporedbe i primjere pomoću kojih je lakše shvatiti i sam rad osnovnih i malo kompliciranijih algoritama sortiranja. Sama vizualizacija algoritama napravljena je u C# console aplikaciji, gdje možemo pratiti svaku promjenu u nizu kojeg sortiramo.

Također vizualizacija i osnovne funckije programa opisane su pomoću UML-dijagrama (dijagram klasa,dijagram slučajeva kroištenja i dijagram aktivnosti).

Lakše shvaćanje brzine rada algoritama pruža nam tablica u kojoj možemo sami logično zaključiti koji od algoritama je koliko brz odnosno koliko vremena mu je potrebno da obavi najbolji, prosječan i najgori slučaj niza.

U prilogu još možemo vidjeti i kodove određenih algoritama sortiranja te dijagrama aktivnosti.

Ključne riječi: Algoritam, Složenost algoritma, Vizualizacija algoritama, UML dijagram, dijagram toka podataka, Usporedba algoritama sortiranja

Summary

In this final thesis, we describe the visualization of sorting algorithms with different comparisons and examples by which it is easier to understand work basic and a bit more complicated sorting algorithms. The mere visualization algorithms is made in C # console application, where we can follow any change in the series, which sort.

Also visualization and main function of the program are described using UML diagrams (class diagram,use case diagram and activity diagram).

Easier understanding of speeds algorithms gives us the table where we can do it logical to conclude that how fast is the algorithm or how much time he needs to do the best, average and worst case series.

Enclosed we can still see the codes of certain algorithms sorting and diagrams activities.

Keywords: algorithm, complexity of the algorithm, visualization algorithms, UML diagrams, data flow, comparison of sorting algorithms

Prilozi

U prilogu su dani c# kodovi i dijagrami aktivnosti za sve vrste sortiranja koje smo obradili u ovom radu.

1. Selection sort

```
///selection sort
static void selectsort(int[] niz2, int n)
{
    int i, j, z;
    for (i = 0; i < n; i++)
    {
        int min = i;
        for (j = i + 1; j < n; j++)
            if (niz2[j] < niz2[min]) min = j; //pronađi najmanju vrijednost
        //i zatim ga zamijeni sa prvim u ne sortiranoj listi
        int temp = niz2[i];
        niz2[i] = niz2[min];
        niz2[min] = temp;
    }
}
```

Slika 18. Primjer Selection sorta u C#-u

2. Insertion sort

```
static void insertsort(int[] niz4, int n)
{
    int i, j;
    for (i = 1; i < n; i++)
    {
        int odabrani_broj = niz4[i];
        int broj2 = 0;
        for (j = i - 1; j >= 0 && broj2 != 1; )
        {
            if (odabrani_broj < niz4[j])
            {
                niz4[j + 1] = niz4[j];
                j--;
                niz4[j + 1] = odabrani_broj;
            }
            else broj2 = 1;
        }
    }
}
```

Slika 19. Primjer Insertion sorta u C#-u

3. Merge sort

```
static void Uradi_Merge(int[] niz5, int lijevo, int sredina, int desno)
{
    int[] temp = new int[5];
    int i, lijevo_kraj, broj_elemenata, tmp_pozicija;

    lijevo_kraj = (sredina - 1);
    tmp_pozicija = lijevo;
    broj_elemenata = (desno - lijevo + 1);

    while ((lijevo <= lijevo_kraj) && (sredina <= desno))
    {
        if (niz5[lijevo] <= niz5[sredina])
            temp[tmp_pozicija++] = niz5[lijevo++];
        else
            temp[tmp_pozicija++] = niz5[sredina++];
    }
    while (lijevo <= lijevo_kraj)
        temp[tmp_pozicija++] = niz5[lijevo++];
    while (sredina <= desno)
        temp[tmp_pozicija++] = niz5[sredina++];
    for (i = 0; i < broj_elemenata; i++)
    {
        niz5[desno] = temp[desno];
        desno--;
    }
}
```

Slika 20. Primjer Merge sorta u C#-u

4. Bubble sort

```
static void bubblesort(int[] niz3, int n)
{
    int i, j;
    for (i = 0; i < n; i++)
        for (j = n - 1; j > i; j--)
            if (niz3[j] < niz3[j - 1])
            {
                int temp2 = niz3[j];
                niz3[j] = niz3[j - 1];
                niz3[j - 1] = temp2;
            }
}
```

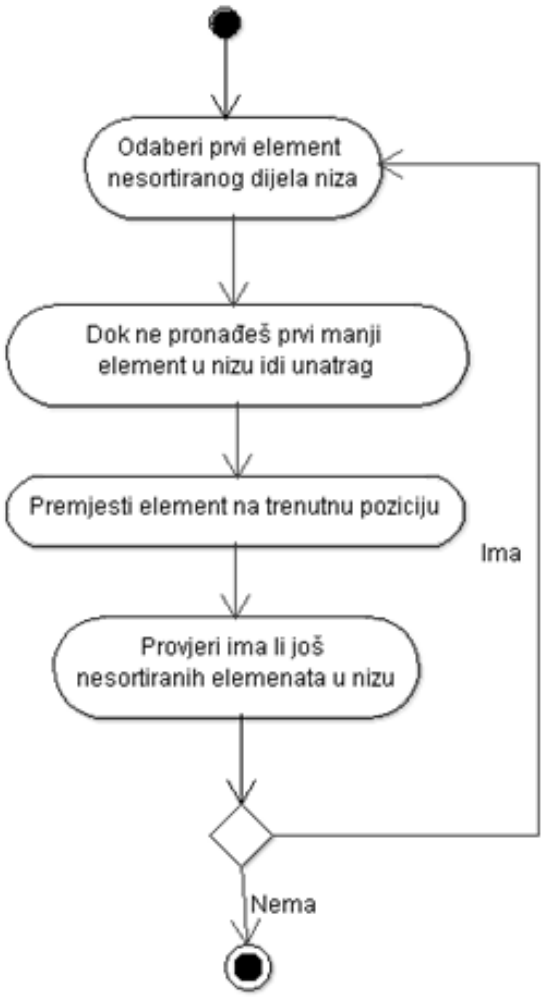
Slika 21. Primjer Bubble sorta u C#-u

5. Quicksort

```
static public int Partition(int[] niz6, int lijevo2, int desno2)
{
    int pivot = niz6[lijevo2];
    while (true)
    {
        while (niz6[lijevo2] < pivot)
            lijevo2++;

        while (niz6[desno2] > pivot)
            desno2--;

        if (lijevo2 < desno2)
        {
            int temp = niz6[desno2];
            niz6[desno2] = niz6[lijevo2];
            niz6[lijevo2] = temp;
        }
        else
        {
            return desno2;
        }

    }

}
```

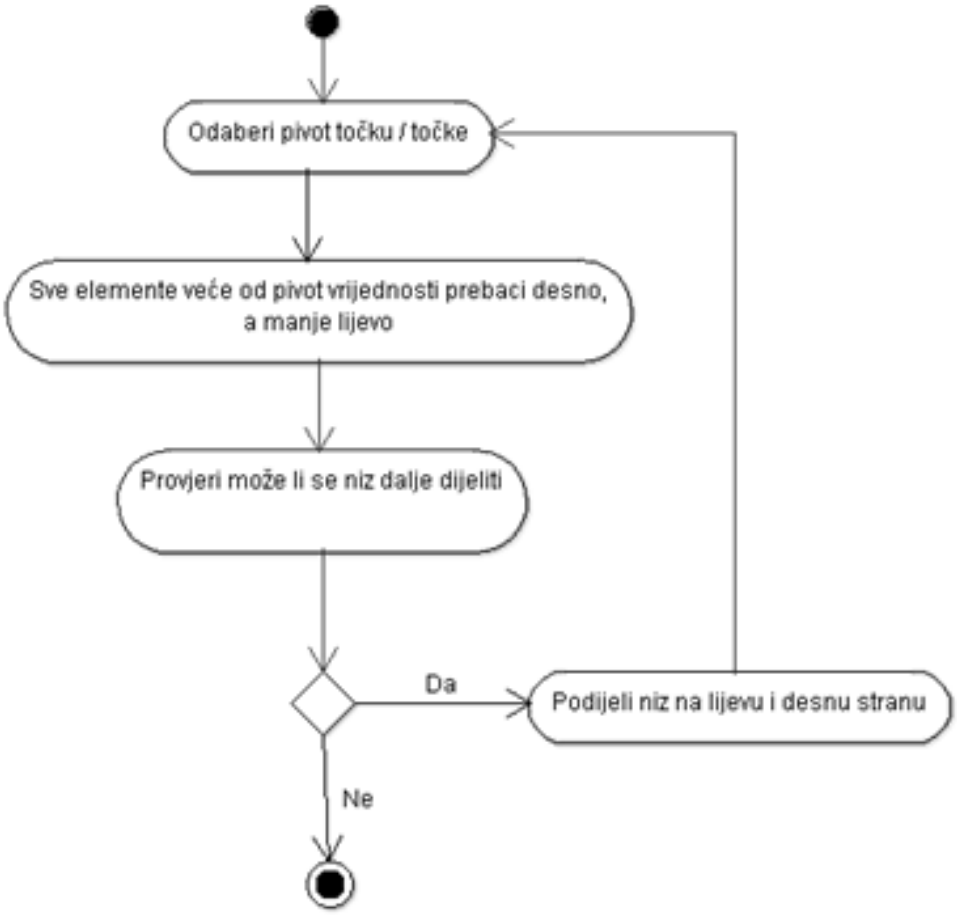
Slika 22. Primjer Quicksorta u C#-u

Dijagrami aktivnosti za preostala 3 sortiranja:



Slika 23. Dijagram aktivnosti Insertion sorta

Slika 24. Dijagram aktivnosti za Merge sort



Slika 25. Dijagram aktivnosti za Quicksort