Elektrotehnički fakultet

Univerzitet u Istočnom Sarajevu



Predmet: Paralelni računarski sistemi

Tema: Paralelizam u algoritmima sortiranja

Autori: Milan Vlaški i Fejzullah Ždralović

Sadržaj

1. Uvod 3

1.1 Svrha 3

1.2 Definicije, akronimi i skraćenice 3

1.3 Reference 4

1.4 Pregled 4

2. Paralelizam 4

2.1 Razlika između paralelizma i konkurentnosti 4

2.2 O paralelizmu 5

2.3 Prednosti paralelizma 5

2.4 Nedostaci paralelizma 6

3. Java 6

3.1 O tehnologiji 7

3.2 Verzije Jave 7

3.3 Instalacija Jave 8

3.3.1 Windows 8

3.3.2 MacOS 8

3.3.3 Linux 8

3.4 Biblioteke 9

4. Java Swing 9

5. Sortiranje 9

5.1 Algoritmi sortiranja 9

5.1.1 Insertion sort 10

5.1.2 Selection sort 11

5.1.3 Bubble sort 12

5.1.4 Merge sort 13

5.1.5 Quick sort 16

6. Paralelizam u Javi 18

6.1 Fork/Join framework 18

6.2 ForkJoinPool 18

6.2.1 Work-stealing algorithm 18

6.3 ForkJoinTask<V> 18

6.4 Predavanje zadataka ForkJoinPool-u 18

7. Paralelizam u algoritmima sortiranja 19

7.1 Implementacija paralelnog selection sorta 19

7.2 Implementacija paralelnog merge sorta 20

7.3 Implementacija paralelnog quicksorta 21

8. Zaključak 22

# Uvod

Prvi dio dokumenta je vezan za sam dokument i tu ćemo moći da saznamo više o samom dokumentu. O svrsi postojanja ovog dokumenta, o definicijama, skraćenicama i akronimima, o referencama i na kraju imamo pregled dokumenta.

U drugom dijelu govorimo o paralelizmu. Pored paralelizma govorit ćemo i o konkurentnosti. Definisati ćemo ta dva pojma, te ćemo govoriti o razlikama između ta dva pojma. Također je u ovom dijelu rečeno nešto više o prednostima i nedostacima paralelizma.

U trećem dijelu će biti nešto više riječi o Java programskom jeziku, o verzijama Jave, o instalaciji Jave na različitim operativnim sistemima, te o bibliotekama koje su korištene u ovome projektu kako bi se obezbjedio paralelni rad različitih algoritama za sortiranje.

Četvrti dio govori o Java Swingu...

U petom dijelu ćemo pričati o algoritmima za sortiranje. Konkretno za algoritme Insertion sort, Selection sort, Bubble sort, Merge sort, Quick sort. Vidjeti ćemo kako rade ti algoritmi, po kojem principu vrše sortiranje, te ćemo i slikovito prikazati kako to izgleda i šta se dešava u pozadini u memoriji.

Šesti dio je rezetvisan za paralelizam u algoritmima sortiranja, tako da ćemo prokomentarisati te algoritme i proći kroz njihov kod. Objasnit ćemo šta se dešava u paralelizaciji sa tim algoritmima i kako smo uspjeli da implementiramo paralelizaciju na te algoritme.

## Svrha

Cilj ovog projekta je vidjeti kako paralelizacija utiče na različite algoritme sortiranja. U ovom projektu smo implementirali pet algoritama za sortiranje (Bubble Sort, Insertion Sort, Selection Sort, Merge Sort i Quick Sort) i omogućili njihovo izvršavanje sekvencijalno (bez paralelizacije) i paralelno (sa paralelizacijom). Pošto ispitujemo uspješnost pojedinih algoritama sortiranja, bitna nam je njihova brzina izvršavanja, tako da prilikom izvršavanja svakog algoritma, mjeri se njegovo vrijeme i rezultat se ispisuje na ekran. Također smo napravili metode koje generišu članove u nizu, pa je moguće unijeti veličinu niza, kao i to da li će brojevi u nizu biti cjelobrojni ili decimalni. Stavili smo da su brojevi, koji se generišu pomoću Random() metode, u rasponu od -10 000 do 10 000. Aplikacija također omogućava rad sa fajlovima, tako da je moguće učitati niz iz fajla, prikazati ga na ekran, te ga sortirati. Moguće je i spremanje rezultata u fajl .

## Definicije, akronimi i skraćenice

* JVM - Java Virtuelna Mašina
* IDE - Integrated Development Environment (Integrisano razvojno okruženje)
* GUI - Graphical User Interface
* API - Application Programming Interface

## Reference

* <https://wiki.haskell.org/Parallelism_vs._Concurrency>
* "Encyclopedia of Parallel Computing" by David Padua
* "Concurrency: State Models & Java Programs" by Jeff Magee and Jeff Kramer
* <https://www.geeksforgeeks.org/introduction-to-parallel-computing>
* <https://www.javatpoint.com/java-versions>
* <https://www.baeldung.com/java-fork-join>
* <https://docs.oracle.com/javase/tutorial/essential/concurrency/forkjoin.html>

## Pregled

Ovaj dokument će pružiti detaljan pregled implementacije algoritama za sortiranje, objasniti paralelizam i kako je implementiran u Javi. Također će obuhvatiti detalje o instalaciji Java okruženja i korištenju Java Swing biblioteke za kreiranje grafičkog korisničkog interfejsa. Dokument će sadržavati i primjere implementacije paralelnih algoritama, te će sadržavati objašnjenje vezano za paralelizaciju u Javi. Zaključak je posljednja stavka koja se nalazi u ovome dokumentu.

# Paralelizam

## Razlika između paralelizma i konkurentnosti

Da ne bi došlo do zabune, na samom početku, napravit ćemo razliku između dva pojma, a to su paralelizam i konkurentnost. Iako su paralelizam i konkurentnost povezani, to su ipak različiti koncepti u računarstvu.

Paralelizam se odnosi na upotrebu više procesora ili procesorskih jezgara za obavljanje zadataka istovremeno kako bi se povećala računska efikasnost i smanjilo vrijeme obrade. Paralelizam uključuje podjelu većeg zadatka na manje podzadatke koje mogu paralelno obraditi različiti procesori ili jezgra za obradu. Glavni cilj paralelizma je postizanje brže i efikasnije obrade zadatka.

Konkurentnost se odnosi na sposobnost različitih dijelova sistema da se izvršavaju nezavisno i istovremeno. Konkurentnost je sposobnost programa da istovremeno obrađuje više zadataka ili zahtjeva, bez čekanja da se jedan zadatak završi prije nego što započne drugi. Glavni cilj konkurentnosti je poboljšanje ukupne odzivnosti i efikasnosti sistema. Na primjer, to može biti izvršavanje više zadataka na jednoprocesorskoj mašini.

Na slikama ispod ćemo predstaviti paralelizam i konkurentnost, kako bismo ih bolje shvatili i zapamtili.



Slika 1



Slika 2

## O paralelizmu

Termin paralelizam odnosi se na tehnike za ubrzavanje programa izvođenjem nekoliko računanja u isto vrijeme. To je zapravo upotreba više procesora ili procesorskih jezgara za istovremeno obavljanje zadataka kako bi se povećala računska efikasnost i smanjilo vrijeme obrade.

Zbog toga je potreban hardver sa više procesorskih jedinica. U mnogim slučajevima, potproračuni su iste strukture. Paralelizam se postiže paralelnom obradom, što uključuje podjelu većeg zadatka na manje podzadatke koje mogu paralelno obraditi različiti procesori ili jezgra za obradu. Na primjer, Grafička izračunavanja na GPU-u (Graphics processing unit) su paralelizovana. Paralelizam se može koristiti u različitim računarskim aplikacijama, uključujući naučne simulacije, analizu podataka, obradu slika i videa i vještačku inteligenciju.

Pošto smo govorili ovdje o procesorima i jezgrima, da ne bi bilo zabune, objasnit ćemo razliku između ta dva pojma:

*Procesor*

Procesor, također poznat kao centralni procesorski jedinica (CPU), je glavna računarska komponenta odgovorna za izvršavanje instrukcija i obavljanje računarskih operacija. To je čip koji obavlja osnovne aritmetičke, logičke, upravljačke i ulazno/izlazne operacije. Procesor se sastoji od više komponenti, uključujući kontrolnu jedinicu (CU) i aritmetičko-logičku jedinicu (ALU).

*Jezgro*

Jezgro je dio procesora koji može samostalno izvršavati naredbe. Procesori se često sastoje od više jezgara, pri čemu svako jezgro može izvršavati zasebne niti instrukcija paralelno. Svako jezgro ima vlastiti set registara i može nezavisno izvršavati instrukcije. Više jezgara omogućava istovremeno izvršavanje više instrukcija i poboljšava performanse multitaskinga i paralelizma.

Razlika između procesora i jezgra je sljedeća:

* Procesor je cijeli čip ili komponenta koja obavlja računarske operacije, dok je jezgro samo jedan dio procesora.
* Procesor može sadržavati jedno ili više jezgara, ovisno o konfiguraciji. Jezgra omogućavaju istovremeno izvršavanje naredbi i mogu se koristiti za postizanje paralelizma.
* Procesor može imati i druge komponente, poput keš memorije, sabirnica i kontrolnih jedinica, dok je jezgro fokusirano na izvršavanje instrukcija.

Broj procesora i jezgara koji se nalaze u računarima zavisi od različitih faktora, uključujući namjenu računara, tip računarskog sistema i tehnološki napredak.

* desktop računari imaju obično jedan fizički procesor (CPU) sa više jezgara, obično 4 do 8, ali postoje i modeli sa više jezgara,
* laptopovi obično imaju jedan procesor sa 2 do 4 jezgra,
* Serverski računari, koji se koriste za hosting web stranica, baze podataka i druge serverske aplikacije, često imaju više procesora i više jezgara u svakom procesoru. Serverski računari mogu imati više od 10 ili čak 20 jezgara po procesoru, a neki napredniji sistemi mogu imati i stotine jezgara,
* HPC (High-Performance Computing) računari, koji se koriste za intenzivne računarske zadatke poput naučnih istraživanja, modeliranja vremenskih prilika ili simulacija, često imaju stotine, hiljade pa čak i milione jezgara. Ovi sistemi koriste se za paralelno izvršavanje velikih brojeva istovremenih zadataka.

## Prednosti paralelizma

Prednosti paralelizma uključuje povećanu brzinu i efikasnost, poboljšane performanse i skalabilnost računarskih sistema, uštedu troškova, veću fleksibilnost, poboljšanu toleranciju na greške, bolje korištenje resursa, poboljšana produktivnost, brža inovacija . U nastavku ćemo nešto reći o svakoj od ovih prednosti.

*Povećana brzina i efikasnost*: paralelizam omogućava da više procesora ili procesorskih jezgara rade zajedno, u isto vrijeme, što može značajno povećati vrijeme izvršenja i efikasnost obrade zadataka.

*Poboljšane performanse i skalabilnost*: Paralelizam može poboljšati performanse i skalabilnost računarskih sistema, čineći mogućim rukovanje većim i složenijim zadacima i skupovima podataka.

*Uštede troškova:* Korištenjem paralelne obrade za poboljšanje efikasnosti obrade, organizacije mogu potencijalno uštediti na troškovima hardvera, energije i ukupnih troškova računarske infrastrukture.

*Veća fleksibilnost*: Paralelizam može pružiti veću fleksibilnost u smislu načina na koji se računski resursi dodjeljuju i koriste, omogućavajući organizacijama da prilagode resurse obrade prema potrebi kako bi zadovoljili promjenjive zahtjeve.

*Poboljšana tolerancija grešaka*: Korištenjem redundantnosti i mehanizama za pronalazak greške, paralelizam može poboljšati toleranciju grešaka i pouzdanost računarskih sistema, smanjujući rizik od kvarova sistema i zastoja.

*Bolje korištenje resursa*: Paralelizam omogućava efikasno korištenje dostupnih računarskih resursa omogućavajući istovremeno pokretanje više procesa, smanjujući vreme mirovanja i poboljšavajući ukupnu iskorišćenost sistema.

*Poboljšana produktivnost*: Smanjenjem vremena obrade i povećanjem efikasnosti računarskih sistema, paralelizam može omogućiti organizacijama da obavljaju zadatke brže i efikasnije, povećavajući produktivnost i propusnost.

*Brža inovacija*: Paralelizam može omogućiti bržu inovaciju omogućavajući istraživačima i programerima da obrađuju velike skupove podataka i izvode složene simulacije brže i efikasnije, što dovodi do bržih uvida i otkrića.

## Nedostaci paralelizma

Jedan od ključnih problema sa paralelizmom je pitanje upravljanja i koordinacije različitih paralelnih procesa kako bi se osiguralo da oni rade zajedno efikasno i bez sukoba. Ovo može biti posebno izazovno u distribuiranim računarskim sistemima, gde se procesi mogu izvoditi na različitim mašinama povezanim mrežom.

Drugi ključni problem je pitanje balansiranja opterećenja, koje uključuje ravnomjernu raspodjelu radnog opterećenja na različite procesorske jedinice kako bi se osiguralo da nijedna jedinica ne miruje dok su druge preopterećene. Balansiranje opterećenja može biti posebno izazovno kada je radno opterećenje dinamično i nepredvidivo, jer može zahtijevati stalno prilagođavanje distribucije zadataka na osnovu promjenjivog radnog opterećenja.

Ostala pitanja vezana uz paralelizam uključuju potrebu za efikasnom komunikacijom i sinhronizacijom između paralelnih procesa, osiguravajući da se podaci pravilno dijele i ažuriraju između procesa i rješavanje potencijalnih uvjeta utrke, zastoja i drugih problema povezanih s konkurentnošću.

Ključni problem paralelizma je smanjenje ovisnosti podataka kako bi se mogli izvoditi proračuni na nezavisnim računskim jedinicama uz minimalnu komunikaciju između njih. U tu svrhu, čak može biti prednost da se isto izračuna dva puta na različitim jedinicama

# Java

Java je objektno orijentisani programski jezik visokog nivoa, koji je prvi put objavio Sun Microsystems 1995. Danas je u vlasništvu Oracle kompanije. Dizajniran je tako da bude nezavisan od platforme, što znači da se Java kod može kompajlirati i pokrenuti na bilo kojoj platformi koja ima Java virtuelnu mašinu (JVM), što ga čini veoma prenosivim.

## O tehnologiji

Spomenut ćemo neke od ključnih karakteristika Java programskog jezika.

Java je potpuno objektno orijentisani programski jezik, što znači da se svo programiranje vrši korištenjem objekata i klasa. Java posjeduje ugrađen Garbage Collector, koji koristi automatsko upravljanje memorijom, što olakšava upravljanje memorijom i smanjuje rizik od curenja memorije. Već smo spomenuli da je Java nezavisna od platforme, pa se Java kod kompajlira u bajt kod, koji se može pokrenuti na bilo kojoj platformi koja ima instaliranu Java virtuelnu mašinu (JVM), bez obzira na osnovni hardver ili operativni sistem. Java ima ugrađenu podršku za višenitnost, koja omogućava da se više niti izvršava istovremeno i asinhrono. Java ima snažan sigurnosni model koji uključuje funkcije kao što su provjera bajt koda i sandboxing za zaštitu od zlonamjernog koda. Java programski jezik ima veoma dobre mogućnosti rukovanja izuzecima, koje olakšavaju pisanje koda koji može rukovati neočekivanim greškama ili izuzecima na zadovoljavajući način. Java ima mnogo popularnih IDE-a (Integrisano razvojno okruženje) kao što su Eclipse, IntelliJ IDEA i NetBeans koji pružaju funkcije kao što su dovršavanje koda, otklanjanje grešaka i refaktorisanje, što olakšava razvoj Java aplikacija.

Java se široko koristi u raznim aplikacijama, uključujući web razvoj, razvoj mobilnih aplikacija, razvoj poslovnog softvera i naučno računarstvo. Ima veliku i aktivnu zajednicu programera i korisnika, što je doprinijelo razvoju bogatog sistema biblioteka, frajmvorka i alata.

## Verzije Jave

Java je imala zavidan broj verzija i izdanja od svog prvog izlaska 1995. godine. U nastavku smo nabrojali neka od glavnih Javainih izdanja:

Java 1.0: Prvo stabilno izdanje Jave, predstavljeno 1996. godine.

Java 1.1: Uvedeno 1997. godine, ovo izdanje je dodalo podršku za unutrašnje klase i JavaBeans.

Java 1.2 (takođe poznat kao Java 2): Uvedeno 1998. godine, ovo izdanje je dodalo podršku za Swing GUI alat, Java imena i interfejs imenika (JNDI) i Java Foundation Classes (JFC).

Java 1.3: Objavljena 2000. godine, ova verzija je uvela Java Sound API i podršku za HotSpot JVM.

Java 1.4: Objavljena 2002. godine, ova verzija je uvela Java Native Interface (JNI), XML obradu sa JAXP-om i regularne izraze sa Java.util.regex.

Java 5 (takođe poznat kao Java 1.5): Objavljeno 2004. godine, ovo izdanje je dodalo podršku za generičke karakteristike, napomene i autoboxing.

Java 6 (takođe poznat kao Java 1.6): Objavljeno 2006. godine, ovo izdanje je uvelo poboljšanja Java virtuelne mašine (JVM), uključujući podršku za dinamičke jezike i Java Compiler API.

Java 7: Objavljena 2011. godine, ova verzija je uvela naredbu try-with-resources, naredbe višestrukog hvatanja i podršku za nizove u naredbama switch.

Java 8: Objavljeno 2014. godine, ovo izdanje je dodalo podršku za lambda izraze, Stream API i Date/Time API.

Java 9: Objavljena 2017. godine, ova verzija je uvela Java Platform Module System (JPMS), koji omogućava modularno programiranje.

Java 10: Objavljeno 2018. godine, ovo izdanje je dodalo podršku za zaključivanje tipa lokalne varijable i interfejs za prikupljanje smeća (Garbage Collector).

Java 11: Objavljena 2018. godine, ova verzija je uvela podršku za HTTP/2 i predstavila nove standardne karakteristike Java Platform Module System (JPMS).

Java 12, 13, 14, 15, 16 i 17: Izdate 2019, 2020 i 2021, ova izdanja su donijela nove funkcije, poboljšanja i poboljšanja performansi Java platforme.

Kao što možemo zaključiti, svako sljedeće izdanje Jave donosi nam nove karakteristike, poboljšanja i unapređenja performansi platforme. Zbog toga se programeri ohrabruju da ažuriraju svoje Java instalacije kako bi iskoristili prednosti najnovijih funkcija.

## Instalacija Jave

Proces instalacije Jave zavisi od operativnog sistema koji koristimo. U zavisnosti od operativnog sistema, navest ćemo postupak instalacije jave.

### Windows

1. Idite na službenu Java web stranicu: <https://www.java.com/en/download/> a zatim kliknite na dugme "Download Java".
2. Prihvatite ugovor o licenci i sačuvajte instalaciju aplikacije na određenu lokaciju na vašem računaru.
3. Kada se preuzimanje završi, dvaput kliknite na instalacionu datoteku i slijedite uputstva sve dok ne dovršite instalaciju.

### MacOS

1. Idite na službenu Java web stranicu: <https://www.java.com/en/download/> a zatim kliknite na dugme "Download Java".
2. Prihvatite ugovor o licenci i sačuvajte instalaciju aplikacije na određenu lokaciju na vašem računaru.
3. Kada se preuzimanje završi, dvaput kliknite na DMG datoteku i slijedite uputstva sve dok ne dovršite instalaciju.

### Linux

Postupak instalacije Jave na Linux operativni sistem se razlikuje u zavisnosti od toga koju distribuciju Linuxa koristite. U slučaju da koristiti Ubuntu evo koraka koje trebate slijediti:

1. Otvorite prozor terminala i unesite sljedeću komandu kako biste ažurirali listu paketa:



1. Unesite sljedeću komandu kako biste instalirali Javu:



(Ovo će komanda instalirati podrazumevanu verziju Jave koja je dostupna u skladišku Ubuntu paketa.)

1. Provjerite koju verziju Jave ste upravu instalirali, tj. Koji verziju sada posjedujete:



## Biblioteke

[bla, bla, bla]

# Java Swing

# Sortiranje

[Uopšteno o sortiranju. Zašto nam treba]

Sortiranje je proces premještanja podataka u rastućem ili opadajućem redoslijedu, kako bismo ih razvrstali i na određeni način uredili. Možemo reći i da je sortiranje proces raspoređivanja podataka u smislenom redoslijed kako bismo ih mogli kasnije efikasnije i lakše analizirati. Sortiranje su najčešća u abecednom ili numeričkom obliku (A-Z ili Z-A; 1-9 ili 9-1).

Postoji nekoliko popularnih algoritama za sortiranje kao što su: bubble sort, selection sort, insertion sort, merge sort i quic sort.

## Algoritmi sortiranja

[O algoritmima sortiranja koji sve postoje]

Algoritmi za sortiranje su osnovni koncept računarske nauke i intenzivno se koriste u mnogim različitim aplikacijama. Možemo reći da je algoritam za sortiranje je zapravo algoritam koji elemente liste postavlja u određenom redoslijedu, u zavisnosti od toga šta se traži u datom kontekstu. Postoji mnogo različitih algoritama za sortiranje, od kojih svaki ima svoje prednosti i nedostatke, a koji ćemo algoritam izabrati to ovisi o specifičnom slučaju korištenja.

Algoritmi za sortiranje se mogu podjeliti na osnovu njihove vremenske i prostorne složenosti, što određuje koliko su efikasni u pogledu upotrebe vremena i memorije. Neki algoritmi su više prikladni za određene tipove podataka ili veličine, pa je važno odabrati pravi algoritam za zadatak koji rješavamo.

U ovome djelu, obradit ćemo pet algoritama sortiranja, a to su:

* insertion sort,
* selection sort,
* bubble sort,
* merge sort
* quick sort

### Insertion sort

Sortiranje umetanjem (eng. Insertion sort) je jednostavan algoritam za sortiranje, koji radi krećući se kroz niz s lijeva na desno, postepeno izgrađujući sortirani dio niza. U svakoj iteraciji, algoritam uzima sljedeći nesortirani element i ubacuje ga na ispravnu poziciju u sortiranom dijelu niza. Niz je podijeljen na sortirani i nesortirani dio. Vrijednosti iz nesortiranog dijela se biraju i postavljaju na ispravan položaj u sortirani dio.

Sortiranje umetanjem ima najgori slučaj vremenske složenosti od O(n^2), ali može biti vrlo efikasno kada su upitanju mali nizovi ili već djelimično sortirani nizovi. Također jedna od njegovih prednosti je to što je ovo algoritam za sortiranje u mjestu, što znači da sortira ulazni niz, tu gdje se nalazi, bez potrebe za zauzimanjem dodatne memorije. Što se tiče prostora, ova tri algoritma, insertion sort, selection sort i bubble sort, koriste O(1) konstantnog prostora, jer sortiraju u mjestu, ali O(n^2) vremena.



Slika 3

****

Slika 4

### Selection sort

Sortiranje odabirom (eng. Selection sort) je jednostavan algoritam sortiranja koji sortira niz ponavljajući postupak pronalaženja minimalnog elementa iz nerazvrstanog dijela i i zamjenjuje ga prvim nesortiranim elementom.

Algoritam održava dva podniza u danom nizu:

* Podniz koji je već sortiran
* Preostali podskup koji još nije sortiran

U svakoj iteraciji, tj. u svakom prolazu se bira minimalni element iz nerazvrstanog podniza i premješta se u razvrstani niz. Selection sort ima svojstvo minimiziranja broja zamjena, stoga je najbolji izbor kada su troškovi zamjene visoki.

Selekciono sortiranje ima vremensku složenost u najgorem slučaju od O(n^2), ali ima prednost što je algoritam za sortiranje u mjestu, što znači da sortira ulazni niz, tu gdje se nalazi, bez potrebe za zauzimanjem dodatne memorije.



Slika 5



Slika 6

### Bubble sort

Bubble sort je algoritam sortiranja koji radi na način da mjenja susjedne elemenate ako su u pogrešnom redoslijedu. Nakon svake iteracije najveći element ide na kraj (u slučaju rastućeg redoslijeda) ili najmanji element dolazi na kraj (u slučaju opadajućeg redoslijeda). Prolazak kroz elemente se ponavlja dok se niz ne sortira. Ovaj algoritam nije prikladan za velike skupove podataka jer su njegova prosječna složenost i složenost u najgorem slučaju O(n^2), što znači da njegove performanse opadaju kako se veličina ulaznih podataka povećava, što ga čini neefikasnim.



Slika 7



Slika 8

### Merge sort

Merge sort algoritam je izumio John von Neumann 1945. Godine. Za razliku od gornja tri algoritma za sortiranje, ovaj se algoritam temelji na tehnici podijeli pa vladaj. To je algoritam zasnovan na upoređivanju koji radi tako što rekurzivno dijeli ulazni niz na manje podnizove, sortira ih, a zatim ih ponovo spaja. Srce Merge Sort algoritma je *merge()* funkcija koja se koristi za spajanje dvije polovice niza. Spajanje je ključni postupak koji pretpostavlja da su dvije polovice sortirane i spaja ih u jednu cjelinu. Ovaj algoritam možete odabrati onda kada vam je potrebno stabilno i brzo sortiranje. Ima zagarantovano vreme rada u najgorem slučaju od O(n\*log(n)), što ga čini jednim od najefikasnijih algoritama za sortiranje. Što se tiče prostora, ovaj algoritam koristi linearno prostor O(n).

Sortiranje spajanjem je algoritam za sortiranje opće namjene koji se može koristiti za sortiranje bilo koje vrste podataka, uključujući brojeve, nizove i objekte. Posebno je pogodan za sortiranje velikih skupova podataka ili podataka koji su pohranjeni na vanjskim uređajima za pohranu (kao što su tvrdi diskovi) koji imaju sporo vrijeme pristupa.



Slika 9

**package** algorithams;

**public** **class** MergeSort {

**public** **static** **void** mergeSort(**int**[] niz, **int** lijevi, **int** desni) {

**if** (lijevi < desni) {

**int** srenji = (lijevi + desni) / 2;

// Sortiramo prvu i drugu polovicu

*mergeSort*(niz, lijevi, srednji);

*mergeSort*(niz, srednji + 1, desni);

// Spajamo sortirane polovice

*merge*(niz, lijevi, srednji, desni);

}

}

**public** **static** **void** merge(**int**[] niz, **int** lijevi, **int** srednji, **int** desni) {

// Velicine dva podniza koji ce se spojiti

**int** n1 = srednji - lijevi + 1;

**int** n2 = desni - srednji;

/\* Kreiranje pomocnih nizova \*/

**int**[] L = **new** **int**[n1];

**int**[] D = **new** **int**[n2];

/\* Upisujemo podatke u nove nizove \*/

**for** (**int** i = 0; i < n1; ++i)

L[i] = niz[lijevi + i];

**for** (**int** j = 0; j < n2; ++j)

D[j] = niz[srednji + 1 + j];

/\* Spajamo podnizove \*/

**int** i = 0, j = 0;

**int** k = lijevi;

**while** (i < n1 && j < n2) {

**if** (L[i] <= D[j]) {

niz[k] = L[i];

i++;

}

**else** {

niz[k] = D[j];

j++;

}

k++;

}

/\* Upisujemo ostale elemente iz L[] ako su ostali(zbog posljednjeg elementa koji moze ostati neupisan) \*/

**while** (i < n1) {

niz[k] = L[i];

i++;

k++;

}

/\* Upisujemo ostale elemente iz D[] ako su ostali (zbog posljednjeg elementa koji moze ostati neupisan)\*/

**while** (j < n2) {

niz[k] = D[j];

j++;

k++;

}

}

}

Slika 10

### Quick sort

Kao i merge sort, quick sort je rekurzivni algoritam koji također radi po principu podijeli pa vladaj. Ovaj algoritam bira jedan element kao pivot, a zatim dijeli listu na dve podliste tako da svi elementi manji od pivota budu u jednoj podlisti, a svi elementi veći od pivota u drugoj podlisti, a onda se rekurzivno sortiraju ove dvije podliste. Ovaj proces se ponavlja dok se cijela lista ne sortira.

Postoji mnogo različitih verzija Quick Sort algoritma koji biraju pivot na različite načine:

* Uvijek bira prvi element kao pivot
* Uvijek bira posljednji element kao pivot
* Bira slučajni element kao pivot
* Bira srednji element kao pivot

Ključni proces u i quick sort algoritmu je „partition()“ metoda. Cilj ove metode je da stavi pivot r na njegov ispravan položaj u sortiranom nizu i da postavi sve elemente manje od r ispred r, a sve veće elemente poslije r.

Brzo sortiranje ima vrijeme rada u najgorem slučaju od O(n^2), ali u praksi obično radi mnogo bolje od ovoga zbog upotrebe dobrog algoritma za odabir pivota i činjenice da podliste obično nisu podijeljene ravnomerno. Prosječno vrijeme rada brzog sortiranja je O(n\*log(n)), što je jedan od najbržih dostupnih algoritama za sortiranje.

Ovo je algoritam za sortiranje u mjestu, što znači da ne zahtijeva dodatnu memoriju za pohranjivanje privremenih nizova. Lako se implementira i može se modifikovati da sortira objekte ili druge složene tipove podataka.



Slika 11



Slika 12

# Paralelizam u Javi

[Da li je moguć? Kako se postiže? Šta dobijamo njime?]

## Fork/Join framework

Fork/Join framework je implementacija ExecutorService interfejsa koja pomaže u iskorištenju više procesora. Dizajnirana je tako da rekurzivno razbija zadatke na manje dijelove. Cilj je iskorištenje dostupne procesorske moći za poboljšanje performansi aplikacije. Dakle, ovaj framework radi „fork“, račvanje zadatka, dok se ne dobije zadatak koji je dovoljno jednostavan da bude izvršen asinhrono. Nakon toga se radi „join“, sabiranje rezultata, ako je to potrebno. Glavne komponente ovog frameworka su ForkJoinPool i ForkJoinTask.

## ForkJoinPool

ForkJoinPool je centar ovog frameworka. On upravlja niti radnice i nudi informacije o stanju i performansama bazena. Niti radnice mogu izvršavati samo po jedan zadatak, ali to ne znači da ForkJoinPool kreira novu nit za svaki novi podzadatak. Umjesto toga, svaka nit ima svoj dvoizlazni red (double ended queue – deque, dek) koji čuva zadatke. Statički ForkJoinPool, vraćen metodom commonPool(), je pogodan za većinu aplikacija te je njime obezbjeđeno smanjeno korištenje resursa. Interesantno je što pri instanciranju novog ForkJoinPool-a možemo specificirati nivo paralelizacije, u suprotnom, nivo paralelizacije biće jednak broju dostupnih procesora.

### Work-stealing algorithm

Work-stealing algoritam ili algoritam koji krade posao, ustvari pokušava „ukrasti“ zadatke iz dekova niti koje su zauzete dok izvršavaju neki zadatak. Po defoltu, nit uzima zadatak sa kraja svog deka. Kada se dek isprazni, nit preuzima zadatak sa repa deka neke druge niti, ili sa globalnog ulaznog reda pošto je tu najvjerovatnije najveći komad posla.

Ovaj pristup smanjuje šansu da će se niti takmičiti oko zadataka. Takođe, nit će manji broj puta morati da traži posao zbog toga što radi prvo na najvećim dijelovima posla.

## ForkJoinTask<V>

ForkJoinTask je osnovni tip za sve zadatke koji se izvršavaju unutar ForkJoinPool-a. U praksi, treba naslijediti jednu od njegove dvije podklase: RecursiveAction za zadatke bez povratne vrijednost i RecursiveTask<V> (gdje je V povratni tip) za zadatke sa povratnom vrijednošću.

## Predavanje zadataka ForkJoinPool-u

Moguće je pokrentui zadatak na nekoliko načina.

* *submit*() ili *execute*() metoda



* *invoke*(), izvršava fork (grananje) zadatka i vraća rezultat, te nije potrebno ručno uraditi nikakav join



* *invokeAll*() metod predstavlja najjednostavniji način za predavanje niza zadataka u bazen. Ona prima zadatke kao parametre, forkuje ih i vraća kolekciju objekata tipa *Future* redoslijedom kojim su i napravljeni
* *fork*() i *join*() metode je moguće koristiti odvojeno. fork() metod predaje zadatak tj priprema ga za asinhrono izvršavanje u bazenu u kojem se trenutno nalazi. Izvršavanje zadatka okida se metodom *join*()



# Zapažanja o implementaciji

Zbog mogućnosti sortiranja cjelobrojnih i decimalnih brojeva jednim te istim algoritmom, postavila se potreba za generalizacijom tipa niza koji sort metode primaju. Jedna od opcija za implementaciju je bila korištenje ArrayList ili Vector klasa, predstavnice Javinog Collections interfejsa. One služe kao zamjena za nizove u Javi i pružaju mnogo pogodnosti kod korištenja. Iako pružaju mnogo opcija za manipulisanje sa nizovima, za našu situaciju ovo nije pogodnost, jer jedina stvar koju zahtijevamo od niza ili elemenata niza je poređenje, zamjena elemenata i slično. Apstrakcija koju ove klase pružaju dolazi po cijenu performansi. A u sortiranju nizova, jedini važan rezultat je vrijeme izvršenja (i korektnost). Takođe, operacije u algoritmima sortiranja su obično strogo određene i unaprijed poznate, te nije problem napisati ih manuelno.

Preostaje opcija korištenja običnih nizova. Pošto su jednostavniji, očekivano je da nizovi pokažu bolje performanse od Collections klasa. No odmah je očigledno ograničenje nizova u programskom jeziku Java i ostalim C-like jezicima a to su: jaki tipovi. To jeste, tip podatka koji, u našem slučaju, prima sort metoda mora biti unaprijed poznat. Iz tog razloa koristilo smo mogućnost Jave dodanu u verziji 5 – Generics ili generični tipovi. Ukratko, moguće je proglasiti generički tip na nivou klase ili metode, i koristiti ga unutar njih, po potrebi. Po konvenciji, novi tip je označen sa velikim slovom T, i u kodu treba da predstavlja generički tip. Generički tip koji smo koristili je **<T extends Comparable<T>>**, što prevedeno znači, bilo koji Tip koji nasljeđuje Comparable klasu. Sama Comparable klasa prima T parametar, te ga je potrebno ponovo tu navesti. Zaista, Integer, Double i Float klase (java wrapper klase za primitivne tipove) nasljeđuju Comparable. Čak i String. Dakle, moguće je alfabetski sortirati niz nekih riječi. Jedina posljedica ove implementacije na klasični način pisanja ovih sort algoritama je zamjena operatora više i manje, sa compareT() metodom, i ovo je uočljivo u kodu.

# Paralelizam u algoritmima sortiranja

## Implementacija paralelnog merge sorta

Prirodno je paralelizaciju merge sort algoritma uraditi koristeći ForkJoinPool iz tog razloga što on podrazumijeva rekurzivno polovljenje niza.



7.1 Osnova sekvencijalnog merge sorta

Primjećujemo da sami algoritam sortiranja predstavlja instanciranje jednog SortTask objekta (koji je tipa RecursiveAction) te njegovo okidanje metodom invoke().



7.2 Početak sortiranja paralelnog mergesorta

Primjećujemo da je osnova paralelnog algoritma sortiranja skoro pa identična sekvencijalnoj. Razlika je u tome što se pozivi metode zamjenjuju instanciranjem novog objekta tipa SortTask, te okidanje svih objekata koji su kreirani metodom invokeAll(). Sam rad prepušten je implementaciji ForkJoinPool-a i nitima koje postoje unutar njega.



7.3 Osnova paralelnog merge sorta

## Implementacija paralelnog quicksorta

U sekvencijalnom quicksort algoritmu zapažamo rekurziju i particioniranje elemenata na lijevu i desnu stranu. Samim tim, logično je lijevu stranu predati jednoj niti a desnu drugoj, pa lijevu stranu lijeve strane dodijeliti nekoj niti, a desnu stranu nekoj drugoj...



8.1 Osnova sekvencijalnog quicksorta

Prirodni opis ovog algoritma odgovara njegovoj stvarnoj podjeli na nove instance taskova.



8.2 Početak sortiranja paralelnog quicksorta

Ova implementacija je slična onoj u mergesortu, s tim što nije potrebno spajati rezultate sortiranja. Sličnosti između paralelne i sekvencijalne implementacije su očigledne. Razlikuju se u tome što se u paralelnoj implementaciji mora okinuti lijeva i desna strana, korištenjem invokeAll() metode. Takođe, poziv metode zamijenjen je instanciranjem SortTask objekta.



8.3 Osnova paralelnog quicksorta

## Implementacija paralelnog selection sorta

Okidanje sortiranja identično je za svaku paralelnu izvedbu. Preostali algoritmi funkcionišu nešto drugačije od prethodnih. Ranije smo primijetili divide and conquer strategiju rekurzivnog dijeljenja nizova na taskove, i to na polovine. To rekurzivno polovljenje urađeno je do najmanje moguće mjere, te je garantovano da je krajnji niz sortiran.



9.1 Osnova paralelnog selection sorta

Strategija razbijanja posla se sada primjenjuje nekom proizvoljnom *granicom*, gdje, ako je dio niza manji od te granice, taj dio se sortira sekvencijalno. Ako ne, nastavlja se razbijanje zadataka na manje, sve dok se ne dobiju dovoljno mali podnizovi. Na kraju, ovako sortirane podnizove potrebno je spojiti, te ponovo koristimo metod MergeSort algoritma.

Granica za selection sort je 500, određena testiranjem nad nizom od 1,000,000 brojeva, gdje je vrijeme izvršenja u opsegu 600ms do 850ms. Ovim je i implicitno rečeno da se za nizove dužine ispod 500 ne isplati paralelizovati ovaj algoritam. To potvrđuje stavku da se paralelizacija ne isplati za manje skupove podataka.

Granica za paralelni bubble sort je određena da bude 200, a za insertion sort 300. Ovi algoritmi su paralelizovani slično kao selection sort.

# Hipoteza

Očekuje se da sekvencijalni algoritmi pokažu sporo vrijeme izvršenja što se više bude povećavala dužina niza. Trebali bi uočiti kvadradtni rast vremena izvršenja uz rast dužine niza za algoritme bubble, insertion i selection, a nešto sporiji za quick i mergesort. Ipak, očekivano je vrijeme izvršenja nekoliko milisekundi za nizove ispod 1000.

Paralelizacijom bi trebali postići bolje performanse u svakom slučaju za nizove od 5000 članova ili više. Trebali bi uočiti ravnanje krive odnosa dužine niza i vremena izvršenja. Pretpostavljamo da ovo izravnjavanje neće biti samo linearno za različite algoritme, jer je očekivani overhead za merge, bubble i insertion dosta veliki. Razlog tome je što se uvodi korak sažimanja podnizova a što više puta moramo sažeti (mergati podnizove), to je veći overhead. Možemo očekivati i da bubblesort ima veći overhead, jer ima najnižu granicu od 200, te je on kandidat za najsporiji od svih paralelnih algoritama. Paralelni quick i mergesort, pored toga što su brži od ostalih algoritama koji se testiraju, nemaju dodatnih koraka koji već ne postoje u njihovoj implementaciji. Dakle, nemamo mnogo overheada, samim tim, izvršenje je nešto brže.

# Rezultati

Sekvencijalno, u početku su ova 3 brza. Ali jako se brzo otmu kontroli. Paralelni selection je bolji od paralelnih quicksort i merge??. (Potvrdi hipotezu ili opovrgni).

# Zaključak

Moguće je da su nam Java objekti preteški pa zato ponestaje memorije. Moguće je da nismo mi krivi, već Java koja smatra sve kao objekat, i za sve radi pass-by-value. Moguće je da je heap pojeden u sekvencijalnom zato što zahtijeva mnogo rekuzrivnih poziva i referenci – a memorija je otišla kod paralelizacije zbog mnogo niti, raznih objekata, i sve što paralelizacija zahtijeva.