

Tartu Ülikool
Füüsika-Keemiateaduskond
Eksperimentaalfüüsika ja tehnoloogia instituut

Kodutöö aines

Algoritmid ja andmestruktuurid

PISTEMEETOD JA KIIRMEETOD

Autor: Laas Toom

Kursus: infotehnoloogia magister I

Juhendaja: Ain Isotamm

Tartu 2003

Sisukord

Ülesande püstitus	3
Pistemeetod	3
Kiirmeetod	4
Mõõtmistulemused.....	5
Kokkuvõte.....	7
Retsensioon.....	7
Kasutatud kirjandus	8
Lisa 1: programmi kood.....	9

Ülesande püstitus

Realiseerida ja esitada piste- ja kiirmeetodi algoritm, määrata nende ajaline keerukus, tabuleerida keskmine ajaline keerukus ning joonistada välja vastavad graafikud. Retsenseerida Erme Reino poolt koostatud kodutööd teemal pistemeetod ja kiirmeetod.

Pistemeetod

Pistemeetod (*insertion method*) on üks lihtsamaid sortimisalgoritme. Massiivi sortimiseks pistemeetodil, paigutatakse iga üksik element a_i temale sobivasse kohta indeksiga j , vaadates seejuures läbi kõik $i-j$ elementi. Olenevalt realisatsioonist ning kasutatavast programmeerimise keelest võib pistemeetodil sorteerimist aeglaseks teha ka veel see, et elemendile a_i koha tekitamiseks tuleb järjest nihutada ühe koha võrra kõrvale kõik temast suuremad elemendid (seda ei pea tegema selliste massiivide puhul, mida hoitakse nõ venivas konteineris, kuhu võib suvalisse kohta elemente juurde lisada, ilma teiste elementide ümberpaigutamise peale aega kulutamata – java keeles on selliseks näiteks `java.util.Vector` klass).

Halvimal juhul nõuab iga i -s samm $i - 1$ elemendi edasi nihutamist (sellisel juhul on tegemist kahanevalt sorteeritud järjendiga), mis annab pistemeetodi keerukuse hinnanguks $O(n^2)$.

Pistemeetodi realisatsioon

Järgnevas meetodis vaadatakse järjendi igat elementi, alates 2. (indeksiga 1) ning otsitakse talle temast eelnevate hulgas kohta. Selleks salvestatakse see element abimuutujasse ning nihutatakse kõiki temast suuremaid elemente ühe koha võrra kaugemale.

```
private void pistemeetod(int[] a) {
    int abi;
    int j=0;
    for (int i = 1;i<a.length;i++){
        abi = a[i];
        j=i;
        while (j>0 && (a[j-1] > abi)){
            a[j] = a[j-1];
            j--;
        }
        a[j]=abi;
    }
}
```

Kiirmeetod

Kiirmeetod on jaga-ja-valitse-tüüpi algoritm, mis leiab praktikas üsna palju kasutust, kuna tema ajaline keerukus on $O(n \log n)$. Selliste algoritmide põhimõte on jagada probleem alamülesanneteks ning nende lahenduste baasil saada lahend ka põhiülesandele. Nii jaotab ka kiirmeetod sorteerimiseks antud järjendi alamosadeks, millest kumbki sorteeritakse eraldi. Sealjuures kindlustab jagamise etapp, et ükski esimeses osajärjendis paiknevatest elementidest ei ole suurem teise osajärjendi ühestki elemendist. Üldjuhul toimub jagamine umbkaudu pooleks, kasutades võrdluseks ühte järjendi enda elementi. Selle elemendi valiku kriteerium pole oluline ning erinevates olukordades võib üks või teine neist omada eeliseid. Siiski, valides jagajaks järjendi esimese elemendi, on oht saada selle konkreetse järjendi sorteerimise ajaliseks keerukuseks hoopis $O(n^2)$. See tuleneb asjaolust, et näiteks juba sorditud järjendi korral jagajaks esimese elemendi valimisel jaguneb massiiv väga ebavõrdse suurusega osadeks ning rekursioonist tulenev ballast muutub arvutiriistvarale koormavaks.

Kiirmeetodi realisatsioon

Järgneva kiirmeetodi realisatsiooni puhul võetakse veelahkmeks osajärjendi keskmine element (alumise täisarvu täpsusega) ning korraldatakse elemendid niiviisi ümber, et veelahkmest eespool ei asu temast suuremaid ning temale ei järgne temast väiksemaid elemente. Seejärel kutsutakse rekursiivselt välja seesama meetod, et sortida kiirmeetodil juba mõlemad alamjärjendid.

```
private void kiirmeetod(int[] a, int v0, int p0){
    // lõpu tunnus, kui v > p
    if (v0 < p0){
        int v = v0;
        int p = p0;
        int kesk = (v0+p0)/2;

        // valime keskmise elemendi jaotajaks
        int b = a[kesk];
        // viime jaotaja algusse eest ära
        int t=a[p0];a[p0]=b;a[kesk]=t;

        // käime antud elemendid läbi
        while(v < p){
            // otsime b-st suuremaid elemente
            while((a[v] <= b) && (v<p)) v++;
            // otsime b-st väiksemaid elemente
            while((a[p] >= b) && (v<p)) p--;

            // vahetame elemendid omavahel ära
```

```

        if (v<p){ // kui nad on võrdsed, siis
raiskaks aega vahetamisega
            int abi=a[v];
            a[v]=a[p];
            a[p]=abi;
        }

// toome b jälle õige koha peale tagasi
a[p0] = a[v];
a[v] = b;

// sordime alumise otsa a[v]=a[p]=b, seda pole vaja
sortida
    kiirmeetod(a,v0,v-1);
// nüüd sordime ülemise otsa
    kiirmeetod(a,p+1,p0);
}
}

```

Juba mainitud rekursiivsuse ballasti tõttu soovitatakse kirjanduses [4], sõltuvalt konkreetsest riistvarast ja protsessorist, muuta pisemate osajärjendite sorteerimise meetod kiirmeetodist pistemeetodiks (~10 elemendilise järjendi korral on pistemeetodil juba märgatav kiiruse tõus). Samuti pakuti kiiruse tõstmise võimalusena välja ka kiirmeetodi algoritmi ümberkirjutamine iteratiivsele kujule, mis puhul kaob rekursiivsusega seotud probleemid.

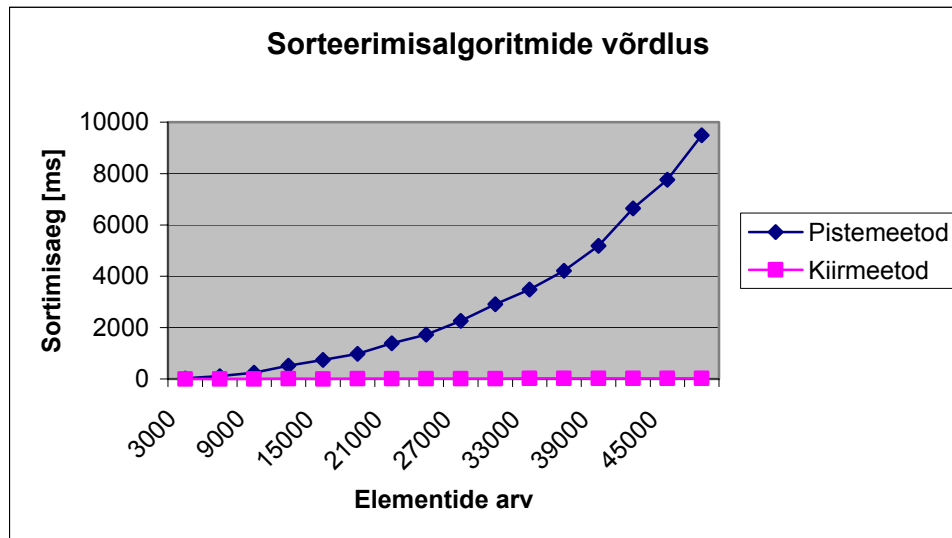
Mõõtmistulemused

Programm on koostatud keeles java ning testimiseks kasutati arvutit: AMD Duron 800 MHz, 100MHz FSB. Katse käigus kasutati mõlemat algoritmi 16 erineva arvujada sorteerimiseks. Igat nendest arvujadadest sorteerisid mõlemad algoritmid 10 korda ning saadud ajad keskmistati. Lõpptulemuses on kajastatud ainult keskmistatud ajad, kuna kõikide andmete välja toomine ei oma mõtet.

Sorteerimisalgoritmide võrdlus		
Ajad millisekundites		
Elemente	Pistemeetod	Kiirmeetod
3000	30	0
6000	108	0
9000	251	3
12000	521	7
15000	743	6
18000	977	7

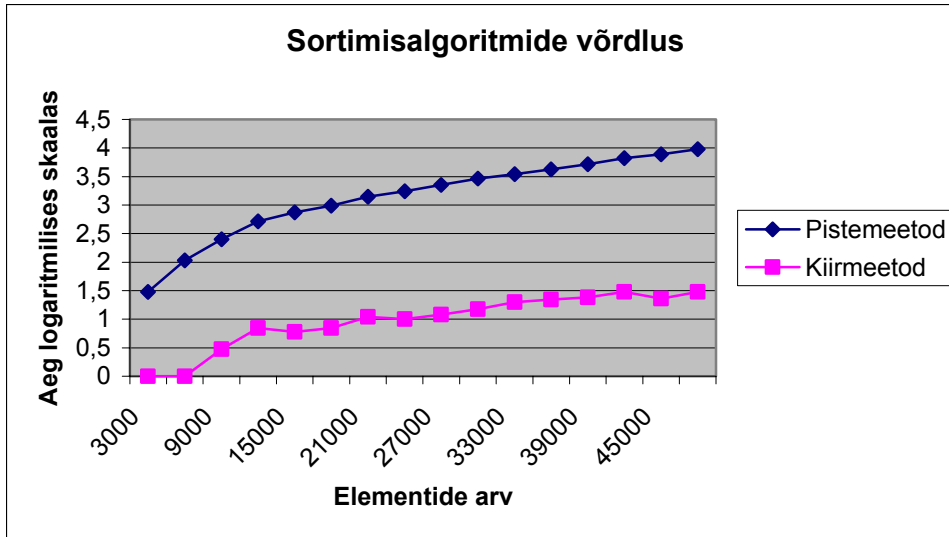
21000	1398	11
24000	1731	10
27000	2266	12
30000	2909	15
33000	3485	20
36000	4211	22
39000	5192	24
42000	6643	30
45000	7764	23
48000	9488	30

Graafikud



Joonis 1: Joonisel on toodud piste- ja kiirmeetodil sorteeritud järjendite elementide arv, ning kummagi meetodi puhul sorteerimiseks kulunud aeg millisekundites.

Kuna joonisel 1 on pistemeetodi ajad palju suuremad kiirmeetodi aegadest, siis et ka kiirmeetodi aegadest paremat ülevaadet saada, on joonisel 2 toodud samad tulemused logaritmilisel skaalal.



Joonis 2: Katse tulemused logaritmilises skaalas

Nagu jooniselt näha, on kiirmeetodi kasvutempo väike ning seda tõestavad ka programmeerimise käigus tehtud katsed ülisuurte massiivide sorteerimisega, mida pistemeetodil sorteerida üritada polnud mõtet.

Kokkuvõte

Katse tulemusena saadud andmete põhjal on selgelt näha, et kiirmeetod on sobilikum suuremõõtmeliste järjendite sorteerimisel. Samas ei tohi unustada, et kiirmeetodil eksisteerib erijuhte, kus ajaline keerukus kasvab $O(n^2)$ tasemele. Seetõttu, kui on oluline kindla aja jooksul tulemust saada, siis peaks kiirmeetodi asemel võib-olla mõnda muud algoritmi kasutama, mille keerukus on samuti $O(n \log n)$, kuid millel ei esine selliseid erijuhte.

Retsensioon

Erme Reino kodutöö on kirjutatud eesti keeles, töö pikkus on 10 lehekülge. Üldine vormistus on korrektne ning vastab esitatud nõuetele (teadustööle omane stiil). Esines küll mõningaid lausestuse ning kirjavigu, kuid muidu oli keelekasutus ladus ning sujuv. Puudusena peab välja tooma, et E. Reino koostatud programmi kood oli lisatud kohe põhitöö järele, pelgalt uue pealkirjaga ning enne kasutatud kirjanduse loetelu, kuigi peaks olema paigutatud hoopis lisana töö lõppu.

Töö üldmulje oli siiski väga hea ning talle omistatud hinde (A-) vääriline.

Kasutatud kirjandus

1. E. Reino, Pistemeetod ja kiirmeetod, Tartu, 2001, 10 lk. (käsikiri)
2. J. Kiho, Algoritmid ja andmestruktuurid, Tartu, 1997, 124 lk
3. P. Laud, Algoritmid ja andmestruktuurid, sügis 2003,
http://math.ut.ee/~peeter_l/teaching/a_ja_a03s/, 2003
4. J. Morris, Data structures and algorithms,
http://ciips.ee.uwa.edu.au/~morris/Year2/PLDS210/ds_ToC.html, 1998
5. Sun Microsystems, Java™ 2 SDK Documentation,
<http://java.sun.com/j2se/1.4.2/docs/>, 2003

Lisa 1: programmi kood

```
import java.util.Date;
import java.util.Random;
import java.util.Vector;

/**
 * @author <a href="mailto:laaz@laaz.org">Laas Toom</a>
 *
 * To change the template for this generated type comment go to
 * Window - Preferences - Java - Code Generation - Code and Comments
 */
public class Sorteerimine {

    int MAX = 50000; // Pistemeetodil suurema massiivi sortimine
    // võtab liialt kaua aega
    int samm = 3000; // samm, mille kaupa massiivi suurust
    // tõstetakse
    int kordi = 10; // mitu korda ühte massiivi sorteerime

    Vector stats;

    /**
     * Loob uue isendi
     */
    public Sorteerimine() {
        super();
        stats = new Vector();
        stats.add("Sorteerimisalgoritmide võrdlus");
        stats.add("Ajad millisekundites");
        stats.add("elemente\tPistemeetod\tKiirmeetod");
    }

    public static void main(String[] args) {
        Sorteerimine s = new Sorteerimine();
        s.run();
    }

    /**
     * Käivitab sorteerija töö
     */
    public void run() {
        int[] massiiv; // massiiv, mida sorteeritakse
        // suurendame elementide arvu iga korraga
        for(int elemente = samm; elemente <= MAX; elemente +=
samm) {
            massiiv = fillArrayInt(elemente);
            sorteeri(massiiv);
        }

        System.out.println(); // lihtsalt outputi ilmestamiseks
        for (int i=0; i<stats.size(); i++){
```

```

        System.out.println(stats.elementAt(i));
    }
}

/**
 * Kutsus kordi arv korda sorteerimise meetodeid välja,
 * et massiivi sorteerida
 * @param massiiv massiiv, mida sorteeritakse
 */
private void sorteerimiseMetodeidVälja(int[] massiiv) {
    long p_keskmine = 0;
    for (int i = 1; i <= kordi; i++) {
        System.out.print("."); // prindime töö indikaatorina
        // midagi välja
        long algus = (new Date()).getTime();
        pistemeetod((int[])massiiv.clone());
        long lopp = (new Date()).getTime();
        p_keskmine = ((p_keskmine * (i-1)) + (lopp - algus)) / i;
    }
    long k_keskmine = 0;
    for (int i = 1; i <= kordi; i++) {
        System.out.print(":"); // samuti indikaator
        long algus = (new Date()).getTime();
        kiirmeetod((int[])massiiv.clone(), 0, massiiv.length -
1);
        long lopp = (new Date()).getTime();
        k_keskmine = ((k_keskmine * (i-1)) + (lopp - algus)) / i;
    }

    System.out.println(); // lihtsalt outputi ilmestamiseks
    String tulemus = String.valueOf(massiiv.length) + "\t" +
String.valueOf(p_keskmine) + "\t" + String.valueOf(k_keskmine);
    stats.add(tulemus);
}

private void kiirmeetod(int[] a, int v0, int p0) {
    // lõpu tunnus, kui v > p
    if (v0 < p0) {
        int v = v0;
        int p = p0;
        int kesk = (v0 + p0) / 2;

        // valime keskmise elemendi jaotajaks
        int b = a[ kesk ];
        // viime jaotaja algusse eest ära
        int t = a[ p0 ]; a[ p0 ] = b; a[ kesk ] = t;

        // käime antud elemendid läbi
        while (v < p) {
            // otsime b-st suuremaid elemente
            while (a[ v ] <= b && (v < p)) v++;
            // otsime b-st väiksemaid elemente
            while (a[ p ] >= b && (v < p)) p--;
        }
    }
}

```

```

        // vahetame elemendid omavahel ära
        if (v<p){ // kui nad on võrdsed, siis
raiskaks aega vahetamisega
            int abi=a[v];
            a[v]=a[p];
            a[p]=abi;
        }
    }

    // toome b jälle õige koha peale tagasi
    a[p0] = a[v];
    a[v] = b;

    // sordime alumise otsa a[v]=a[p]=b, seda pole vaja
sortida
    kiirmeetod(a,v0,v-1);
    // nüüd sordime ülemise otsa
    kiirmeetod(a,p+1,p0);
}
}

```

```

/**
 * Sorteerib <em>massiivi</em> pistemeetodil
 * @param massiiv
 */
private void pistemeetod(int[] a) {
    int abi;
    int j=0;
    for (int i = 1;i<a.length;i++){
        abi = a[i];
        j=i;
        while (j>0 && (a[j-1] > abi)){
            a[j] = a[j-1];
            j--;
        }
        a[j]=abi;
    }
}

/**
 * Tekitab etteantud pikkusega massiivi ning täidab selle
suvaliste arvudega
 * @param elemente      massiivi elementide arv
 * @return tagastab loodud massiivi
 */
private int[] fillArrayInt(int elemente) {
    Random r = new Random();
    int[] array = new int[elemente];
    for (int i = 0; i< array.length; i++){
        array[i] = r.nextInt();
    }
}

```

```
    }  
    return array;  
  }  
}
```