1. Iseseisev töö

I Sorteerimismeetodite ajalise keerukuse empiiriline hindamine

Tuleb koostada programme, mis kuvavad uuritavate meetodite tööaja või siis ajalise keerukuse graafikuid (abstsissiks järjekordse sorteeritava järjendi pikkus).

Järjendite sorteerimismeetodite algoritme leiab kas Internetist või õpikust või loengumaterjalidest.

Lahendusi võib esitada mudelprogrammide *Ajagraafikud.java* või *AjalKgraafikud.java* modifikatsioonidena. Võib kasutada ka muid graafikujoonistamise vahendeid, "sakilised" graafikud on aga igal juhul taunitavad.

1.1. **Ülesanne**

Ülesanne 7.47 ülesannete kogust ([2], lk 48).

Aeglaseks sorteerimismeetodiks mitte võtta pistemeetodit ega päkapikumeetodit (*gnome sort*). Kiireks sorteerimismeetodiks võtta kiirmeetod õpikust ([1], lk 64), kus jaotamise „veelahkmeks“ võetakse jaotatava järjendi esimese elemendi väärtus.

1.2. **Ülesanne**

a) Valida välja, realiseerida ja testida üks ruutkeerukusega (keskmise ajalise keerukuse hinnanguga $\Theta(n^2)$) algoritm järjendi elementide ümberpaigutamiseks **mittekasvavasse** järjestusse. Sellisteks algoritmideks on näiteks mullimeetod (*bubble sort*), valikumeetod (*selection sort*) jt. Pistemeetodit (*insertion sort*) ega päkapikumeetodit (*gnome sort*) mitte võtta! Testimisel mitte unustada nn äärejuhte (järjendid pikkusega 0, 1 või 2; sorteeritud järjend; tagurpidi sorteeritud järjend).

b) Valida välja, realiseerida ja testida üks keskmise ajalise keerukuse hinnanguga $\Theta(n \log n)$ algoritm järjendi elementide ümberpaigutamiseks **mittekasvavasse** järjestusse. Sellisteks algoritmideks on näiteks kiirmeetod (*quick sort*), ühildusmeetod (*merge sort*), Shelli meetod (*Shell sort*) jt. Testimisel mitte unustada nn äärejuhte!

c) Koostada nende meetodite tööaegade graafikud, mis demonstreeriksid kahe valitud sorteerimismeetodi **tööaja** kasvu vastavalt sorteeritavate järjendite pikkuste kasvule. Kolmandaks olgu joonisele lisatud ka Java süsteemse sorteerimismeetodi *Arrays.sort()* tööaja graafik.

1.3. **Ülesanne**

a) Valida välja, realiseerida ja testida üks ruutkeerukusega (keskmise ajalise keerukuse hinnanguga $\Theta(n^2)$) algoritm järjendi elementide ümberpaigutamiseks **mittekahanevasse** järjestusse. Sellisteks algoritmideks on näiteks mullimeetod (*bubble sort*), valikumeetod (*selection sort*) jt. Pistemeetodit (*insertion sort*) ega päkapikumeetodit (*gnome sort*) mitte võtta! Testimisel mitte unustada nn

äärejuhte (järjendid pikkusega 0, 1 või 2; sorteeritud järjend; tagurpidi sorteeritud järjend).

b) Valida välja, realiseerida ja testida üks keskmise ajalise keerukuse hinnanguga $\Theta(n \log n)$ algoritm järjendi elementide ümberpaigutamiseks **mittekahanevasse** järjestusse. Sellisteks algoritmideks on näiteks kiirmeetod (*quick sort*), ühildusmeetod (*merge sort*), Shelli meetod (*Shell sort*) jt. Testimisel mitte unustada nn äärejuhte!

c) Koostada nende meetodite ajalise keerukuse graafikud, mis demonstreeriksid kahe valitud sorteerimismeetodi **ajalise keerukuse** kasvu vastavalt sorteeritavate järjendite pikkuste kasvule.

II Keskmine ajaline keerukus empiiriliselt

1.4. **Ülesanne**

(a) Uurida katseliselt järgmise meetodi keskmist ajalist keerukust ([1], lk 19).

```
static int lõpunulle(String b){
// Antud: b -- bitijärjend sõnena
// Tulemus: tagastatakse b lõpus olevate nullide arv
   int tulem = 0;
   for(int i = b.length()-1; i >= 0; i--){
      if(b.charAt(i) == '1') // (põhitehe)
         break;
      tulem++;
   }//for
   return tulem;
}//lõpunulle
```

Sel eesmärgil täita tabel 1, lugedes põhitehteks bitijärjendi elemendi võrdlemise sümboliga `'1'`. Tabeli teise veeru väärtuste leidmiseks võiks koostada ka vastava abiprogrammi.

Tabel 1: Keskmise ajalise keerukuse väärtusi.

Bitisõne pikkus n	**Sooritatud põhitehete arv** keskmiselt
1	...
2	...
...	...
25	...

Mida saab selle tabeli põhjal oletada vaadeldava meetodi keskmise ajalise keerukuse kohta?

(b) Uurida samamoodi järgmise rekursiivse meetodi keskmist ajalist keerukust.

```
static int lõpunulleRek(String b){
// Antud: b -- bitijärjend sõnena
// Tulemus: tagastatakse b lõpus olevate nullide arv
   int n = b.length();
   if (n == 0) // baasjuht 1
      return 0;
   if(n == 1)  // baasjuht 2
      if(b.charAt(0) == '1')
         return 0;
      else
         return 1;

   int k = n/2; // keskkoht
   int tulem2 = lõpunulleRek(b.substring(k, n));
   if(tulem2 == n-k) // teises pooles ainult nullid
      return lõpunulleRek(b.substring(0,k)) + tulem2;
   else
      return tulem2;
}//lõpunulleRek
```

(c) Koostada võimalikult lühikese lähtekoodiga rekursiivne meetod lõpunullide arvu leidmiseks etteantud bitisõnes. Uurida samamoodi (katsete tabeli näol) ka selle meetodi keskmist ajalist keerukust.

(d) Formuleerida võimalikult lihtsa avaldisena eespool käsitletud kolme meetodi keskmine ajaline keerukus $f_{ac}(n)$, kus n on töödeldavate bitisõnede (üle mille keskmine võetakse) ühine pikkus. [Tõestada selle kehtivus.]

Viited

[1] J. Kiho. *Algoritmid ja andmestruktuurid.* Kolmas, parandatud ja täiendatud trükk. TÜ, 2003, 147 lk. http://dspace.ut.ee/bitstream/handle/10062/16872/9985567676.pdf?sequence=1 (16.04.2017)

[2] A. Peder, J. Kiho, H. Nestra. *Algoritmid ja andmestruktuurid. Ülesannete kogu.* TÜ, 2017, 131 lk.