Algoritmid ja andmestruktuurid
Arvutipraktikumi ülesanded I

Ajalise keerukuse empiiriline hindamine

Algoritmi efektiivsuse hindamisel on üks olulisi kriteeriume algoritmi täitmiseks tehtavate operatsioonide arv (mis määrab ka vastava programmi tööaja). Algoritmi ajaline keerukus on funktsioon k, mis algandmete mahule n seab vastavusse algoritmi sellise mahuga ülesannete täitmiseks tehtavate operatsioonide keskmise arvu $k(n)$. Ütleme, et algoritmi ajalise keerukuse k O-hinnang on $f(n)$ (lühemalt: k on $O(f)$), kui leiduvad konstandid $c > 0$ ja n_0 nii, et iga $n > n_0$ korral $k(n) < cf(n)$. Piltlikult: funktsiooni k graafiku kasvukiirus ei ole „ägedam“ kui funktsioonil f.

Algoritmi ajalise keerukuse hinnang määrab ka algoritmi praktilise rakendatavuse piiri: väga kiiresti kasvava keerukusfunktsiooniga algoritmi tasub realiseerida vaid väikesemahuliste ülesannete jaoks.

Ajalise keerukuse empiiriliseks hindamiseks (ja vastavate graafikute joonistamiseks) tuleb mõõta programmi tööaega erinevate algandmemahtude korral. Keeles Python saab seda teha näiteks moodulis `time` defineeritud funktsiooni `time()` abil:

```
from time import *

a = time()
# PROGRAMMIOSA, MILLE TÄITMISAEGA MÕÕDAME (algus)

# PROGRAMMIOSA, MILLE TÄITMISAEGA MÕÕDAME (lõpp)
b = time()

print(b - a) # PROGRAMMIOSA täitmisaeg sekundites
print((b - a)*1000) # PROGRAMMIOSA täitmisaeg millisekundites
```

Ülesanne 1

[Praktikumitöö] Fibonacci arvud defineeritakse seostega $f(0) = 0$, $f(1) = 1$ ja iga $n > 1$ korral $f(n) = f(n-1) + f(n-2)$.

a) Leida suurim indeks, millele vastava Fibonacci arvu on arvuti võimeline välja arvutama 1 sekundi jooksul rekursiivse meetodiga — Pythonis näiteks sellise meetodiga:

```
def fibo(n):
   if n < 1: return 0
   if n < 3: return 1
   return fibo(n - 1) + fibo(n - 2)

print (fibo(9))
```

b) Leitud väärtuse puhul teha kindlaks sellise meetodi tööaeg, kus Fibonacci arvude arvutamine on realiseeritud mitterekursiivse algoritmiga.

c) Määrata ka uue algoritmi puhul suurim indeks, millele vastava Fibonacci arvu suudab arvuti välja arvutada 1 sekundi jooksul.

Ülesanne 2

[Iseseisev töö]

a) Valida välja, realiseerida ja testida läbi üks ruutkeerukusega (keskmise ajalise keerukuse hinnanguga $\Theta(n^2)$) algoritm järjendi elementide ümberpaigutamiseks mittekahanevasse järjekorda. Sellisteks algoritmideks on näiteks mullimeetod (*bubble sort*), valikumeetod (*selection sort*), pistemeetod (*insertion sort*) jt. Testimisel mitte unustada nn äärejuhte (järjendid pikkusega 0, 1 või 2; ette antakse sorditud järjend, või siis vastupidises suunas sorditud järjend).

b) Valida välja, realiseerida ja testida läbi üks keskmise ajalise keerukuse hinnanguga $\Theta(n \log n)$ algoritm järjendi elementide ümberpaigutamiseks mittekahanevasse järjekorda. 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 ilmekalt demonstreeriksid kahe valitud sortimismeetodi tööaja kasvu vastavalt sorditavate järjendite pikkuste kasvule. Kolmandaks olgu joonisele lisatud ka süsteemse sortimismeetodi tööaja graafik (nt Pythoni korral meetod `sort()`).

Soovitus. Sorteerimismeetodite algoritme leiab kas Internetist või õpikust või loengumaterjalidest.