Vilniaus
universitetas

MATEMATIKOS IR INFORMATIKOS
FAKULTETAS

Užduotis – 27

K-ASIS SEKOS ELEMENTAS

Projektas

Parengė: Duomenų mokslo 2k. 2gr.

studentė Simona Gelžinytė

Vilnius, 2021

Turinys

1. Uždavinio formuluotė

Duota aibė A, turinti n skirtingų elementų, ir skaičius k, $1 \leq k \leq n$. **Rasti** k-ąjį pagal dydį aibės A elementą, t.y., tokį elementą $x \in A$, kuris yra didesnis už lygiai k – 1 aibės A elementą.

2. Realizuotų algoritmų aprašymas

Turime neišrūšiuotą skirtingų n elementų aibę ir ieškome k-ojo pagal dydį aibės A elemento.

a) jeigu aibės A elementų skaičius (elementų skaičius = n) yra mažiau nei 50, tuomet išrūšiuojame aibę didėjimo tvarka ir randame k-ąjį elementą.
b) jeigu aibės A elementų skaičius yra daugiau nei 50, tuomet aibę daliname į poaibius po 5 elementus (paskutinis poaibis gali būti mažesnis, jeigu n nesidalija be liekanos iš 5). Surūšiuojame visus poaibius didėjimo tvarka. Susikuriame naują aibę M, kurioje talpinsime kiekvieno poaibio rastą medianą. Aibė M talpins n/5 elementų skaičių ir taip 5 kartus greičiau rasime medianą nei imtumėm visą A aibę. Surandame aibės M medianą (aibės M mediana = m), jos ieškodami kviečiame tą pačią rekursyvią funkciją, kur k-asis ieškomas elementas lygus M/2 ir taip randame M medianą. Tuomet vėl einame per visą Aibę A ir lyginame kiekvieną elementą su m (ar jis mažesnis, didesnis arba lygus m elementui). Jeigu A aibės elementas mažesnis už m, įrašome į A1; jeigu lygus – į A2; jeigu didesnis į A3. Tuomet ieškome k-ojo pagal dydį skaičiaus, t.y. jeigu k < A1 dydį, vadinasi ieškomas elementas bus šioje aibėje ir vėl kviečiame tą pačią rekursyvią funkciją, tik jau su mažesne aibę, jei aibės dydis yra < 50 jau grąžiname atsakymą, jeigu ne – einame per visą funkciją dar kartą, kol gauname atsakymą. Jeigu k < (A1 dydis + A2 dydis), tiesiog grąžiname elementą m, nes A2 aibėje ir bus tik vienas elementas, kuris lygus m. Paskutiniu atveju iš k atimame A1 dydį ir A2 dydį, kadangi tose aibėse ieškomo elemento jau nebus. Taip rekursyviai naudodami vieną funkciją randame **k-ąjį pagal dydį aibės A elementą.**

Algoritmo kintamieji:
Uždaviniui spręsti naudoju:
1) vektorių ***A***, kuriame bus įrašomi visi elementai aibės A elementai.
2) kintamąjį ***k***, kuris nurodys kelintą pagal dydį norime rasti A elementą.
3) dvimatis vektorius ***poaibis***, kuriame bus įrašyta po 5 elementus (paskutiniajame gali būti nuo 1 iki 5 imtinai elementų) iš aibės *A*.
4) vektorių ***M***, kuriame bus įrašoma kiekvieno poaibio gauta mediana.
5) kintamasis ***m*** bus lygus *M* medianai.
6) vektorius ***A1*** – jame įrašomi elementai, kurie yra mažesni už *m.*
7) vektorius ***A2*** – jame įrašomas elementas, lygus *m.*
8) vektorius ***A3*** – jame įrašomi elementai, kurie yra didesni už *m*.

3. Pavyzdys su mažais duomenimis

Turime aibę A = { 1, 5, 9, 4, 6, 20, 101, 78, 99, 100, 39, 34, 36, 30, 33, 31, 32, 35, 37, 38, 50, 59, 51, 58, 53, 57, 54, 56, 55, 52, 49, 40, 48, 41, 47, 42, 46, 43, 45, 44, 80, 89, 81, 88, 83, 87, 84, 86, 85, 82, 97, 2, 7, 18 }. Ją sudaro 54 skirtingi elementai. **Ieškome 30-ojo pagal dydį elemento.**

1) dalijame A į poaibius (paskutiniajame bus 4 elementai)
   gauti poaibiai:
   p1 = {1, 5, 9, 4, 6}
   p2 = {20, 101, 78, 99, 100}
   p3 = {39, 34, 36, 30, 33}
   p4 = {31, 32, 35, 37, 38}
   p5 = {50, 59, 51, 58, 53}
   p6 = {57, 54, 56, 55, 52}
   p7 = {49, 40, 48, 41, 47}
   p8 = {42, 46, 43, 45, 44}
   p9 = {80, 89, 81, 88, 83}
   p10 = {87, 84, 86, 85, 82}
   p11 = {97, 2, 7, 18}.

2) išrūšiuojame kiekvieną poaibį ir randame jo medianą, o gautas medianas surašome į aibę M.
   p1 = {1, 4, 5, 6, 9}, mediana = 5;
   p2 = {20, 78, 99, 100, 101}, mediana = 99;
   p3 = {30, 33, 34, 36, 39}, mediana = 34;
   p4 = {31, 32, 35, 37, 38}, mediana = 35;
   p5 = {50, 51, 53, 58, 59}, mediana = 53;
   p6 = {52, 54, 55, 56, 57}, mediana = 55;
   p7 = {40, 41, 47, 48, 49}, mediana = 47;
   p8 = {42, 43, 44, 45, 46}, mediana = 44;
   p9 = {80, 81, 83, 88, 89}, mediana = 83;
   p10 = {82, 84, 85, 86, 87}, mediana = 85;
   p11 = {2, 7, 18, 97}, mediana = 12.5.

   Aibė M = {5, 99, 34, 35, 53, 55, 47, 44, 83, 85, 12.5}.

3) išrūšiuojame M aibę ir ieškome jos medianos,
   M = {5, 12, 34, 35, 44, 47, 53, 55, 83, 85, 99}
   šiuo atveju ji bus lygi 47 (m = 47).

4) einame per kiekvieną aibės A elementą ir lyginame jį su m. Gaunamos naujos A1, A2, A3 aibės:
   A1 = {1, 5, 9, 4, 6, 20, 39, 34, 36, 30, 33, 31, 32, 35, 37, 38, 40, 41, 42, 46, 43, 45, 44, 2, 7, 18};
   A2 = {47};
   A3 = {101, 78, 99, 100, 50, 59, 51, 58, 53, 57, 54, 56, 55, 52, 49, 48, 80, 89, 81, 88, 83, 87, 84, 86, 85, 82, 97}.

5) aibės A1 dydis yra 26, vadinasi toje aibėje ieškomo elemento nebus. Aibės A2 dydis yra 1, A1 dydis + A2 dydis = 27, kas yra mažiau už 30. Gauname, kad ieškomas elementas bus aibėje A3.

6) iš k (k = 30) atimame aibės A1 dydį ir A2 dydį, gauname 3. Vadinasi, ieškomas elementas pagal dydį išrūšiuotoje aibėje A3 bus 3-ias. Ir gauname atsakymą:

   30-asis pagal dydi elementas: 50

4. Algoritmo realizavimas

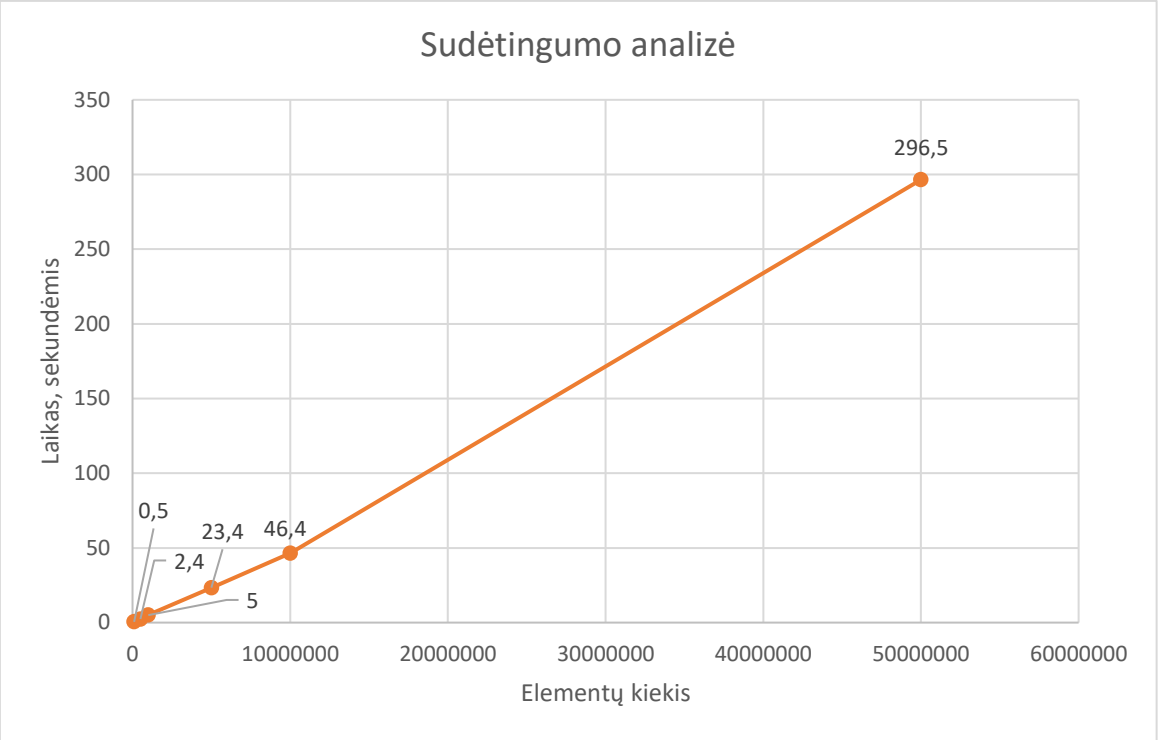
Algoritmas realizuotas C++ programavimo kalba. VisualStudio programa. (pilna programa pateikta priede)

```
long double Elemento_paieska(long double k, vector<long double>& A)
{
    if (A.size() < 50)
    {
        sort(A.begin(), A.end());
        return A.at(k - 1);
    }
    else
    {
        int n = 5;
        vector<vector<long double>> poaibis;
        long double poaibiu_skaicius = ceil(A.size() / 5.0);
        poaibis.resize(poaibiu_skaicius);
        vector<long double> M;

        for (long double i = 0; i < poaibiu_skaicius; i++)
        {
            poaibis[i].resize(n);
            int start_itr = i * n;
            int end_itr = i * n + n;

            if (end_itr > A.size())
            {
                end_itr = A.size();
                poaibis[i].resize(A.size() - start_itr);
            }

            copy(A.begin() + start_itr, A.begin() + end_itr,
poaibis[i].begin());
        }

        for (long double i = 0; i < poaibiu_skaicius; i++)
```

```
        {
            sort(poaibis[i].begin(), poaibis[i].end());
            long double med = mediana(poaibis[i]);
            M.push_back(med);
        }

        long double m = Elemento_paieska(ceil(M.size() / 2.0), M);

        vector<long double> A1;
        vector<long double> A2;
        vector<long double> A3;

        for (long double i = 0; i < A.size(); i++)
        {
            if (A.at(i) < m)
                A1.push_back(A.at(i));
            else if (A.at(i) == m)
                A2.push_back(A.at(i));
            else
                A3.push_back(A.at(i));
        }

        if (A1.size() >= k)
            return Elemento_paieska(k, A1);
        else if (A1.size() + A2.size() >= k)
            return m;
        else
            return Elemento_paieska(k - A1.size() - A2.size(), A3);
    }

}
```

5. Sudėtingumo analizė

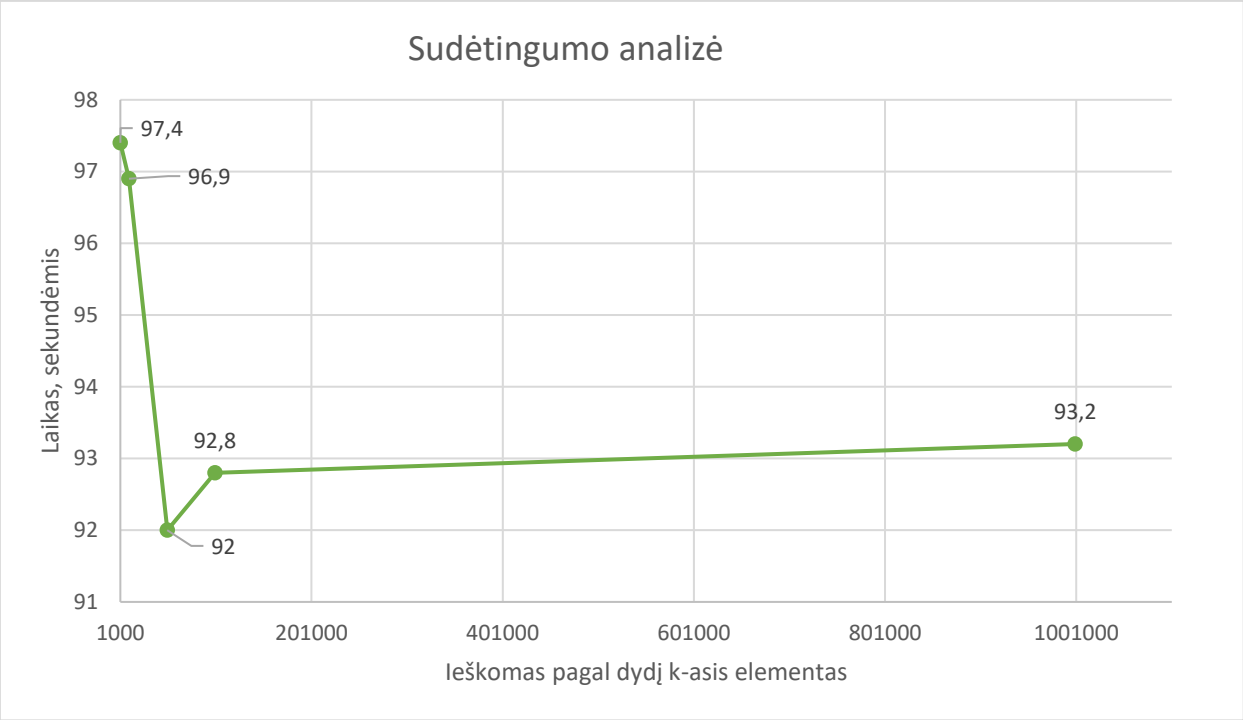
1 pav. vaizduoja kaip kinta programos vykdymo laikas, kai kinta tik n (elementų kiekis). Vykdant programą visada naudojau k = 105 t.y. kelinto pagal dydį ieškau elemento. Matome, kad didėjant elementų skaičiui, ilgėja ir programos vykdymo laikas, vadinasi didėja ir algoritmo sudėtingumas.



1 pav. Sudėtingumo analizė, kai kinta n (elementų kiekis)

Elementų kiekis	100000	500000	1000000	5000000	10000000	50000000
Laikas, sekundėmis	0,5	2,4	5	23,4	46,4	296,5

2 pav. vaizduoja kiek laiko trunka programa, kai kinta tik k (ieškomas pagal dydį elementas). Šioje analizėje n (elementų kiekis) visada bus lygus 20000000. Iš grafiko matome, kad ir kaip renkantis didelį k, tai nelabai turi įtakos programos vykdymo laikui, skirtumas tarp vykdymų laikų vos kelios sekundės.

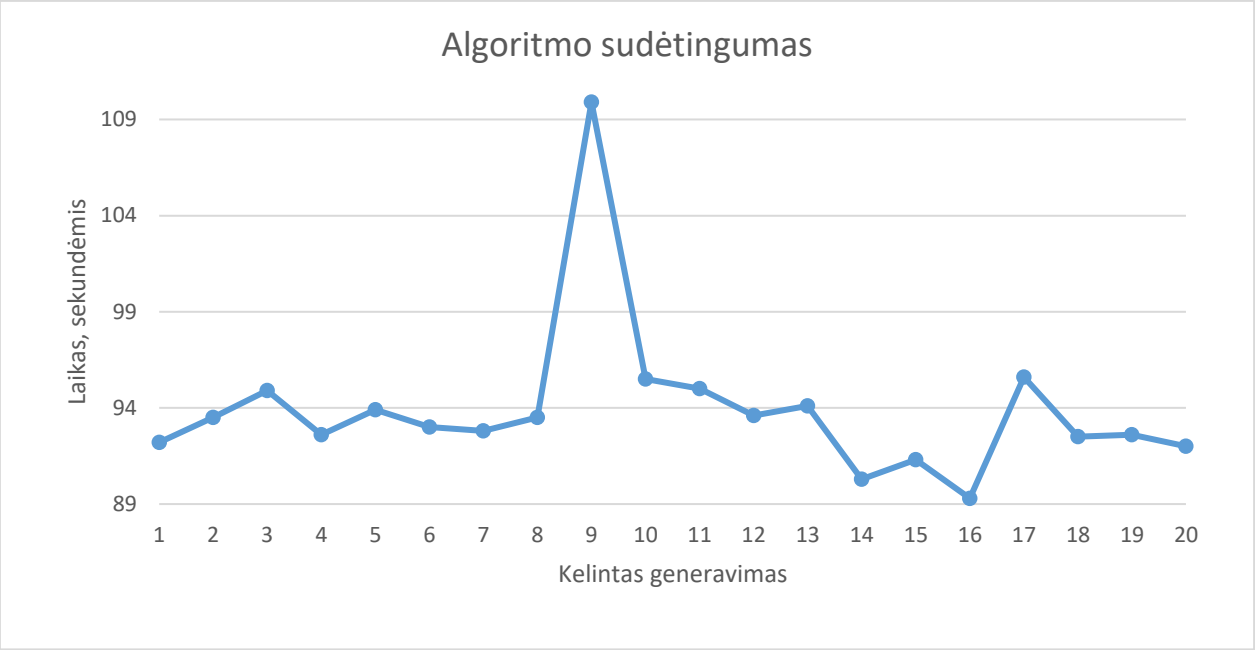


2 pav. Sudėtingumo analizė, kai kinta k (ieškomas pagal dydį k-asis elementas)

Ieškomas elementas pagal dydį	1000	10000	50000	100000	1000000
Laikas, sekundėmis	97,4	96,9	92	92,8	93,2

3 pav. atvaizduoja programos vykdymo laika, n ir k kiekvieno generavimo metu išlieka tokie patys.

Čia n = 20000000, o k = 50000. Vidutinis programos veikimo laikas lygus 93,91 sek. Algoritmo sudėtingumas lygus *O(n)*, kur n yra elementų skaičius.



3 pav. Algoritmo sudėtingumas

6. Eksperimentai

Eksperimentas nr. 1

Norime surasti 45 pagal dydį elementą. Sugeneruota aibė A sudaryta iš 80 skirtingų elementų

A = { 402, 353, -602, -426, -145, -361, -639, 781, -482, -577, 336, -542, 578, 286, 95, 616, -737, -420, -92, 146, -397, 371, -248, 294, -678, 778, 579, 527, 389, -65, 588, 59, 0, 218, -389, -759, -306, -699, -116, -676, 797, -512, 479, -646, -747, -31, -653, 587, -629, -64, 63, 185, -446, -285, 320, -265, 581, 266, -17, -119, -660, -321, -579, -110, 792, 675, -668, 413, -738246126, -633, 497, 83, 736, 782, -347, 551, 771, 408, 664, 653}

Kiek reikės poaibių, kad kiekviename poaibyje būtų 5 elementai (paskutiniame gali buti ir mažiau elementų): 16

m (Aibės M mediana): -92

45-asis pagal dydį elementas: 83

Programos vykdymo laikas : 0.0 sec

Eksperimentas nr. 2

Norime surasti 1 pagal dydį elementą. Sugeneruota aibė A sudaryta iš 92 skirtingų elementų

A = { 395, -28, 50, 28, -95, -369, -632, -391, -341, -604, -577, -374, -479, -769, -555, 881, -33, -260, 174, 799, 361, -427, 723, -635, -163, 30, 45, -578, -176, -381, 859, -244, -382, -14, 3.75311e+08, 469, -835, -610, -419, 617, -105, -456, -80, 186, 858, 8, 689, 236, -776, -298, -279, -592, -99, 834, -120, 275, -692, -518, -562, -124, 488, -901, 473, -865, -2, 524, 348, -41, 587, 260, 569, 305, -222, -329, 191, -490, -328, -201, 580, 225, 58, 791, -131, 721, 728, -228, -915, -525, 221, -63, 403, -2826}

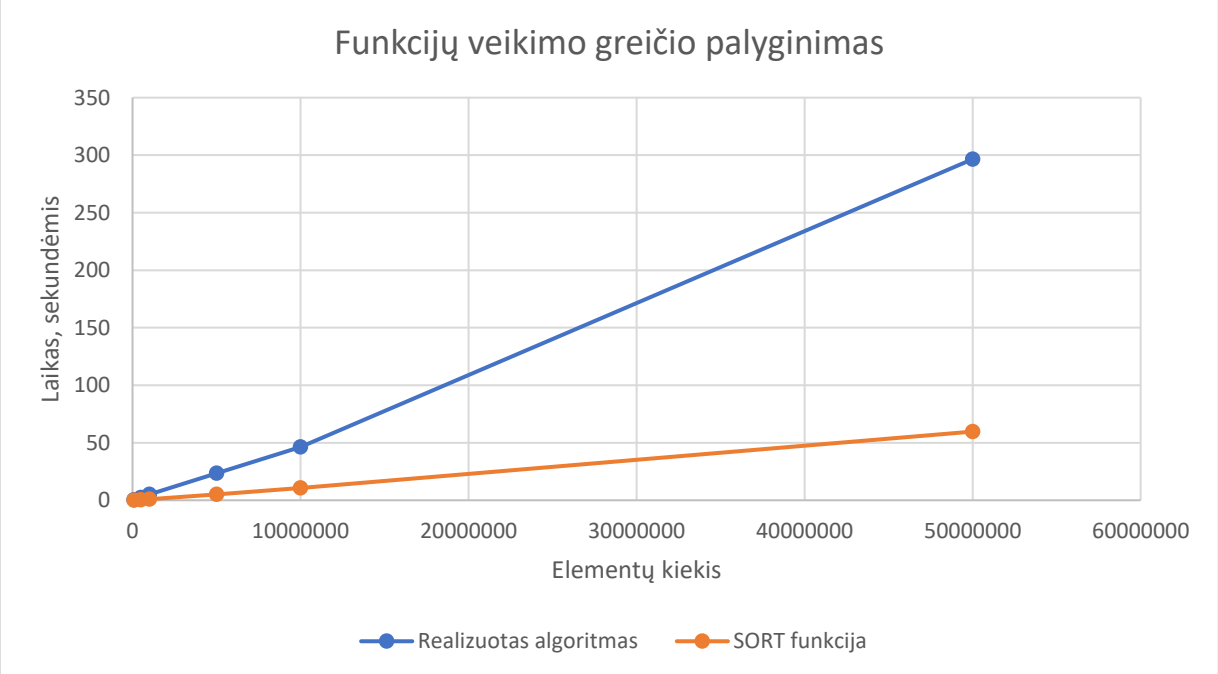
Kiek reikės poaibių, kad kiekviename poaibyje būtų 5 elementai (paskutiniame gali buti ir mažiau elementų): 19

m (Aibės M mediana): -120

1-asis pagal dydį elementas: -2826

Programos vykdymo laikas : 0.0 sec

7. Palyginimas su SORT funkcija



4 pav. Funkcijų veikimo greičio palyginimas

SORT funkcija:

Elementų kiekis	100000	500000	1000000	5000000	10000000	50000000
Laikas, sekundėmis	0,1	0,5	0,9	5	10,7	59,7

Realizuotas algoritmas:

Elementų kiekis	100000	500000	1000000	5000000	10000000	50000000
Laikas, sekundėmis	0,5	2,4	5	23,4	46,4	296,5

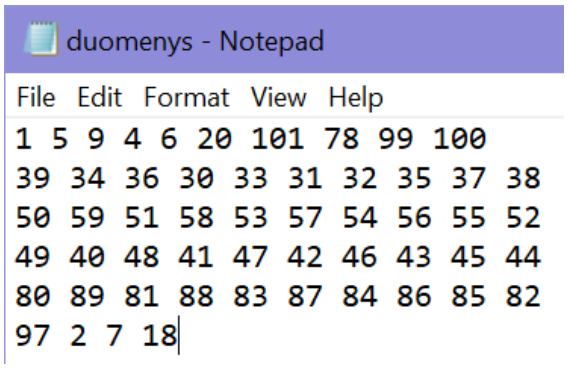
Iš grafiko matome, kad SORT funkcija esanti C++ programavimo kalboje veikia greičiau nei realizuotas algoritmas.

8. Išvados

1. Algoritmo sudėtingumas didėja, kai yra didinamas aibės A elementų skaičius (n).
2. Algoritmo sudėtingumas nepriklauso nuo k, buvo atliktas tyrimas, kai yra didinamas tik k (kelinto pagal dydį ieškosime elemento), jis yra 2 pav.
3. SORT funkcija esanti C++ programavimo kalboje veikia greičiau nei realizuotas algoritmas.

9. Naudojimosi instrukcija

1. Atidarykite Projektas_27.exe failą.
2. Atsidarys langas ir jame sekite nurodymus, įvedę 0 arba 1 paspauskite Enter klavišą.
   a) jeigu įrašėte 0, tuomet jums reikės įrašyti, kurio pagal dydį elemento ieškote ir paspausti Enter klavišą. Savo norimus elementus galite įrašyti duomenys.txt, kuris yra pateiktas priede (šiuo metu jame yra įrašyti elementai, kurie buvo naudojami pavyzdyje). Elementus atskirkite tarpu kaip pavaizduota 4 pav. (skirtumo nėra ar skaičiai bus išdėstyti viename stulpelyje ar vienoje eilutėje, aš juos išdėsčiau keliomis eilutėmis, nes man taip patogiau)



5 pav.

   b) jeigu įrašėte 1, tuomet jums reikės įrašyti, kurio pagal dydį elemento ieškote ir paspausti Enter klavišą. Toliau turėsite įrašyti kokio dydžio sugeneruotų duomenų failo norite ir

paspauskite Enter klavišą. Sugeneruotų duomenų failas taip pat atsiras aplankale pavadinimu pvz.: duomenys65.txt (skaičius nurodo kokio dydžio failo norėjote).

3. Rezultatas bei programos vykdymo laikas išvedamas į ekraną ir taip pat į Rezultatai.txt failą (šiuo metu ten yra įrašytas pavyzdžio rezultatas, tačiau paleidus Projektas_27.exe failą jame bus jau įrašomas naujas atsakymas).

10. Literatūra

1. A.V. Aho, J.E. Hopcroft, J.D. Ullman, The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading, MA, 1974, pp. 97--102.

2. T.H. Cormen, C.E. Leiserson and R.L. Rivest, Introduction to Algorithms, 3rd edition, MIT Press, Cambridge, MA, 2011, pp. 213--222.