

RŪŠIAVIMO ALGORITMAI IR JŲ REALIZACIJA C++

Laboratorinis darbas Nr. 3

TURINYS


1 UŽDUOTIS: INDIVIDUALUS PRISTATYMAS	1
REIKALAVIMAI PRISTATYMUJ	1
2 UŽDUOTIS: ALGORITMO REALIZACIJA C++ APLINKOJE	1
PAGALBINIAI NURODYMAI.....	3
RODYKLĖS TIPO (ANGL. POINTER) PANAUDOJIMAS PROGRAMŲ REALIZACIJOSE	3
REKOMENDUOJAMA LITERATŪRA	4

1 UŽDUOTIS: INDIVIDUALUS PRISTATYMAS

Parenkite **individualius** pristatymus (jų nereikės viešai pristatyti visuomenei, o tik parodyti ginantis darbus. Pačius geriausius kiekvieno rūšiavimo algoritmo pristatymus patalpinsiu FTP) apie šiuos „sparčius“ rūšiavimo algoritmus:

 Spartųjį rūšiavimo algoritmą (**angl. quick sort**).

Rengia tos grupės studentai, kurie darė pranešimą apie **stack** duomenų struktūrą.

 Suliejimo (sąlajos) rūšiavimo algoritmą (**angl. merge sort**).

Rengia tos grupės studentai, kurie darė pranešimą apie **queue** duomenų struktūrą.

 Piramidinį (krūvos) rūšiavimo algoritmą (**angl. heap sort**)

Rengia tos grupės studentai, kurie darė pranešimą apie **heap** duomenų struktūrą.

REIKALAVIMAI PRISTATYMUJ

Pristatyme turėtų būti:

- Paaiškintas (vaizdiniais pavyzdžiais pademonstruotas) algoritmo veikimas. Gali būti pateiktas algoritmo **pseudo kodas**.
- Pavyzdžiu (keliais pavyzdžiais) pademonstruotas jo veikimas. Kuo įdomesni ir „netikėtesni“ pavyzdžiai, tuo tik geriau.
- Įvertintas algoritmos sudėtingumas $T(N)$ pagrindinių operacijų: *elementų palyginimo* bei jų *sukeitimo* atžvilgiu.

2 UŽDUOTIS: ALGORITMO REALIZACIJA C++ APLINKOJE

Realizuokite Jums priklausančią spartųjį algoritmą C++ kalboje bei savo realizaciją „prijunkite“ prie jau esančių trijų standartinių rūšiavimo algoritmų realizacijos (**sort.cpp** failas patalpintas FTP):

```
1. #include <iostream>
2. #include <stdlib.h>
3. template <class T> void exch(T &, T &);
4. template <class T> void compexch(T &, T &);
5. template <class T> void selection(T *, int, int);
6. template <class T> void insertion(T *, int, int);
7. template <class T> void bubble(T *, int, int);
8. using namespace std;
9. int main(int argc, char *argv[])
10. {
11.     int i, N = atoi(argv[1]), KaDaryt = atoi(argv[2]);
12.     int *a = new int[N];
13.     if (KaDaryt) // Atsitiktinai sugeneruoja N skaičių intervale: 1,...,1000.
14.         for (i = 0; i < N; i++)
15.             a[i] = 1000*(1.0*rand())/RAND_MAX;
16.     else // Savo nuožiūra įvedame skaičius. Norėdami baigti, įvedame ne skaičių.
17.         { N = 0; while (cin >> a[N]) N++; }
18.     cout << "Įvestas skaičių masyvas yra:" << endl;
19.     for (i = 0; i < N; i++) cout << a[i] << " ";
20.     cout << endl;
21.     selection(a, 0, N-1);
22.     cout << "Surūšiuotas skaičių masyvas yra:" << endl;
23.     for (i = 0; i < N; i++) cout << a[i] << " ";
24.     cout << endl;
25. }
26.
27. // Sukeičia elementus vietomis
28. template <class T>
29. void exch(T &A, T &B)
30. { T t = A ; A = B; B = t; }
31. // Sukeičia elementus vietomis tik jei patenkinta sąlyga
32. template <class T>
33. void compexch(T &A, T &B)
34. { if (B < A) exch(A, B); }
35. // Išrinkimo algoritmo realizacija
36. template <class T>
37. void selection(T a[], int l, int r)
38. { for (int i = l; i < r; i++)
39.     { int min = i;
40.       for (int j = i+1; j <= r; j++)
41.           if (a[j] < a[min]) min = j;
42.       exch(a[i], a[min]);
43.     }
44. }
45. // Įterpimo algoritmo realizacija
46. template <class T>
47. void insertion(T a[], int l, int r)
48. { int i;
49.   for (i = r; i > l; i--) compexch(a[i-1], a[i]);
50.   for (i = l+2; i <= r; i++)
51.       { int j = i; T v = a[i];
52.         while (v < a[j-1])
53.             { a[j] = a[j-1]; j--; }
54.         a[j] = v;
55.       }
56. }
57. // Burbulo algoritmas
58. template <class T>
59. void bubble(T a[], int l, int r)
60. { for (int i = l; i < r; i++)
61.     for (int j = r; j > i; j--)
62.         compexch(a[j-1], a[j]);
63. }
```

PAGALBINIAI NURODYMAI

RODYKLĖS TIPO (ANGL. POINTER) PANAUDOJIMAS PROGRAMŲ REALIZACIJOSE

Norėdami iš funkcijos „gauti“ daugiau negu vieną reikšmę, C++ kalboje yra keletas būdų kaip tai padaryti.

1 būdas

Turime funkcijai perduoti pagrindinėje programoje susikurtą ir kompiuterio atmintyje saugomų kintamųjų adresus, su kuriais iškviesta funkcija dirbų tarsi „lokaliai“, tačiau pakoreguotos kintamųjų reikšmės būtų pasiekiamos ir už funkcijos ribų, nes visi pakeitimai būtų vykdomi į perduotą adresą vietą kompiuterio atmintyje. Norint tai padaryti, turime naudoti rodyklės tipą (angl. pointer'ius). Pavyzdžiui, norėdami kad išrinkimo rūšiavimo algoritmas suskaičiuotų kiek rūšiuodamas atliko lyginimo (kintamasis **L**) ir sukeitimo (kintamasis **S**) operacijų ir tos reikšmės būtų pasiekiamos už rūšiavimo algoritmo realizuojančios funkcijos **selection** ribų, turime atlikti tokius papildymus. Pirmiausia papildyti funkcijos deklaraciją:

```
template <class T> void selection(T *, int, int, int *, int *);
```

Tuomet pagrindinėje programoje kreipdamiesi į išrinkimo algoritmo funkciją, turime perduoti jau egzistuojančių kintamųjų **L** ir **S** adresus - **&L**, **&S**:

```
selection(a, 0, N-1, &L, &S);
```

Galiausiai pakoreguoti algoritmo realizuojančią funkciją **selection**:

```
void selection(T a[], int l, int r, int * L, int * S)
{
    for (int i = l; i < r; i++)
    { int min = i;
      for (int j = i+1; j <= r; j++)
        if (a[j] < a[min]) {
            min = j; *L = *L + 1;      // Atlikome lyginimo operaciją
        }
      exch(a[i], a[min]); *S = *S + 1; // Atlikome sukeitimo operaciją
    }
}
```

Plačiau apie rodyklės tipą, galite pasiskaityti: <http://www.cplusplus.com/doc/tutorial/pointers/>

Šioje situacijoje galima išsisukti ir be rodyklės tipo kintamųjų. Algoritmo realizuojančioje funkcijoje galima susikurti lokalų kintamąjį, kurio adresas kompiuterio atmintyje bus tas pats, kaip programoje susikurto kintamojo. Todėl reikšmę, esančią kompiuterio atmintyje tuo pačiu adresu, galėsime modifikuoti dviejų skirtingų kintamųjų pagalba. Norint tai atlikti, pirmiausia turime

Pirmiausia papildyti funkcijos deklaraciją:

```
template <class T> void selection(T *, int, int, int &, int &);
```

Tuomet pagrindinėje programoje kreipdamiesi į išrinkimo algoritmo funkciją, turime perduoti jau egzistuojančius kintamuosius **L** ir **S**:

```
selection(a, 0, N-1, L, S);
```

Galiausiai pakoreguoti algoritmo realizuojančią funkciją **selection**:

```
void selection(T a[], int l, int r, int &L, int &S)
{
    for (int i = l; i < r; i++)
    {
        int min = i;
        for (int j = i+1; j <= r; j++)
            if (a[j] < a[min]) {
                min = j; L++; // Atlikome lyginimo operaciją
            }
        exch(a[i], a[min]); S++; // Atlikome sukeitimo operaciją
    }
}
```

REKOMENDUOJAMA LITERATŪRA

1. R. Čiegis. *Duomenų struktūros, algoritmai ir jų analizė*.
2. A. Juozapavičius. *Duomenų struktūros ir efektyvūs algoritmai*.
3. R. Sedgewick. *Algorithms in C++, Parts 1–4: Fundamentals, Data Structure, Sorting, Searching*.
4. http://en.wikipedia.org/wiki/Quick_sort
5. http://en.wikipedia.org/wiki/Merge_sort
6. http://en.wikipedia.org/wiki/Heap_sort
7. www.google.lt ☺