ლაბ 11: გადათვლით დახარისხება - Counting sort

განსახილველი საკითხები:

- ამოცანა 1 /გადათვლის დახარისხების ალგორითმი, პარამეტრიზებული ვექტორებით/
- ამოცანა 2 /გადათვლის დახარისხების ალგორითმი, პარამეტრიზებული ორმხრივი და სწრაფი წვდომის იტერატორებით/ >>>
- სავარჯიშოები >>>

ამოცანა 1 /გადათვლის დახარისხების ალგორითმი, პარამეტრიზებული ვექტორებით/. "data.in" ფაილში ჩაწერილია არაუარყოფითი მთელი რიცხვები. მაგალითად:

31 1 1 1 1 1 2 1 2 1 7 1 5 0 13 7 17 4

შექმენით ფუნქცია **void** countingSort(vector<**int**>& a, vector<**int**>& b); რომელიც გადათვლის (counting sort) ალგორითმით დაახარისხებს **a** ვექტორს და დახარისხებულ მნიშვნელობებს **b** ვექტორში ჩაწერს.

"**main.cpp**" ფაილში მოათავსეთ ძირითადი პროგრამა, რომელიც "data.in" ფაილიდან მონაცემებს წაითხავს a ვექტორში, გააკეთებს განაცხადს დამხმარე b ვექტორზე, გამოიძახებს დახარისხების (countingSort) ფუნქციას და დაბეჭდავს b ვექტორს.

შენიშვნა: რადგან არ ვიცით ვექტორის ელემენტების დიაპაზონის ზედა ზღვარი (ვიცით მხოლოდ რომ ეს რიცხვები არის არაუარყოფითი მთელი რიცხვები, ანუ ნულზე ნაკლები არ იქნება), შესაბამისად დაგვჭირდება ვექტორში მაქსიმალური ელემენტის მოძებნა. ამისთვის ვსარგებლობთ ალგორითმით max_element, რომელიც გარკვეულ დიაპაზონში ეძებს მაქსიმალურ ელემენტს და აბრუნებს მის იტერატორს.

ამოხსნა: თუ დახარისხების ფუნქციას და ძირითად პროგრამას მოვათავსებთ ერთ ფაილში, გვექნება:

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <fstream>

using namespace std;

void countingSort(vector<int>& a, vector<int>& b)
{
    int k = *max_element(a.begin(), a.end());
    k++;
    vector<int> c(k);

    for (int i=0; i< a.size(); i++)
         c[a[i]]++;

    for (int i=1; i<k; i++)
         c[i] += c[i-1];

    for (int i=a.size()-1; i>=0; i--)
    {
        b[c[a[i]]-1] = a[i];
        c[a[i]]--;
    }
}

int main() {
    int n, k;
    vector<int> a;
```

```
    ifstream ifs("data.in");
    while(ifs >> n)
        a.push_back(n);

    vector<int> b(a.size());

    countingSort(a, b);

    for(int i=0; i<b.size(); ++i)
    {
        cout << b[i] << "\t";
        if( 0 == (i+1)%5 )
           cout << endl;
    }
    cout << endl;
    system("PAUSE");
    return (0);
}
```

<<< **ამოცანა 2** /გადათვლის დახარისხების ალგორითმი, პარამეტრიზებული ორმხრივი და სწრაფი წვდომის იტერატორებით/. გადავჭრათ იგივე ამოცანა სწრაფი ორმხრივი და წვდომის კონტეინერების გამოყენებით, რაც ბევრად მეტ მოქნილობას მიანიჭებს ალგორითმს. მაგალითად, პროგრამა-დრაივერში, ერთი და იგივე ფუნქციის გამოყენებით, დახარისხდება ვექტორის ელემენტები და ისინი ჩაიწერება დეკში. შემდეგ დახარისხდება სიის ელემენტები და ისინი ჩაიწერება მასივში. ორივე შემთხვევაში, მონაცემების გამოსატანად ვიყენებთ ostream_iterator<Y>-ს. შემდეგი კოდი გვიჩვენებს, რომ პირველი კონტეინერის იტერატორისგან, რომელიც შეიცავს დასახარისხებელ მონაცემებს, საკმარისია ორჯერ შემოიაროს კონტეინერი, ჯერ ++, შემდეგ -- ოპერატორების გამოყენებით. პირველ შემთხვევაში ჩვენს ვიცით თუ როგორ გავჩერდეთ, - last იტერატორთან. ამავე დროს ვიმახსოვრებთ რამდენჯერ გამოვიყენეთ ++ ოპერატორი, რომ იმდენჯერვე შევამციროთ first, რათა შებრუნებული რიგით შემოვიაროთ საწყისი კონტეინერი. კოდს აქვს სახე:

```
#include <iostream>
#include <vector>
#include <deque>
#include <list>
#include <algorithm>
#include <fstream>
#include<iterator>
using namespace std;

template<typename BidirectionalIterator,
        typename RandomAccessIterator>
void countingSort(BidirectionalIterator first,
                  BidirectionalIterator last,
                  RandomAccessIterator result
                  )
{
    int k = *max_element(first, last);
    k++;

    vector<int> c(k);

    int size(0);
    while(first != last)
    {
         c[*first]++;
         first++;
         size++;
```

```
    }
    first--;
    for (int i=1; i<k; i++)
         c[i] += c[i-1];

    for (int i=0; i<size; i++)
    {
        result[c[*first]-1] = *first;
        c[*first]--;
        first--;
    }
}

int main() {
    int n, k;
    //New vector container
    vector<int> a;

    ifstream ifs("data.in");
    while(ifs >> n)
        a.push_back(n);
     //New deque container to store sorted sequence
    deque<int> b(a.size());
    //Sorting data
    countingSort(a.begin(),a.end(), b.begin());
    //Printing data using ostream_iterator
    ostream_iterator<int> out(cout,"\t");
    copy(b.begin(),b.end(),out);
    cout << endl << endl;
    //New list container with unordered data
    list<int> lst(a.begin(), a.end());
    //New array to store sorted sequence
    int mas[lst.size()];
    //Sorting data
    countingSort(lst.begin(),lst.end(), mas);
    //Printing data using ostream_iterator
    copy(mas,mas+lst.size(),out);
    cout << endl << endl;

    system("PAUSE");
    return (0);
 }
```

<<< სავარჯიშოები:

1. დახარისხების ფუნქცია (ორივე ვარიანტი) გადავაკეთოთ ისე, რომ იგი მუშაობდეს მთელი რიცხვებისთვის, მათი ნიშნისგან დამოუკიდებლად.
2. დახარისხების ფუნქცია გადავტვირთოთ, რომ იგი მუშაობდეს არა მხოლოდ მთელ რიცხვებთან, არამედ კლასის ობიექტებთან, რომლებისთვისაც განსაზღვრულია გასაღები- ანუ ფუნქცია ამ კლასის ობიექტებზე მნიშვნელობებით მთელ რიცხვებში.