პრაქტიკული 11: **გადათვლით დახარისხება (Counting sort)**

სააუდიტორიო სამუშაო:

- ამოცანა
- ალგორითმის კოდი >>>
- ტრასირება >>>
- სავარჯიშოები >>>

ამოცანა. მთელი არაუარყოფითი რიცხვების `a` ვექტორი

i	0	1	2	3	4	5	6	7	8
a[i]	3	5	0	3	3	1	5	3	2

დაახარისხეთ გადათვლის (CountingSort) ალგორითმით. აჩვენეთ როგორ იცვლება დამხმარე `c` ვექტორი, რომლის ელემენტების რაოდენობა არის `k+1`

`(k=max{a[0],a[1],...,a[a.size()-1]}),`

და საბოლოო `b` ვექტორები.

ამოხსნა: გადათვლით დახარისხების ალგორითმის იდეა შემდეგია: თუ კონტეინერის ნებისმიერი x ელემენტისათვის წინასწარ დავთვლით ამ კონტეინერის რამდენი ელემეტია x -ზე ნაკლები (ვთქვათ m), მაშინ ის შეგვიძლია საბოლოო კონტეინერში პირდაპირ ჩავწეროთ (m +1) – ე ადგილზე, ანუ ინდექსით m. თუ შემავალ კონტეინერში გვხვდება ერთმანეთის ტოლი რიცხვები, მაშინ დამატებით უნდა ვიზრუნოთ, რომ ტოლი რიცხვები ერთმანეთის მეზობლად და ძველი რიგის შენარჩუნებით განლაგდეს.

მოვიყვანოთ გადათვლით დახარისხების ალგორითმის ფსევდოკოდი და ამოცანის მონაცემიდან გამომდინარე თვალი გავადევნოთ ალგორითმის სტრიქონების მიმდევრობას.

<<< ალგორითმის კოდი:

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

3    vector<int> c(k);

4    for (int i=0; i< a.size(); i++)
5         c[a[i]]++;
6    for (int i=1; i<k; i++)
7         c[i] += c[i-1];

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

დამხმარე c ვექტორის ბოლო ინდექსია k=max{3, 5, 0, 3, 3,1, 5, 3, 2}=5, ამიტომ დამხმარე ვექტორი არის ექვსელემენტიანი. მასზე განაცხადი კეთდება სტრიქონში 3; ყურადღება მივაქციოთ, რომ განაცხადის გაკეთების მომენტშივე ხდება ვექტორის ინიციალიზაცია ნულებით. მაქსიმალური ელემენტის იტერატორის განსაზღვრისთვის ვიყენებთ STL-ის ალგორითმს `max_element`, რომლის რამდენიმე გადატვირთული ვერსია არსებობს.

i	0	1	2	3	4	5
c[i], განაცხადის გაკეთების შემდეგ	0	0	0	0	0	0
c[i], I for-ის შემდეგ	1	1	1	4	0	2
c[i], II for-ის შემდეგ	1	2	3	7	7	9

I for შეტყობინება საშუალებას გვაძლევს განვსაზღვროთ, თუ საწყისი ვექტორის თითოეული ელემენტი რამდენ ეკზემპლიარად გვხდება ამ ვექტორში, II for შეტყობინება საშუალებას გვაძლევს განვსაზღვროთ, თუ თითოეული ელემენტი რამდენ ელემენტზე არის მეტი ან ტოლი.

III for-ის მუშაობის შედეგი ბიჯების მიხედვით, ანუ

<<< ალგორითმის ტრასირება:

```
1) i=8,  a[8]=2,  c[a[8]]=c[2]=3 ⇒ b[3-1]=b[2]=2, c[2]=2;
2) i=7,  a[7]=3,  c[a[7]]=c[3]=7 ⇒ b[7-1]=b[6]=3, c[3]=6;
3) i=6,  a[6]=5,  c[a[6]]=c[5]=9 ⇒ b[9-1]=b[8]=5, c[5]=8;
4) i=5,  a[5]=1,  c[a[5]]=c[1]=2 ⇒ b[2-1]=b[1]=1, c[1]=1;
5) i=4,  a[4]=3,  c[a[4]]=c[3]=6 ⇒ b[6-1]=b[5]=3, c[3]=5;
6) i=3,  a[3]=3,  c[a[3]]=c[3]=5 ⇒ b[5-1]=b[4]=3, c[3]=4;
7) i=2,  a[2]=0,  c[a[2]]=c[0]=1 ⇒ b[1-1]=b[0]=0, c[0]=0;
8) i=1,  a[1]=5,  c[a[1]]=c[5]=8 ⇒ b[8-1]=b[7]=5, c[5]=7;
9) i=0,  a[0]=3,  c[a[0]]=c[3]=4 ⇒ b[4-1]=b[3]=3, c[3]=3;
```

b ვექტორი, 9-ელემენტიანი

i	0	1	2	3	4	5	6	7	8
i=8			**2**						
i=7			2				**3**		
i=6			2				3		**5**
i=5		**1**	2				3		5
i=4		1	2			**3**	3		5
i=3		1	2		**3**	3	3		5
i=2	**0**	1	2		3	3	3		5
i=1	0	1	2		3	3	3	**5**	5
i=0	0	1	2	**3**	3	3	3	5	5

c ვექტორი, 9-ელემენტიანი

1	2	3	7	7	9
1	2	**2**	7	7	9
1	2	2	**6**	7	9
1	2	2	**6**	7	**8**
1	**1**	2	6	7	**8**
1	1	2	**5**	7	8
1	1	2	**4**	7	8
0	1	2	4	7	8
0	1	2	**4**	7	**7**
0	1	2	**3**	7	7

<<< სავარჯიშოები

1. მოცემულია **მთელი არაუარყოფით** რიცხვების n =10 ელემენტიანი ვექტორი

ა. a[n]={2, 3, 1, 2, 3, 0, 5, 7, 1, 2 },

ბ. a[n]={1, 3, 1, 6, 3, 0, 5, 6, 1 },

გ. a[n]={0, 2, 1, 2, 3, 0, 5, 3},

დაახარისხეთ იგი გადათვლის (CountingSort) ალგორითმით და აჩვენეთ ბიჯების მიხედვით როგორ იცვლება დამხმარე c ვექტორი,

(k=max{a[0],a[1],...,a[a.size()-1]}),

და საბოლოო b[n] ვექტორი.

2*. მოცემულია **მთელი** რიცხვების n ელემენტიანი a[n] ვექტორი,, რომელშიც

k1=min{a[0],a[1],..., a[a.size()-1]}, k2=max{a[0],a[1],..., a[a.size()-1]}.

დაწერეთ გადათვლით დახარისხების (CountingSort) ალგორითმის მოდიფიცირებული ვარიანტი, რომელიც დამხმარე მასივის ელემენტების რაოდენობა იქნება (k2 - k1).

3. ალგორითმის რომელი ვერსიის გამოყენებაა მიზანშეწონილი, თუ გვინდა დავახარისხოთ 1000 მთელი რიცხვი, მოთავსებული [3002001, 3004002] შუალედში?