

**PRAKTIKUM ANALISIS ALGORITMA**

**Untuk Memenuhi Tugas**



**Disusun oleh :**

**Mutia Karimah**

**140810170002**

**TEKNIK INFORMATIKA**

**FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM**

**UNIVERSITAS PADJADJARAN**

**2019**

## Tugas 2

### CountingSort.cpp

```
#include<iostream>

using namespace std;

int k=0;

/*Method to sort the array*/

void Counting_Sort(int A[],int B[],int n)
{
    int C[k];

    for(int i=0;i<k+1;i++)
    {
        /*It will initialize the C with zero*/

        C[i]=0;
    }

    for(int j=1;j<=n;j++)
    {
        /*It will count the occurrence of every element x in A
        and increment it at position x in C*/

        C[A[j]]++;
    }

    for(int i=1;i<=k;i++)
    {
```

```

        /*It will store the last
        occurence of the element i */

        C[i]+=C[i-1];

    }

    for(int j=n;j>=1;j--)

    {

        /*It will place the elements at their
        respective index*/

        B[C[A[j]]]=A[j];

        /*It will help if an element occurs
        more than one time*/

        C[A[j]]=C[A[j]]-1;

    }

}

int main()

{

    int n;

    cout << "Masukkan jumlah array :";

    cin>>n;

    cout << "Masukkan element : ";

    /*A stores the elements input by user */

    /*B stores the sorted sequence of elements*/

```

```

int A[n],B[n];

for(int i=1;i<=n;i++)
{
    cin>>A[i];

    if(A[i]>k)
    {

        /*It will modify k if an element
        occurs whose value is greater than k*/

        k=A[i];

    }
}

Counting_Sort(A,B,n);

/*It will print the sorted sequence on the
console*/

cout << "=====Hasil Counting Sort===== " <<endl ;

for(int i=1;i<=n;i++)
{

    cout<<B[i]<<" ";

}

cout<<endl;

return 0;

}

```

```
input
Masukkan jumlah array :6
Masukkan element : 18
14
10
6
2
9
=====Hasil Counting Sort=====
2 6 9 10 14 18
...Program finished with exit code 0
Press ENTER to exit console.
```

### Kompleksitas waktu dan big-O :

#### *Pseudo-code*

```
Input : A : array [1..n] of integer, k: max (A)
Output : B : array [1..n] of integer
    for i = 1 to k do
        C[i] = 0
    for j = 1 to length(A) do
        C[A[j]] = C[A[j]] + 1
    for i = 1 to k do
        C[i] = C[i] + C[i-1]
    for j = 1 to length(A) do
        B[C[A[j]]] = A[j]
        C[A[j]] = C[A[j]] - 1
    return B
```

Waktu yang dibutuhkan untuk mengurutkan data menggunakan counting sort bisa didapatkan dari perhitungan sebagai berikut :

- For pertama membutuhkan waktu  $O(k)$ ,
- For kedua membutuhkan waktu  $O(n)$ ,
- For ketiga membutuhkan waktu  $O(k)$ , dan
- For keempat membutuhkan waktu  $O(n)$ .

Jadi secara total membutuhkan waktu  $O(k+n)$ , yang seringkali dianggap  $k = O(n)$

### Step by step :

Misal array data yang akan diurutkan adalah A. Counting sort membutuhkan sebuah array C berukuran k, yang setiap elemen  $C[i]$  merepresentasikan jumlah elemen dalam A yang nilainya adalah i. Di array inilah penghitungan (counting) yang dilakukan dalam pengurutan ini disimpan. Misal kita akan melakukan pengurutan pada array A sebagai berikut, dengan n adalah 10 dan diasumsikan bahwa rentang nilai setiap  $A[i]$  adalah 1..5

A

1	3	5	4	5	2
1	2	3	4	5	6

Dan array C setelah diinisialisasi adalah :

C

0	0	0	0	0
1	2	3	4	5

Kemudian proses penghitungan pun dimulai, proses ini linier, dilakukan dengan menelusuri array A, Langkah 1 : pembacaan pertama mendapat elemen  $A[1]$  dengan isi 1, maka  $C[1]$  ditambah 1.

A

1	3	5	4	5	2
1	2	3	4	5	6

C

1	0	0	0	0
1	2	3	4	5

Langkah 2 : pembacaan kedua mendapat elemen  $A[2]$  dengan isi 3, maka  $C[3]$  ditambah 1.

*A*

1	3	5	4	5	2
1	2	3	4	5	6

*C*

1	0	1	0	0
1	2	3	4	5

Langkah 3 : pembacaan ketiga mendapat elemen  $A[3]$  dengan isi 5, maka  $C[5]$  ditambah 1.

*A*

1	3	5	4	5	2
1	2	3	4	5	6

*C*

1	0	1	1	1
1	2	3	4	5

Demikian dilakukan terus menerus hingga semua elemen  $A$  telah diakses. Lalu array  $C$  diproses sehingga setiap elemen  $C$ ,  $C[i]$  tidak lagi merepresentasikan jumlah elemen dengan nilai sama dengan  $i$ , namun setiap  $C[i]$  menjadi merepresentasikan jumlah elemen yang lebih kecil atau sama dengan  $i$ . Dalam proses ini kita mengakses elemen  $A[i]$ , kemudian memposisikannya di posisi sebagaimana tercatat dalam  $C[A[i]]$ , kemudian kita mengurangkan  $C[A[i]]$  dengan 1, yang dengan jelas untuk memberikan posisi untuk elemen berikutnya dengan yang isinya sama dengan  $A[i]$ . Proses ini memerlukan sebuah array bantu  $B$  yang ukurannya sama dengan array  $A$ , yaitu  $n$ . Yang pada awalnya semua  $B[i]$  diinisialisasi dengan nil

*B*

-	-	-	-	-	-
1	2	3	4	5	6

Langkah 1 : elemen  $A[10]$  adalah 2, maka karena  $C[5]$  adalah 10, maka  $B[6]$  diisi dengan 5, dan  $C[5]$  dikurangi 1.

*A*

1	3	5	4	5	2
1	2	3	4	5	6

*B*

-	-	-	-	-	-
1	2	3	4	5	6

*C*

1	3	6	7	9
1	2	3	4	5

Demikian proses dilakukan hingga elemen  $A[1]$  selesai diproses, sehingga didapatkan hasil akhir

*B*

1	2	2	3	3	3
1	2	3	4	5	6

**Contoh soal dan running time :**

	Keys Type	Average run-time	Worst case run-time	Extra Space	In Place	Stable
Insertion Sort	Any	$O(n^2)$	$O(n^2)$	$O(1)$	✓	✓
Merge Sort	Any	$O(n \log n)$	$O(n \log n)$	$O(n)$	X	✓
Heap Sort	Any	$O(n \log n)$	$O(n \log n)$	$O(1)$	✓	X
Quick Sort	Any	$O(n \log n)$	$O(n^2)$	$O(1)$	✓	X
Counting Sort	integers [1..k]	$O(n+k)$	$O(n+k)$	$O(n+k)$	X	✓
TPS Sort	integers [1..n]	$O(n)$	$O(n)$	$O(n)$	X	✓
Radix Sort	d digits in base b	$O(d(b+n))$	$O(d(b+n))$	Depends on the stable sort used	Depends on the stable sort used	✓
Bucket sort	[0,1)	$O(n)$	$O(n^2)$	$O(n)$	X	✓

Count Sort adalah algoritma pengurutan bilangan bulat, bukan algoritma berbasis perbandingan. Sementara setiap algoritma sorting membutuhkan perbandingan  $\Omega(n \log n)$ . Sementara count sort membutuhkan  $O(n)$  untuk running timenya. Perubahan nilai k akan mempengaruhi jumlah running time.