Algoritma Count Sort

Counting Sort adalah algoritma pengurutan efektif dan efisien yang melakukan pengurutan dengan ide dasar meletakkan elemen pada posisi yang benar, dimana penghitungan posisi yang benar dilakukan dengan cara menghitung (counting) elemen-elemen dengan nilai lebih kecil atau sama dengan elemen tersebut. Contoh sederhana saja jika terdapat 12 elemen yang lebih kecil daripada x, maka x akan mendapatkan posisinya di posisi 13.

1. Count Sort efisien jika rentang input data tidak jauh lebih besar dari jumlah objek yang akan diurutkan. Pertimbangkan situasi di mana urutan input berada di antara rentang 1 hingga 10K dan data adalah 10, 5, 10K, 5K.
1. Ini bukan penyortiran berdasarkan perbandingan. Ini menjalankan kompleksitas waktu adalah O (n) dengan ruang proporsional dengan rentang data.
2. Ini sering digunakan sebagai sub-rutin untuk algoritma pengurutan lainnya seperti radix sort.
3. Counting sort menggunakan hashing parsial untuk menghitung kemunculan objek data di O (1).
4. Jenis penghitungan dapat diperluas untuk bekerja untuk input negatif juga.

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 2 = 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-Step Count Sort (6 Input)

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.

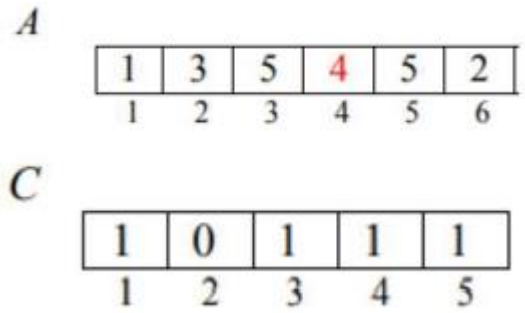
A

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

C

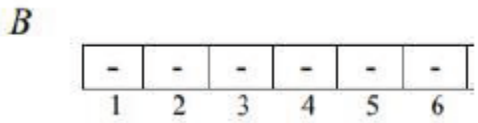
1	0	1	0	0
1	2	3	4	5

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

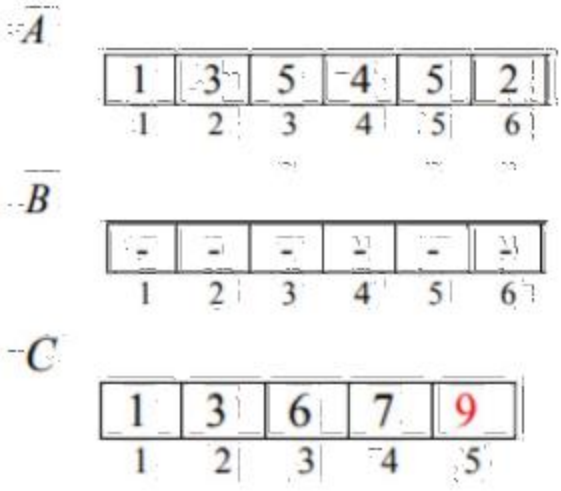


Demikian dilakukan terus menerus hingga semua elemen A telah diakses. Lalu array C diproses sehingga setiap elemen C, C[i] tidak lagi merepresentasikan jumlah elemen dengan nilai sama dengan i, namun setiap C[i] menjadi merepresentasikan jumlah elemen yang lebih kecil atau sama dengan i.

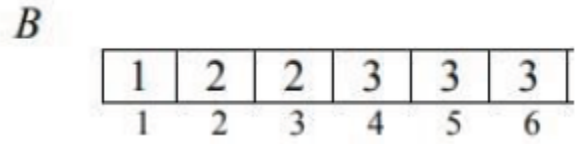
Dalam proses ini kita mengakses elemen A[i], kemudian memposisikannya di posisi sebagaimana tercatat dalam C[A[i]], kemudian kita mengurangkan C[A[i]] dengan 1, yang dengan jelas untuk memberikan posisi untuk elemen berikutnya dengan yang isinya sama dengan A[i]. Proses ini memerlukan sebuah array bantu B yang ukurannya sama dengan array A, yaitu n. Yang pada awalnya semua B[i] diinisialisasi dengan nil.



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



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



Running Time

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

	Keys Type	Average run-time	Worst case run-time	Extra Space	In Place	Stable
Insertion Sort	Any	O(n2)	O(n2)	O(1)	√	√
Merge Sort	Any	O(nlogn)	O(nlogn)	O(n)	X	√
Heap Sort	Any	O(nlogn)	O(nlogn)	O(1)	√	X
Quick Sort	Any	O(nlogn)	O(n2)	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(n2)	O(n)	X	√

Maka, kompleksitas waktu counting sort O(k) + O(n) + O(k) + O(n) = O($k + n$)