**Insertion Sort.**

Dalam teknik Insertion Sort, kami mulai dari elemen kedua dan membandingkannya dengan elemen pertama dan meletakkannya di tempat yang tepat. Kemudian kami melakukan proses ini untuk elemen selanjutnya.

1. Psudeocode.

**procedure** insertionSort(array A,n)

*A - array to be sorted*

*n - number of elements*

**for** i = 0 **to** n

key = A[i]

j = i - 1

**while** j >= 0 **and** A[j] > key **do**

A[j + 1] = A[j]

j = j - 1

**end** **while**

A[j + 1] = key

**end** **for**

1. C(n), T(n), Kelas Efisiensi.

Kami mengasumsikan Biaya setiap operasi i sebagai Ci dimana i ∈ {1,2,3,4,5,6,8}. Lalu menghitung berapa kali ini dijalankan. Oleh karena itu, Biaya Total untuk satu operasi semacam itu akan menjadi produk dari Biaya satu operasi dan berapa kali itu dijalankan. Kami dapat mencantumkannya seperti di bawah ini:

Ci where i ∈ {1,2,3,4,5,6,8}.

**C1** n

**C2** n - 1

**C3** n - 1

**C4** Σn - 1j = 1(tj)

**C5** Σn - 1j = 1 (tj - 1)

**C6** Σn - 1j = 1(tj - 1)

**C8** n - 1

Total Running Time:

**T(n)** = C1 \* n + (C2+C3) \* (n-1) + C4 \* Σn - 1j = 1(tj) + (C5+C6) \* Σn - 1j = 1(tj) + C8 \* (n–1)

Kelas Efisiensi

**Best Case Analysis**

Array telah disortir, tj = 1.

T(n) = C1 \* n + (C2+C3) \* (n-1) + C4 \* (n-1 + (C5+ C) \* (n-2) + C8 \* (n-1)

Yang bila disederhanakan lebih lanjut memiliki faktor dominan **n** dan:

**T(n)** = C \* (n) or **O(n)**

**Worst Case Analysis**

Array tersusun descending tj = j.

T(n) = C1 \* n + (C2+C3) \* (n-1) + C4 \* (n-1) (n)/2 + (C5+C6) \* ((n-1)(n)/2-1) + C8 \* (n-1)

Yang bila disederhanakan lebih lanjut memiliki faktor dominan **n2** dan:

**T(n)** = C \* (n2) or **O(n2)**

**Average Case Analysis**

Asumsikan tj = (j-1)/2 untuk menghitung Average Case.

T(n) = C1 \* n + (C2+C3) \* (n-1) + C4/2 \* (n-1) (n)/2 + (C5+C6)/2 \* ((n-1) (n)/2-1) + C8 \* (n-1)

Yang bila disederhanakan lebih lanjut memiliki faktor dominan **n2** dan:

**T(n)** = C \* (n2) or **O(n2)**

1. Ilustrasi pengurutan bilangan acak 10 bilangan.

|  |  |  |  |
| --- | --- | --- | --- |
| Pass | Unsorted List | Comparison | Sorted List |
| 1 | {12,3,5,10,81,15,6,4,2} | {12,3} | {3,12,5,10,8,1,15,6,4,2} |
| 2 | {3,12,5,10,8,1,15,6,4,2} | {3,12,5} | {3,5,12,10,8,1,15,6,4,2} |
| 3 | {3,5,12,10,8,1,15,6,4,2} | {3,5,12,10} | {3,5,10,12,8,1,15,6,4,2} |
| 4 | {3,5,10,12,8,1,15,6,4,2} | {3,5,10,12,8} | {3,5,8,10,12,1,15,6,4,2} |
| 5 | {3,5,8,10,12,1,15,6,4,2} | {3,5,8,10,12,1} | {1,3,5,8,10,12,15,6,4,2} |
| 6 | {1,3,5,8,10,12,15,6,4,2} | {1,3,5,8,10,12,15,6} | {1,3,5,6,8,10,12,15,4,2} |
| 7 | {1,3,5,6,8,10,12,15,4,2} | {1,3,5,6,8,10,12,15,4} | {1,3,4,5,6,8,10,12,15,2} |
| 8 | {1,3,4,5,6,8,10,12,15,2} | {1,3,4,5,6,8,10,12,15,2} | {1,2,3,4,5,6,8,10,12,15} |

Seperti yang ditunjukkan pada ilustrasi di atas, kita mulai dengan elemen ke-2 karena kami mengasumsikan bahwa elemen pertama selalu diurutkan. Jadi kita mulai dengan membandingkan elemen kedua dengan yang pertama dan menukar posisi jika elemen kedua lebih kecil dari yang pertama.

Proses perbandingan dan pertukaran ini memposisikan dua elemen di tempat yang tepat. Selanjutnya, kami membandingkan elemen ketiga dengan elemen sebelumnya (pertama dan kedua) dan melakukan prosedur yang sama untuk menempatkan elemen ketiga di tempat yang tepat.

Dengan cara ini, untuk setiap operan, kami menempatkan satu elemen di tempatnya. Untuk lintasan pertama, kami menempatkan elemen kedua di tempatnya. Jadi secara umum, untuk menempatkan elemen N di tempat yang tepat, kita membutuhkan N-1 pass.

Visualisasi:

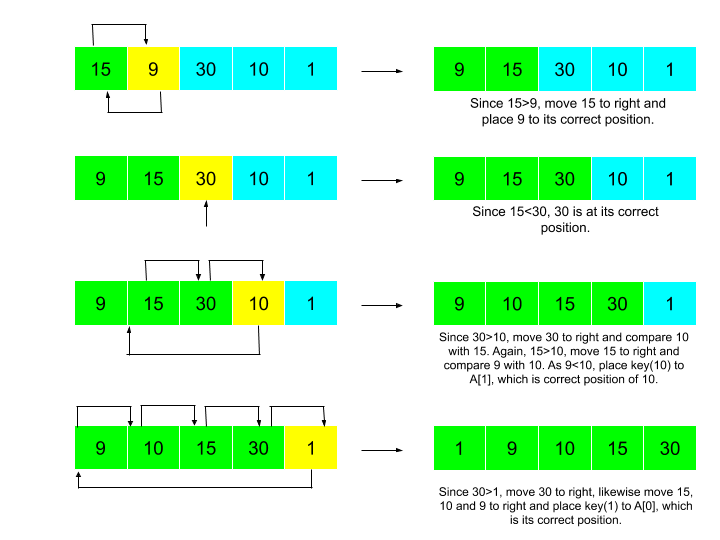
Video:

<insertionsort.mp4>.

Gif:

<InsertionSort.gif>; <InsertionSort2.gif>; <InsertionSort3.gif>.

Photo:



1. Aplikasi yang mengimplementasikan algoritma.

// C++ program for insertion sort

#include <bits/stdc++.h>

using namespace std;

/\* Function to sort an array using insertion sort\*/

void insertionSort(int arr[], int n)

{

int i, k, j;

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

{

k = arr[i];

j = i - 1;

/\*shifting elements of arr[0..i-1] towards right

if are greater than k \*/

while (j >= 0 && arr[j] > k)

{

arr[j + 1] = arr[j];

j = j - 1;

}

arr[j + 1] = k;

}

}

int main()

{

int n,i;

// size of array

cin>>n;

int arr[n];

for (i = 0; i < n; i++)

cin >> arr[i];

// input the array

insertionSort(arr, n);

// sort the array

for (i = 0; i < n; i++)

cout << arr[i] << " ";

cout << endl;

// print the array

return 0;

}

1. Tabel dan grafik perubahan running time untuk nilai inputan n berbeda-beda.

Berikut adalah hasil untuk Insertion Sort setelah 50 iterasi:

|  |  |  |  |
| --- | --- | --- | --- |
| n | Unsorted | Descending | Ascending |
| ... | ... | ... | ... |
| 32,768 | 87.86 ms | 175.80 ms | 0.042 ms |
| 65,536 | 350.43 ms | 697.59 ms | 0.084 ms |
| 131,072 | 1,398.92 ms | 2,840.00 ms | 0.168 ms |
| 262,144 | 5,706.82 ms | 11,517.36 ms | 0.351 ms |
| 524,288 | 23,009.68 ms | 46,309.37 ms | 0.710 ms |
| 1,048,576 | - | - | 1.419 ms |
| ... | ... | ... | ... |
| 536,870,912 | - | - | 693.310 ms |

Chart, line chart

Description automatically generated

Chart, line chart

Description automatically generated

Ref:

<https://iq.opengenus.org/insertion-sort-analysis/>;

<https://www.softwaretestinghelp.com/insertion-sort/>;

<https://www.hackerearth.com/practice/algorithms/sorting/insertion-sort/visualize/>;

<https://www.happycoders.eu/algorithms/insertion-sort/>;

<https://stackoverflow.com/questions/23331997/how-to-calculate-the-theoretical-running-time-of-insertion-sort-for-any-input-n>.