Insertion Sort

Implementation:

```
void insertion_sort(int arr[],int n){
  int i,j,key;
  for(i=1; i<n; i++){
    key = arr[i];
    j = i-1;
    while(j>=0 && arr[j] > key){
       arr[j+1] = arr[j];
       j--;
    }
    arr[j+1] = key;
  }
}
```

Analysis:

We start analyzing the Insertion Sort procedure with the time "cost" of each statement and the number of times each statement is executed.

pseudo code of insertion sort:

		Cost	Time
1	for *i* <- 2 to *n*	c1	n
2	do *key* <- *A[j]*	c2	n-1
3	*j* <- *i* – 1	c3	n-1
4	while *j* > 0 and *A[j]* > *key*	c4	$\sum_{i=2}^{n} t_i$
5	do *A[j + 1]* <- *A[j]*	c5	$\sum_{i=2}^{n}(t_i - 1)$
6	*j* <- *j* - 1	c6	$\sum_{i=2}^{n}(t_i - 1)$
7	*A[j + 1]* <- *key*	c7	n-1

For each i= 2,3,……..n, where n is the size of the array, we let t_i denote the number of times the while loop test in line 4 is executed for that value of i. When a for or while loop exits in the usual way (i.e., due to the test in the loop header), the test is executed one time more than the loop body.

The running time of the algorithm is the sum of running times for each statement executed; a statement that takes c_i steps to execute and executes n times will contribute $c_i * n$ times to the total running time.

Md. Foysal Ahmed
ID: 191-15-12486

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4 \sum_{i=2}^{n} t_i + c_5 \sum_{i=2}^{n} (t_i - 1) + c_6 \sum_{i=2}^{n} (t_i - 1) + c_7(n-1)$$

In insertion sort, the **Best case** occurs if the array is already sorted. For each value of i = 2,3,……n, the while loop will execute only once per iteration for comparison that means line 5 and 6 will not execute at all. When $i = 2$ the minimum comparison will be 1 and movement 0

$i = 3$ the minimum comparison will be 2 and movement 0

$i = 4$ the minimum comparision will be 3 and movement 0

.

.

.

$i = n$ the minimum comparison will be (n-1) and movement 0

Therefore,

$$\begin{aligned} T(n) &= c_1 n + c_2(n-1) + c_3(n-1) + c_4(n-1) + c_7(n-1) \\ &= c_1 n + c_2 n - c_2 + c_3 n - c_3 + c_4 n - c_4 + c_7 n - c_7 \\ &= (c_1 + c_2 + c_3 + c_4 + c_7)n - (c_2 + c_3 + c_4 + c_7) \end{aligned}$$

This equation can be written as,

$T(n)$ = $an + b$ (where a & b are some constant) **which is a linear function.**

If the array is in reverse sorted order—that is, in decreasing order the **worst case** occurs. We must compare each element $A[i]$ with each element in the entire subarray $A[1 \ldots \ldots \ldots i-1]$ and t_i = i for i =2,3,……n. which means,

For i = 2+3+……+n

$$= \frac{n(n+1)}{2} - 1$$

Therefore,

$$\sum_{i=2}^{n} i = \frac{n(n+1)}{2} - 1$$

And

For $j - 1$, the loop will execute for 2+3+……..+(n-1) times

$$\sum_{i=2}^{n} (i-1) = \frac{n(n-1)}{2}$$

Therefore, the running time of INSERTION-SORT is,

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4\left(\frac{n(n+1)}{2} - 1\right) + c_5\left(\frac{n(n-1)}{2}\right) + c_6\left(\frac{n(n-1)}{2}\right) + c_7(n-1)$$

Md. Foysal Ahmed
ID: 191-15-12486

$= c_1 n + c_2(n-1) + c_3(n-1) + \frac{c_4}{2} n(n+1) - c_4 + \frac{c_5}{2} n(n-1) - c_5 + \frac{c_6}{2} n(n-1) - c_6 + c_7(n-1)$

$= c_1 n + c_2(n-1) + c_3(n-1) + \frac{c_4}{2}(n^2+n) - c_4 + \frac{c_5}{2}(n^2-n) - c_5 + \frac{c_6}{2}(n^2-n) - c_6 + c_7(n-1)$

$= c_1 n + c_2 n - c_2 + c_3 n - c_3 + \frac{c_4}{2} n^2 + \frac{c_4}{2} n - c_4 + \frac{c_5}{2} n^2 - \frac{c_5}{2} n + \frac{c_6}{2} n^2 - \frac{c_6}{2} n + c_7 n - c_7$

$= (\frac{c_4}{2} + \frac{c_5}{2} + \frac{c_6}{2}) n^2 + \left(c_1 + c_2 + c_3 + \frac{c_4}{2} - \frac{c_5}{2} - \frac{c_6}{2} + c_7 \right) n - (c_2 + c_3 + c_4 + c_7)$

This equation can be written as,

$T(n)$ = $an^2 + bn - c$ (where a ,b & c are some constant) **which is quadratic function.**

Time Complexity:

Worst Case: The time function we get from the analysis is $T(n)$ = $an^2 + bn - c$

So the complexity of insertion sort in worst case is $O(n^2)$

Best Case: The time function we get from the analysis is $T(n)$ = $an + b$

So the complexity of insertion sort in best case is $O(n)$

Md. Foysal Ahmed
ID: 191-15-12486