**Pseudocode:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Id no:** | **Statement** | **Cost** | **Times** | **Reasons** |
|  | **for j <— 2 to length[A]** | C1 | n | The outer for loop will execute n times and the statement will execute n+1 time i.e., for j= 2, 3, 4…. n, n+1.  For n+1, the condition of the outer for loop will become false and will end the program. |
|  | **do key <— A[j]** | C2 | n-1 | Statement 2 is an assignment statement inside the for loop, it will run (n-1) times as it will not run after the outer loop condition will become false. |
|  | **// Insert A[j] into the sorted sequence AR -j —1]** | 0 | n-1 | This is a comment, the processor will ignore this statement **(cost=0)** and will be executed n-1 times as it is under the outer for loop |
|  | **i=j —1** | C4 | n-1 | Statement 4 is an assignment statement inside the for loop, it will run (n-1) times as it will not run after the outer loop condition will become false. |
|  | **while i > 0 and A[i]> key** | C5 |  | Statement 5 is an inner loop for comparisons, the variable i used in this loop is directly dependent on the value of outer loop variable j. (Statement 4) |
|  | **do A[i +1] <— A[i]** | C6 |  | Statement 6 is used inside inner loop for swapping the sorted array from the end if greater than key. As it is used under the inner while loop it will execute one less than statement 5 times. |
|  | **i =i-1** | C7 |  | Statement 7 is used inside inner loop to decrement the counter. As it is used under the inner while loop it will execute one less than statement 5 times. |
|  | **A[i +1] < — key** | C8 | n-1 | Statement 8 is an assignment statement inside the for loop, it will run (n-1) times as it will not run after the outer loop condition will become false. |

**Dry Run:**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Iteration no**  0 | **Key** | **4** | **3** | **2** | **10** | **12** | **1** | **5** | **6** | **No of Swaps** | **No of Comparisons** |
| 0.1 | 3 | 3 | 4 | 2 | 10 | 12 | 1 | 5 | 6 | 0 | 0 |
| 0.2 | 3 | 4 | 4 | 2 | 10 | 12 | 1 | 5 | 6 | 1 | 1 |
| 0.3 | 3 | 3 | 4 | 2 | 10 | 12 | 1 | 5 | 6 | 1 | 0 |
| **1** | **3** | **3** | **4** | **2** | **10** | **12** | **1** | **5** | **6** | **1** | **1** |
| 1.1 | 3 | 3 | 4 | 2 | 10 | 12 | 1 | 5 | 6 | 0 | 0 |
| 1.2 | 3 | 3 | 4 | 4 | 10 | 12 | 1 | 5 | 6 | 1 | 1 |
| 1.3 | 3 | 3 | 3 | 4 | 10 | 12 | 1 | 5 | 6 | 1 | 1 |
| 1.4 | 3 | 2 | 3 | 4 | 10 | 12 | 1 | 5 | 6 | 1 | 0 |
| **2** | **2** | **2** | **3** | **4** | **10** | **12** | **1** | **5** | **6** | **2** | **2** |
| 2.1 | 2 | 2 | 3 | 4 | 10 | 12 | 1 | 5 | 6 | 0 | 0 |
| **3** | **10** | **2** | **3** | **4** | **10** | **12** | **1** | **5** | **6** | **0** | **1** |
| 3.1 | 10 | 2 | 3 | 4 | 10 | 12 | 1 | 5 | 6 | 0 | 1 |
| **4** | **12** | **2** | **3** | **4** | **10** | **12** | **1** | **5** | **6** | **0** | **1** |
| 4.1 | 12 | 2 | 3 | 4 | 10 | 12 | 1 | 5 | 6 | 0 | 0 |
| 4.2 | 12 | 2 | 3 | 4 | 10 | 12 | 12 | 5 | 6 | 1 | 1 |
| 4.3 | 12 | 2 | 3 | 4 | 10 | 10 | 12 | 5 | 6 | 1 | 1 |
| 4.4 | 12 | 2 | 3 | 4 | 4 | 10 | 12 | 5 | 6 | 1 | 1 |
| 4.5 | 12 | 2 | 3 | 3 | 4 | 10 | 12 | 5 | 6 | 1 | 1 |
| 4.5 | 12 | 2 | 2 | 3 | 4 | 10 | 12 | 5 | 6 | 1 | 1 |
| 4.6 | 12 | 1 | 2 | 3 | 4 | 10 | 12 | 5 | 6 | 1 | 0 |
| **5** | **1** | **1** | **2** | **3** | **4** | **10** | **12** | **5** | **6** | **5** | **5** |
| 5.1 | 1 | 1 | 2 | 3 | 4 | 10 | 12 | 5 | 6 | 0 | 0 |
| 5.2 | 1 | 1 | 2 | 3 | 4 | 10 | 12 | 12 | 6 | 1 | 1 |
| 5.3 | 1 | 1 | 2 | 3 | 4 | 10 | 10 | 12 | 6 | 1 | 1 |
| 5.4 | 1 | 1 | 2 | 3 | 4 | 5 | 10 | 12 | 6 | 1 | 0 |
| **6** | **5** | **1** | **2** | **3** | **4** | **5** | **10** | **12** | **6** | **2** | **2** |
| 6.1 | 5 | 1 | 2 | 3 | 4 | 5 | 10 | 12 | 6 | 0 | 0 |
| 6.2 | 5 | 1 | 2 | 3 | 4 | 5 | 10 | 12 | 12 | 1 | 1 |
| 6.3 | 5 | 1 | 2 | 3 | 4 | 5 | 10 | 10 | 12 | 1 | 1 |
| 6.4 | 5 | 1 | 2 | 3 | 4 | 5 | 6 | 10 | 12 | 1 | 0 |
| **7** | **6** | **1** | **2** | **3** | **4** | **5** | **6** | **10** | **12** | **2** | **2** |

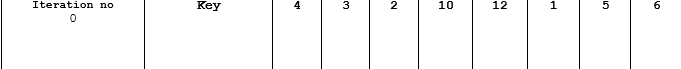
**Loop Invariant:**

The condition that the subarray A [0 to i-1] is always sorted.

**Justification:**

* **Initialization**

We consider the first element to be sorted at start, the shaded part is sorted.



* **Maintenance**

As it is true before an iteration, then it remains true before the next iteration

Before an iteration, the shaded area of array is sorted:



Before the next iteration starts, the shaded area of array is still sorted:



* **Termination**

At the end of program, the final array is also sorted.

Resulting array:

****

**Analysis:**

1. **Best Case:**

The best case for insertion sort is that the array is sorted.

T(n)= c1(n) + c2(n-1) + c4(n-1) + c5(n-1) + c8(n-1)

T(n)= (c1+c2+c4+c5+c8) \*n - (c2+c4+c5+c8)

T(n)=a\*n +b

The program will not execute for statement 6 and 7 as it will compare the values in the inner while loop and when the condition will become false (sorted array), it will come out of the while loop.

**Running time:**

**O (n)**

1. **Worst Case:**

The worst case of insertion sort is that your array is sorted is descending order and we have to arrange it in ascending order, so we have to perform n number of comparisons and swaps, i.e. for all values of j the inner loop statements will also be executed.

For j=2, 3, 4, …., n



T(n)= c1(n) + c2(n-1) + c4(n-1) + c5(n(n+1)/2-1) + c6(n(n-1)/2) + c7(n(n-1)/2) + c8(n-1)

T(n)= (c5/2+c6/2+c7/2) \*n2 +(c1+c2+c4+c5/2-c6/2-c7/2+c8) \*n

T(n)=an2 + b(n) +c

**Running time:**

O(n2)

**Selection Sort**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Id no:** | **Statement** | **Cost** | **Times** | **Reasons** |
|  | n <- length[A] | C1 | 1 | Statement 1 is an assignment statement where we assign n the length of array and it will execute 1 time only. |
|  | for j <— 1 to n - 1 | C2 | n | Statement 2 is the outer for loop, it will run n times because this line will execute for j=1, 2, 3, ..., n-1  For n, the condition of outer for loop will become false |
|  | do smallest <— j | C3 | n-1 | Statement 3 is an assignment statement as it is inside the outer for loop so it will execute one less than statement 1 times. |
|  | for i<-j+1 to n | C4 |  | Statement 4 is an assignment statement inside the for loop, it will run (n-j+1) times and as it depends on outer for loop.  It will run one time extra and the condition will become false and will come out of the inner loop. |
|  | do if *A[i] <*A[smallest] | C5 |  | Statement 5 is used inner loop for comparing values in the array to find the smallest element, thus it will execute one less than statement 4 times. |
|  | **then** smallest <- i |  |  | Statement 6 is used inside inner loop for exchanging the value of variable smallest of array only if it finds a smaller value than the current smallest through comparison (statement 5) |
|  | exchange A[j]<->A[smallest] | C6 | n-1 | Statement 7 is used after inner loop to swap the values and insert the smallest value present in array at its right position.  This statement is inside the outer for loop and will be executed one less than statement 1 times. |

**Dry Run:**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Iteration no**  0 | **Minimum**  **value** | **4** | **3** | **2** | **10** | **12** | **1** | **5** | **6** | **No of Swaps** | **No of Comparisons** |
| 0.1 | 4 | 4 | 3 | 2 | 10 | 12 | 1 | 5 | 6 | 0 | **0** |
| 0.2 | 3 | 4 | 3 | 2 | 10 | 12 | 1 | 5 | 6 | 0 | 1 |
| 0.3 | 2 | 4 | 3 | 2 | 10 | 12 | 1 | 5 | 6 | 0 | 1 |
| 0.4 | 2 | 4 | 3 | 2 | 10 | 12 | 1 | 5 | 6 | 0 | 1 |
| 0.5 | 2 | 4 | 3 | 2 | 10 | 12 | 1 | 5 | 6 | 0 | 1 |
| 0.6 | 1 | 4 | 3 | 2 | 10 | 12 | 1 | 5 | 6 | 0 | 1 |
| 0.7 | 1 | 4 | 3 | 2 | 10 | 12 | 1 | 5 | 6 | 0 | 1 |
| 0.8 | 1 | 4 | 3 | 2 | 10 | 12 | 1 | 5 | 6 | 0 | 1 |
| 0.9 | 1 | 1 | 3 | 2 | 10 | 12 | 4 | 5 | 6 | 1 | 0 |
| **1** | **1** | **1** | **3** | **2** | **10** | **12** | **4** | **5** | **6** | **1** | **7** |
| 1.1 | 3 | 1 | 3 | 2 | 10 | 12 | 4 | 5 | 6 | 0 | 0 |
| 1.2 | 2 | 1 | 3 | 2 | 10 | 12 | 4 | 5 | 6 | 0 | 1 |
| 1.3 | 2 | 1 | 3 | 2 | 10 | 12 | 4 | 5 | 6 | 0 | 1 |
| 1.4 | 2 | 1 | 3 | 2 | 10 | 12 | 4 | 5 | 6 | 0 | 1 |
| 1.5 | 2 | 1 | 3 | 2 | 10 | 12 | 4 | 5 | 6 | 0 | 1 |
| 1.6 | 2 | 1 | 3 | 2 | 10 | 12 | 4 | 5 | 6 | 0 | 1 |
| 1.7 | 2 | 1 | 3 | 2 | 10 | 12 | 4 | 5 | 6 | 0 | 1 |
| 1.8 | 2 | 1 | 2 | 3 | 10 | 12 | 4 | 5 | 6 | 1 | 0 |
| **2** | **2** | **1** | **2** | **3** | **10** | **12** | **4** | **5** | **6** | **1** | **6** |
| 2.1 | 2 | 1 | 2 | 3 | 10 | 12 | 4 | 5 | 6 | 0 | 0 |
| 2.2 | 3 | 1 | 2 | 3 | 10 | 12 | 4 | 5 | 6 | 0 | 1 |
| 2.3 | 3 | 1 | 2 | 3 | 10 | 12 | 4 | 5 | 6 | 0 | 1 |
| 2.4 | 3 | 1 | 2 | 3 | 10 | 12 | 4 | 5 | 6 | 0 | 1 |
| 2.5 | 3 | 1 | 2 | 3 | 10 | 12 | 4 | 5 | 6 | 0 | 1 |
| 2.6 | 3 | 1 | 2 | 3 | 10 | 12 | 4 | 5 | 6 | 0 | 1 |
| 2.7 | 3 | 1 | 2 | 3 | 10 | 12 | 4 | 5 | 6 | 0 | 0 |
| **3** | **3** | **1** | **2** | **3** | **10** | **12** | **4** | **5** | **6** | **0** | **5** |
| 3.1 | 10 | 1 | 2 | 3 | 10 | 12 | 4 | 5 | 6 | 0 | 0 |
| 3.2 | 10 | 1 | 2 | 3 | 10 | 12 | 4 | 5 | 6 | 0 | 1 |
| 3.3 | 4 | 1 | 2 | 3 | 10 | 12 | 4 | 5 | 6 | 0 | 1 |
| 3.4 | 4 | 1 | 2 | 3 | 10 | 12 | 4 | 5 | 6 | 0 | 1 |
| 3.5 | 4 | 1 | 2 | 3 | 10 | 12 | 4 | 5 | 6 | 0 | 1 |
| 3.6 | 4 | 1 | 2 | 3 | 4 | 12 | 10 | 5 | 6 | 1 | 0 |
| **4** | **4** | **1** | **2** | **3** | **4** | **12** | **10** | **5** | **6** | **1** | **4** |
| 4.1 | 12 | 1 | 2 | 3 | 4 | 12 | 10 | 5 | 6 | 0 | 0 |
| 4.2 | 10 | 1 | 2 | 3 | 4 | 12 | 10 | 5 | 6 | 0 | 1 |
| 4.3 | 5 | 1 | 2 | 3 | 4 | 12 | 10 | 5 | 6 | 0 | 1 |
| 4.4 | 5 | 1 | 2 | 3 | 4 | 12 | 10 | 5 | 6 | 0 | 1 |
| 4.5 | 5 | 1 | 2 | 3 | 4 | 12 | 10 | 5 | 6 | 1 | 0 |
| **5** | **5** | **1** | **2** | **3** | **4** | **5** | **10** | **12** | **6** | **1** | **3** |
| 5.1 | 10 | 1 | 2 | 3 | 4 | 5 | 10 | 12 | 6 | 0 | 0 |
| 5.2 | 10 | 1 | 2 | 3 | 4 | 5 | 10 | 12 | 6 | 0 | 1 |
| 5.3 | 6 | 1 | 2 | 3 | 4 | 5 | 10 | 12 | 6 | 0 | 1 |
| 5.4 | 6 | 1 | 2 | 3 | 4 | 5 | 10 | 12 | 6 | 1 | 0 |
| **6** | **6** | **1** | **2** | **3** | **4** | **5** | **6** | **12** | **10** | **1** | **2** |
| 6.1 | 12 | 1 | 2 | 3 | 4 | 5 | 6 | 12 | 10 | 0 | 0 |
| 6.2 | 10 | 1 | 2 | 3 | 4 | 5 | 6 | 12 | 10 | 0 | 1 |
| 6.3 | 10 | 1 | 2 | 3 | 4 | 5 | 6 | 12 | 10 | 1 | 0 |
| **7** | **-** | **1** | **2** | **3** | **4** | **5** | **6** | **10** | **12** | **1** | **1** |

**Analysis:**

T(n)= c1\*1 + c2\*n +c3(n-1)+ c4( + c5( + c6(n-1)

**Best Case:**

Best case is that the array is already sorted, still (n-1), (n-2), (n-3) ...0 times time comparison will be done and perform no swaps (O (1)) as the array is sorted. For comparisons we get:

n(n-1)/2

Running Time:

**O(n2)**

**Worst Case:**

Worst case is that the array is sorted in descending order, (n-1), (n-2), (n-3) ...0 times comparison will be done and perform O(n) swaps. For comparisons we get:

n(n-1)/2

Running Time:

**O(n2)**

**Average Case:**

In average case, (n-1), (n-2), (n-3) ...0 times comparison will be done. Thus we get:

n(n-1)/2

Running Time:

**O(n2)**