| **Experiment No. 2** |
| --- |
| **To implement Insertion Sort** |
| Date of Performance: 15/2/24 |
| Date of Submission: 22/2/24 |

## Experiment No. 2

**Title:** Insertion Sort

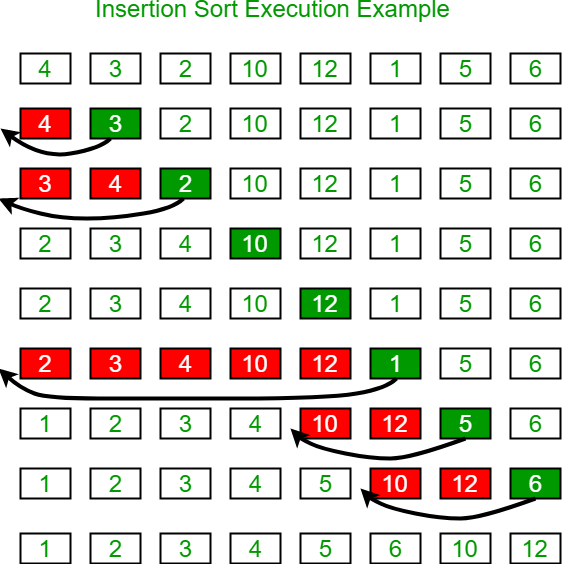
**Aim:** To study, implement and Analyze Insertion Sort Algorithm

**Objective:** To introduce the methods of designing and analyzing algorithms

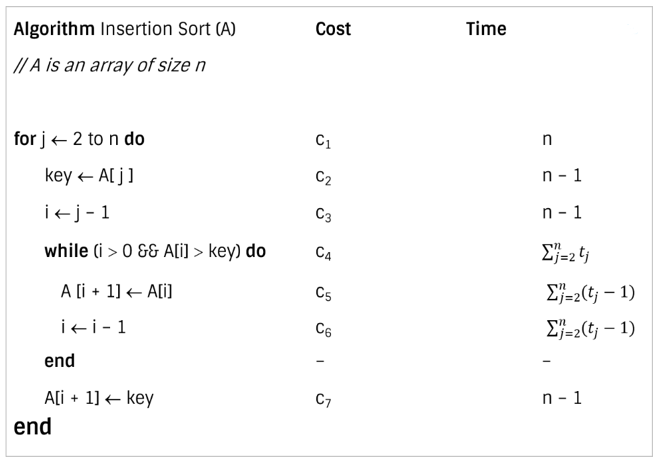
#### Theory:

Insertion sort is a simple sorting algorithm that works similar to the way you sort the playing cards in your hands. The array is virtually split into a sorted and an unsorted part. Values from the unsorted part are picked and placed at the correct position in the sorted part.

#### Example:



**Algorithm and Complexity:**



**Best case analysis:**

* Let size of the input array is n. Total time taken by algorithm is the summation of time taken by each of its instruction.

A black and white math equation

Description automatically generated with medium confidence

* The best case offers the lower bound of the algorithm’s running time.
* When data is already sorted, the best scenario for insertion sort happens.
* In this case, the condition in the while loop will never be satisfied, resulting in tj = 1.

A screenshot of a math problem

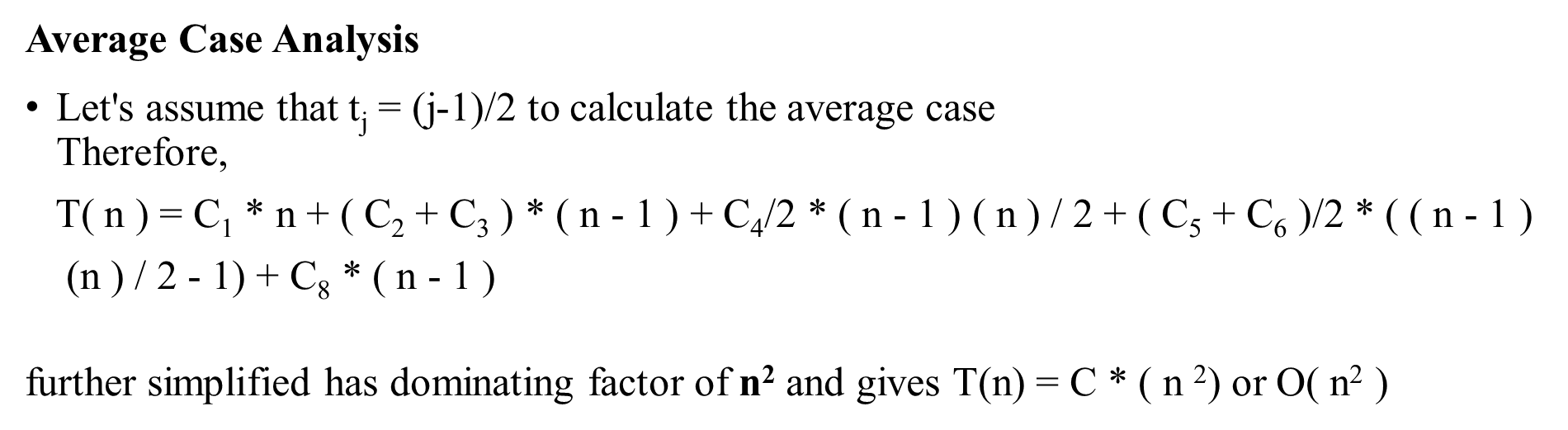
Description automatically generated

**Worst case analysis:**

* The worst-case running time gives an upper bound of running time for any input.
* The running time of algorithm cannot get worse than its worst-case running time.
* Worst case for insertion sort occurs when data is sorted in reverse order.
* So we must have to compare A[j] with each element of sorted array A[1 … j – 1]. So, tj = j

A math equations on a white background

Description automatically generated



**Code:**

**#include<stdio.h>**

**void InsertionSort(int a[], int n){**

**int key ,i;**

**for(int j =1; j<=n; j++){**

**key = a[j];**

**i = j-1;**

**while(i>0 && a[i]>key){**

**a[i+1] = a[i];**

**i= i-1;**

**}**

**a[i+1]= key;**

**}**

**printf("After Sorting the Elements are :");**

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

**printf("%d ",a[i]);**

**}**

**}**

**int main(){**

**int a[10];**

**int n ;**

**printf("Enter the n Elements:");**

**scanf("%d",&n);**

**printf("Enter %d elements:\n",n);**

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

**scanf("%d",&a[i]);**

**}**

**InsertionSort(a,n);**

**return 0;**

**}**

**Output:**

**Enter the n Elements:10**

**Enter 10 elements:**

**5 7 9 2 3 1 7 8 9 4**

**After Sorting the Elements are :1 2 3 4 5 7 7 8 9 9**

**Conclusion:**

**In concluding our experiment on Insertion Sort, we've observed its performance across various dataset sizes. Insertion Sort, known for its simplicity, exhibits linear time complexity of O(n) in best-case scenarios, making it highly efficient for nearly sorted arrays or small datasets.**

**However, its average and worst-case time complexities of O(n^2) render it less efficient for larger datasets, where more sophisticated algorithms like Merge Sort or Quick Sort would be preferable. Despite its limitations in scalability, Insertion Sort remains a valuable tool in situations where simplicity and efficiency for small datasets are prioritized.**