# L P S - 1

{ Design and Analysis of Algorithms }

*Rachit Bhalla*
*21BAI1869*

1. Inputs required from the theory:
   (a) Pseudocode of Insertion sort
   (b) T(n) of the insertion-sort
   (c) Best-case, worst-case and average-case time-complexity of the algorithm

## Problem Statement:

To use insertion sort on random numbers generated by a python script and using concept of vectors and iterations for insertion sort.
In other words, sorting a vector array using insertion sort.

## Logic:

In insertion sort, we compare each element to it's next element while storing the next element in a temporary variable and then comparing it to the previous ones and thus sorting the given array. Each element is moved from unsorted position to sorted position in every iteration till we get the sorted array.

## Algorithm:

1. Import the necessary libraries
2. for i=1 to array .length
{key = elements[i]
j = i-1

while j>=0 && elements[j]>key //perform the following till while block is true.
{elements[j+1] = elements[j]
j = j – 1}
elements[j+1] = key }
3. print the sorted array elements to verify our results.

## (A)Pseudocode:

```
#include<iostream>
#include<vector>
#include<ctime>
using namespace std;

int main(){
    vector<int> elements;
    int key,i,j,n,ele;
    cin>>n;
    for(i=0;i<n;i++){
        cin>>ele;
        elements.push_back(ele);
    }
    clock_t Tstart=clock();
    for(i=1;i<n;i++){
        key=elements[i];
        j=i-1;
        while((j>=0)&&(elements[j]>key)){
            elements[j+1]=elements[j];
            j=j-1;
        }
        elements[j+1]=key;
    }
    double time1=(double)(clock()-Tstart)/CLOCKS_PER_SEC;
    cout<<time1<<endl;
    for(i=0;i<n;i++){
        cout<<elements.at(i)<<" ";
    }
}
```

## (B)Time Complexity: $O(N^2)$

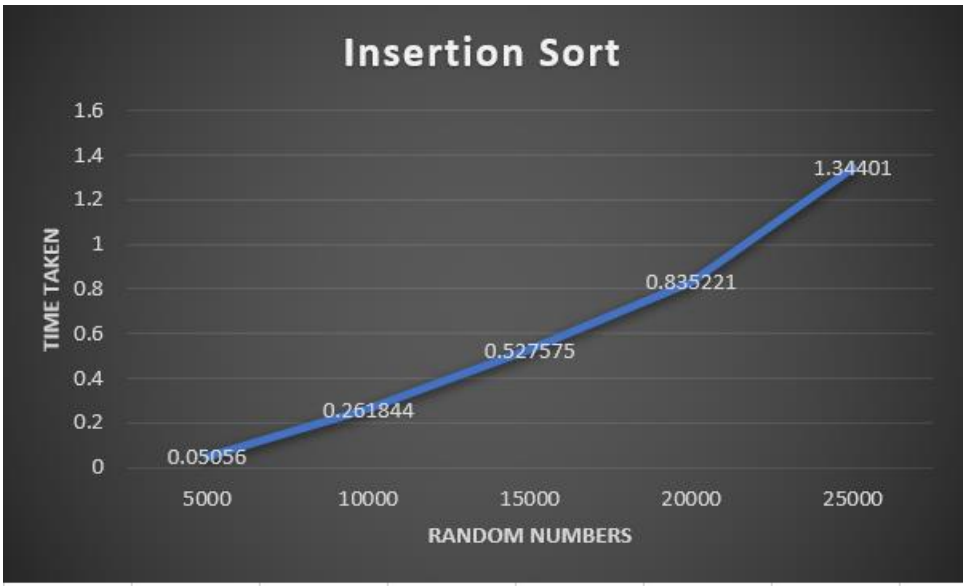
**~>**As there is a ‘while’ loop incorporated inside the ‘for’ loop such that the while loop will be run N times and as it is inside a for loop which is also running N times the combined time comes out to be N*N times.
**~>**in short two loops are being used one inside the other hence we get a time complexity of $N^2$

# Proof of Correctness:

**Initialization:**
Before the first iteration even happens once, first elements of the array is always considered to be at the correct position, i.e. that if only 1 element is present there’s no need to sort it, so it is already sorted. The condition is met.
**Maintenance:** After each iteration of the outer loop, the subarray consisting of first j-1 elements are already sorted and the inner while loop inserts the ith element into it correct position in the sorted subarray.
**Termination:** Both the loops will be terminated when , for the last iteration of inner loop, the while condition turns out to be false i.e. when element[j]<key, and the outer loop terminates when it finishes all it’s iterations.
Thus, the insertion sort is terminated when all it’s elements are put into correct positions,i.e. array is sorted now.

# Graph:

| Random number | Time Taken |
|---|---|
| 5000 | 0.05056 |
| 10000 | 0.261844 |
| 15000 | 0.527575 |
| 20000 | 0.835221 |
| 25000 | 1.34401 |
| 30000 | 1.89904 |




## (C)

**Best-case time complexity: [ O(N) ]**

If the array is already sorted ,i.e. the input vector array is already in ascending order, then the inner while loop would never execute. The time complexity would be O(N) as only the outer for loop gets executed for N times.

**Average-case time complexity: [ $O(N^2)$]**

It is $O(N^2)$, this is because, on average every element in the unsorted array will need to compared to at least half of the elements in the sorted subarray side, which results in a lot of computations i.e.both the loops will compute for some times, let's say outer loop runs for N times and the inner while loop runs for about half comparisons,i.e. N/2. Thus final time comes out to be $N^2/2$ which is nothing but $O(N^2)$.

**Worst-case time complexity: [$O(N^2)$]**

For example, when the elements are present in the descending order, so each element is at the wrong position and has to be compared with every other element so as to be at the right position. This results in a lot of comparisons at every level of loop i.e. both inner while and outer while loop will have to run for N times and the time becomes N*N and hence the time complexity is $O(N^2)$.

## 2. Design related problems:

(a) Translate the Insertion-sort algorithm discussed in the class into a program which takes n numbers (real or integers). Inputs for the program is n and the n numbers

# Problem Statement:

To use insertion sort on random numbers generated by a python script and using concept of vectors and iterations for insertion sort.
In other words, sorting a vector array using insertion sort.

# Logic:

In insertion sort, we compare each element to it's next element while storing the next element in a temporary variable and then comparing it to the previous ones and thus sorting the given array. Each element is moved from unsorted position to sorted position in every iteration till we get the sorted array.

# Algorithm:

1. Import the necessary libraries
2. for i=1 to array .length
{key = elements[i]
j = i-1
while j>=0 && elements[j]>key //perform the following till while block is true.
{elements[j+1] = elements[j]
j = j – 1}
elements[j+1] = key }
3. print the sorted array elements to verify our results.

# Pseudocode:

```
#include<iostream>
#include<vector>
#include<ctime>
using namespace std;

int main(){
    vector<int> elements;
    int key,i,j,n,ele;
    cin>>n;
    for(i=0;i<n;i++){
        cin>>ele;
        elements.push_back(ele);
    }
```

```
    clock_t Tstart=clock();
    for(i=1;i<n;i++){
        key=elements[i];
        j=i-1;
        while((j>=0)&&(elements[j]>key)){
            elements[j+1]=elements[j];
            j=j-1;
        }
        elements[j+1]=key;
    }
    double time1=(double)(clock()-Tstart)/CLOCKS_PER_SEC;
    cout<<time1<<endl;
    for(i=0;i<n;i++){
        cout<<elements.at(i)<<" ";
    }
}
```

(b) Given a sequence of n numbers (real or integers) and a number k (k is one among the n numbers ), write an algorithm and the corresponding code to compute the position of k if the given n numbers are arranged in an increasing order, using insertion sort. If the 2, −1, 3, 0, 7 and 3 are the input , your program should output 4 since 3 will be in the fourth position (starting from 1), in the sorted (increasing ) order. You are expected to code the problem two different ways, say, c1 , c2 using two different approaches. Decide whether c1 is efficient or c2 is efficient based on the running time T(n) of the respective codes.

## Problem Statement:

To search for an element In the sorted array using given algorithms and printing the position of that element

## Logic:

Algorithm 1 uses insertion sort and linear search and algorithm 2 uses binary search technique.

## Algorithm c1:

1.use insertion sort to sort the array in ascending order.
2. then we apply linear search on the sorted array.

```
    For(i<0;i in n){
    If element[i]== required element;
```

Return i+1; //instead of returning the index we return the position if we say index 0 as 1

# Algorithm c2:

Initialize two variables, left and right, which point to the first and last elements of the sequence respectively.

While left <= right:

mid= (low + high) / 2.

If k = elements[mid]

return mid + 1.

If k < elements[mid]

right = mid - 1.

else

left = mid + 1.

If k is not found, return -1.

# Pseudocode c1:

```
#include <iostream>
#include <vector>

using namespace std;

void insertionSort(vector<int> elements, int n, int ele) {
    for (int i = 1; i < n; i++) {
        int key = elements[i];
        int j = i - 1;
        while (j >= 0 && elements[j] > key) {
            elements[j+1] = elements[j];
            j--;
```

```
        }
        elements[j+1] = key;
    }
}

int Linear_search(vector<int>& elements,int n,int k){
        for (int i = 0; i < n; i++) {
        if (elements[i] == k) {
            return i + 1;
        }
    }
    return -1;
}
int main() {
    int n, req_ele;
    cout<<"enter the number of elements: ";
    cin >> n;
    vector<int> elements(n);
    for (int i = 0; i < n; i++) {
        cin >> elements[i];
    }
    cout<<"enter the element: ";
    cin >> req_ele;
    insertionSort(elements, n , req_ele);
    int position= Linear_search(elements,n,req_ele);
    cout << position << endl;
    return 0;
}
```

# Pseudocode c2:

```
#include <iostream>
#include <vector>

using namespace std;

int binary_st(vector<int> element, int left, int right, int req_ele) {
    while (left <= right) {
        int mid = (left + right) / 2;
        if (element[mid] == req_ele) {
```

```
            return mid + 1;
        }
        else if (element[mid] > req_ele) {
            right = mid - 1;
        }
        else {
            left = mid + 1;
        }
    }
    return -1;
}

int main() {
    int k, req_ele;
    cout<<"enter the number of elements: ";
    cin >> k;
    vector<int> element(k);
    for (int i = 0; i < k; i++) {
        cin >> element[i];
    }
    cout<<"enter the element: ";
    cin >> req_ele;
    int pos = binary_st(element, 0, k-1,req_ele);
    cout << pos << endl;
    return 0;
}
```

(c) All the alphabets(lower case) of English language a, b, c, ..., y, z are assigned values 1, 2, 3, ..., 25, 26. Given a sequence of n symbols from english alphabet (only lower case), write an 2 insertion-sort based algorithm to arrange the given n symbols, in an increasing order of their values. You are expected to code the problem two different ways, say, c1 , c2 using two different approaches. Decide whether c1 is efficient or c2 is efficient based on the running time T(n) of the respective codes.

# Problem Statement:

To sort the given alphabetical array using insertion sort.using 2 different approaches.

**Logic:** we can solve this problem using insertion sort. By giving each lowercase letter a numeric value and then sorting the supplied

sequence in increasing order using those values. we take two approaches i.e. we approach the problems using vectors and arrays.

# Algorithm c1:

1. Get the input as vectors.
2. Create function charValue which will tell us the integer value of alphabets.

```
int charValue(char p1){
    int s = p1 - 'a';
    return s + 1;
}
```

3. Use insertion sort to sort the given alphabetical value that we obtained using charvalue fxn.

```
For (int i=1;i<=n;i++){
Key=arr[i];
j=i-1
while(j>=0 and arr[j]>key){
        arr[j+1]=arr[j];
        j=j-1;
}
Arr[j+1]=key
}
```

# Algorithm c2:

1. Get the input as array
2. Create function charValue which will tell us the integer value of alphabets.

```
int charValue(char p1){
    int s = p1 - 'a';
    return s + 1;
}
```

3. Use insertion sort to sort the given alphabetical value that we obtained using charvalue fxn.

```
For (int i=1;i<=n;i++){
Key=arr[i];
j=i-1
while(j>=0 and arr[j]>key){
        arr[j+1]=arr[j];
```

```
            j=j-1;
        }
    Arr[j+1]=key
    }
```

# Pseudocode c1:

```
#include <iostream>
#include <algorithm>
#include <vector>
#include <ctime>

using namespace std;

int charValue(char p1){
    int s = p1 - 'a';
    return s + 1;
}

void insertionSort(vector<int>& arr)
{
    int n = arr.size();
    for (int i = 1; i < n; i++) {
        int key = arr[i];
        int j = i - 1;

        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}

int main(){

    clock_t start, end;
    int pp = 'a';
```

```
    int n;
    cout << "Enter the number of characters: ";
    cin >> n;

    vector<int> arr(n);
    cout << "Enter the characters:\n";
    for(int i = 0; i < n; i++){
        char c;
        cin >> c;
        int s = charValue(c);
        arr[i] = s;
    }

    start = clock();
    insertionSort(arr);

    for(int i = 0; i < n; i++){
        int a = arr[i] + pp - 1;
        char s1 = a;
        cout << s1 << "\n";
    }

    end = clock();
    double time_taken = double(end - start) /
double(CLOCKS_PER_SEC);
    cout << "Time taken by program is: " << time_taken;
    cout << " sec " << endl;

    return 0;
}
```

# Pseudocode c2:

```
#include <iostream>
#include <algorithm>
#include <ctime>

using namespace std;
```

```
int charValue(char p1) {
    int s = p1 - 'a';
    return s + 1;
}

void insertionSort(int arr[], int n) {
    int i, key, j;
    for (i = 1; i < n; i++) {
        key = arr[i];
        j = i - 1;

        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}

int main() {
    clock_t start, end;
    int pp = 'a';

    int n;
    cout << "Enter the number of characters: ";
    cin >> n;

    int* arr = new int[n];
    cout << "Enter the characters:\n";
    for (int i = 0; i < n; i++) {
        char c;
        cin >> c;
        int s = charValue(c);
        arr[i] = s;
    }

    start = clock();
    insertionSort(arr, n);
```

```
    for (int i = 0; i < n; i++) {
        int a = arr[i] + pp - 1;
        char s1 = a;
        cout << s1 << "\n";
    }

    end = clock();
    double time_taken = double(end - start) /
double(CLOCKS_PER_SEC);
    cout << "Time taken by program is: " << time_taken;
    cout << " sec " << endl;

    return 0;
}
```

(d) Given a sequence of n numbers (real or integers), write an algorithm and the corresponding code to arrange the given n numbers are arranged in such a way that all the negative numbers (if any) are arranged in a descending order and all the positive numbers are arranged in an increasing order with zero (if it is in the input) appearing between the smallest negative number and the smallest positive number. If 7, 3, 2, 4 the output should be 2, 3, 4, 7. If −7, −3, 2, 4 the output should be −3, −7, 2, 4 should be the output. If 7, 3, −1, 0, 2, 4 the output should be −1, 0, 3, 4, 7.

# Problem Statement:

Sort and arrange negative numbers in descending order and positive numbers including 0 in ascending order.

# Logic:

If elements of array are negative sort them in an inverted manner else If positive including zero use general insertion sort.

# Algorithm:

1. Read the inputs as vectors.
2. for(i=0 to n) {  // case where element at j is -ve

```
            temp = nums[i];
            int j = i - 1;
            while(j >= 0 && nums[j] < 0 && nums[j] < temp) {
                nums[j+1] = nums[j];
                j--;
            }
            nums[j+1] = temp;
        }
```

```
3.  for(i=0 to n) {       // case where element at j is either +ve or 0
        temp = nums[i];
        int j = i - 1;
        while(j >= 0 && nums[j] >= 0 && nums[j] > temp) {
            nums[j+1] = nums[j];
            j--;
        }
        nums[j+1] = temp;
    }
```

# Pseudocode:

```
#include <iostream>
#include <vector>
using namespace std;

int main() {
    vector<int> nums;
    int n, temp;

    // input n numbers
    cout << "Enter the number of elements: ";
    cin >> n;

    cout << "Enter the numbers: ";
    for(int i = 0; i < n; i++) {
        cin >> temp;
        nums.push_back(temp);
    }

    // insertion sort for negative numbers in descending order
    for(int i = 1; i < n; i++) {
        temp = nums[i];
        int j = i - 1;
        while(j >= 0 && nums[j] < 0 && nums[j] < temp) {
            nums[j+1] = nums[j];
            j--;
        }
        nums[j+1] = temp;
```

```
    }

    // insertion sort for positive numbers in ascending order
    for(int i = 1; i < n; i++) {
        temp = nums[i];
        int j = i - 1;
        while(j >= 0 && nums[j] >= 0 && nums[j] > temp) {
            nums[j+1] = nums[j];
            j--;
        }
        nums[j+1] = temp;
    }

    // output the sorted numbers
    cout << "Sorted numbers: ";
    for(int i = 0; i < n; i++) {
        cout << nums[i] << " ";
    }
    cout << endl;

    return 0;
}
```

(e) Given n points P1, P2, ..., Pn with the coordinates (x1, y1),(x2, y2), ...,(xn, yn) respectively, write an insertion-sort based algorithm and the corresponding code to arrange the points in an increasing order of the distance of the point from the origin (with (0, 0) as the coordinates. Distance between any two points (a1, b1) and (a2, b2) is p (a1 − a2) 2 + (b1 − b2) 2 . Input for the code is : n and the coordinates of the n points entered with x−coordinate first and then the y−coordinate.

# Problem Statement:

Given n points arrange the points in an increasing order of the distance of the point from the origin

# Logic:

Applying the distance formula, comparing the distance between two points, and storing the results(the coordinates) in ascending order is similar to how we perform an insertion sort.

# Algorithm:

1. read the points and make a pair of x, y

2. for j = 1 to n
pains<int, int> key = points[j];
i = j - 1;
while (i >= 0 && sqrt(pow(points[i].first, 2) + pow(points[i].second, 2)) > sqrt(pow(key.first, 2) + pow(key.second, 2))
points[i+1] = points[i];
i--;
points[i+1] = key;
3. print the points in ascending order.

# Pseudocode:

```
#include <iostream>
#include <vector>
#include <utility>
#include <cmath>
using namespace std;

int main() {
    int n;
    cout << "Enter number of points: ";
    cin >> n;

    vector<pair<int, int>> points(n);

    for (int i = 0; i < n; i++) {
        int x, y;
        cout << "x: ";
        cin >> x;
        cout << "y: ";
        cin >> y;
        points[i] = pair<int, int>(x, y);
    }

    for (int j = 1; j < n; j++) {
        pair<int, int> key = points[j];
        int i = j - 1;
        while (i >= 0 && sqrt(pow(points[i].first, 2) + pow(points[i].second, 2)) > sqrt(pow(key.first, 2) + pow(key.second, 2))) {
            points[i + 1] = points[i];
```

```
            i--;
        }
        points[i + 1] = key;
    }

    for (int i = 0; i < n; i++) {
        cout << "(" << points[i].first << ", " << points[i].second << ")" <<
endl;
    }
return 0;
}
```

## 3. Analysis related problems

(a) We know that the running time of insertion-sort algorithm T(n) is

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^{n} t_j + c_6 \sum_{j=2}^{n} (t_j - 1) + c_7 \sum_{j=2}^{n} (t_j - 1) + c_8(n-1) .$$

where n is the size of the problem (number of inputs given), ci is the time taken to execute the i-th line in the algorithm and tj is the number of times the 'while-loop' is executed for the given input. Take the value of ci 's as 1. Execute the insertion sort algorithm (through the respective code) and compute 3 T(n) for different input size n. Based on the the values of n and T(n), draw the graph with n as the x-axis and T(n) as the y-axis. Based on the graph, conclude whether T(n) is a linear or a parabola or a combination of line and a parabola.

ANS:-

FOR different n values:-

N=2; t(n)=9

N=3; t(n)=20

N=4; t(n)=34

N=5; t(n)=51

N=8; t(n)=120

**Graph:-**

| T(n) | N |
|---|---|
| 9 | 2 |
| 20 | 3 |
| 34 | 4 |
| 51 | 5 |
| 120 | 8 |


(b) Execute the insertion-sort algorithm many time with different inputs and Justify the statement : Best-case running time of T(n) is a linear function of n.

# Time Complexity: [O(N)]

If the array is already sorted ,i.e. the input vector array is already in ascending order, then the inner while loop would never execute. The time complexity would be O(N) as only the outer for loop gets executed for N times.

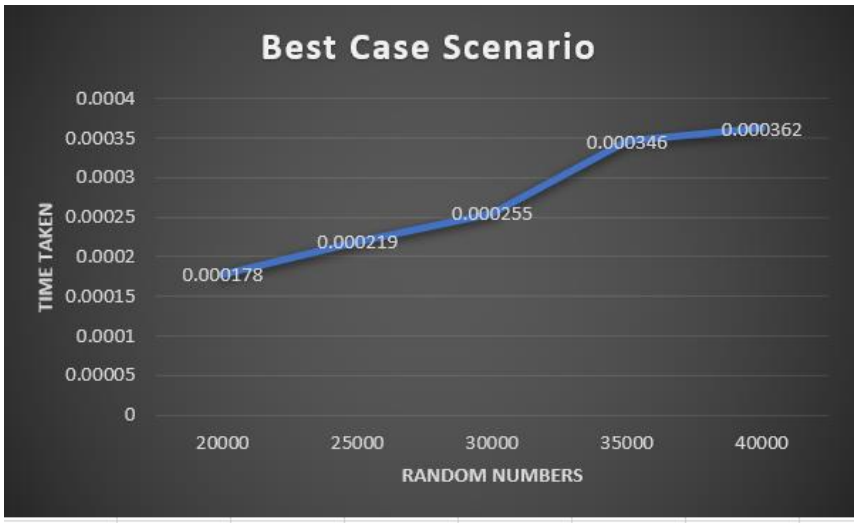
# Proof of Correctness:

**Initialization:**
Before the first iteration even happens once, first elements of the array is always considered to be at the correct position, i.e. that if only 1 element is present there's no need to sort it, so it is already sorted. The condition is met.
**Maintenance:** After each iteration of the outer loop, the subarray consisting of first j-1 elements are already sorted and the inner while loop inserts the ith element into it correct position in the sorted subarray.
**Termination:** Both the loops will be terminated when , for the last iteration of inner loop, the while condition turns out to be false i.e. when

element[j]<key, and the outer loop terminates when it finishes all it's iterations.
Thus, the insertion sort is terminated when all it's elements are put into correct positions,i.e. array is sorted now.

# Graph:

| Random number | Time Taken |
|---|---|
| 20000 | 0.000178 |
| 25000 | 0.000219 |
| 30000 | 0.000255 |
| 35000 | 0.000346 |
| 40000 | 0.000362 |
| 45000 | 0.000527 |




(c) Execute the insertion-sort algorithm many time with different inputs and Justify the statement : Worst-case running time of T(n) is a quadratic function of n.

# Time Complexity:[O($N^2$)]

For example, when the elements are present in the descending order, so each element is at the wrong position and has to be compared with every other element so as to be at the right position. This results in a lot of comparisons at every level of loop i.e. both inner while and outer while loop will have to run for N times and the time becomes N*N and hence the time complexity is O($N^2$).

# Proof of Correctness:

**Initialization:**

Before the first iteration even happens once, first elements of the array is always considered to be at the correct position, i.e. that if only 1 element is present there's no need to sort it, so it is already sorted. The condition is met.

**Maintenance:**

After each iteration of the outer loop, the subarray consisting of first j-1 elements are already sorted and the inner while loop inserts the ith element into it correct position in the sorted subarray.

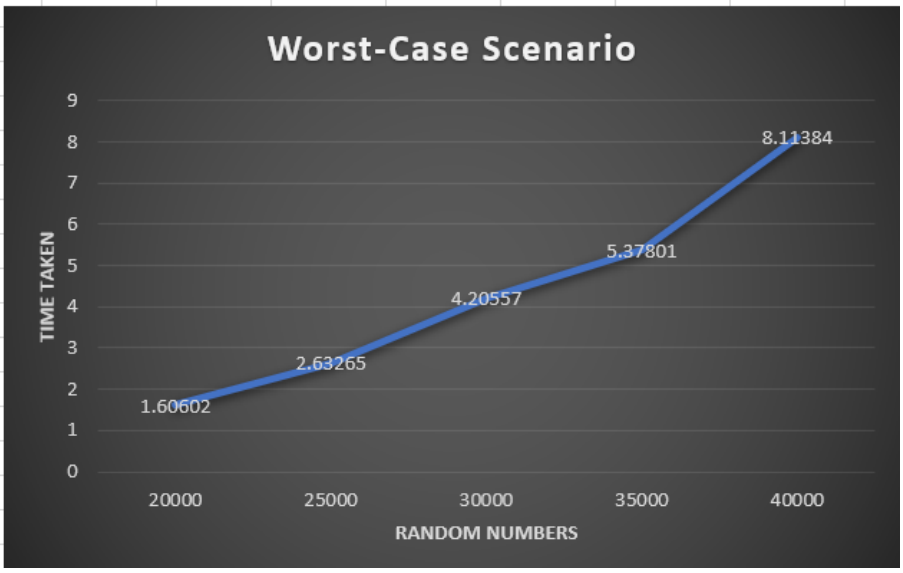
**Termination:**

Both the loops will be terminated when , for the last iteration of inner loop, the while condition turns out to be false i.e. when element[j]<key, and the outer loop terminates when it finishes all it's iterations.

Thus, the insertion sort is terminated when all it's elements are put into correct positions,i.e. array is sorted now.

# Graph:

| Random number | Time Taken |
|---|---|
| 20000 | 1.60602 |
| 25000 | 2.63265 |
| 30000 | 4.20557 |
| 35000 | 5.37801 |
| 40000 | 8.11384 |
| 45000 | 10.3292 |




(d) Execute the insertion-sort algorithm many time with different inputs and Justify the statement : Average -case running time of T(n) is a quadratic function of n.

# Time Complexity: : [ $O(N^2)$]

It is $O(N^2)$, this is because, on average every element in the unsorted array will need to compared to at least half of the elements in the sorted subarray side, which results in a lot of computations i.e.both the loops will compute for some times, let's say outer loop runs for N times

and the inner while loop runs for about half comparisons,i.e. N/2. Thus final time comes out to be $N^2/2$ which is nothing but $O(N^2)$.

# Proof of Correctness:

**Initialization:**
Before the first iteration even happens once, first elements of the array is always considered to be at the correct position, i.e. that if only 1 element is present there's no need to sort it, so it is already sorted. The condition is met.

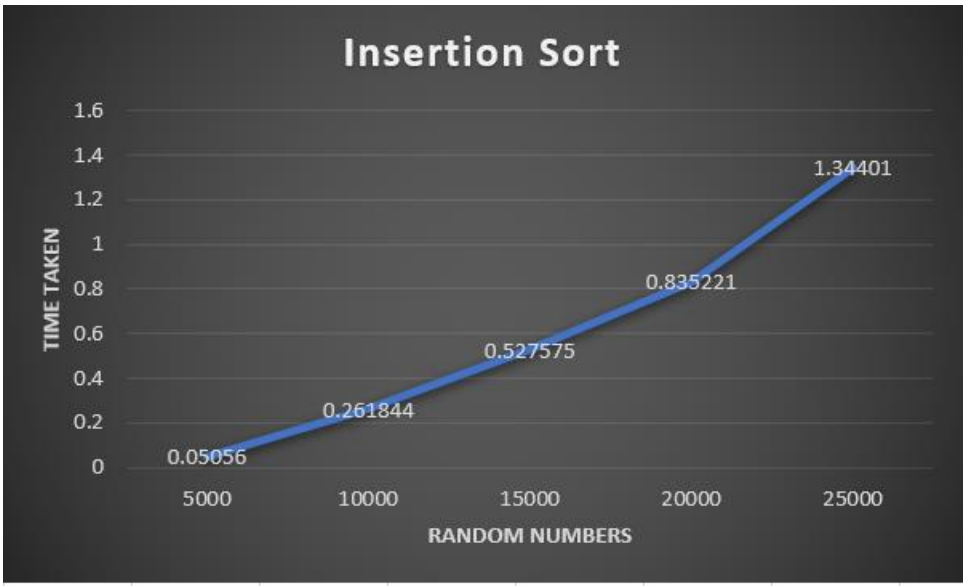
**Maintenance:**
After each iteration of the outer loop, the subarray consisting of first j-1 elements are already sorted and the inner while loop inserts the ith element into it correct position in the sorted subarray.

**Termination:** Both the loops will be terminated when , for the last iteration of inner loop, the while condition turns out to be false i.e. when element[j]<key, and the outer loop terminates when it finishes all it's iterations.

Thus, the insertion sort is terminated when all it's elements are put into correct positions,i.e. array is sorted now.

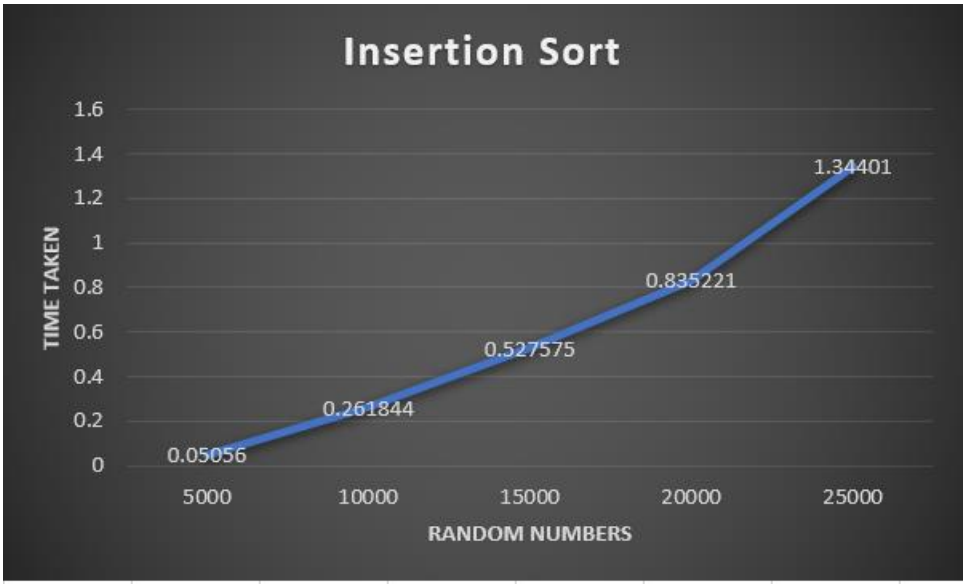
# Graph:

| Random number | Time Taken |
|---|---|
| 5000 | 0.05056 |
| 10000 | 0.261844 |
| 15000 | 0.527575 |
| 20000 | 0.835221 |
| 25000 | 1.34401 |
| 30000 | 1.89904 |




(e) compute the running time of P, t(P) in seconds for the insertion sort algorithm for different inputs and draw the graph n − V s − t(p).

| Random number | Time Taken |
|---|---|
| 5000 | 0.05056 |
| 10000 | 0.261844 |
| 15000 | 0.527575 |
| 20000 | 0.835221 |
| 25000 | 1.34401 |
| 30000 | 1.89904 |




(f) Compute the t(P) for all the codes requested in Q.No 2.

Ans:-

## (a)Time Complexity: $O(N^2)$

**~>**As there is a ‘while’ loop incorporated inside the ‘for’ loop such that the while loop will be run N times and as it is inside a for loop which is also running N times the combined time comes out to be N*N times.
**~>**in short two loops are being used one inside the other hence we get a time complexity of $N^2$

## (b)Time Complexity:

**~>**$O(N^2)$ for insertion sort, O(n) for linear search as it O(log n) for binary search as it uses divide and conquer strategy to search through the array.

## (c)Time Complexity: $O(N^2)$

**~>** $O(N^2)$ for insertion sort , we just take 2 different approaches like vectors and arrays.

## (d)Time Complexity: $O(N^2)$

**~>** $O(N^2)$ for insertion sort we just use insertion sort twice , once for ascending and once for descending sorting of elements based on positive or negative values.

## (e)Time Complexity: $O(N^2)$

**~>** $O(N^2)$ for insertion sort