Experiment No. : 1

Title: Basic Sorting algorithm and its analysis

Batch: A2 **Roll No.: 16010421063** **Experiment No.: 1**

Aim: To implement and analyse time complexity of insertion sort & Heap sort.

Explanation and Working of insertion sort & Heap sort:

Insertion Sort:

Working Principle

- Compare the element with its adjacent element.
- If at every comparison, we could find a position in sorted array where the element can be inserted, then create space by shifting the elements to right and insert the element at the appropriate position.
- Repeat the above steps until you place the last element of unsorted array to its correct position

Heap Sort:

In heap sort, basically, there are two phases involved in the sorting of elements. By using the heap sort algorithm, they are as follows –

- The first step includes the creation of a heap by adjusting the elements of the array.
- After the creation of heap, now remove the root element of the heap repeatedly by shifting it to the end of the array, and then store the heapstructure with the remaining elements.

Algorithm of insertion sort & Heap sort:

```
1: for j = 2 to A.length do
2:     key = A[j]
3:     i = j − 1
4:     while i > 0 and A[i] > key do
5:         A[i + 1] = A[i]
6:         i = i − 1
7:     end while
8:     A[i + 1] = key
9: end for
```

Heap Sort:

Here is the pseudocode for Heap Sort, modified to include a counter:

```
count ← 0
HeapSort(A)
1     Build_Max_Heap(A)
2     for i ← length[A] downto 2 do
3           exchange A[1] ↔ A[i]
4           heap-size[A] ← heap-size[A] - 1
5           Max_Heapify(A, 1)
```

And here is the algorithm for the Max_Heapify function used by Heap Sort:

```
Max_Heapify(A, i)
1     l ← LEFT(i)
2     r ← RIGHT(i)
3     if l ≤ heap-size[A] and A[l] > A[i]
4           then largest ← l
5           else largest ← i
6     if r ≤ heap-size[A] and A[r] > A[largest]
7           then largest ← r
8     if largest ≠ i
9           then  exchange A[i] ↔ A[largest]
9.5               count ← count + 1
10                Max_Heapify (A, largest)
```

And here is the algorithm for the Build_Max_Heap function used by Heap Sort:

```
Build_Max_Heap(A)
1     heap-size[A] ← length[A]
2     for i ← floor(length[A]/2) downto 1 do
3           Max_Heapify (A, i)
```

Derivation of Analysis insertion sort & Heap sort:
Insertion Sort

Worst Case Analysis:
In Worst Case i.e., when the array is reversly sorted (in descending order), tj = j
Therefore,T(n) = C1 * n + (C2 + C3) * (n - 1) + C4 * (n - 1) (n) / 2 + (C5 + C6) * ((n - 1) (n) / 2 - 1) + C8 * (n - 1)
which when further simplified has dominating factor of n2 and gives T(n) = C * (n 2) or O(n2)

Best Case Analysis
In Best Case i.e., when the array is already sorted, tj = 1
Therefore, T(n) = C1 * n + (C2 + C3) * (n - 1) + C4 * (n - 1) + (C5 + C6) * (n - 2) + C8 * (n - 1)

Average Case Analysis:
Let's assume that tj = (j-1)/2 to calculate the average case
Therefore,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) which when further simplified has dominating factor of n2 and gives T(n) = C * (n 2) or O(n2)

Heap Sort:

Worst Case Analysis:
The worst case for heap sort might happen when all elements in the list are distinct. Therefore, we would need to call max-heapify every time we remove an element. In such a case, considering there are 'n' number of nodes-

- The number of swaps to remove every element would be log(n), as that is the max height of the heap
- Considering we do this for every node, the total number of moves would be n * (log(n)).

Therefore, the runtime in the worst case will be O(n(log(n)).

Best Case Analysis:
The best case for heapsort would happen when all elements in the list to be sorted are identical. In such a case, for 'n' number of nodes-
Removing each node from the heap would take only a constant runtime, O(1). There would be no need to bring any node down or bring max valued node up, as all items are identical.

- Since we do this for every node, the total number of moves would be n * O(1). Therefore, the runtime in the best case would be O(n).

Average Case Analysis:
In the final function of heapsort, we make use of create_heap, which runs once to create a heap and has a runtime of O(n). Then using a for-loop, we call the max_heapify for each node, to maintain the max-heap property whenever we remove or insert a node in the heap. Since there are 'n' number of nodes, therefore, the total runtime of the algorithm turns out to be O(n(log(n)), and we use the max-heapify function for each node. Mathematically, we see that-

- The first remove of a node takes log(n) time
- The second remove takes log(n-1) time
- The third remove takes log(n-2) time
- and so on till the last node, which will take log(1) time So summing up all the terms, we get-

log(n) + log(n-1) + log(n-2) +log(1) as log(x) + log(y) = log(x * y), we get
=log(n (n−1) (n−2) … 2 1)
=log(n!)
Upon further simplification (using Stirling's approximation), log(n!) turns out to be =n log(n)−n+O(log(n))
Taking into account the highest ordered term, the total runtime turns out to be O(n(log(n)).

Program(s) of insertion sort & Heap sort:

```
#include <bits/stdc++.h>
#define int long long
using namespace std;

void createBest(int arr[], int n)
{
    int x = rand();
    for (int i = 0; i < n; i++)
    {
        arr[i] = x + i;
    }
}
void createAverage(int arr[], int n)
{
    for (int i = 0; i < n; i++)
    {
        arr[i] = rand();
    }
}
void createWorst(int arr[], int n)
{
    int x = rand();
    for (int i = 0; i < n; i++)
    {
        arr[i] = x + n - i;
    }
}

int insertion_sort(int arr[], int n)
{
    int count = 0;
    for (int i = 1; i < n; i++)
    {
        int j = i;
        for (int j = i; j > 0; j--)
        {
            count++;

            if (arr[j] < arr[j - 1])
            {
                arr[j] += arr[j - 1];
                arr[j - 1] = arr[j] - arr[j - 1];
```

```
                arr[j] = arr[j] - arr[j - 1];
            }
            else
            {
                break;
            }
        }
    }

    return count;
}

int32_t main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
#ifndef ONLINE_JUDGE
    freopen("input.txt", "r", stdin);
    freopen("output.txt", "w", stdout);
#endif

    int n;
    cin >> n;
    int best[n], avg[n], worst[n];
    createBest(best, n);
    createAverage(avg, n);
    createWorst(worst, n);

    cout << "Best Case: " << insertion_sort(best, n) << "\n";
    cout << "Average Case: " << insertion_sort(avg, n) << "\n";
    cout << "Worst Case: " << insertion_sort(worst, n);
}
```

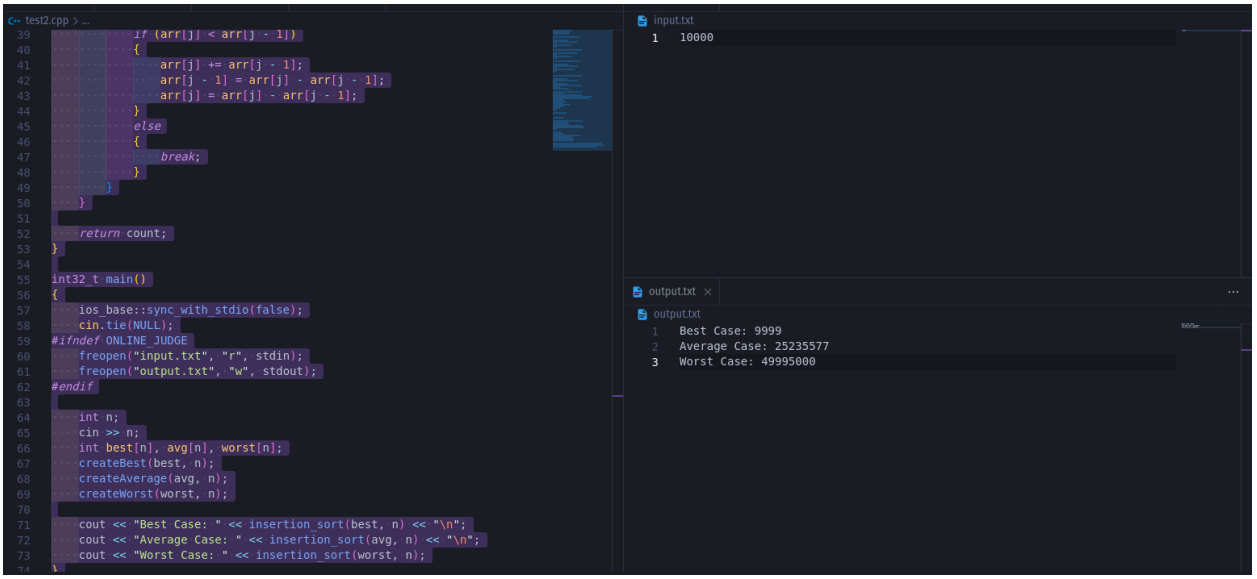
Heap Sort

```
#include <bits/stdc++.h>
// #define int long long
using namespace std;
int counter=0;

void adjust(vector<int> &a, int i, int n)
{
```

```
    int x = a[i];
    int j = i * 2;
    while (j <= n)
    {
        if (a[j] < a[j + 1] && j < n)
            j = j + 1;
        if (x > a[j])
            break;
        a[j / 2] = a[j];
        j = j * 2;

        counter++;
    }
    a[j / 2] = x;
}
void heapSort(vector<int> &v, int n)
{
    for (int i = n / 2; i >= 1; i--)
    {
        adjust(v, i, n);
    }
    for (int i = n; i >= 2; i--)
    {
        swap(v[i], v[1]);
        adjust(v, 1, i - 1);
    }
}

void createBest(vector<int> &arr, int n)
{
    int x = rand();
    for (int i = 0; i < n + 1; i++)
    {
        arr[i] = x + i;
    }
}
void createAverage(vector<int> &arr, int n)
{
    for (int i = 0; i < n + 1; i++)
    {
        arr[i] = rand();
    }
}
```

```
void createWorst(vector<int> &arr, int n)
{
    int x = rand();
    for (int i = 0; i < n + 1; i++)
    {
        arr[i] = x + n - i;
    }
}

int main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
#ifndef ONLINE_JUDGE
    freopen("input.txt", "r", stdin);
    freopen("output.txt", "w", stdout);
#endif

    int n;
    cin >> n;
    vector<int> best(n + 1), avg(n + 1), worst(n + 1);
    createBest(best, n);
    createAverage(avg, n);
    createWorst(worst, n);
    heapSort(best, n);
    cout<<"Best Case- "<<counter<<"\n";
    counter=0;
    heapSort(avg, n);
    cout<<"Average Case- "<<counter<<"\n";
    counter=0;
    heapSort(worst, n);
    cout<<"Worst Case- "<<counter<<"\n";

}
```

Output(o) of insertion sort & Heap sort:
Insertion Sort



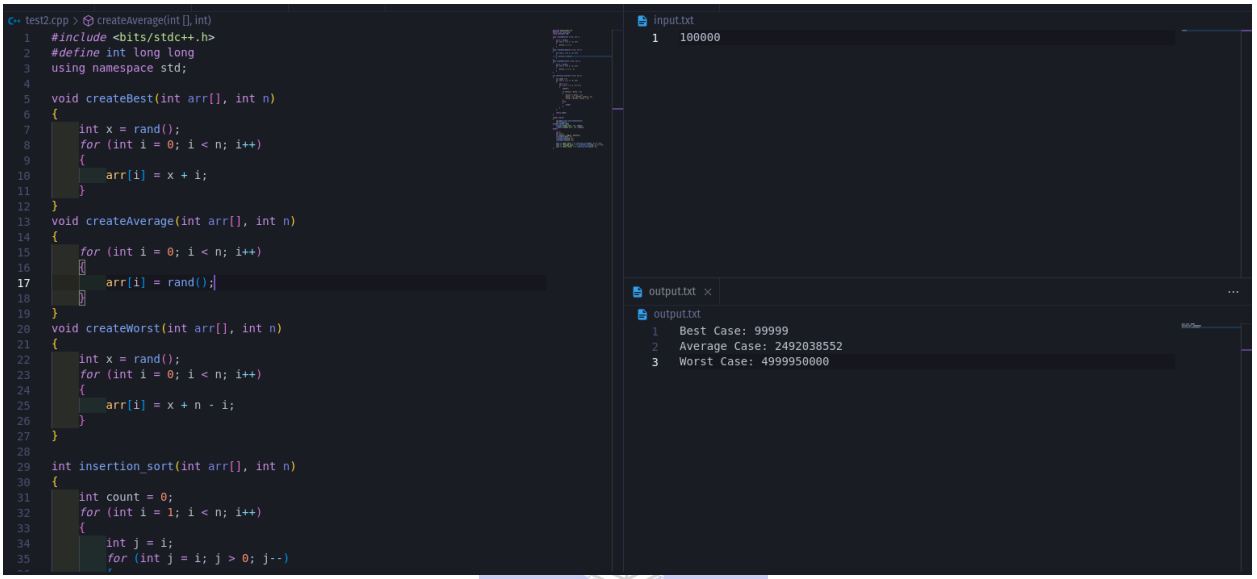
```
            if (arr[j] < arr[j - 1])
            {
                arr[j] += arr[j - 1];
                arr[j - 1] = arr[j] - arr[j - 1];
                arr[j] = arr[j] - arr[j - 1];
            }
            else
            {
                break;
            }
        }
    }

    return count;
}

int32_t main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
#ifndef ONLINE_JUDGE
    freopen("input.txt", "r", stdin);
    freopen("output.txt", "w", stdout);
#endif

    int n;
    cin >> n;
    int best[n], avg[n], worst[n];
    createBest(best, n);
    createAverage(avg, n);
    createWorst(worst, n);

    cout << "Best Case: " << insertion_sort(best, n) << "\n";
    cout << "Average Case: " << insertion_sort(avg, n) << "\n";
    cout << "Worst Case: " << insertion_sort(worst, n);
}
```

input.txt

```
10000
```

output.txt

```
Best Case: 9999
Average Case: 25235577
Worst Case: 49995000
```



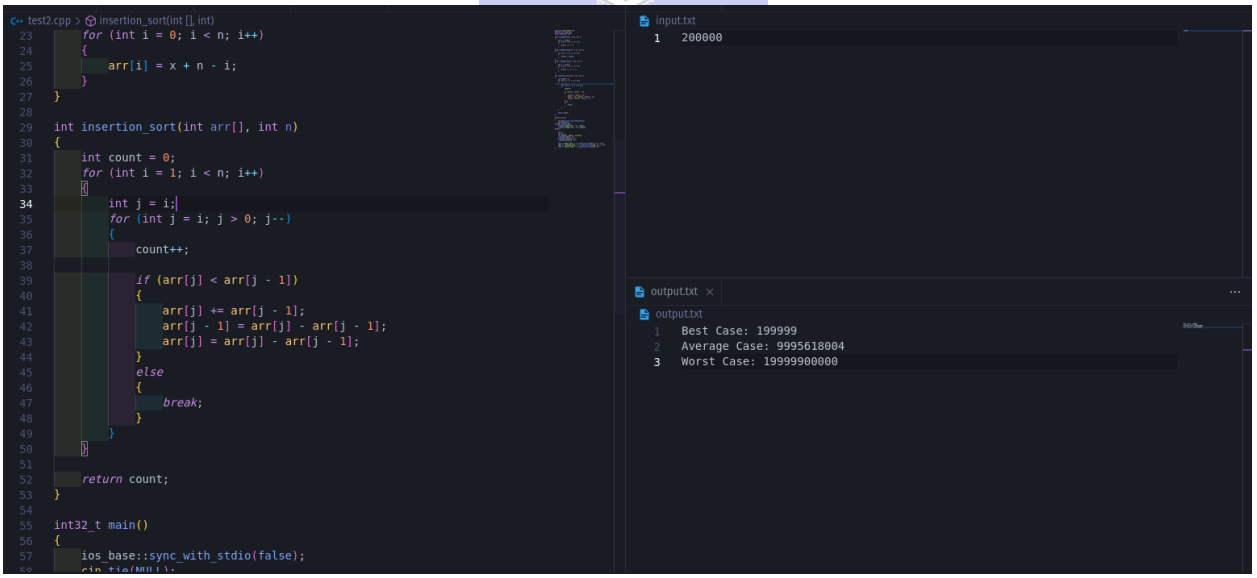
```
#include <bits/stdc++.h>
#define int long long
using namespace std;

void createBest(int arr[], int n)
{
    int x = rand();
    for (int i = 0; i < n; i++)
    {
        arr[i] = x + i;
    }
}
void createAverage(int arr[], int n)
{
    for (int i = 0; i < n; i++)
    {
        arr[i] = rand();
    }
}
void createWorst(int arr[], int n)
{
    int x = rand();
    for (int i = 0; i < n; i++)
    {
        arr[i] = x + n - i;
    }
}

int insertion_sort(int arr[], int n)
{
    int count = 0;
    for (int i = 1; i < n; i++)
    {
        int j = i;
        for (int j = i; j > 0; j--)
```

input.txt

```
100000
```

output.txt

```
Best Case: 99999
Average Case: 2492038552
Worst Case: 4999950000
```



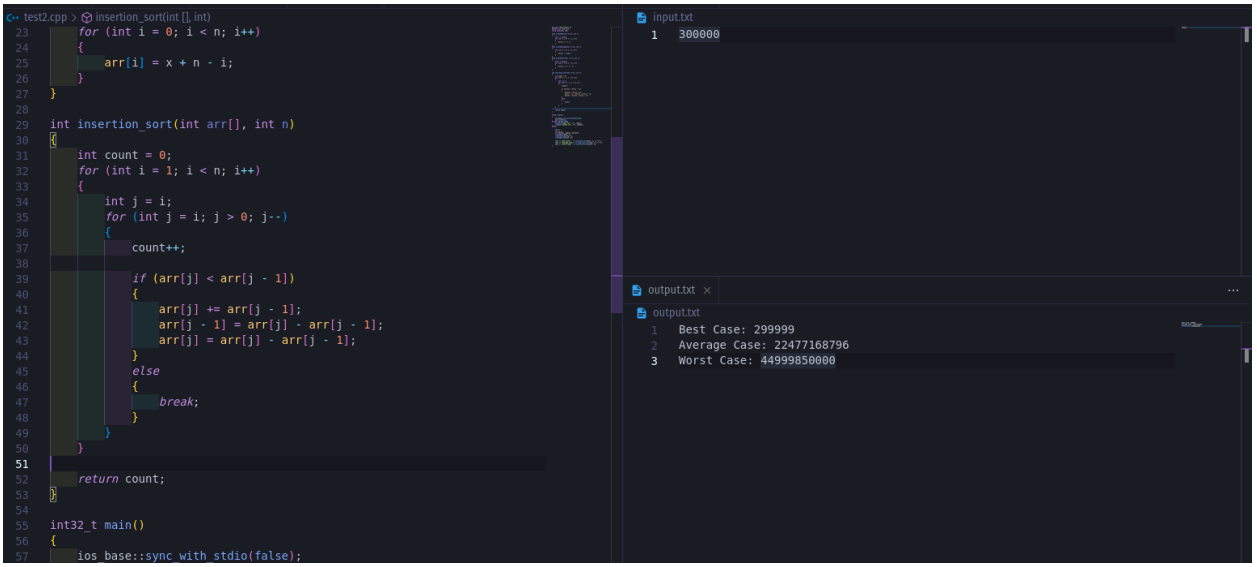
```
    for (int i = 0; i < n; i++)
    {
        arr[i] = x + n - i;
    }
}

int insertion_sort(int arr[], int n)
{
    int count = 0;
    for (int i = 1; i < n; i++)
    {
        int j = i;
        for (int j = i; j > 0; j--)
        {
            count++;

            if (arr[j] < arr[j - 1])
            {
                arr[j] += arr[j - 1];
                arr[j - 1] = arr[j] - arr[j - 1];
                arr[j] = arr[j] - arr[j - 1];
            }
            else
            {
                break;
            }
        }
    }

    return count;
}

int32_t main()
{
    ios_base::sync_with_stdio(false);
```

input.txt

```
200000
```

output.txt

```
Best Case: 199999
Average Case: 9995618004
Worst Case: 19999900000
```



```cpp
    for (int i = 0; i < n; i++)
    {
        arr[i] = x + n - i;
    }
}

int insertion_sort(int arr[], int n)
{
    int count = 0;
    for (int i = 1; i < n; i++)
    {
        int j = i;
        for (int j = i; j > 0; j--)
        {
            count++;

            if (arr[j] < arr[j - 1])
            {
                arr[j] += arr[j - 1];
                arr[j - 1] = arr[j] - arr[j - 1];
                arr[j] = arr[j] - arr[j - 1];
            }
            else
            {
                break;
            }
        }
    }

    return count;
}

int32_t main()
{
    ios_base::sync_with_stdio(false);
```

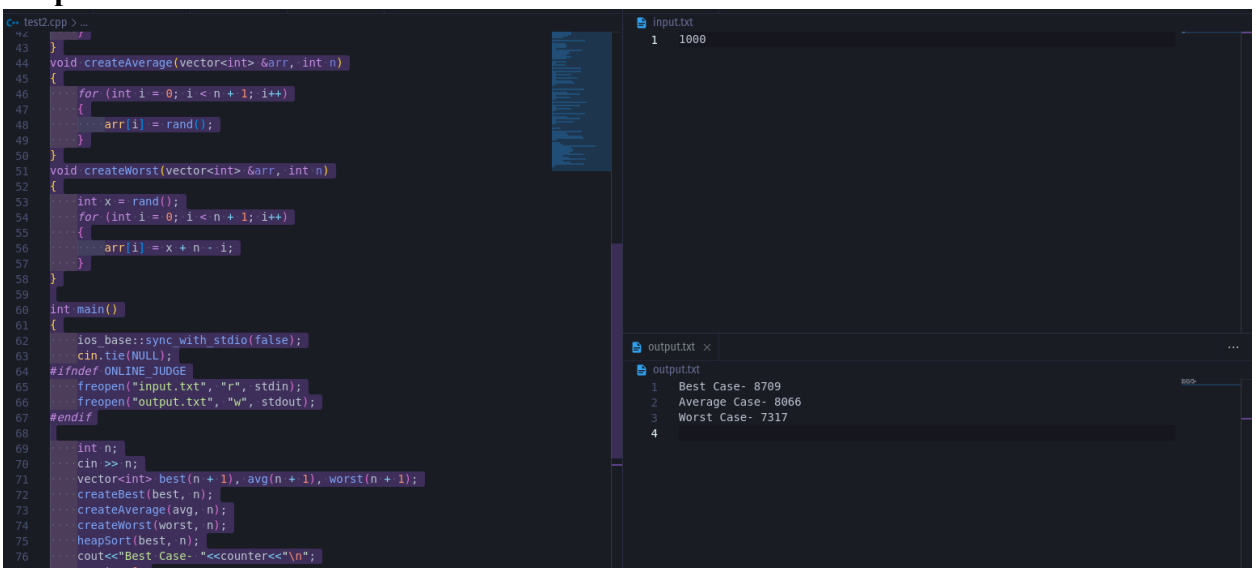
input.txt

```
300000
```

output.txt

```
Best Case: 299999
Average Case: 22477168796
Worst Case: 44999850000
```

Heap Sort



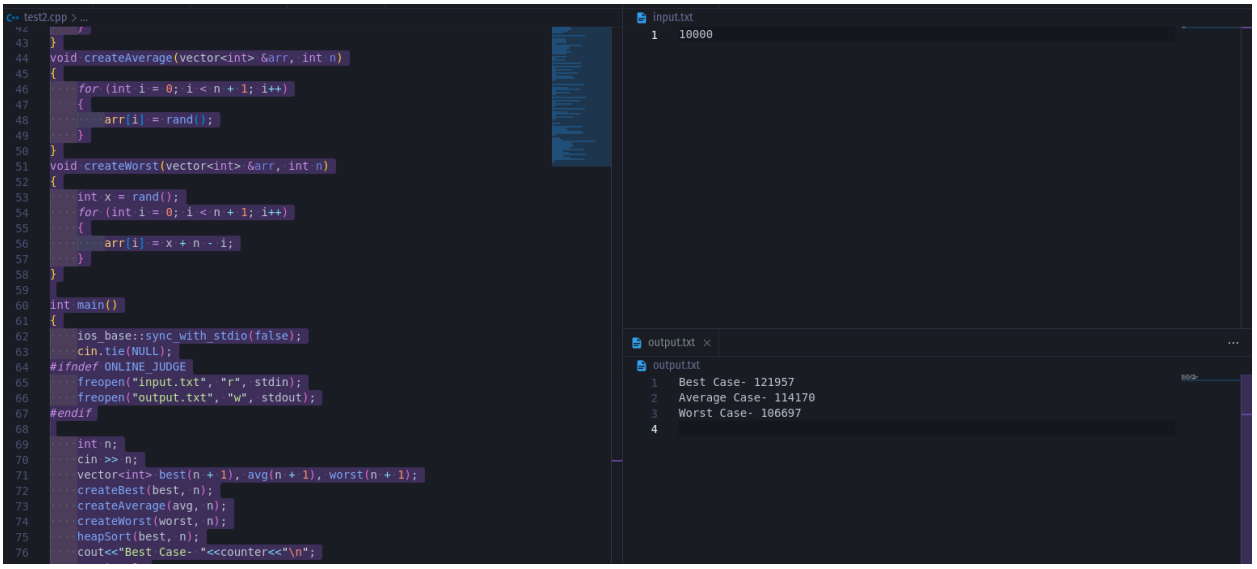
```cpp
}
void createAverage(vector<int> &arr, int n)
{
    for (int i = 0; i < n + 1; i++)
    {
        arr[i] = rand();
    }
}
void createWorst(vector<int> &arr, int n)
{
    int x = rand();
    for (int i = 0; i < n + 1; i++)
    {
        arr[i] = x + n - i;
    }
}

int main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
#ifndef ONLINE_JUDGE
    freopen("input.txt", "r", stdin);
    freopen("output.txt", "w", stdout);
#endif

    int n;
    cin >> n;
    vector<int> best(n + 1), avg(n + 1), worst(n + 1);
    createBest(best, n);
    createAverage(avg, n);
    createWorst(worst, n);
    heapSort(best, n);
    cout<<"Best Case- "<<counter<<"\n";
```

input.txt

```
1000
```

output.txt

```
Best Case- 8709
Average Case- 8066
Worst Case- 7317
```



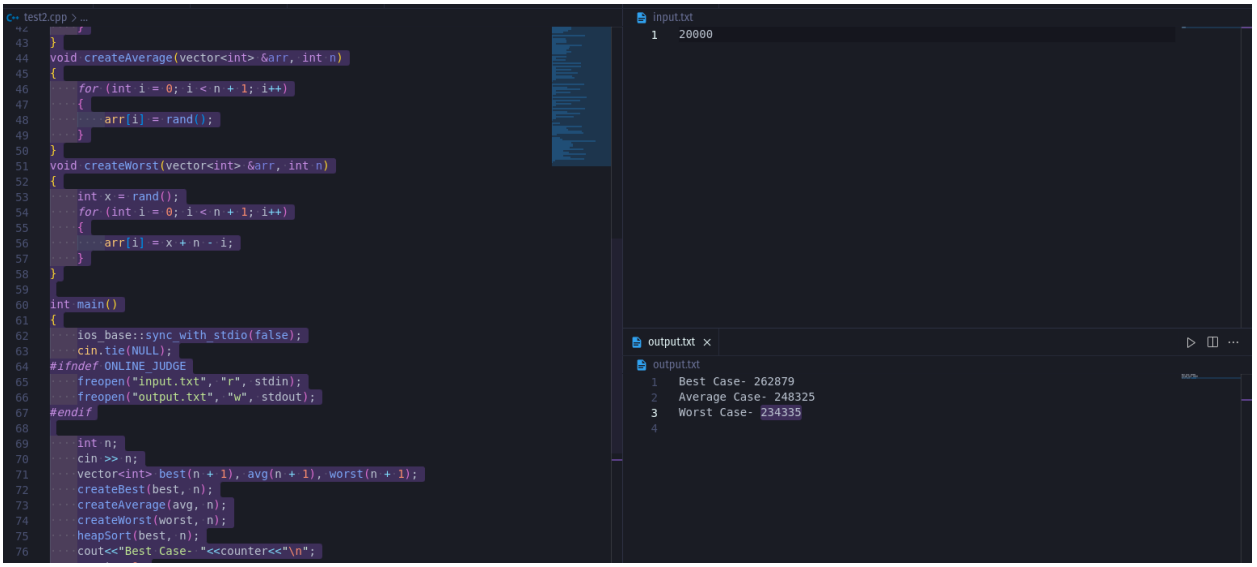
```cpp
}
void createAverage(vector<int> &arr, int n)
{
    for (int i = 0; i < n + 1; i++)
    {
        arr[i] = rand();
    }
}
void createWorst(vector<int> &arr, int n)
{
    int x = rand();
    for (int i = 0; i < n + 1; i++)
    {
        arr[i] = x + n - i;
    }
}

int main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
#ifndef ONLINE_JUDGE
    freopen("input.txt", "r", stdin);
    freopen("output.txt", "w", stdout);
#endif

    int n;
    cin >> n;
    vector<int> best(n + 1), avg(n + 1), worst(n + 1);
    createBest(best, n);
    createAverage(avg, n);
    createWorst(worst, n);
    heapSort(best, n);
    cout<<"Best Case- "<<counter<<"\n";
```

input.txt

```
10000
```

output.txt

```
Best Case- 121957
Average Case- 114170
Worst Case- 106697
```



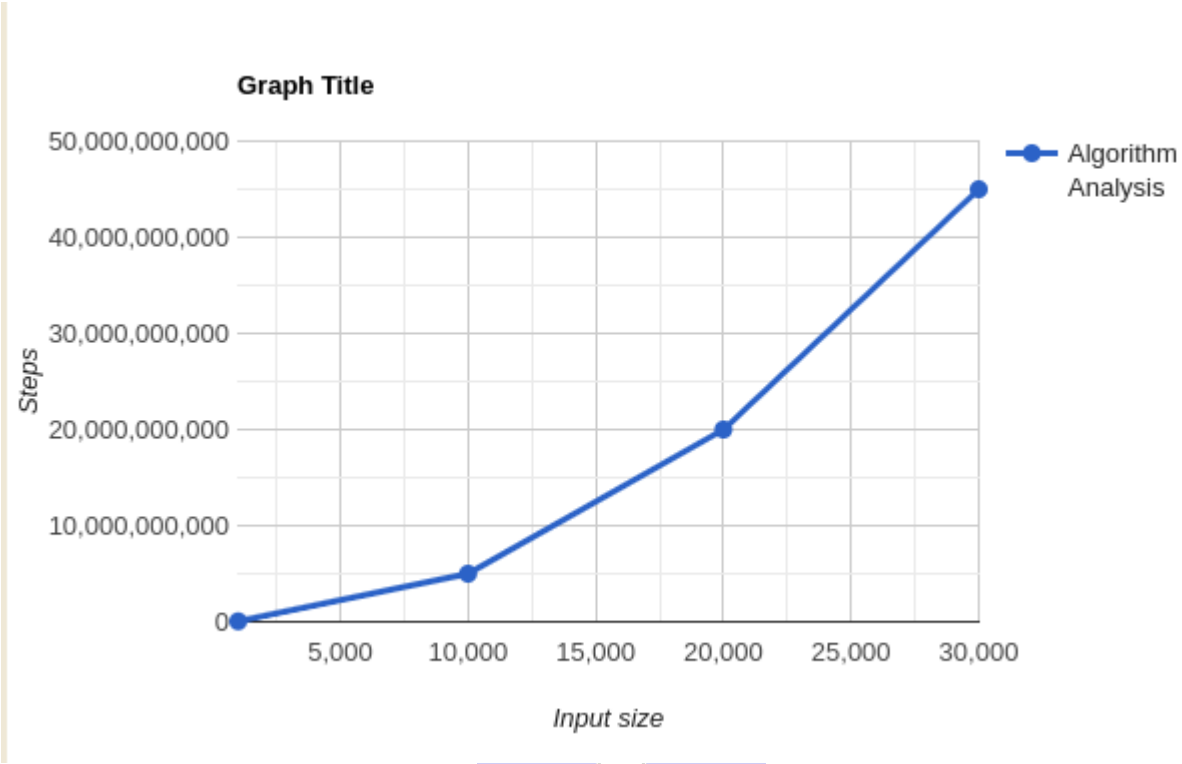
Results:

Time Complexity of Insertion sort:

Worst Case Analysis:

Sr. No.	Input size	No: of steps from Algorithm analysis	No: of steps from Theoretical analysis
	1000	49995000	1000000
	10000	4999950000	100000000
	20000	19999900000	400000000
	30000	44999850000	900000000

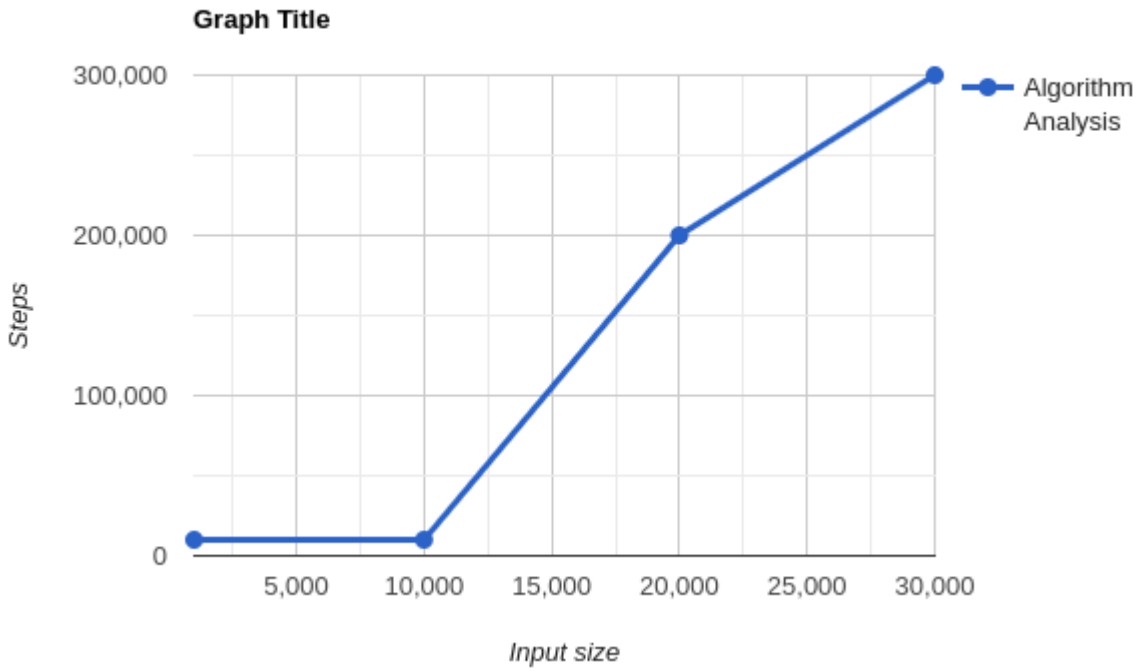
GRAPH:



Best Case Analysis:

Sr. No.	Input size	No: of steps from Algorithm analysis	No: of steps from Theoretical analysis
	1000	9999	1000000
	10000	9999	100000000
	20000	199999	400000000
	30000	299999	900000000

GRAPH

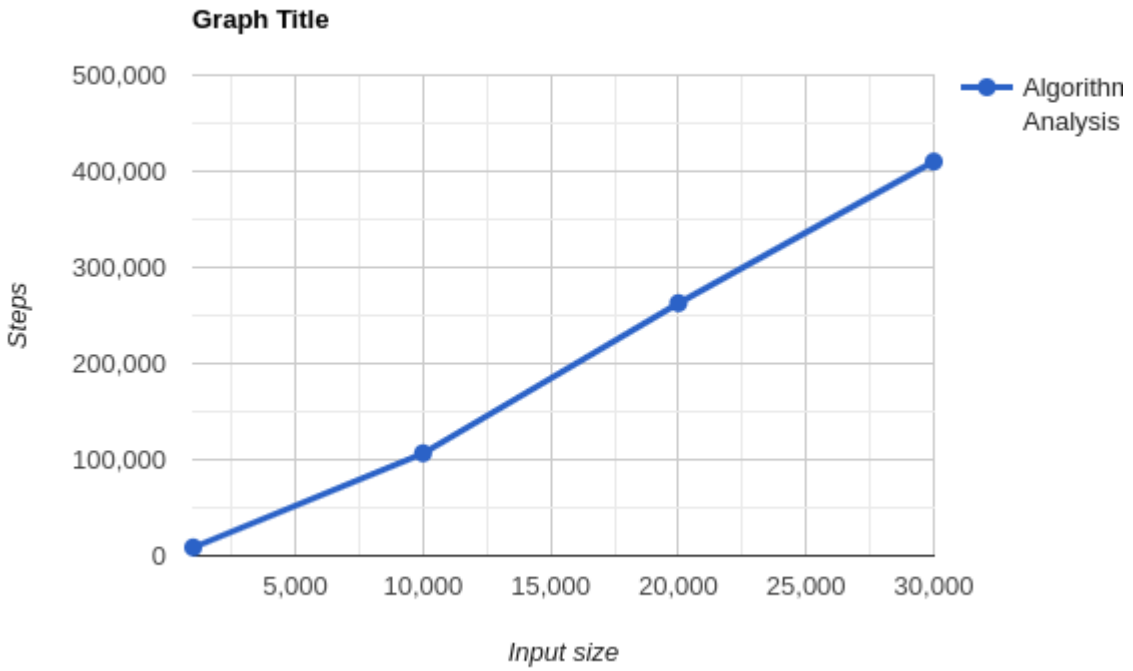


Time Complexity of Heap sort:

Worst Case Analysis:

Sr. No.	Input size	No: of steps from Algorithm analysis	No: of steps from Theoretical analysis
	1000	8709	1000000
	10000	106697	100000000
	20000	262879	400000000
	30000	410101	900000000

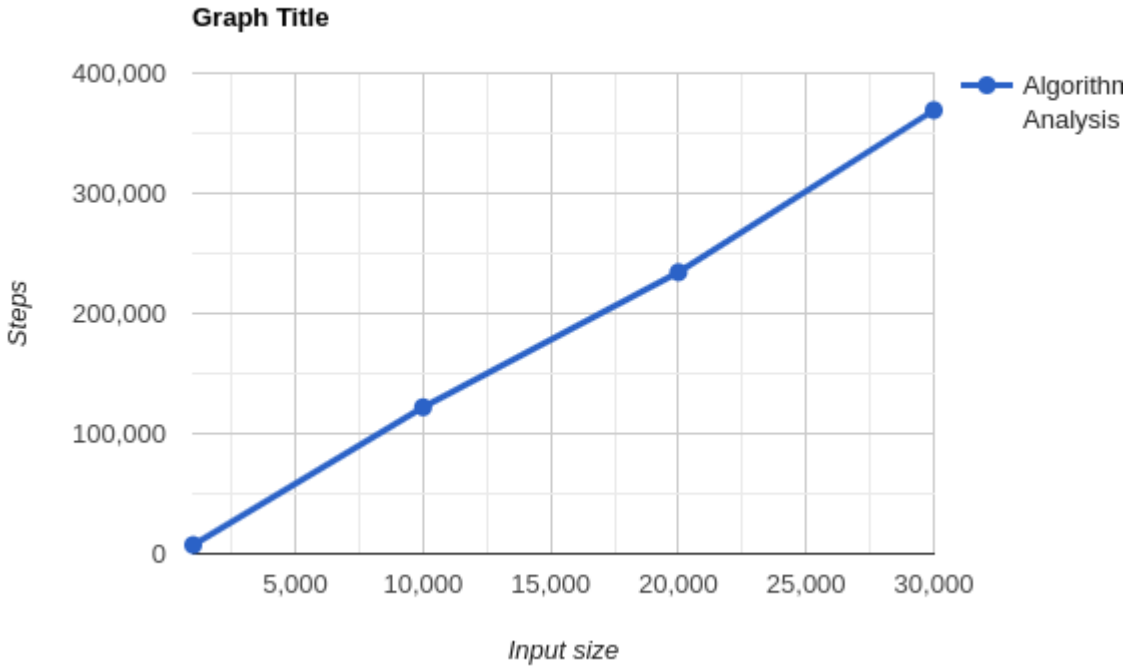
GRAPH:



Best Case Analysis:

Sr. No.	Input size	No: of steps from Algorithm analysis	No: of steps from Theoretical analysis
	1000	7317	1000000
	10000	121957	100000000
	20000	234335	400000000
	30000	369213	900000000

GRAPH



Conclusion: (Based on the observations):
Successfully understood and implemented the Insertion Sort and Heap Sort Algorithm

Outcome: CO1: Analyze time and space complexity of basic algorithms

References:

1. Richard E. Neapolitan, " Foundation of Algorithms ", 5th Edition 2016, Jones & Bartlett Students Edition
2. Harsh Bhasin , " Algorithms : Design & Analysis", 1st Edition 2013, Oxford Higher education, India
3. T.H. Coreman ,C.E. Leiserson,R.L. Rivest, and C. Stein, " Introduction to algorithms", 3rd Edition 2009, Prentice Hall India Publication
4. Jon Kleinberg, Eva Tardos, " Algorithm Design", 10th Edition 2013, Pearson India Education Services Pvt. Ltd.