ALGORITHM LABORATORY

ASSIGNMENT-2

PROBLEM STATEMENT: Implement Bubble sort, straight insertion sort and straight selection sort.

ALGORITHM (BUBBLE SORT):

a. BOUND <= n.
b. t <= 0. Perform step c for i = 1,2, …., BOUND - 1
c. If $K_i > K_i + 1$ then $R_i \Leftrightarrow R_i + 1$ and t <= i.
d. If t=0 , terminate the algorithm. Otherwise BOUND <= t and go to Step 2.

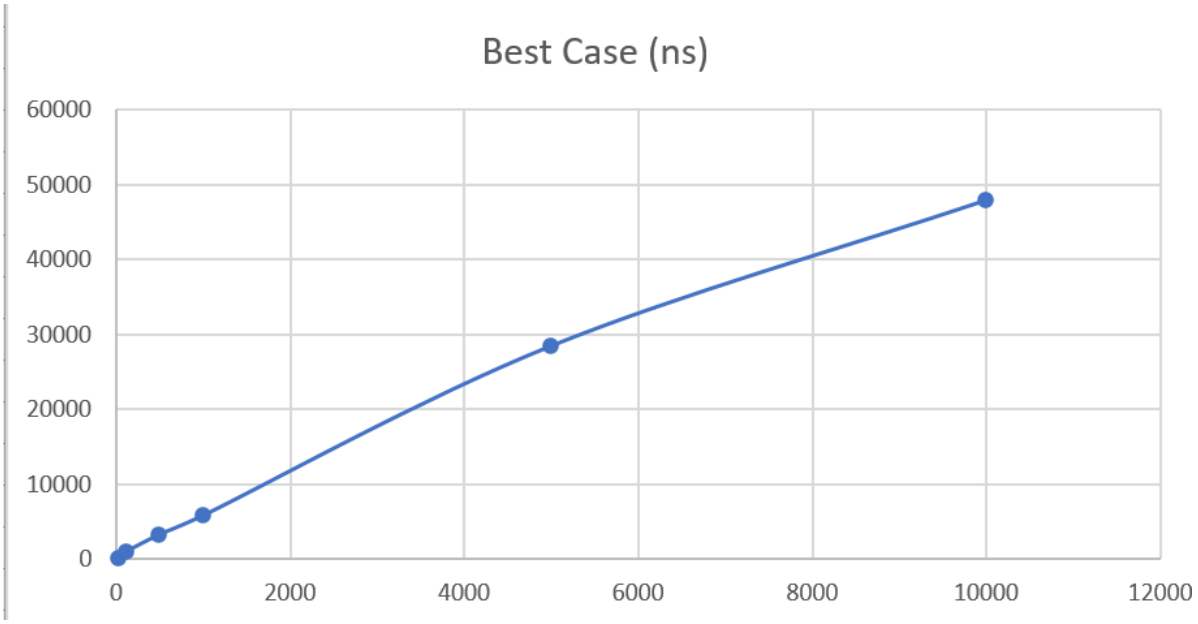
PROGRAM CODE:

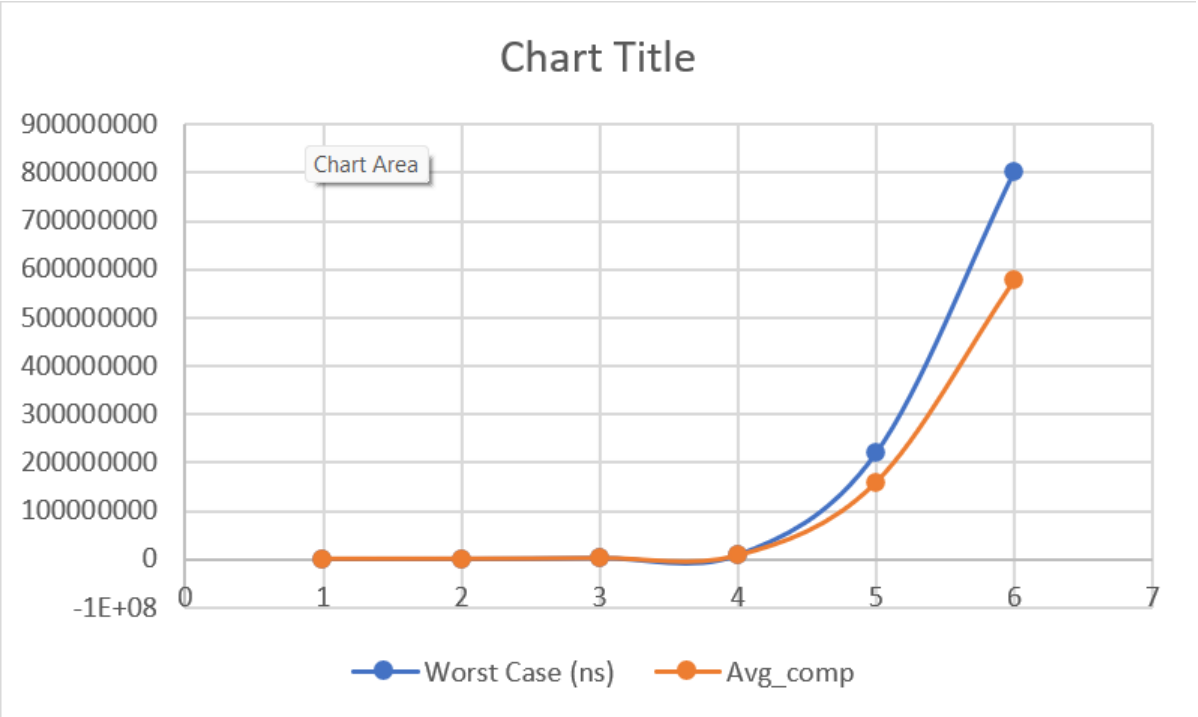
```
#include <bits/stdc++.h>
using namespace std;
using namespace std::chrono;
void bubble_sort(vector<int> &a, int n){
    for (int i = 0; i < n - 1; i++){
        int t = 0;
        for (int j = 0; j < n - i - 1; j++){
            if (a[j] > a[j + 1]){
                swap(a[j], a[j + 1]);
                t=1;
            }
        }
        if(t==0) return;
    }
}


int main(){
    srand(time(NULL));
    vector<int> size = {10,100,500,1000,5000,10000};
    for (int k : size){
        double best_case_average_time = 0;
        double worst_case_average_time = 0;
        double random_case_average_time = 0;
        int num_trials = 100;
        if(k==10000) num_trials = 10;
```

```
        for(int i=0;i<num_trials;i+=1){
            unordered_set<int> s;
            while (s.size() < k){
                s.insert(rand() % (5 * k));
            }
            vector<int> a(s.begin(), s.end());
            auto start = chrono::high_resolution_clock::now();
            bubble_sort(a, k);
            auto end = chrono::high_resolution_clock::now();
            auto time = duration_cast<nanoseconds> (end - start);
            random_case_average_time+=time.count();
            start = chrono::high_resolution_clock::now();
            bubble_sort(a, k);
            end = chrono::high_resolution_clock::now();
            time = duration_cast<nanoseconds>(end-start);
            best_case_average_time+=time.count();
            reverse(a.begin(), a.end());
            start = chrono::high_resolution_clock::now();
            bubble_sort(a, k);
            end = chrono::high_resolution_clock::now();
            time = duration_cast<nanoseconds>(end-start);
            worst_case_average_time+=time.count();
        }
        if(k<10000){
            worst_case_average_time/=100;
            best_case_average_time/=100;
            random_case_average_time/=100;
        }
        else{
            worst_case_average_time/=10;
            best_case_average_time/=10;
            random_case_average_time/=10;
        }
        cout<<"N = "<<k<<" | best_case_average = "<<best_case_average_time<<"
ns | worst_case_average = "<<worst_case_average_time<<" ns |
random_case_average = "<<random_case_average_time<<" ns"<<endl;
    }
}
```

OUTPUT and PLOT:

N value	Best Case (ns)	Worst Case (ns)	Avg_comp
10	120.7	1267.8	1086.3
100	886.2	106167	91646.8
500	3290.65	2.00E+06	1.48E+06
1000	5820.56	7.90E+06	6.75E+06
5000	28446.4	2.18E+08	1.58E+08
10000	47899.9	8.01E+08	5.77E+08





ALGORITHM (STRAIGHT INSERTION SORT):

a. Repeat step b to step e for j = 2,3,4, …., n
b. i <= j – 1, K <= K_j , R <= R_j
c. If K >= K_j go to step e.
d. Ki <= Ki-1, Ri <= Ri-1, i <= i-1. If i>0, then go to c.
e. K_i <= K, R_i <= R

PROGRAM CODE:

```
#include <bits/stdc++.h>
using namespace std;
using namespace std::chrono;
void starightInsertionSort(vector<int> &a, int n){
    for (int i = 1; i < n; i++){
        int key = a[i];
        int j = i - 1;
        while (j >= 0 && a[j] > key)
        {
            a[j + 1] = a[j];
            j--;
        }
        a[j + 1] = key;
    }
}


int main(){
    srand(time(NULL));
    vector<int> size = {10,100,500,1000,5000,10000};
    for (int k : size){
        double best_case_average_time = 0;
        double worst_case_average_time = 0;
        double random_case_average_time = 0;
        int num_trials = 100;
        if(k==10000) num_trials = 10;
        for(int i=0;i<num_trials;i+=1){
            unordered_set<int> s;
            while (s.size() < k){
                s.insert(rand() % (5 * k));
            }
            vector<int> a(s.begin(), s.end());
            auto start = chrono::high_resolution_clock::now();
            starightInsertionSort(a, k);
            auto end = chrono::high_resolution_clock::now();
            auto time = duration_cast<nanoseconds> (end - start);
            random_case_average_time+=time.count();
```

```
            start = chrono::high_resolution_clock::now();
            starightInsertionSort(a, k);
            end = chrono::high_resolution_clock::now();
            time = duration_cast<nanoseconds>(end-start);
            best_case_average_time+=time.count();
            reverse(a.begin(), a.end());
            start = chrono::high_resolution_clock::now();
            starightInsertionSort(a, k);
            end = chrono::high_resolution_clock::now();
            time = duration_cast<nanoseconds>(end-start);
            worst_case_average_time+=time.count();
        }
        if(k<10000){
            worst_case_average_time/=100;
            best_case_average_time/=100;
            random_case_average_time/=100;
        }
        else{
            worst_case_average_time/=10;
            best_case_average_time/=10;
            random_case_average_time/=10;
        }
        cout<<"N = "<<k<<" | best_case_average = "<<best_case_average_time<<"
ns | worst_case_average = "<<worst_case_average_time<<" ns |
random_case_average = "<<random_case_average_time<<" ns"<<endl;
    }
}
```

OUTPUT and PLOT:

N value	Best Case (ns)	Worst Case (ns)	Avg_comp
10	143.2	544.5	505.9
100	709.2	32460.6	17590.8
500	3919.78	838710	425153
1000	8910.91	3.65E+06	1.81E+06
5000	37746.9	9.13E+07	4.68E+07
10000	81354.8	4.08E+08	2.17E+08

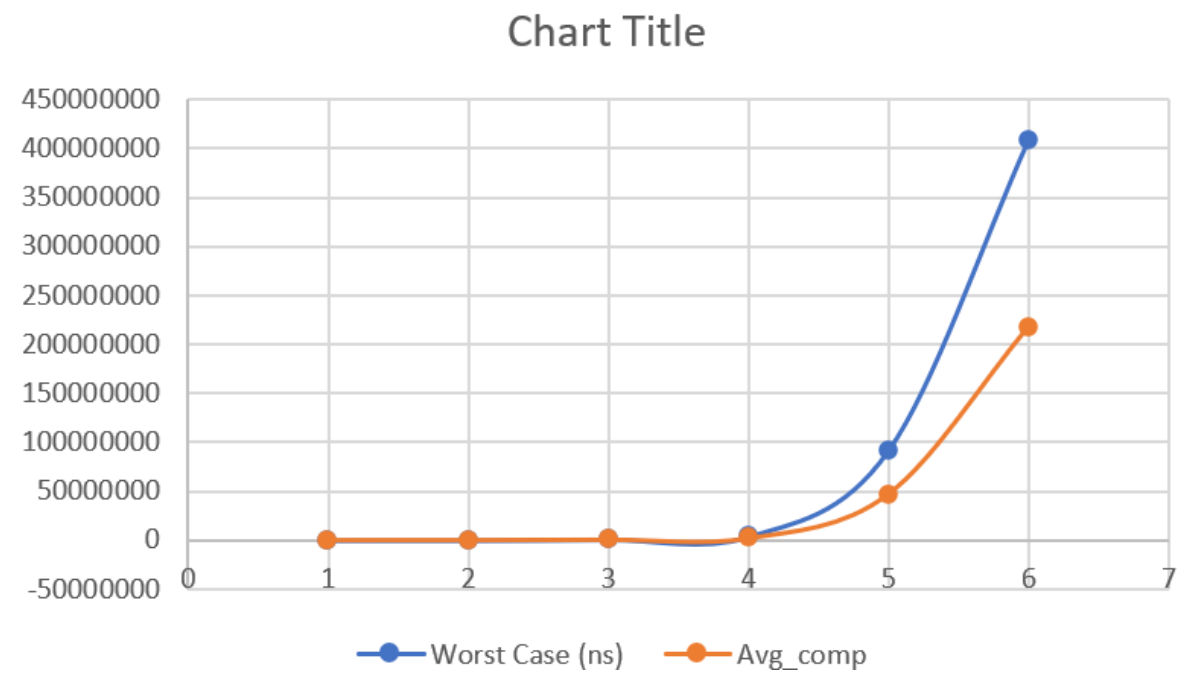
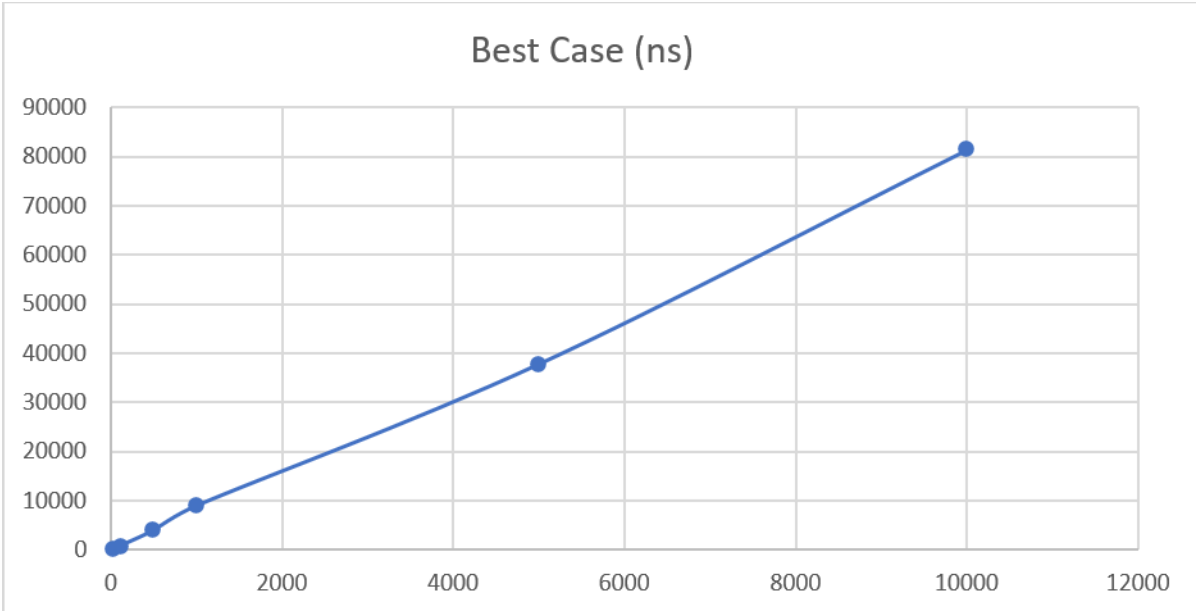


Chart Title
450000000
400000000
350000000
300000000
250000000
200000000
150000000
100000000
50000000
0
-50000000
0
1
2
3
4
5
6
7
Worst Case (ns)
Avg_comp


Best Case (ns)
90000
80000
70000
60000
50000
40000
30000
20000
10000
0
0
2000
4000
6000
8000
10000
12000

ALGORITHM (STRAIGHT SELECTION SORT):

a. SET MIN to location 0.
b. Search the minimum element in the list.
c. Swap with value at location MIN.
d. Increment MIN to point to next element.
e. Repeat until the list is sorted.

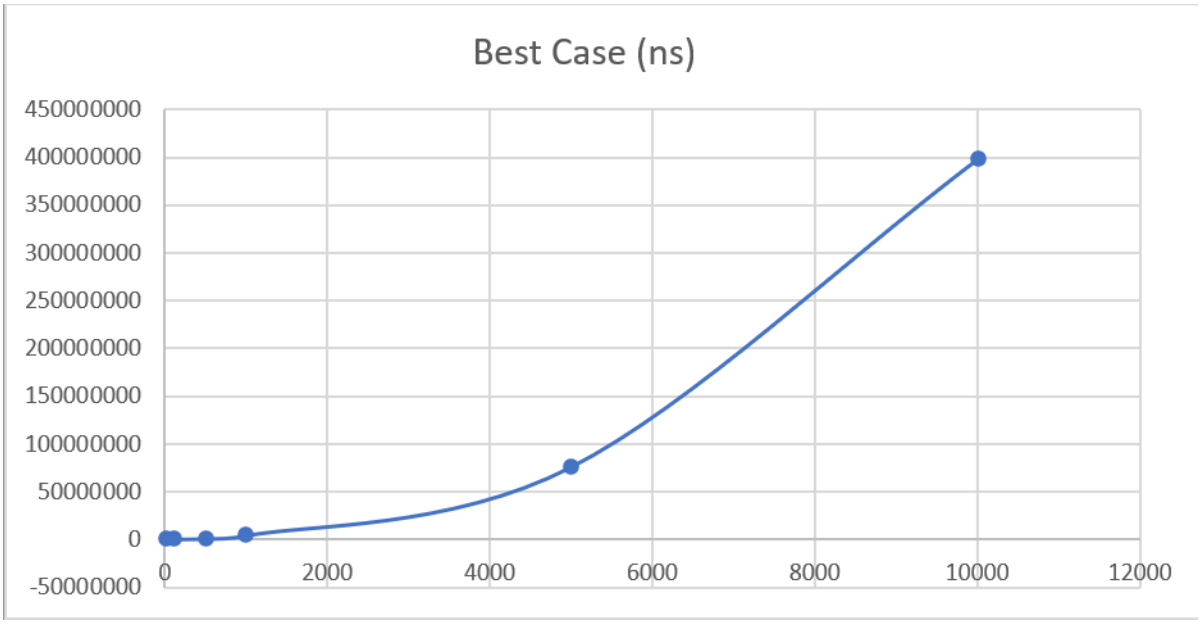
PROGRAM CODE:

```
#include <bits/stdc++.h>
using namespace std;
using namespace std::chrono;
void starightSelectionSort(vector<int> &a, int n){
    for(int i=0;i<n-1;i++){
        int min_index=i;
        for(int j=i+1;j<n;j++){
            if(a[j]<a[min_index]){
                min_index=j;
            }
        }
        swap(a[min_index],a[i]);
    }
}


int main(){
    srand(time(NULL));
    vector<int> size = {10,100,500,1000,5000,10000};
    for (int k : size){
        double best_case_average_time = 0;
        double worst_case_average_time = 0;
        double random_case_average_time = 0;
        int num_trials = 100;
        if(k==10000) num_trials = 10;
        for(int i=0;i<num_trials;i+=1){
            unordered_set<int> s;
            while (s.size() < k){
                s.insert(rand() % (5 * k));
            }
            vector<int> a(s.begin(), s.end());
            auto start = chrono::high_resolution_clock::now();
            starightSelectionSort(a, k);
            auto end = chrono::high_resolution_clock::now();
            auto time = duration_cast<nanoseconds> (end - start);
            random_case_average_time+=time.count();
            start = chrono::high_resolution_clock::now();
```

```
            starightSelectionSort(a, k);
            end = chrono::high_resolution_clock::now();
            time = duration_cast<nanoseconds>(end-start);
            best_case_average_time+=time.count();
            reverse(a.begin(), a.end());
            start = chrono::high_resolution_clock::now();
            starightSelectionSort(a, k);
            end = chrono::high_resolution_clock::now();
            time = duration_cast<nanoseconds>(end-start);
            worst_case_average_time+=time.count();
        }
        if(k<10000){
            worst_case_average_time/=100;
            best_case_average_time/=100;
            random_case_average_time/=100;
        }
        else{
            worst_case_average_time/=10;
            best_case_average_time/=10;
            random_case_average_time/=10;
        }
        cout<<"N = "<<k<<" | best_case_average = "<<best_case_average_time<<"
ns | worst_case_average = "<<worst_case_average_time<<" ns |
random_case_average = "<<random_case_average_time<<" ns"<<endl;
    }
}
```

OUTPUT and PLOT:

N value	Best Case (ns)	Worst Case (ns)	Avg_comp
10	517.3	313.5	571.9
100	23628.9	24905.2	28152.2
500	680111	748439	716508
1000	3.90E+06	3.48E+06	3.53E+06
5000	7.58E+07	7.62E+07	7.63E+07
10000	3.98E+08	3.73E+08	3.83E+08


Best Case (ns)
450000000
400000000
350000000
300000000
250000000
200000000
150000000
100000000
50000000
0
-50000000
0
2000
4000
6000
8000
10000
12000

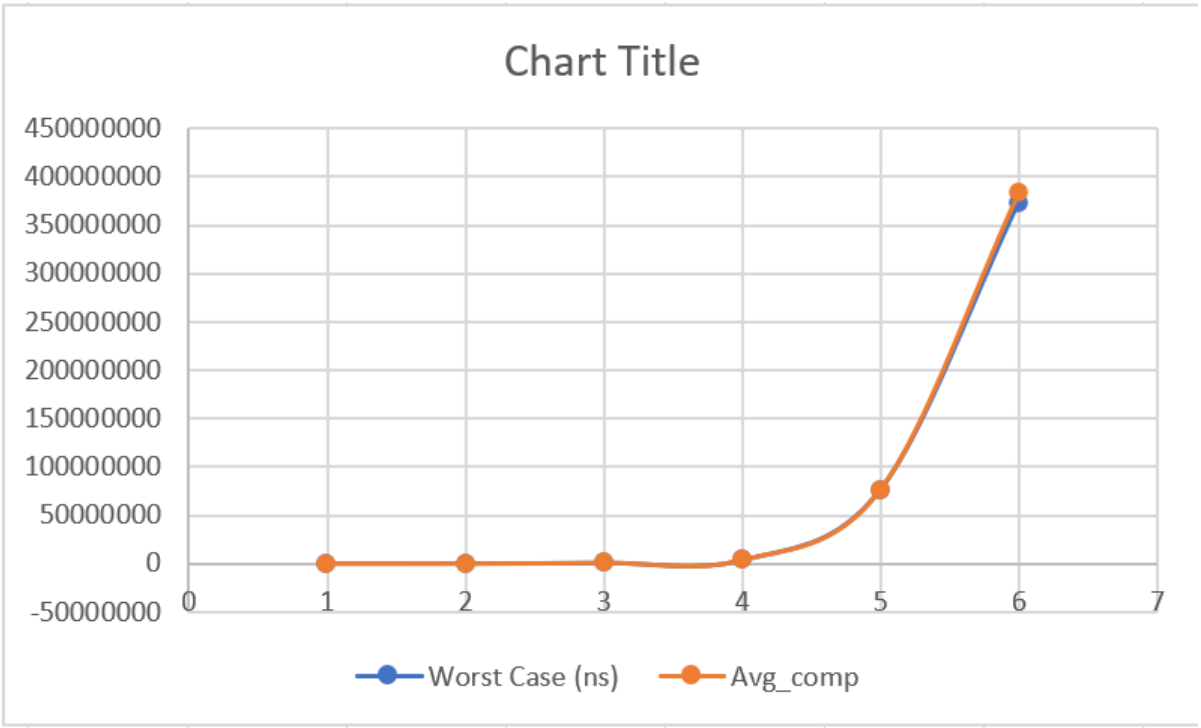


Chart Title
450000000
400000000
350000000
300000000
250000000
200000000
150000000
100000000
50000000
0
-50000000
0
1
2
3
4
5
6
7
Worst Case (ns)
Avg_comp