Design and Analysis of Algorithms (CS206)

Assignment - 1

U19CS012

1. Given the following algorithms, answer the questions.

• Linear Search: Searching Problem

Input: A Sequence of n numbers, **a1,a2,…,an** & Element to Search **key**

Output:
– find **key**: return true, or
– you have unsuccessfully examined all the elements of the array: return false

• Bubble Sort & Selection Sort : Sorting Problem

Input: A Sequence of Unsorted 'n' numbers, **a1,a2,…,an**

Output: A Permutation (Reordering) (**a1',a2',…,an'**) of Input Sequence such that **a1'≤ a2'≤ … ≤ an'**

1.1. (T) Analyze the time complexity of above algorithms using the RAM model

• Linear Search

U19CS012 (1)

A.) Running Time of Linear Search [RAM Model Analysis]

n = length(A)

STEPS	COST	TIMES BEST	TIMES WORST
for i=0 to (A).length-1	c_1	1	$n+1$
if A[i] = key	c_2	1	n
return true	c_3	1	0
return false	c_4	0	1

(A) BEST CASE : The first element is 'key'

$T(n) = c_1 * 1 + c_2 * 1 + c_3 * 1 + c_4 * 0$

n: size of input Array $= c_1 + c_2 + c_3$

$= \boxed{a}$ where $a = c_1 + c_2 + c_3$ (constant)

$= \boxed{O(1)}$

(B) WORST CASE : The element is not present in whole array

$T(n) = c_1(n+1) + c_2 * (n) + c_3 * 0 + c_4(1)$

$= (c_1 + c_2) n + c_1 + c_4$

$= a n + b$ {a, b ∈ constants}

= linear function of n $= O(n)$

• Bubble Sort

B.) Running Time of Bubble Sort

n = size of input array

STEPS	COST	TIMES (BEST)	TIMES (WORST)
for i=0 to (length(A)-1)	c_1	n	n
for j=0 to (length(A)-i-1)	c_2	$\sum_{i=0}^{n-1}(n-i+1)$	$\sum_{i=0}^{n-1}(n-i+1)$
if A[j] > A[j+1]	c_3	$\sum_{i=0}^{n-1}(n-i)$	$\sum_{i=0}^{n-1}(n-i)$
tmp = A[j]	c_4	0	$\sum_{i=0}^{n-1}(n-i)$
A[j] = A[j+1]	c_5	0	$\sum_{i=0}^{n-1}(n-1)$
A[j+1] = tmp	c_6	0	$\sum_{i=0}^{n-1}(n-1)$

U19CS012

2

$$\sum_{i=0}^{n-1}(n-i+1) = (n+1)(n) + (-)\sum_{i=0}^{(n-1)} i$$

Standard series $\left[\sum_{i=0}^{n-1} = \frac{(n-1)(n)}{2}\right]$

$$= n(n+1) + (-)\frac{n(n-1)}{2}$$

$$= n\left((n+1) - \frac{(n-1)}{2}\right)$$

$$= n\left(\frac{2n+2-n+1}{2}\right)$$

$$= \boxed{\frac{n(n+3)}{2}}$$

$$\sum_{i=0}^{n-1}(n-i) = (n)(n) - \sum_{i=0}^{n-1}(i)$$

$$= n^2 - \frac{n(n-1)}{2}$$

$$= \frac{2n^2-n^2+n}{2} = \boxed{\frac{n(n+1)}{2}}$$

(A) BEST CASE : If the array is already sorted
(No swaps occurs)

$$T(n) = C_1 * n + C_2 \sum_{i=0}^{n-1} (n-i+1) + C_3 \sum_{i=0}^{n-1} (n-i) + (C_4+C_5+C_6)(0)$$

$$= C_1 * n + C_2 \frac{n(n+3)}{2} + C_3 \frac{n(n+1)}{2} + 0$$

$$= \left(\frac{C_2}{2} + \frac{C_3}{2}\right) n^2 + n\left(C_1 + \frac{3n}{2} C_2 + \frac{C_3 n}{2}\right)$$

$$= \boxed{an^2 + bn} = \text{quadratic function of } n, \ (a, b \in \text{constants})$$

$$= \boxed{O(n^2)}$$

(B) WORST CASE : If the array is reverse sorted (descending).

$$T(n) = C_1 * (n) + C_2 * \sum_{i=0}^{n-1} (n-i+1) + C_3 \sum_{i=0}^{n-1} (n-i) + (C_4+C_5+C_6) \sum_{i=0}^{n-1} (n-i)$$

$$= C_1 * n + C_2 * \left(\frac{n(n+3)}{2}\right) + (C_3 + C_4 + C_5 + C_6)\left(\frac{n(n+1)}{2}\right)$$

Bubble sort Worst case continued ...

$$T(n) = \left(\frac{C_2}{2} + \frac{C_3 + C_4 + C_5 + C_6}{2}\right) n^2 + \left(C_1 + \left\{\frac{3}{2} C_2 + \frac{(C_3+C_4+C_5+C_6)}{2}\right\}\right) n + \frac{3C_2}{2}$$

$$= \boxed{an^2 + bn + c}, \ \{a, b, c \in \text{Constant}\}$$

$$= \boxed{O(n^2)} : \text{quadratic function of } (n)$$

(C) AVERAGE CASE : All the inputs of particular size are equiprobable. If the element of Array [0 to j-1] are randomly chosen, we can assume half of elements are greater than A[j] while other half are less. ∴ no. of swaps $= \left(\frac{\text{worst case}}{2}\right)$

$$T(n) = C_1 * n + C_2 * \frac{n * (n+3)}{2} + (C_3 + C_4 + C_5 + C_6) \frac{n(n+1)}{2} \left(\frac{1}{2}\right)$$

$$= \left(\frac{C_2}{2} + \frac{C_3 + C_4 + C_5 + C_6}{4}\right) n^2 + \left(C_1 + \frac{3C_2}{2} + \frac{C_3 + C_4 + C_5 + C_6}{4}\right) n + \frac{3C_2}{2}$$

$$= \boxed{an^2 + bn + c} \ (\text{quadratic function of } n)$$

$$= \boxed{O(n^2)} \ \{a, b, c \in \text{constants}\}$$

But in some cases, average case may tilt towards Best case.

• Selection Sort

c> Running Time of Selection Sort

n = size of Input Array

SELECTION SORT	COST	TIMES (BEST)	TIMES (WORST)
for i=1 to n-1	c_1	n	n
min = i	c_2	$(n-1)$	$(n-1)$
for j = i+1 to n-1	c_3	$\sum_{i=1}^{n-1}(n-i+1)$	$\sum_{i=1}^{n-1}(n-i+1)$
if (A[min] > A[j])	c_4	$\sum_{i=1}^{n-1}(n-i)$	$\sum_{i=1}^{n-1}(n-i)$
min = j	c_5	0	$\sum_{i=1}^{n-1}(n-i)$
if (min != i)	c_6	$(n-1)$	$(n-1)$
temp = A[i]	c_7	0	n-1
A[i] = A[min]	c_8	0	n-1
A[min] = temp	c_9	0	n-1

U19CS012 ④

Ⓐ $\boxed{\text{BEST CASE}}$: If the Array is Already sorted

$$T(n) = c_1 \times n + c_2 \times (n-1) + c_3 \times \sum_{i=1}^{n-1}(n-i+1) + c_4 \times \sum_{i=1}^{n-1}(n-i) + 0 \times c_5 + c_6 \times (n-1) + 0 \times (c_7 + c_8 + c_9)$$

$$= c_1 \times n + c_2 \times (n-1) + c_3\left(n \times (n+1) - \frac{n(n-1)}{2}\right) + c_4\left(n(n-1) - \frac{(n-1)(n)}{2}\right) + c_6 \times (n-1)$$

$$= c_1 \times n + c_2 \times (n-1) + c_3\left(\frac{n^2+3n}{2}\right) + c_4\left(\frac{n(n-1)}{2}\right) + c_6 \times (n-1)$$

$$= \left(\frac{c_3}{2} + \frac{c_4}{2}\right)n^2 + \left(c_1 + c_2 + \frac{3c_3}{2} - \frac{c_4}{2} + c_6\right)n + (-c_2 - c_6)$$

$$= \boxed{an^2 + bn + c}\ ,\ [a, b, c \in \text{constants}]$$

$= \boxed{\text{quadratic function of } n} = \Theta(n^2)$

(B) **WORST CASE**: If the Array is sorted in Reverse order (descending)

$$T(n) = c_1 * n + c_2 * (n-1) + c_3\left(\frac{n^2+3n}{2}\right) + (c_4+c_5)\left(\frac{n(n-1)}{2}\right) + (c_7+c_8+c_9)(n-1)$$

$$= n^2\left(\frac{c_3}{2} + \frac{(c_4+c_5)}{2}\right) + n\left(c_1 + c_2 - \frac{c_4 - c_5}{2} + (c_7+c_8+c_9)\right) + (-c_2 - c_7 - c_8 - c_9)$$

$$\boxed{= an^2 + bn + c = \text{quadratic function of } n = O(n^2)}$$

(C) **AVERAGE CASE**: All the inputs of particular size are equally possible & suppose min element is found at middle of (i+1 to n-1)

$$T(n) = c_1 * n + c_2 * (n-1) + c_3\left(\frac{n^2+3n}{2}\right) + c_4 + c_5\left(\frac{n(n-1)}{4}\right) + (c_7+c_8+c_9)(n-1)$$

$$= n^2\left(\frac{c_3}{2} + \frac{c_4+c_5}{4}\right) + n\left(c_1 + c_2 - \frac{c_4 - c_5}{4} + c_7+c_8+c_9\right) + (-c_2 - c_7 - c_8 - c_9)$$

$$\boxed{= an^2 + bn + c = \text{quadratic function of } n = O(n^2)}$$

1.2. (L) Implement the above algorithms using the programming language of your choice.

- Linear Search

```
• Linear_Search(A,key)
for i=0 to (length(A)-1)
 if A[i] = key
     return true // Element Found
// In case No Element Found in Array, Return False
return false
```

- Bubble Sort

```
• Bubble-sort(A)

for i=0 to (length(A)-1)
   for j=0 to (length(A)-i-1)
      if A[j]>A[j+1]
         tmp = A[j]
         A[j] = A[j+1]
         A[j+1] = tmp
```

- Selection Sort

```
• Selection-sort(A)

for i=0 to (length(A)-1)
   min_idx = i
   for j=i+1 to (length(A)-1)
      if(A[j]<A[min_idx])
         min_idx = j
   tmp = A[min_idx]
   A[min_idx] = A[i]
   A[i] = tmp
```

1.3. (L) Provide the details of Hardware/Software you used to implement algorithms and to measure the time.

Hardware Details of My Laptop:

PARAMETER	LAPTOP CONFIGURATION
Operating System	Microsoft Windows **10** .0.19042
Processor	Intel(R) Core(TM) i5-10210U [Core **i5 10th Gen**]
CPU	**1.60GHz**, 2112 Mhz, **4** Core(s), 8 Logical Processor(s)
System Type	x64-based PC [**64 Bit**]
RAM	**8.00** GB
Hard Drive/SSD	512 GB **SSD**

Software Used:

PARAMETER	LAPTOP CONFIGURATION
Code Editor	**Visual Studio** Code [Version 1.52]
Compiler	gcc (MinGW.org **GCC-8.2.0-5**) 8.2.0
Time	Measured using **chrono** Library in C++
Programming Language Used	**C++**

1.4. (L) Submit the code (complete programs).

• Linear Search

```
// HEADERS AND NAMESPACE
#include <bits/stdc++.h>
// INSTEAD OF ALL THESE
#include <iostream>
// For Creating File
#include <fstream>
#include <vector>
// For set - precision
#include <iomanip>
// For Time Calculation
#include <chrono>
// For File Name and Output File Name
#include <string>

using namespace std;
```

```cpp
using namespace std::chrono;

// COMMONLY USED TYPES
typedef long long ll;
typedef vector<ll> vll;

// Basic Algorithm Implementation of Binary Search
bool linear_search(vll arr, ll key)
{
    ll sz = arr.size(), i;
    for (i = 0; i < sz; i++)
    {
        if (arr[i] == key)
            return true;
    }
    return false;
}

int main()
{
    // For Read & Write from "Input File" and  Return Output to "Output" File
    freopen("output.txt", "w", stdout);

    // EDIT THIS FILE NUMBER , LIMIT and Number of Times File Runs
    int file_no = 1;
    int limit = 10;
    int each_file_runs = 2;

    for (; file_no <= limit; file_no++)
    {
        string inp_file = "File";
        string num = to_string(file_no);
        string ext = ".txt";
        inp_file += num;
        inp_file += ext;

        ifstream File;
        File.open(inp_file);

        vector<ll> arr;

        ll number, idx = 0;
        while (!File.eof())
        {
            File >> number;
            arr.push_back(number);
        }

        ll Best_Duration = 0, Worst_Duration = 0, Average_Duration = 0;
        auto start = high_resolution_clock::now();
```

```cpp
        auto end = high_resolution_clock::now();
        auto time_taken = duration_cast<nanoseconds>(end - start);
        ll sz = arr.size();
        for (int f = 0; f < each_file_runs; f++)
        {
            // ------------------------AVERAGE CASE [O(n/2)]----------------------------
            // Search for Random Number in Array
            start = high_resolution_clock::now();
            // Function Here
            linear_search(arr, arr[sz / 2]);
            // Function Ends here
            end = high_resolution_clock::now();
            time_taken = duration_cast<nanoseconds>(end - start);
            Average_Duration += time_taken.count();

            // ------------------------BEST CASE [O(1)]----------------------------
            // Search for First Value in Array
            start = high_resolution_clock::now();
            // Function Here
            linear_search(arr, arr[0]);
            // Function Ends here
            end = high_resolution_clock::now();
            time_taken = duration_cast<nanoseconds>(end - start);
            Best_Duration += time_taken.count();

            // ------------------------WORST CASE [O(n)]----------------------------
            // Search for Value Not Present in Array [Negative Value]
            start = high_resolution_clock::now();
            // Function Here
            linear_search(arr, -1);
            // Function Ends here
            end = high_resolution_clock::now();
            time_taken = duration_cast<nanoseconds>(end - start);
            Worst_Duration += time_taken.count();
        }

        cout << "---------------------------------------------------------" << endl;
        cout << inp_file << endl;
        cout << "AVERAGE CASE : ";
        double avg = (double)Average_Duration / (double)each_file_runs;
        avg *= 1e-9;
        cout << fixed << avg << setprecision(9);
        cout << " seconds" << endl;
        cout << "BEST CASE    : ";
        double best = (double)Best_Duration / (double)each_file_runs;
        best *= 1e-9;
        cout << fixed << best << setprecision(9);
        cout << " seconds" << endl;
        cout << "WORST CASE   : ";
        double worst = (double)Worst_Duration / (double)each_file_runs;
```

```
        worst *= 1e-9;
        cout << fixed << worst << setprecision(9);
        cout << " seconds" << endl;
    }

    return 0;
}
```

• Bubble Sort

```
// HEADERS AND NAMESPACE
#include <bits/stdc++.h>
// INSTEAD OF ALL THESE
#include <iostream>
// For Creating File
#include <fstream>
#include <vector>
// For set - precision
#include <iomanip>
// For Time Calculation
#include <chrono>
// For File Name and Output File Name
#include <string>

using namespace std;
using namespace std::chrono;

// COMMONLY USED TYPES
typedef long long ll;
typedef vector<ll> vll;

// Basic Algorithm Implementation of Bubble Sort
void bubble_sort(vll &arr)
{
    ll n = arr.size(), i, j, tmp;
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < n - i - 1; j++)
        {
            if (arr[j] > arr[j + 1])
            {
                tmp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = tmp;
            }
        }
    }
}
```

```cpp
int main()
{
    // For Read & Write from "Input File" and  Return Output to "Output" File
    freopen("output.txt", "a+", stdout);

    // EDIT THIS FILE NUMBER , LIMIT and Number of Times File Runs
    int file_no = 1;
    int limit = 5;
    int each_file_runs = 2;

    for (; file_no <= limit; file_no++)
    {
        string inp_file = "File";
        string num = to_string(file_no);
        string ext = ".txt";
        inp_file += num;
        inp_file += ext;

        ifstream File;
        File.open(inp_file);

        vector<ll> arr;

        ll number, idx = 0;
        while (!File.eof())
        {
            File >> number;
            arr.push_back(number);
        }

        ll Best_Duration = 0, Worst_Duration = 0, Average_Duration = 0;
        auto start = high_resolution_clock::now();
        auto end = high_resolution_clock::now();
        auto time_taken = duration_cast<nanoseconds>(end - start);
        for (int f = 0; f < each_file_runs; f++)
        {
            // ------------------------AVERAGE CASE [O(n^2)]----------------------------

            start = high_resolution_clock::now();
            // Function Here
            bubble_sort(arr);
            // Function Ends here
            end = high_resolution_clock::now();
            time_taken = duration_cast<nanoseconds>(end - start);
            Average_Duration += time_taken.count();

            // ------------------------BEST CASE [0(n)]----------------------------
            // The Array is Already Sorted from Average Case, So it Becomes out Best Case
            // sort(arr.begin(), arr.end());
```

```cpp
            start = high_resolution_clock::now();
            // Function Here
            bubble_sort(arr);
            // Function Ends here
            end = high_resolution_clock::now();
            time_taken = duration_cast<nanoseconds>(end - start);
            Best_Duration += time_taken.count();

            // -----------------------WORST CASE [O(n^2)]---------------------------
            // This will Reverse the Sorted Array, Therfore we will Get the Worst Case

            reverse(arr.begin(), arr.end());
            // sort(arr.begin(), arr.end(), greater<ll>());
            start = high_resolution_clock::now();
            // Function Here
            bubble_sort(arr);
            // Function Ends here
            end = high_resolution_clock::now();
            time_taken = duration_cast<nanoseconds>(end - start);
            Worst_Duration += time_taken.count();
        }

        cout << "---------------------------------------------------------" << endl;
        cout << inp_file << endl;
        cout << "AVERAGE CASE : ";
        double avg = (double)Average_Duration / (double)each_file_runs;
        avg *= 1e-9;
        cout << fixed << avg << setprecision(9);
        cout << " seconds" << endl;
        cout << "BEST CASE    : ";
        double best = (double)Best_Duration / (double)each_file_runs;
        best *= 1e-9;
        cout << fixed << best << setprecision(9);
        cout << " seconds" << endl;
        cout << "WORST CASE   : ";
        double worst = (double)Worst_Duration / (double)each_file_runs;
        worst *= 1e-9;
        cout << fixed << worst << setprecision(9);
        cout << " seconds" << endl;
    }

    return 0;
}
```

- Selection Sort

```
// HEADERS AND NAMESPACE
#include <bits/stdc++.h>
// INSTEAD OF ALL THESE
#include <iostream>
// For Creating File
#include <fstream>
#include <vector>
// For set - precision
#include <iomanip>
// For Time Calculation
#include <chrono>
// For File Name and Output File Name
#include <string>

using namespace std;
using namespace std::chrono;

// COMMONLY USED TYPES
typedef long long ll;
typedef vector<ll> vll;

// Basic Algorithm Implementation of Selection Sort
void selection_sort(vll &arr)
{
    ll n = arr.size(), i, j, tmp, min_idx;
    for (i = 0; i < n - 1; i++)
    {
        min_idx = i;
        for (j = i + 1; j < n; j++)
        {
            if (arr[j] < arr[min_idx])
            {
                min_idx = j;
            }
        }

        // Swap a[min_idx] and a[i]
        tmp = arr[min_idx];
        arr[min_idx] = arr[i];
        arr[i] = tmp;
    }
}

int main()
{
    // For Read & Write from "Input File" and  Return Output to "Output" File
    freopen("output.txt", "a+", stdout);
```

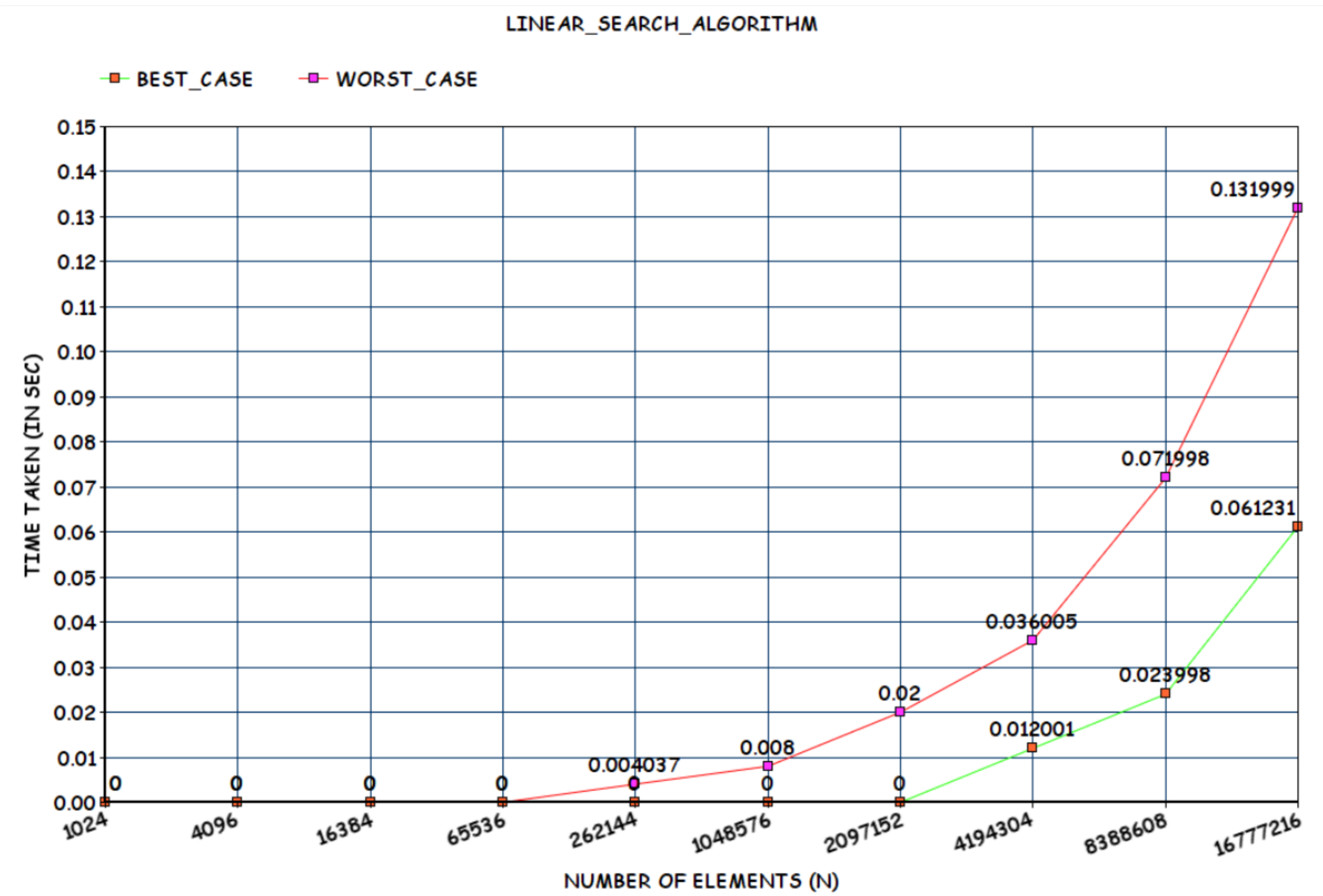
```cpp
// EDIT THIS FILE NUMBER , LIMIT and Number of Times File Runs
int file_no = 1;
int limit = 5;
int each_file_runs = 1;

for (; file_no <= limit; file_no++)
{
    string inp_file = "File";
    string num = to_string(file_no);
    string ext = ".txt";
    inp_file += num;
    inp_file += ext;

    ifstream File;
    File.open(inp_file);

    vector<ll> arr;

    ll number, idx = 0;
    while (!File.eof())
    {
        File >> number;
        arr.push_back(number);
    }

    ll Best_Duration = 0, Worst_Duration = 0, Average_Duration = 0;
    auto start = high_resolution_clock::now();
    auto end = high_resolution_clock::now();
    auto time_taken = duration_cast<nanoseconds>(end - start);
    for (int f = 0; f < each_file_runs; f++)
    {
        // -------------------------AVERAGE CASE [O(n^2)]----------------------------

        start = high_resolution_clock::now();
        // Function Here
        selection_sort(arr);
        // Function Ends here
        end = high_resolution_clock::now();
        time_taken = duration_cast<nanoseconds>(end - start);
        Average_Duration += time_taken.count();

        // -------------------------BEST CASE [0(n)]------------------------------
        // The Array is Already Sorted from Average Case, So it Becomes out Best Case
        // sort(arr.begin(), arr.end());
        start = high_resolution_clock::now();
        // Function Here
        selection_sort(arr);
        // Function Ends here
        end = high_resolution_clock::now();
```

```cpp
            time_taken = duration_cast<nanoseconds>(end - start);
            Best_Duration += time_taken.count();

            // ------------------------WORST CASE [0(n^2)]----------------------------
            // This will Reverse the Sorted Array, Therfore we will Get the Worst Case

            reverse(arr.begin(), arr.end());
            // sort(arr.begin(), arr.end(), greater<ll>());
            start = high_resolution_clock::now();
            // Function Here
            selection_sort(arr);
            // Function Ends here
            end = high_resolution_clock::now();
            time_taken = duration_cast<nanoseconds>(end - start);
            Worst_Duration += time_taken.count();
        }

        cout << "---------------------------------------------------------" << endl;
        cout << inp_file << endl;
        cout << "AVERAGE CASE : ";
        double avg = (double)Average_Duration / (double)each_file_runs;
        avg *= 1e-9;
        cout << fixed << avg << setprecision(9);
        cout << " seconds" << endl;
        cout << "BEST CASE    : ";
        double best = (double)Best_Duration / (double)each_file_runs;
        best *= 1e-9;
        cout << fixed << best << setprecision(9);
        cout << " seconds" << endl;
        cout << "WORST CASE   : ";
        double worst = (double)Worst_Duration / (double)each_file_runs;
        worst *= 1e-9;
        cout << fixed << worst << setprecision(9);
        cout << " seconds" << endl;
    }

    return 0;
}
```

1.5. (L) Measure the best-case time and worst-case time of linear search for all ten files. Plot a graph.

LINEAR SEARCH ALGORITHM

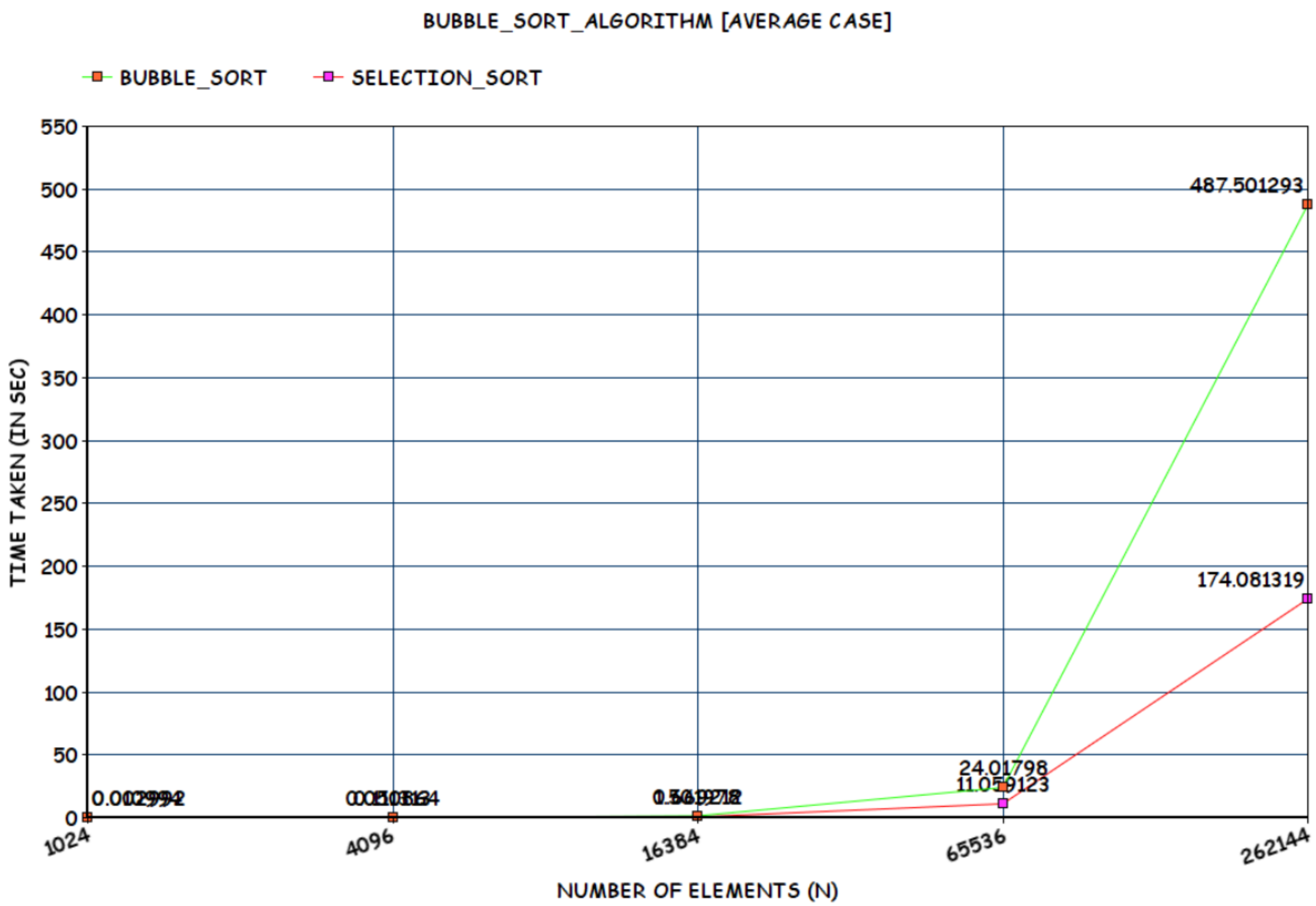
FILE	No. Of Elements(n)	BEST CASE [in sec]	WORST CASE [in sec]
1	1024 = 2^10	0.000000000	0.000000000
2	4096 = 2^12	0.000000000	0.000000000
3	16384 = 2^14	0.000000000	0.000000000
4	65536 = 2^16	0.000000000	0.000000000
5	262144 = 2^18	0.000000000	0.004037000
6	1048576 = 2^20	0.000000000	0.008000000
7	2097152 = 2^21	0.000000000	0.020000000
8	4194304 = 2^22	0.012000500	0.036005000
9	8388608 = 2^23	0.023998500	0.071998000
10	16777216 = 2^24	0.061231000	0.131999000



1.6. (L) Measure the average-case time (considering current data of ten files) of bubble sort and selection sort for all ten files. Plot a graph.

AVERAGE CASE

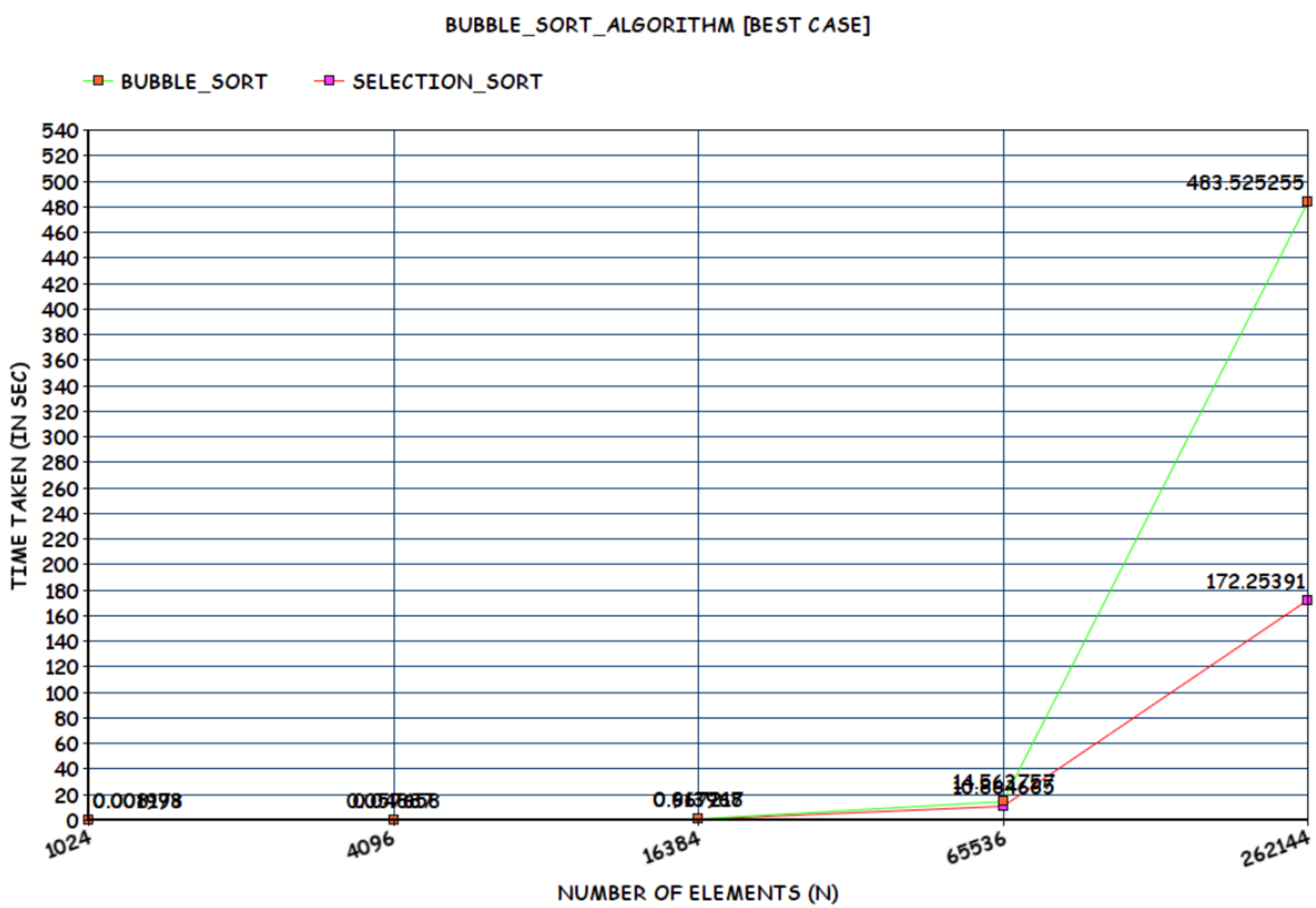
FILE	No. Of Elements(n)	BUBBLE SORT [in sec]	SELECTION SORT [in sec]
1	1024 = 2^10	0.01099400	0.002992000
2	4096 = 2^12	0.111313000	0.050864000
3	16384 = 2^14	1.501978000	0.669212000
4	65536 = 2^16	24.017980000	11.059123000
5	262144 = 2^18	487.501293000	174.081319000



1.7. (L) Measure the best-case time of bubble sort and selection sort for all ten files. Plot a graph.

BEST CASE

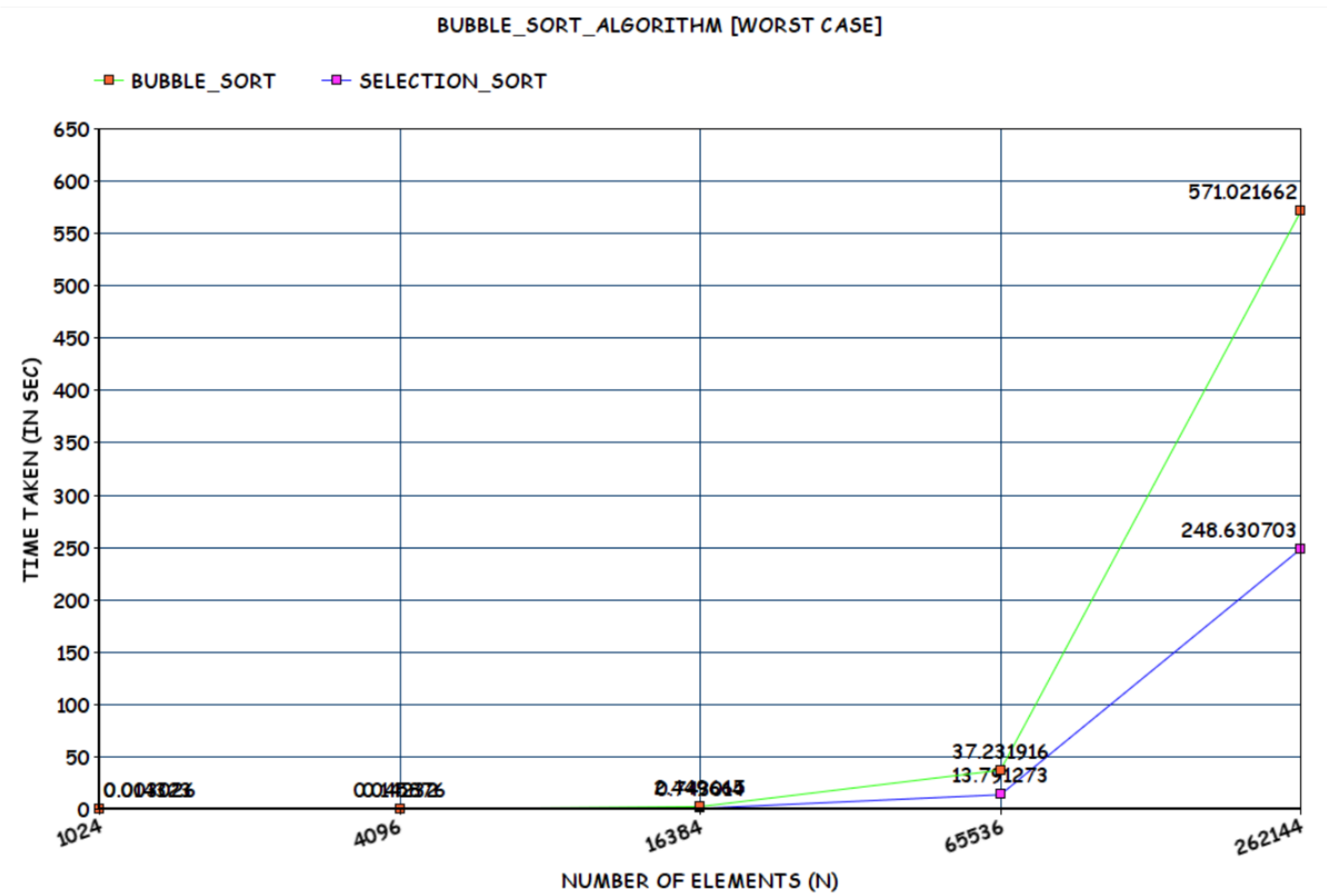
FILE	No. Of Elements(n)	BUBBLE SORT [in sec]	SELECTION SORT [in sec]
1	1024 = 2^10	0.008177500	0.001993000
2	4096 = 2^12	0.057657500	0.048870000
3	16384 = 2^14	0.913967500	0.667217000
4	65536 = 2^16	14.563756500	10.864665000
5	262144 = 2^18	483.525254500	172.253910000



1.8. (L) Measure the worst-case time of bubble sort and selection sort for all ten files. Plot a graph.

WORST CASE

FILE	No. Of Elements(n)	BUBBLE SORT [in sec]	SELECTION SORT [in sec]
1	1024 = 2^10	0.014323500	0.003026000
2	4096 = 2^12	0.142320000	0.045876000
3	16384 = 2^14	2.449665000	0.743014000
4	65536 = 2^16	37.231916500	13.791273000
5	262144 = 2^18	571.021661500	248.630703000



1.9. (T) Assume that you don't know the time complexity of above algorithms.

1.9.1. Can you predict the same based on your implementation of above algorithms?
Definitely Yes.

Since 1 sec takes 10^8 Operations [Approximation]

X sec takes '?' Operations

So From Time Taken we can get the Number of Operations it performs.

Eg:

No of Operations [in File 5 Worst Case] = 571.0216615 * (10^8) = 57102166150

= [Approximately Equal to 68719476736 = 2^36 = N^2]

= O(N^2)

Therefore, Time Complexity for **Worst Case Bubble Sort** [Prediction] = **O(N^2)**

1.9.2. Do they match with theoretical time complexity? **Yes**~~/No.~~

1.9.3. If yes, then write the time complexity of each algorithm. If no, then write the difference.

Time Complexity of Linear Search

BEST CASE = If First Element Checked is key

Running Time is Constant

WORST CASE = If Element is Not there in Array

Running Time is Linear Function of N [Since it has to Check All Elements]

AVERAGE CASE = O(N/2) [Approximately]

Instead of Input of Particular Type [Sorted or Reverse Sorted]

, All the Inputs of Given Sizes are Equally Probable

Time Complexity of Bubble Sort [Blindly All Possible Pairs]

BEST CASE = If the Array is Already Sorted = O(N^2)

Running Time is Quadratic Function of N

WORST CASE = If the Array is Reverse Sorted = O(N^2)

Running Time is Quadratic Function of N

AVERAGE CASE = O(N^2) [Approximately]

Instead of Input of Particular Type [Sorted or Reverse Sorted]

, All the Inputs of Given Sizes are Equally Probable

Time Complexity of Selection Sort [Save Some Time, depending upon values]

BEST CASE = If the Array is Already Sorted = O(N^2)

Running Time is Quadratic Function of N

WORST CASE = If the Array is Reverse Sorted = O(N^2)

Running Time is Quadratic Function of N

AVERAGE CASE = O(N^2) [Approximately]

Instead of Input of Particular Type [Sorted or Reverse Sorted]

, All the Inputs of Given Sizes are Equally Probable

If First Half , We can assume that half the elements are greater than A[j] while half are less.

On the average, thus tj=j/2. [In RAM Model]

Plugging this value into T(n) [RAM Model Equation] still leaves it Quadratic.

Thus, in this case Average case is Equivalent to Worst Case Time Complexity.

Remark : Since the Input is Random, Average Case may Tilt Towards Best Case as well.

BEST CASE [THEORATICAL CALCULATION]

FILE	NUMBER OF ELEMENTS	NO OF OPERATIONS [CASE] = O(N)	APPROX TIME TAKEN [OP/10^8]
FILE 1	1024 = 2^10	1024	0.00001024
FILE 2	4096 = 2^12	4096	0.00004096
FILE 3	16384 = 2^14	16384	0.00016384
FILE 4	65536 = 2^16	65536	0.00065536
FILE 5	262144 = 2^18	262144	0.00262144
FILE 6	1048576 = 2^20	1048576	0.01048576
FILE 7	2097152 = 2^21	2097152	0.02097152
FILE 8	4194304 = 2^22	4194304	0.04194304
FILE 9	8388608 = 2^23	8388608	0.08388608
FILE 10	16777216 = 2^24	16777216	0.16777216

WORST/AVERAGE CASE [THEORATICAL CALCULATION]

FILE	NUMBER OF ELEMENTS	NO OF OPERATIONS [CASE] = O(N^2)	APPROX TIME TAKEN [OP/10^8]
FILE 1	1024 = 2^10	2^20	0.0104 seconds = 0.01 sec
FILE 2	4096 = 2^12	2^24	0.167 seconds = 0.16 sec
FILE 3	16384 = 2^14	2^28	2.684 seconds = 2.6 sec
FILE 4	65536 = 2^16	2^32	43 seconds = 43 sec
FILE 5	262144 = 2^18	2^36	687 seconds = 11 mins
FILE 6	1048576 = 2^20	2^40	10995 seconds = 3 hrs 3 mins
FILE 7	2097152 = 2^21	2^42	43980 seconds = 12 hrs 13 mins
FILE 8	4194304 = 2^22	2^44	175922 seconds = 2 days 52 hrs 2 mins
FILE 9	8388608 = 2^23	2^46	703687 seconds = 8 days 3 hrs 28 mins
FILE 10	16777216 = 2^24	2^48	2814750 seconds = 32 days 13 hrs 52 mins

CONCLUSION:

1.) Linear Search is Brute Force Searching Algorithm Which Checks for given KEY by iterating all Elements in Array O(N)

2.) Bubble Sort is Easy to Implement, Stable and In-Place Algorithm and Space Requirement is Minimum

But The Process is Blindly Considering all Possible Pairs O(N^2) [Expensive]

3.) Selection Sort Performs Well on Small Lists and Good In-Place Algorithm.

SUBMITTED BY:

U19CS012

BHAGYA VINOD RANA