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

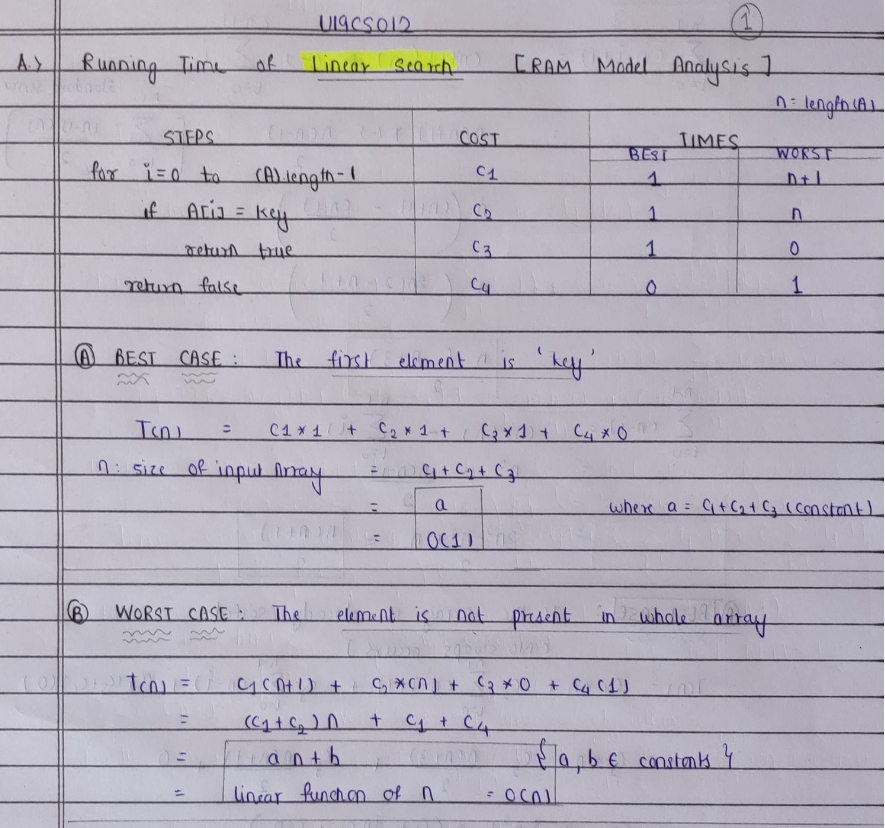
Input: A Sequence of Unsorted ‘n’ numbers, **a1,a2,…,an**

Output: A Permutation (Reordering) (**a1’,a2’,…,an’**) of Input Sequence

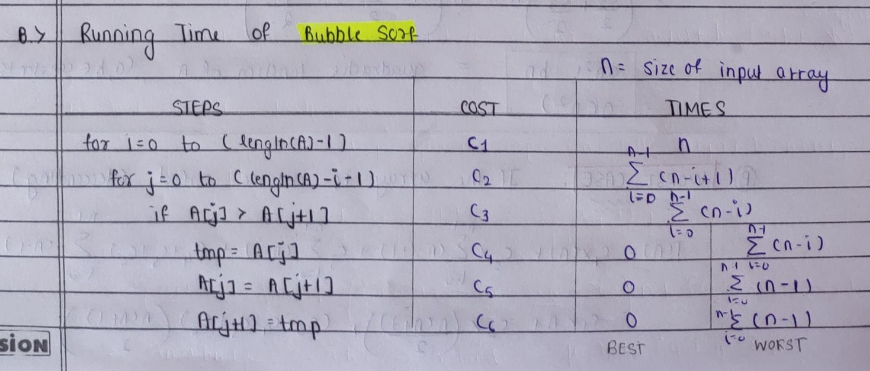
such that **a1’≤ a2’≤ … ≤ an’**

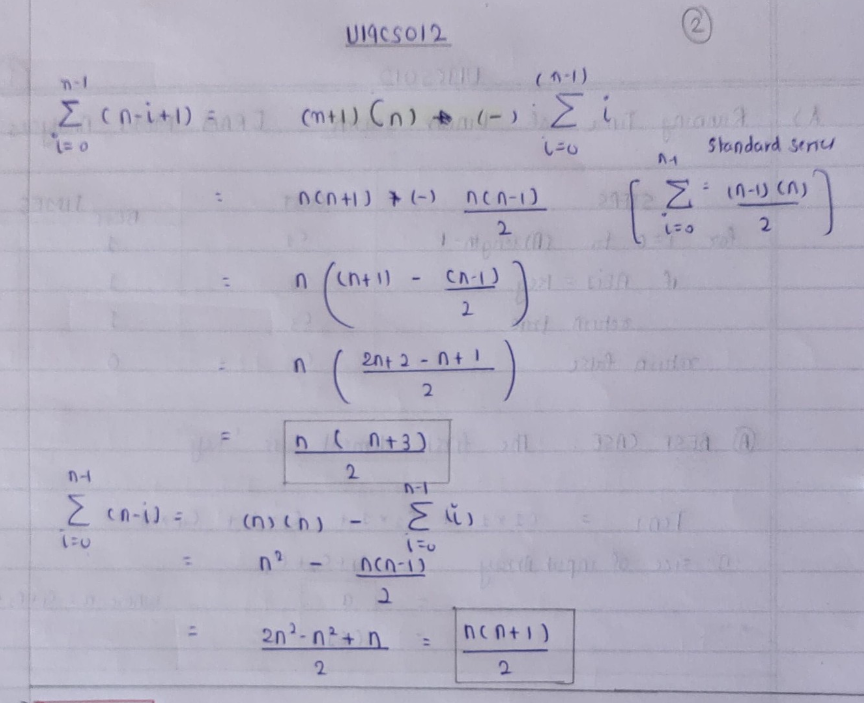
1.1. (T) Analyze the time complexity of above algorithms using the RAM model

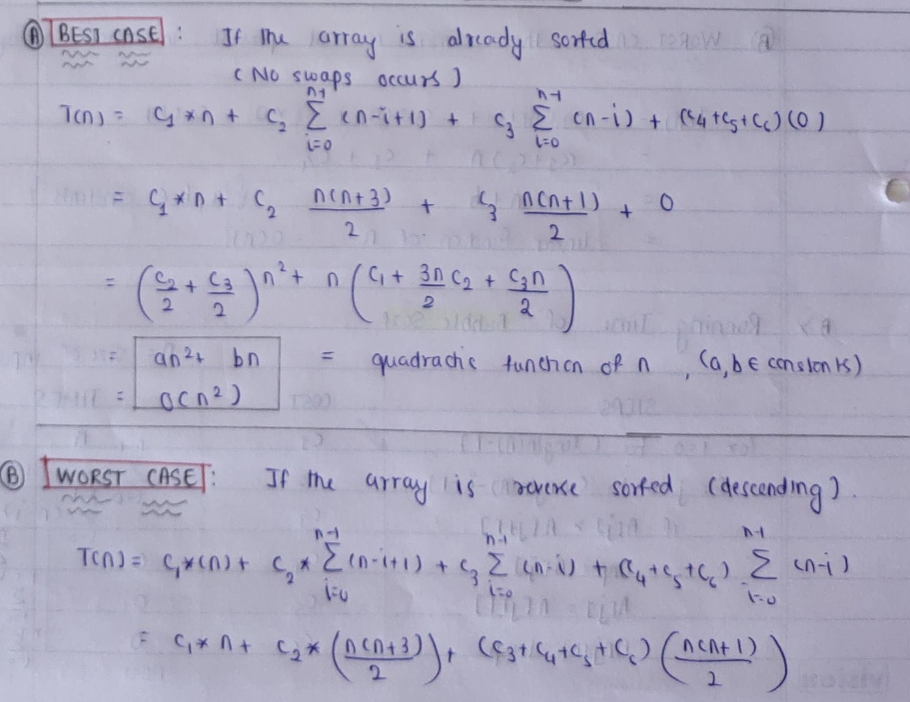
• Linear Search

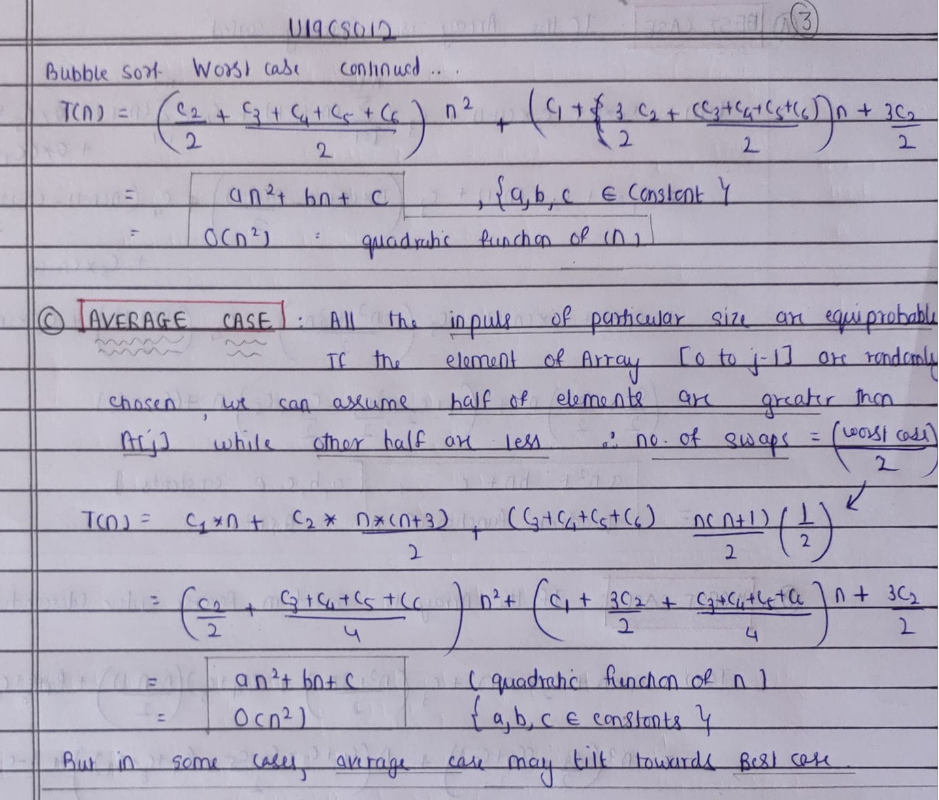


• Bubble Sort

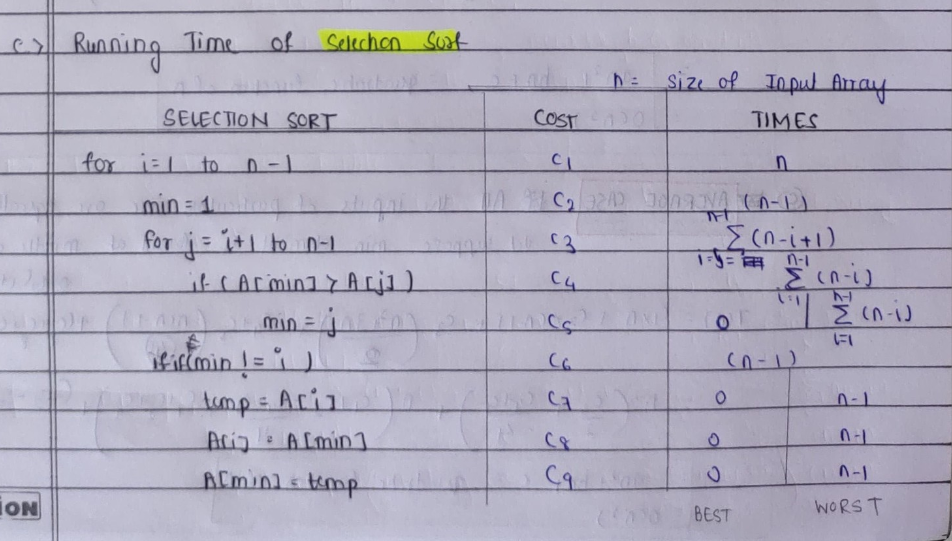


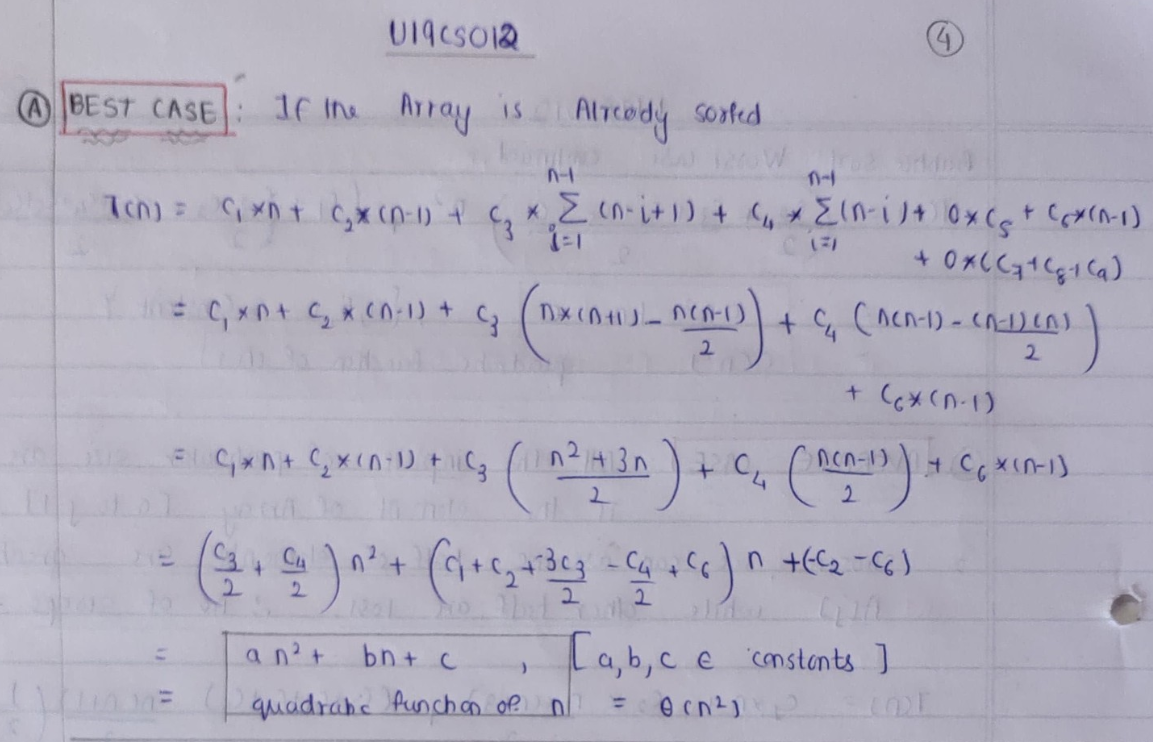


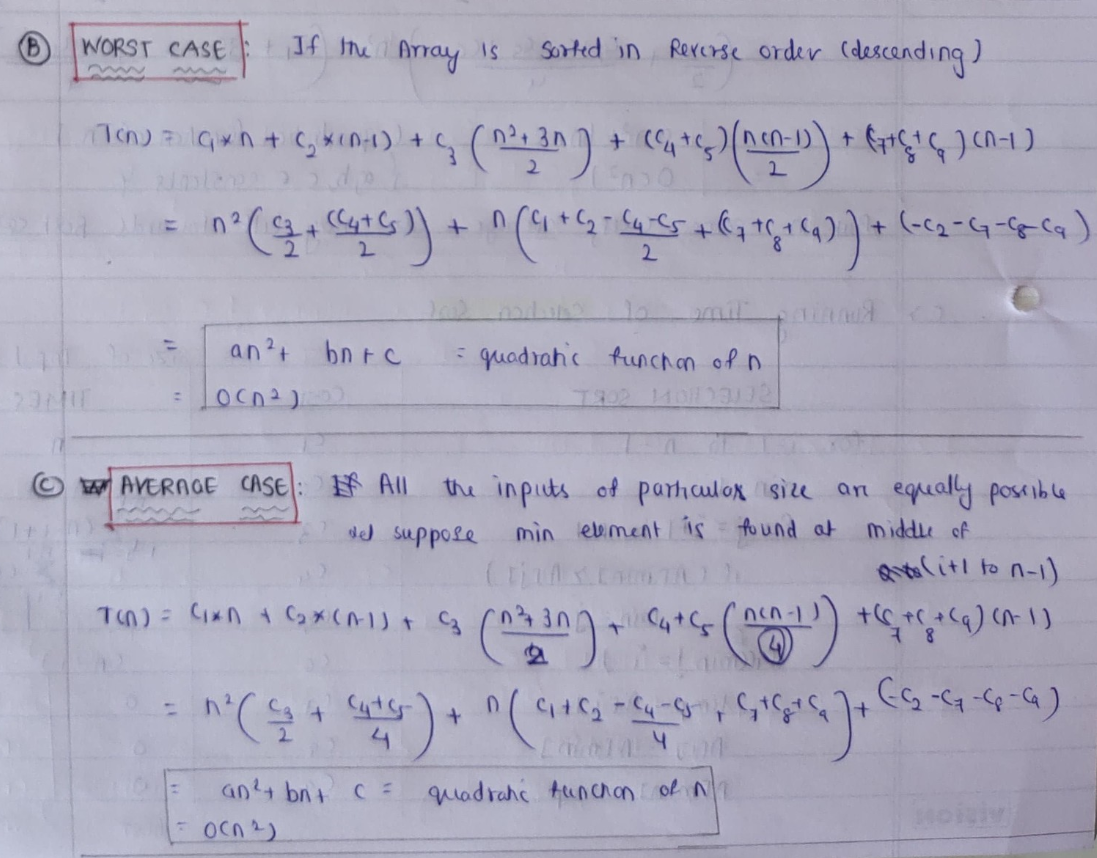




• Selection Sort



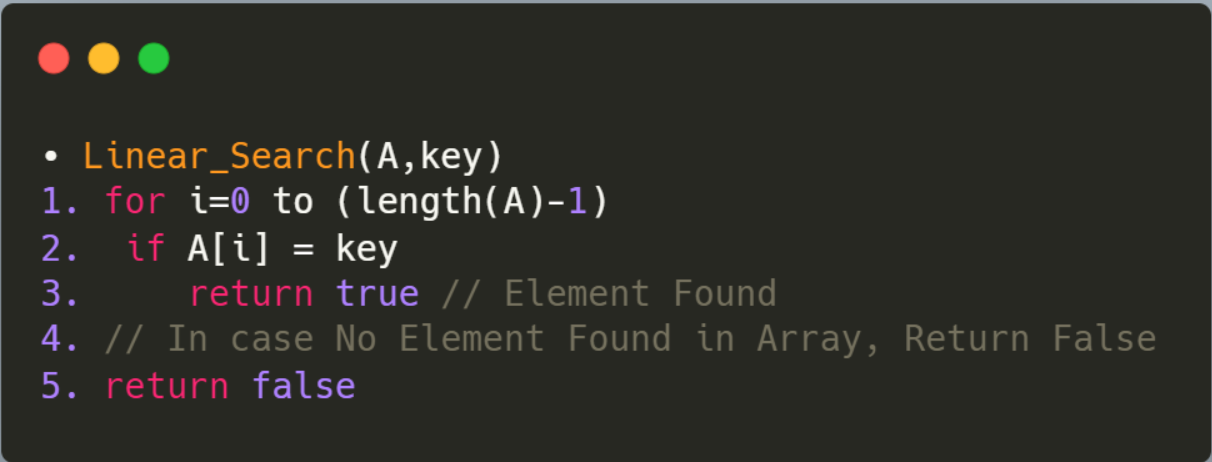




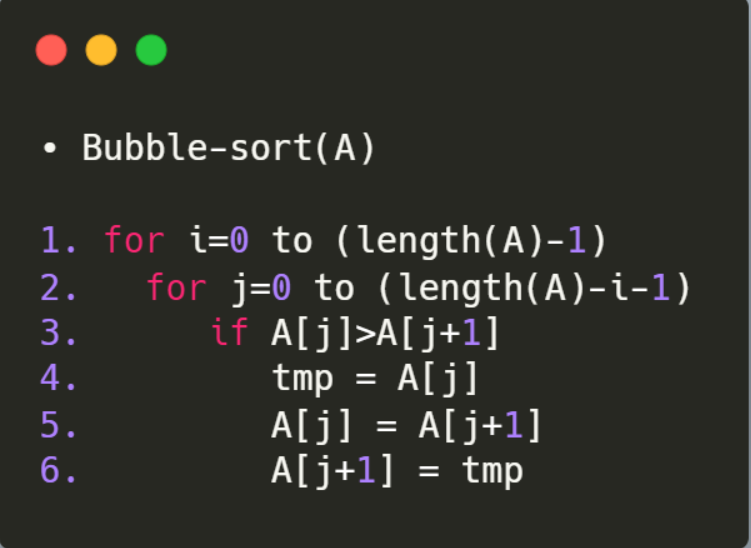
1.2. (L) Implement the above algorithms using the programming language of your

choice.

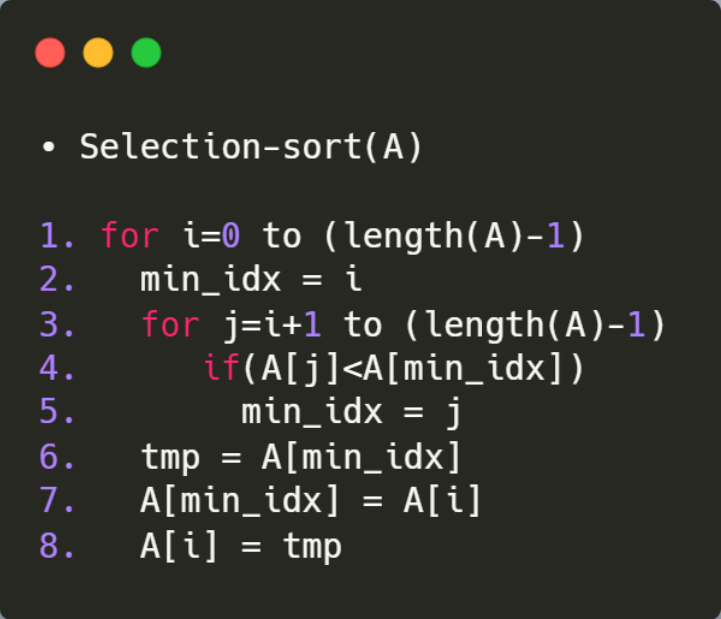
• Linear Search



• Bubble Sort



• Selection Sort



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;

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();

        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 [0(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 [0(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;

            }

        }

    }

}

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());*

            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 [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*

            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);

*// 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();

            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;

    }

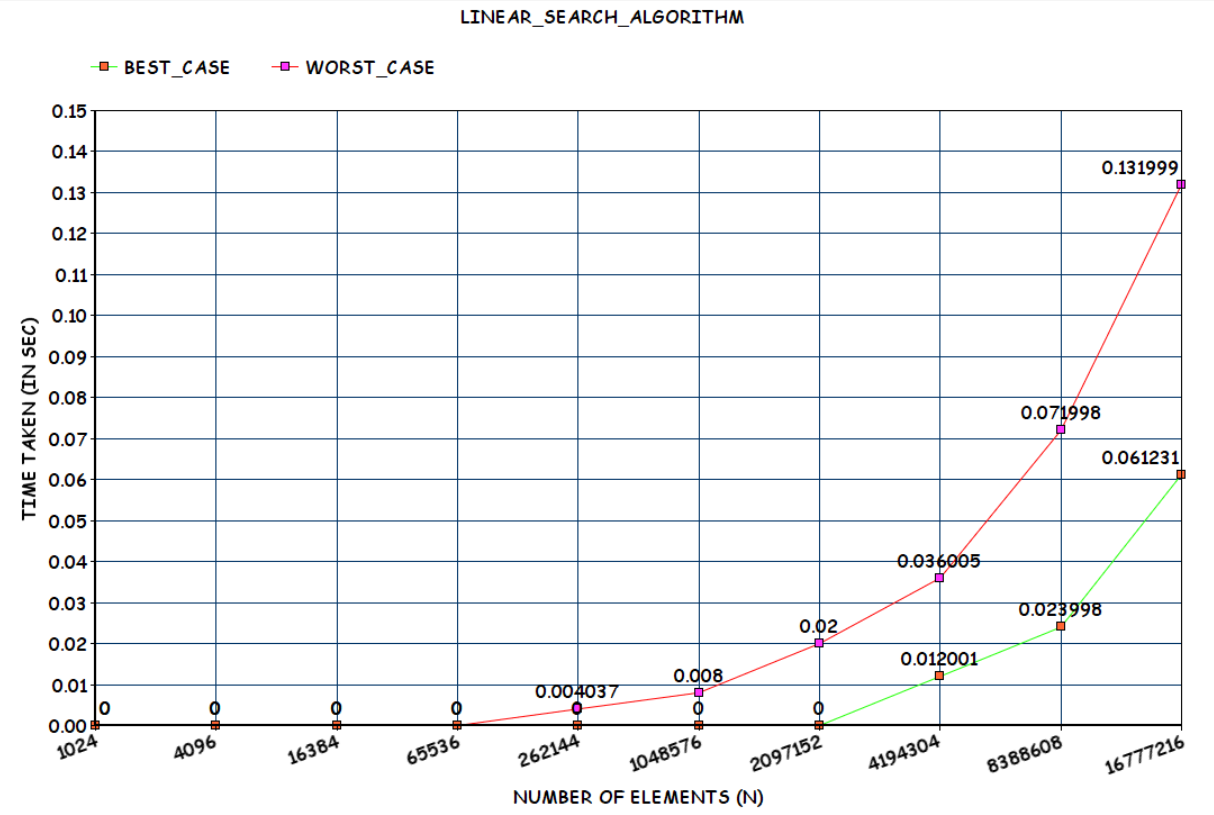
*return* 0;

}

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

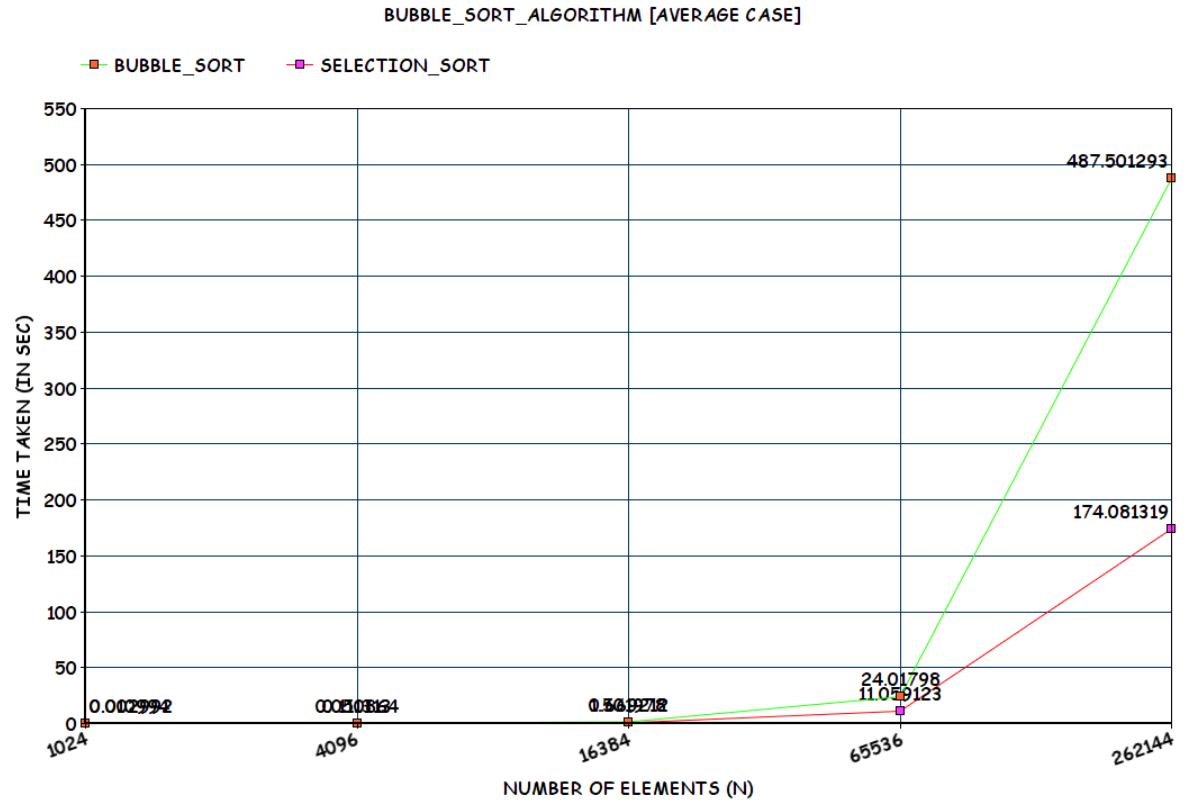
|  |  |  |  |
| --- | --- | --- | --- |
| FILE | No. Of Elements(n) | BEST CASE [in sec] | WORST CASE [in sec] |
| 1 | 1024 = 2^10 | 0.000000000 | 0.000000000 |
| 2 | 4096 = 2^12 | 0.000000000 | 0.000000000 |
| 3 | 16384 = 2^14 | 0.000000000 | 0.000000000 |
| 4 | 65536 = 2^16 | 0.000000000 | 0.000000000 |
| 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

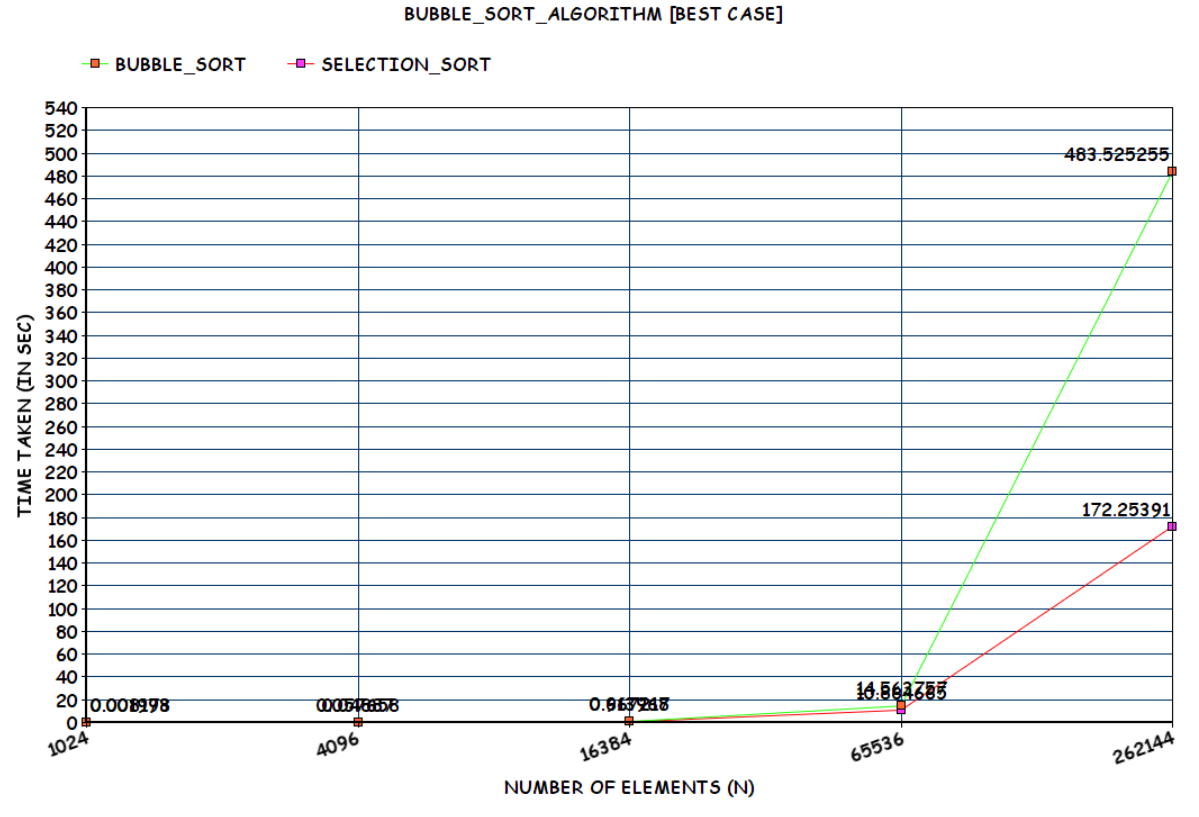
|  |  |  |  |
| --- | --- | --- | --- |
| FILE | No. Of Elements(n) | BUBBLE SORT [in sec] | SELECTION SORT  [in sec] |
| 1 | 1024 = 2^10 | 0.01099400 | 0.002992000 |
| 2 | 4096 = 2^12 | 0.111313000 | 0.050864000 |
| 3 | 16384 = 2^14 | 1.501978000 | 0.669212000 |
| 4 | 65536 = 2^16 | 24.017980000 | 11.059123000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

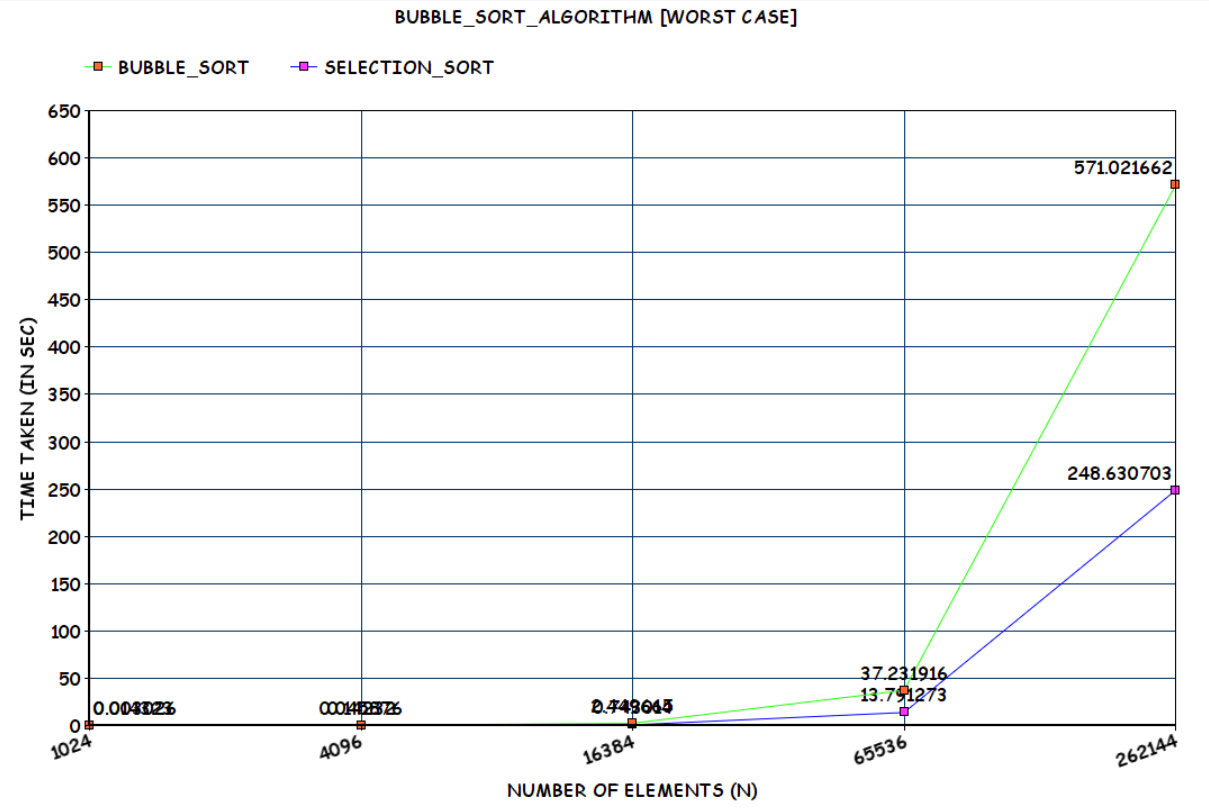
|  |  |  |  |
| --- | --- | --- | --- |
| FILE | No. Of Elements(n) | BUBBLE SORT [in sec] | SELECTION SORT  [in sec] |
| 1 | 1024 = 2^10 | 0.008177500 | 0.001993000 |
| 2 | 4096 = 2^12 | 0.057657500 | 0.048870000 |
| 3 | 16384 = 2^14 | 0.913967500 | 0.667217000 |
| 4 | 65536 = 2^16 | 14.563756500 |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