Design and Analysis of Algorithms (CS206)

Assignment - 2

**U19CS012**

1. Given the following algorithms, answer the questions.

• Insertion 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

• Lets **Analyze** Insertion Sort

• The time taken to sort depends on the fact that we are sorting how many numbers

• Also, the time to sort may change depending upon whether the *array is*

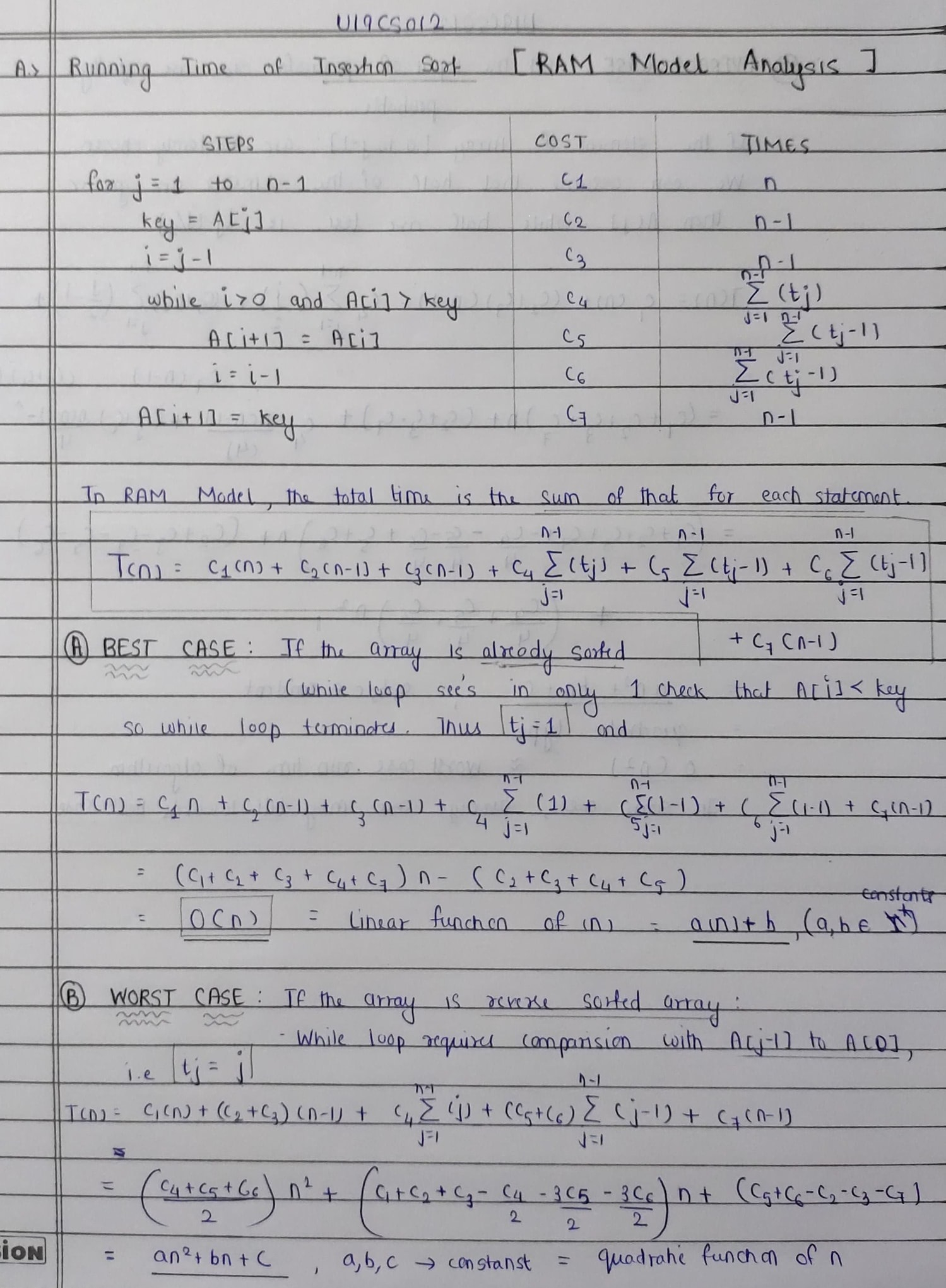
*almost sorted* (can you see if the array was sorted we had very little job).

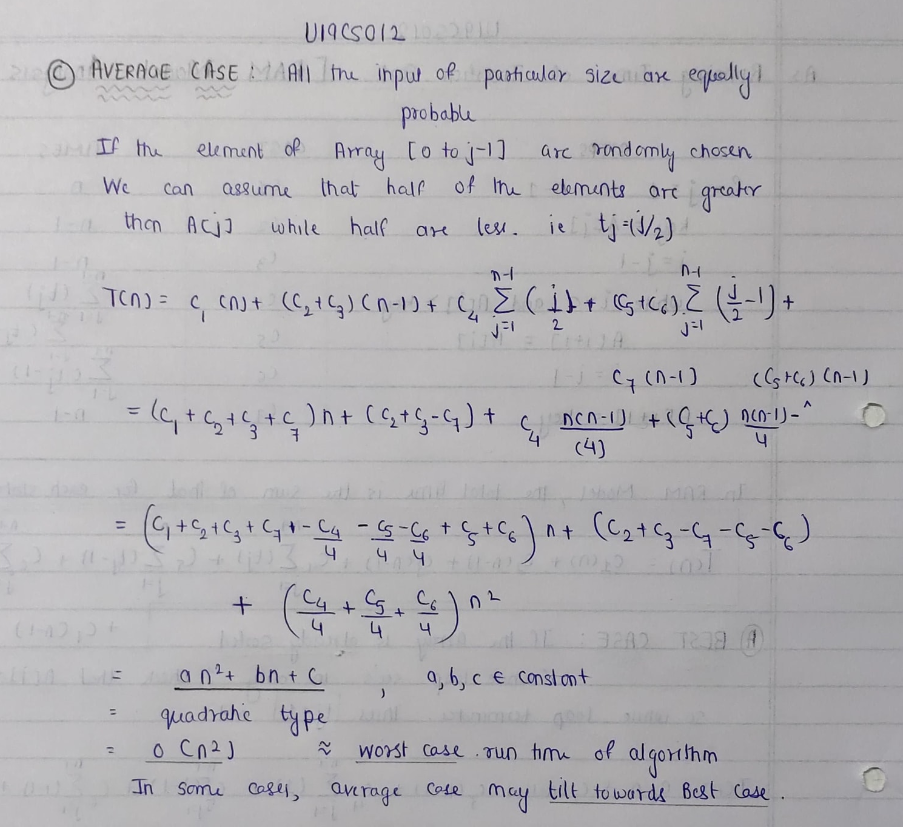
• So, we need to define the meaning of the **input size** and **running time**.

In Sorting Problem,

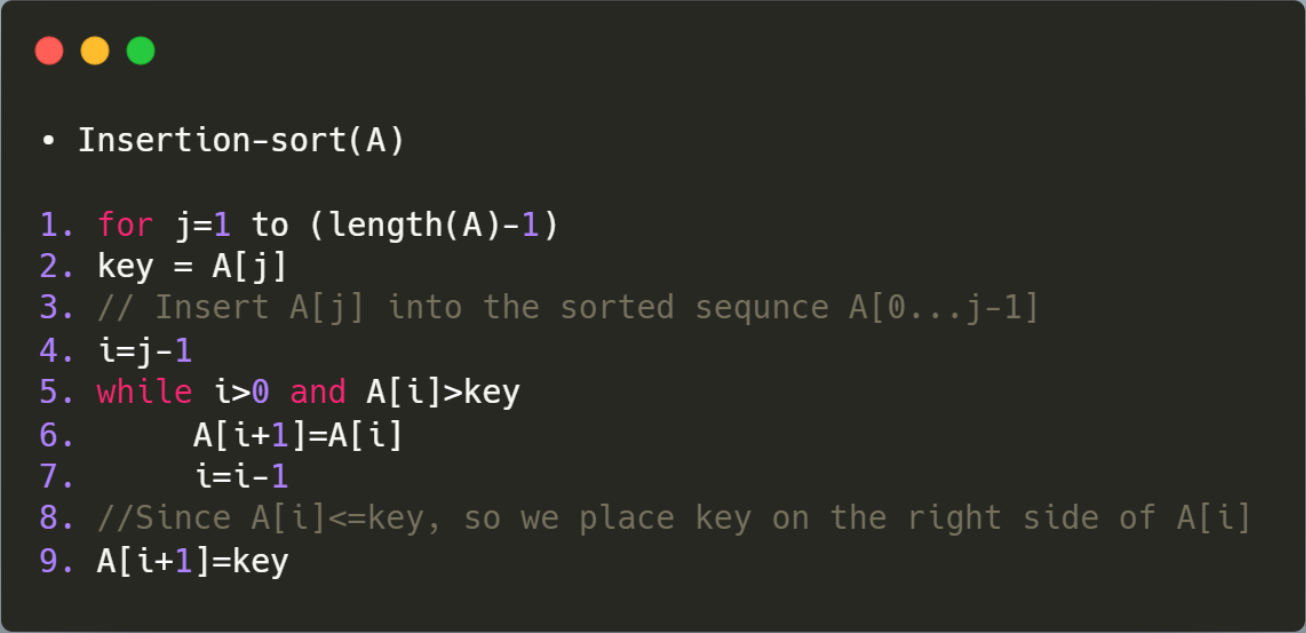
**Input Size** = Number of Integers we are Sorting

**Running Time** = Proportional to the Number of Operations Performed





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



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).

*// 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 Insertion Sort*

void insertion\_sort(vll &arr)

{

    ll sz = arr.size(), key, i, j;

*for* (j = 1; j < sz; j++)

    {

        key = arr[j];

*// Insert arr[j] into sorted sequence A[0...j-1]*

        i = j - 1;

*while* (i >= 0 && arr[i] > key)

        {

            arr[i + 1] = arr[i];

            i = i - 1;

        }

*// Since A[i]<=key, so we place key on right side of arr[i]*

        arr[i + 1] = key;

    }

*return*;

}

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*

            insertion\_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*

            insertion\_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*

            insertion\_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;

}

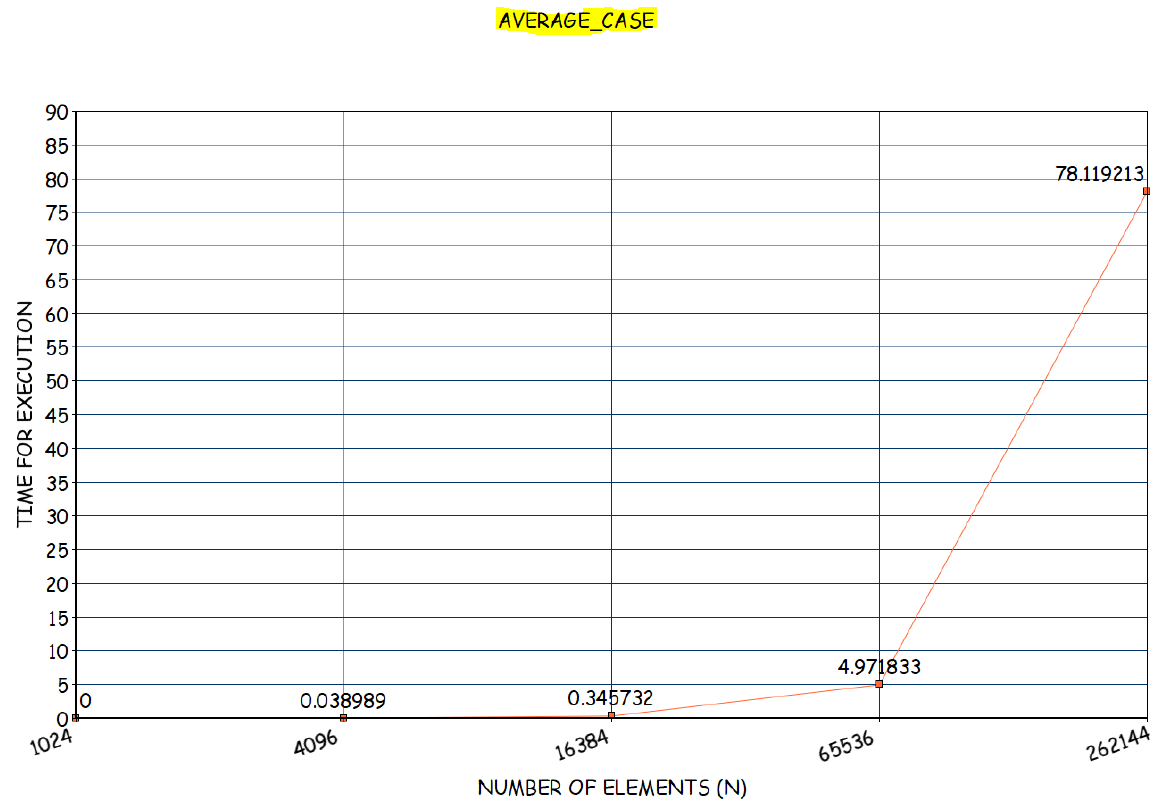
1.5. (L) Measure the average-case time (considering current data of ten files) of insertion sort for all ten files. Plot a graph.

INSERTION SORT ALGORITHM

|  |  |  |
| --- | --- | --- |
| FILE | Number of Elements | AVERAGE CASE [in sec] |
| 1 | 1024 = 2^10 | 0.000000000 |
| 2 | 4096 = 2^12 | 0.038989000 |
| 3 | 16384 = 2^14 | 0.345732500 |
| 4 | 65536 = 2^16 | 4.971833000 |
| 5 | 262144 = 2^18 | 78.119213000 |

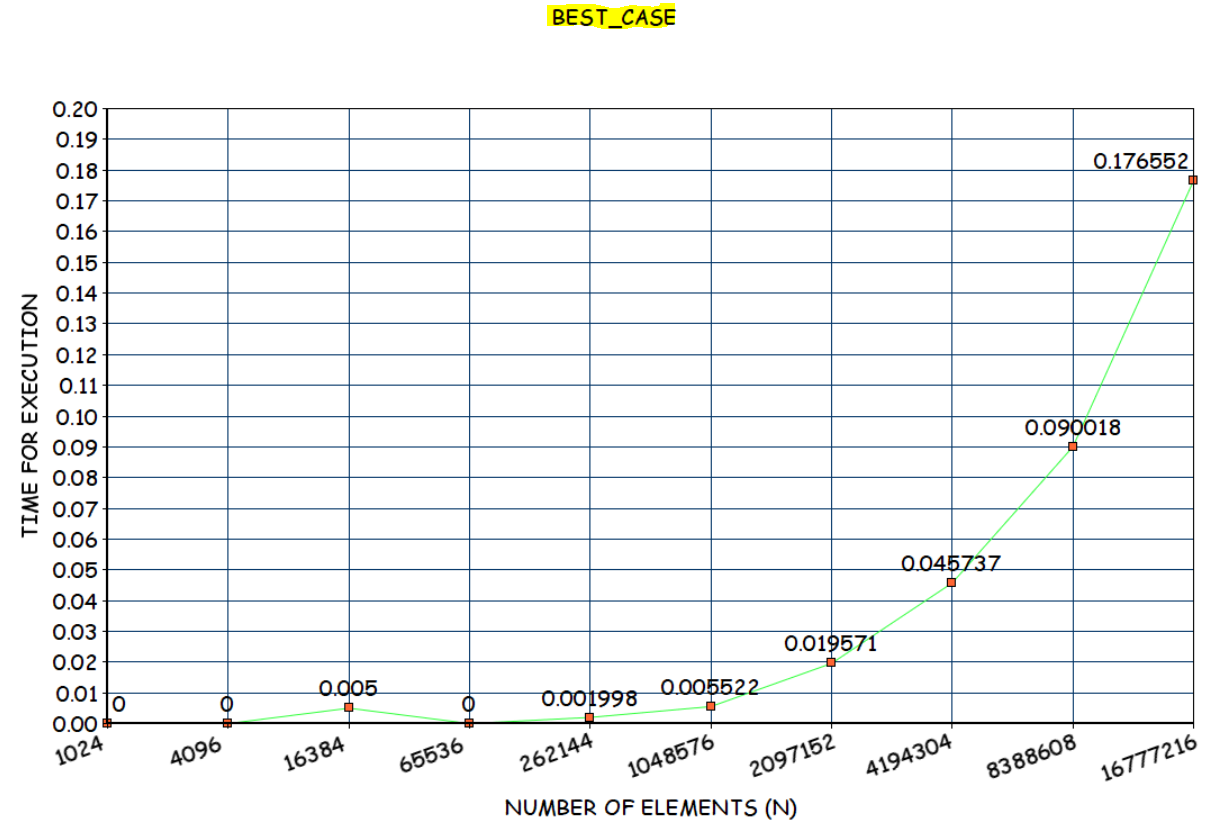
After File 5 Onwards, It would take a Minimum of 2 hrs for Each File Execution.

So Avoided Executing for Rest of the Files.



1.6. (L) Measure the best-case time of insertion sort for all ten files. Plot a graph

|  |  |  |
| --- | --- | --- |
| FILE | Number of Elements | BEST CASE [in sec] |
| 1 | 1024 = 2^10 | 0.000000000 |
| 2 | 4096 = 2^12 | 0.000000000 |
| 3 | 16384 = 2^14 | 0.005000000 |
| 4 | 65536 = 2^16 | 0.000000000 |
| 5 | 262144 = 2^18 | 0.001998500 |
| 6 | 1048576 = 2^20 | 0.005522500 |
| 7 | 2097152 = 2^21 | 0.019571000 |
| 8 | 4194304 = 2^22 | 0.045737000 |
| 9 | 8388608 = 2^23 | 0.090017500 |
| 10 | 16777216 = 2^24 | 0.176551500 |

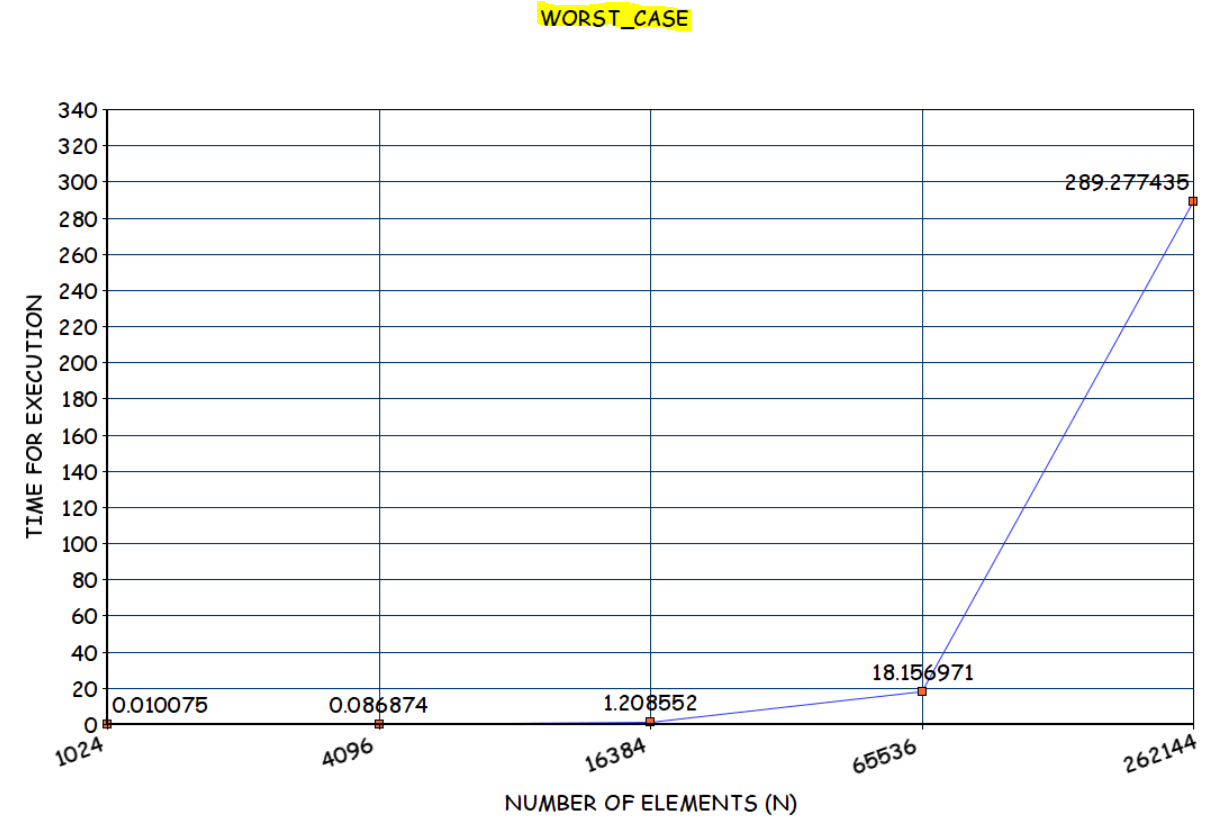


1.7. (L) Measure the worst-case time of insertion sort for all ten files. Plot a graph.

|  |  |  |
| --- | --- | --- |
| FILE | Number of Elements | WORST CASE [in sec] |
| 1 | 1024 = 2^10 | 0.010075500 |
| 2 | 4096 = 2^12 | 0.086874000 |
| 3 | 16384 = 2^14 | 1.208551500 |
| 4 | 65536 = 2^16 | 18.156971000 |
| 5 | 262144 = 2^18 | 289.277434500 |

After File 5 Onwards, It would take a Minimum of 3 hrs for Each File Execution.

So Avoided Executing for Rest of the Files.



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

1.8.1. Can you predict the same based on your implementation?

*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 10 Best Case] = 0.176551500 \* (10^8) = 17655150

= [Approximately Equal to 16777216 = 2^24 = N]

= O(N)

Therefore, Time Complexity for **Best Case** [Prediction] = **O(N)**

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

1.8.3. If yes, then write the time complexity of algorithm. If no, then write

the difference.

Time Complexity of Insertion Sort

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

Running Time is Linear 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.) Efficient for sorting small numbers*

*2.) In place sort: Takes an array A[0..n-1] (sequence of n elements) and arranges*

*them in place, so that it is sorted.*

3.) Maintains relative order of the input data in case of two equal values (stable)

& Algorithm is Also Adaptive.

**SUBMITTED BY:**

**U19CS012**

**BHAGYA VINOD RANA**