Assignment 1

**1. Given the following algorithms, answer the questions.**

**• Linear Search**

**• Bubble Sort**

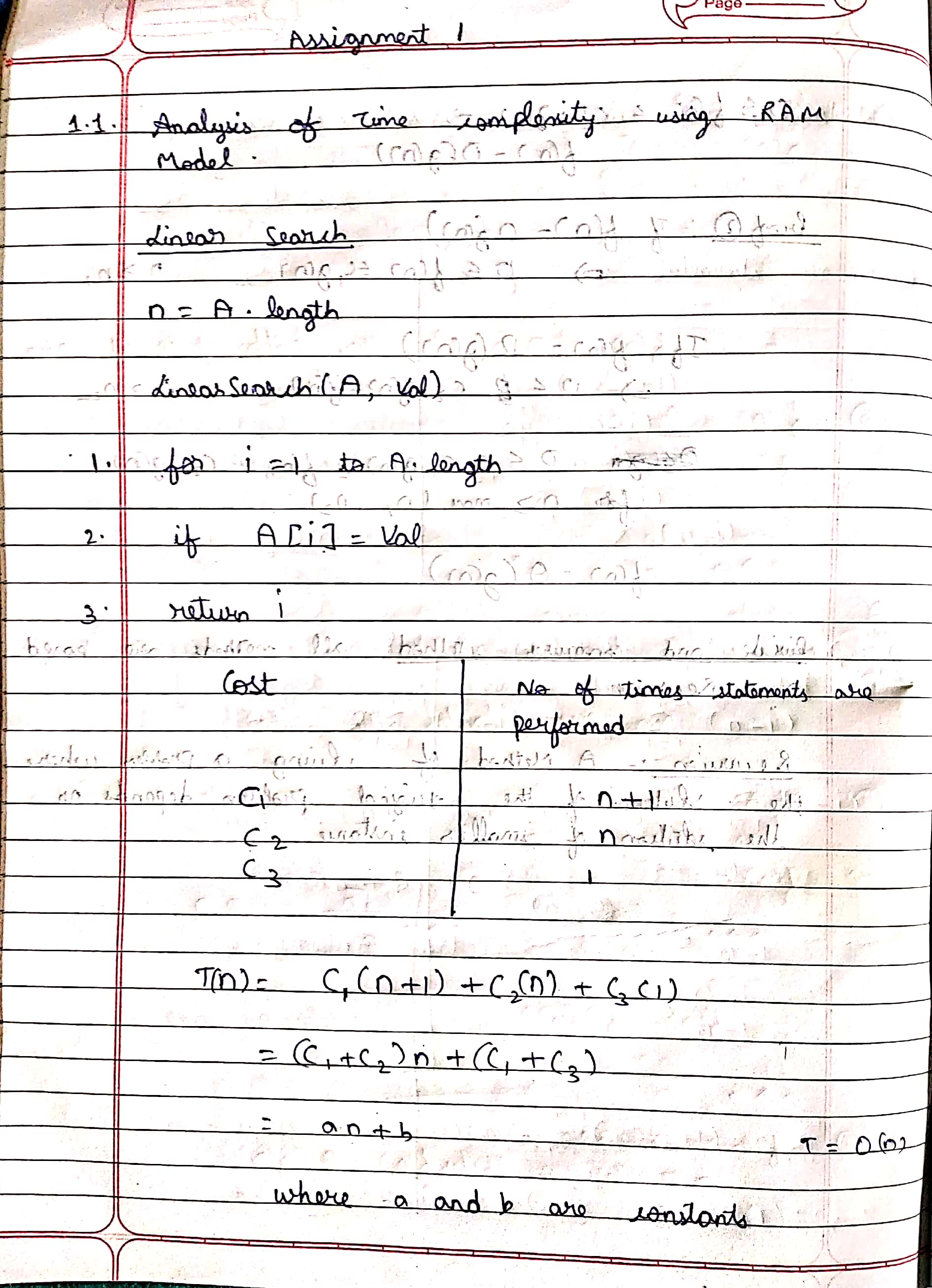
**• Selection Sort**

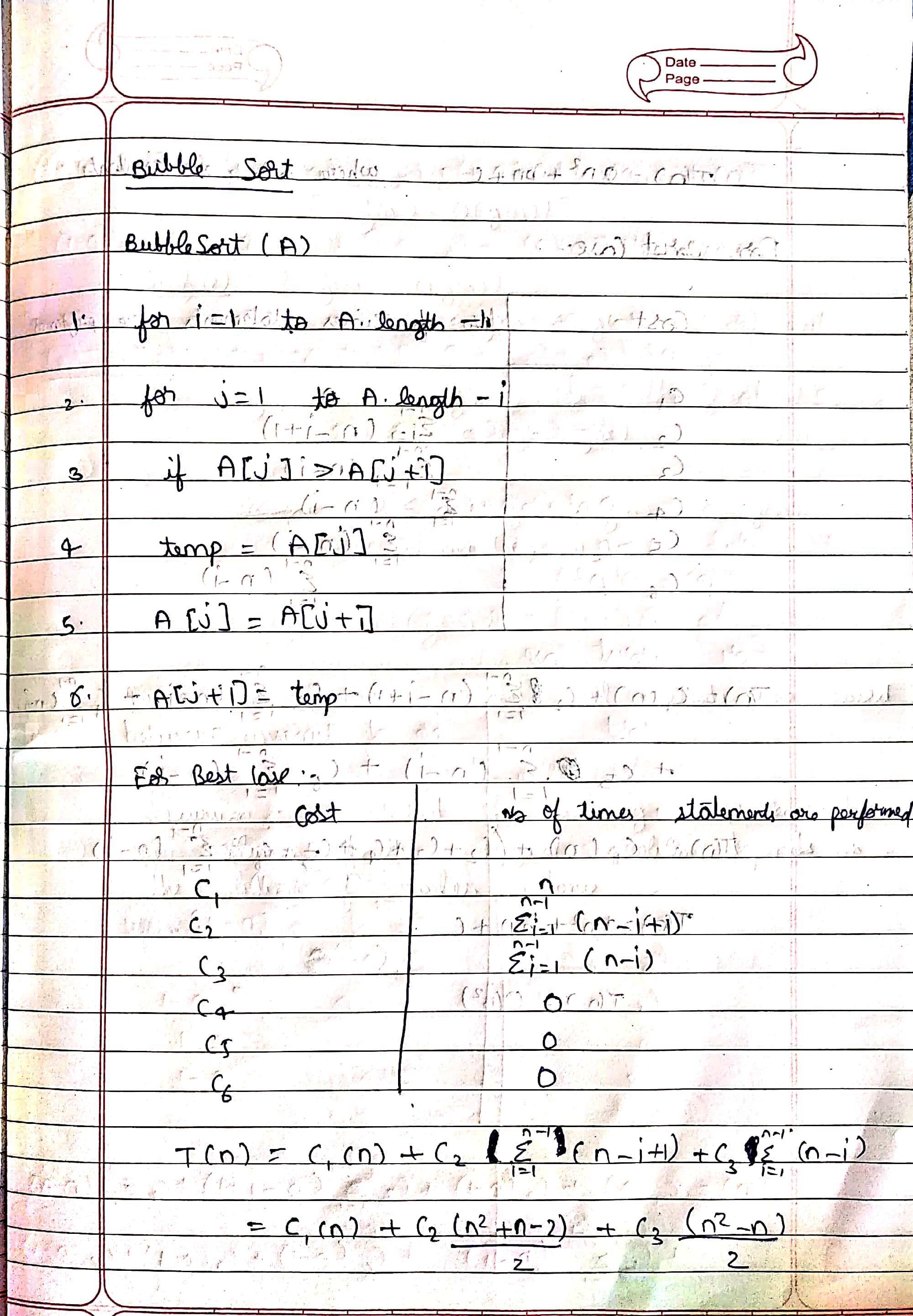
**• Insertion sort**

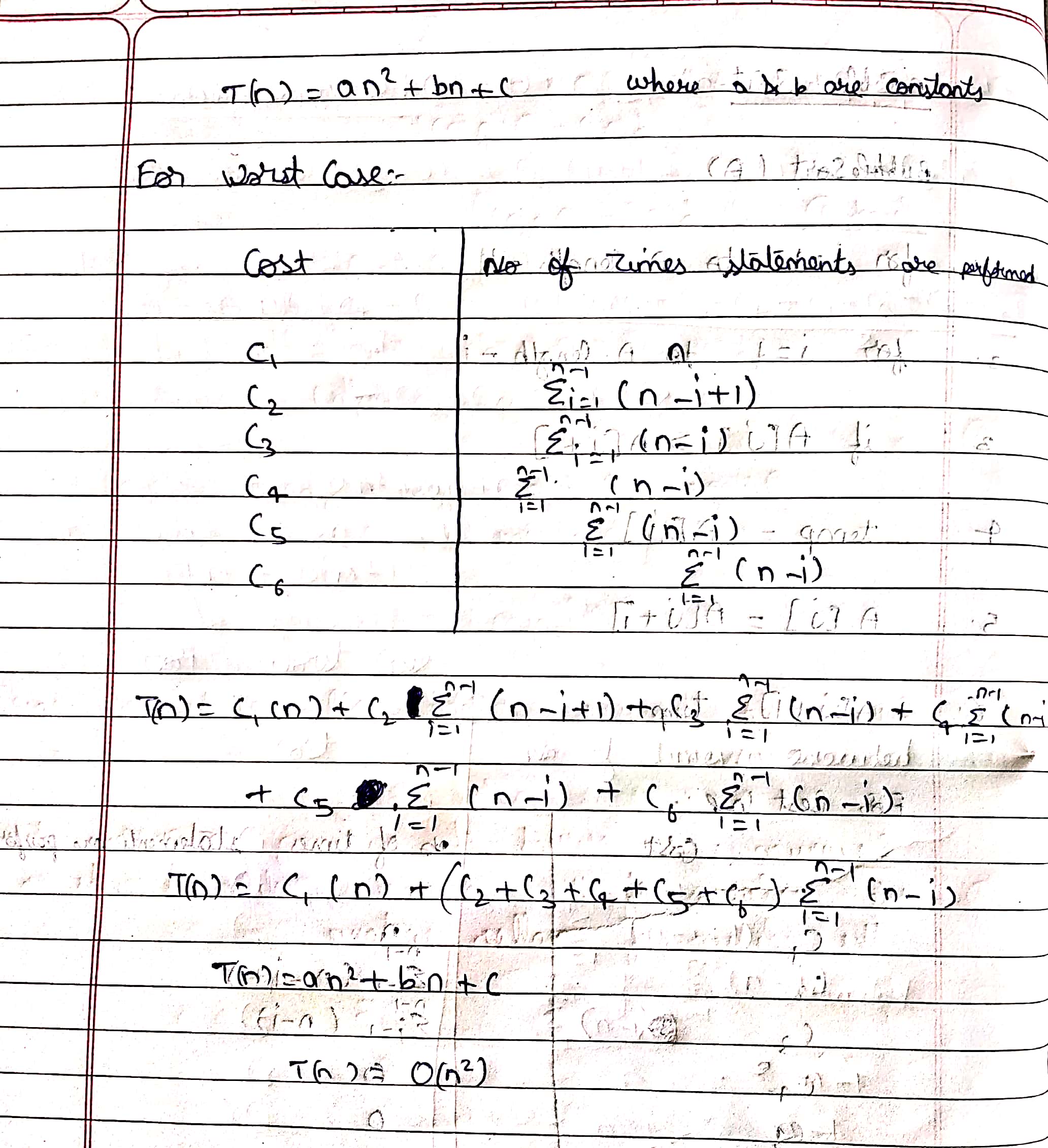
**1.1. Analyze the time complexity of above algorithms using the RAM model (Include**

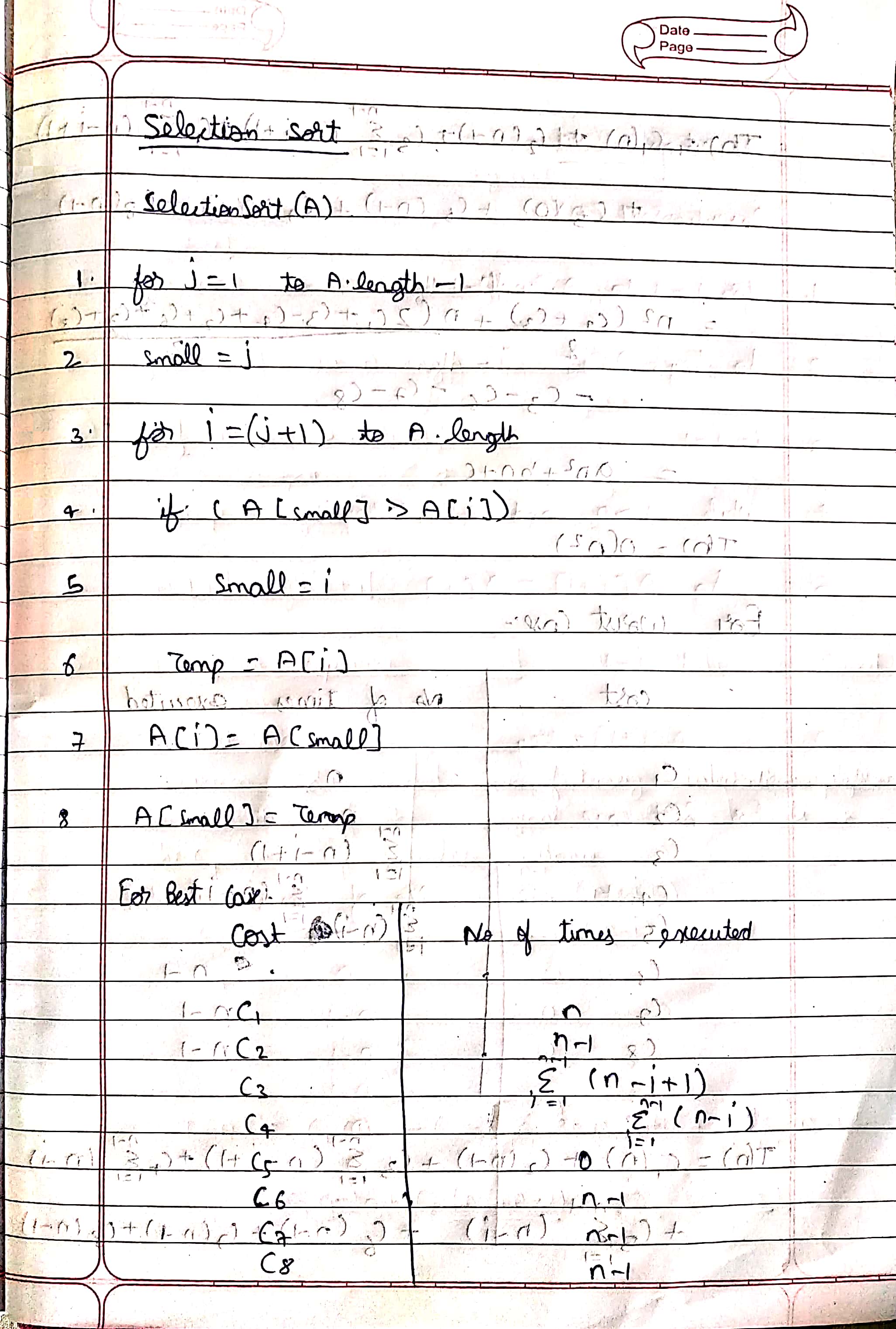
**the handwritten analysis of these algorithms as an image in the latex file. Make**

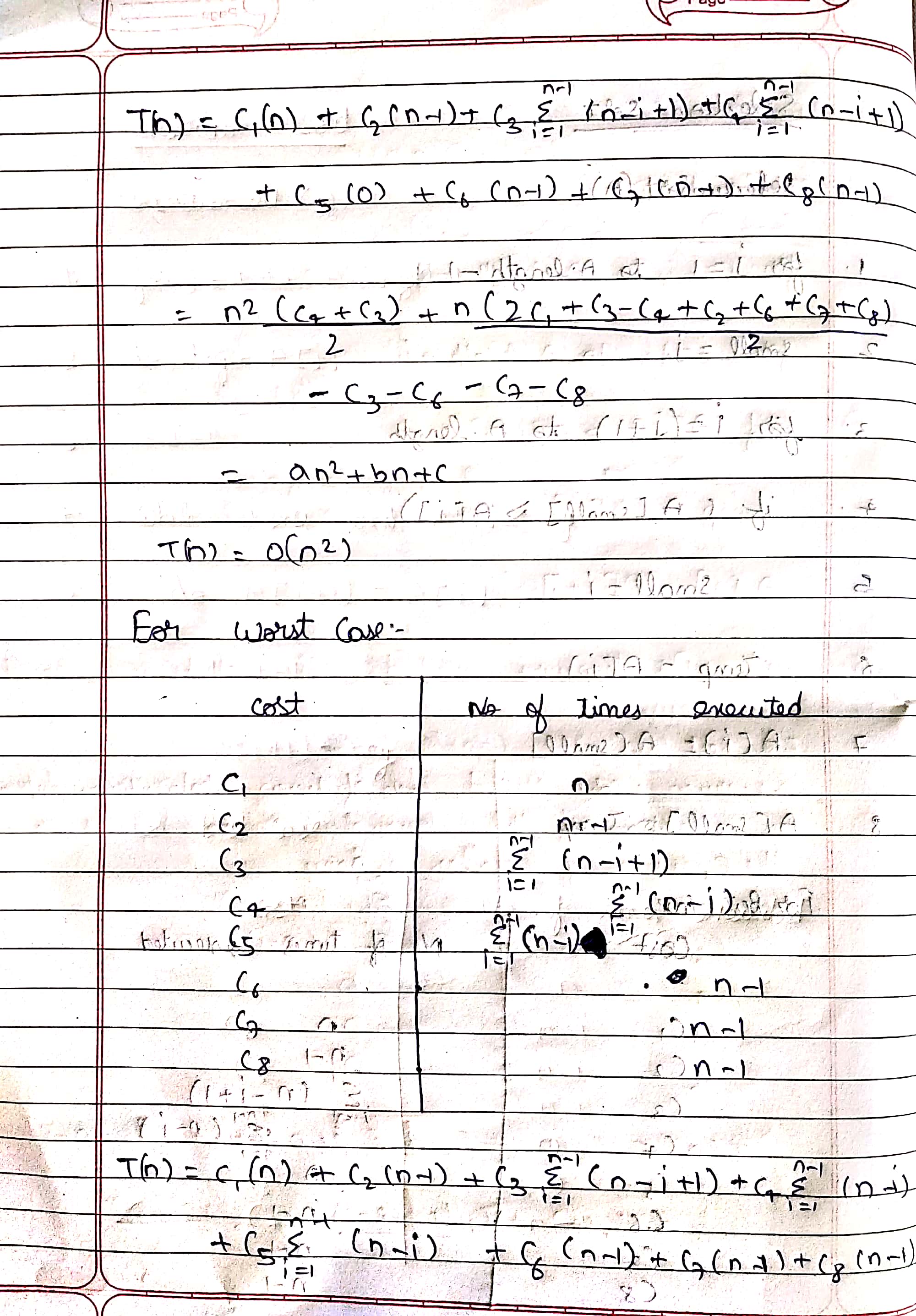
**sure the images/contents are readable.).**

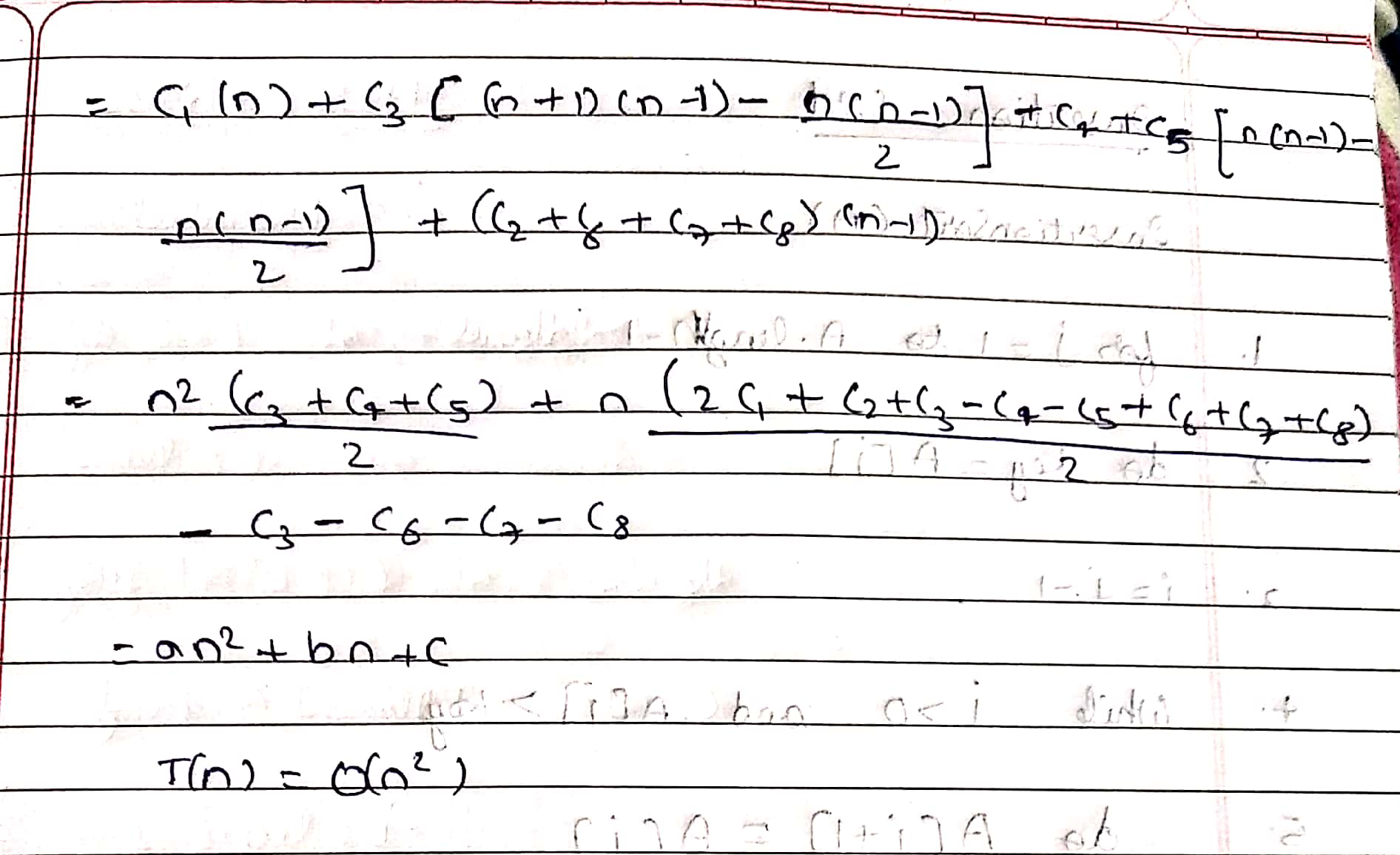


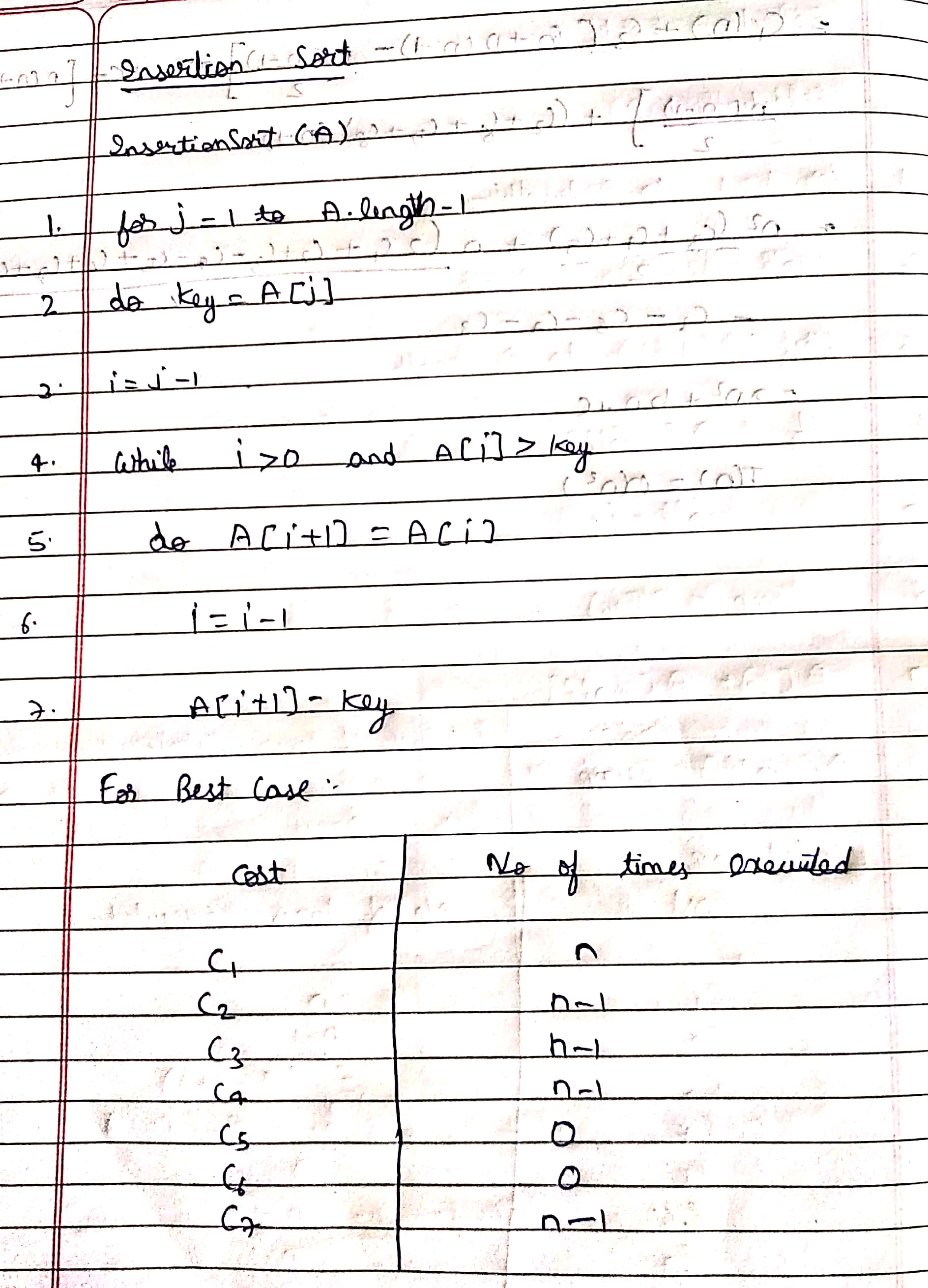


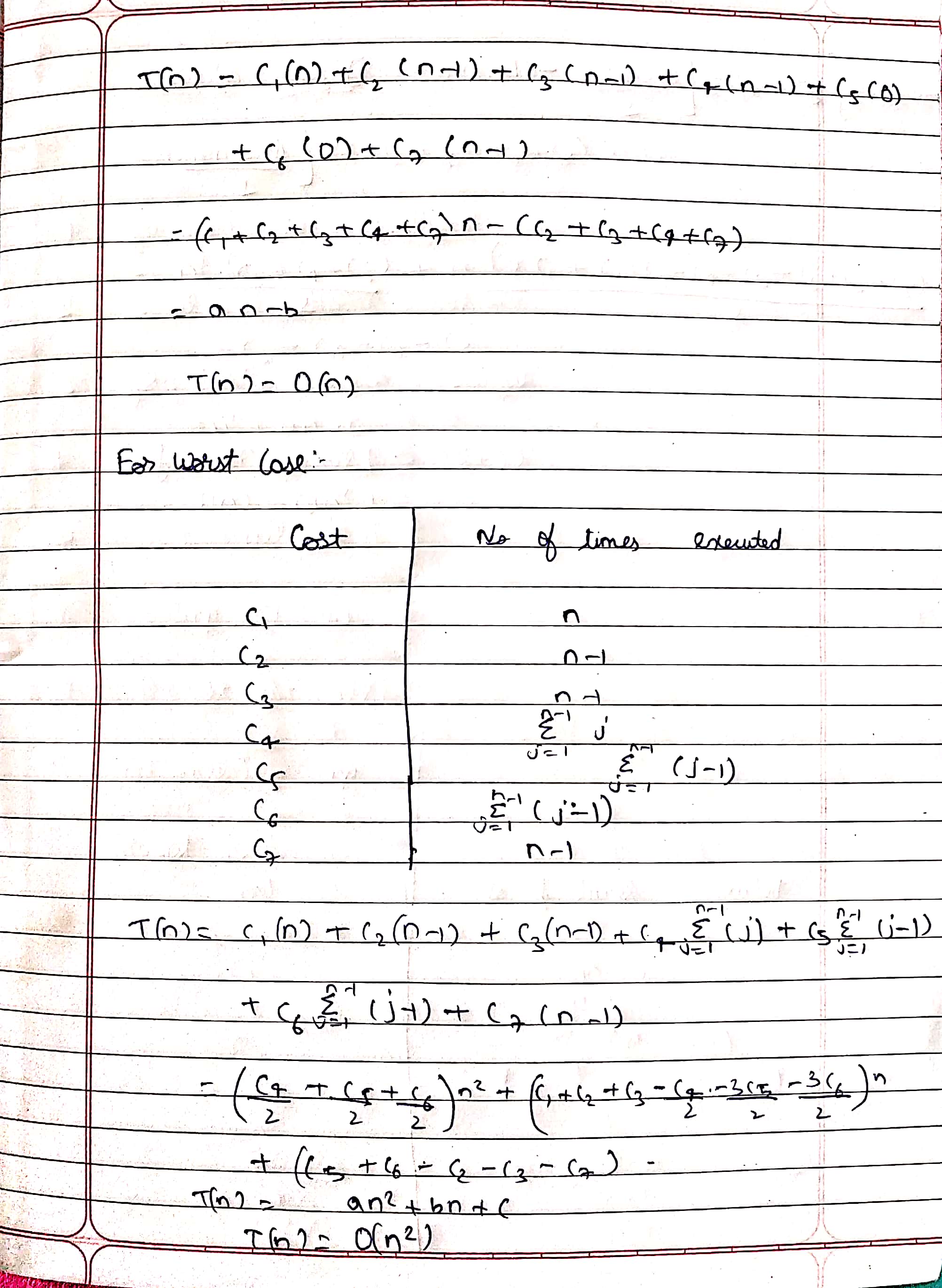












**1.3. Provide the details of Hardware/Software you used to implement algorithms and**

**to measure the time.**

Software: Visual Studio Code

Hardware: intel core i5, 8th gen, 8GB RAM

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

Linear Search:

import java.util.\*;

import java.io.\*;

class LinearSearch

{

public static void main(String args[])throws IOException

{

BufferedReader br=new BufferedReader(new FileReader("File 1.txt"));

Scanner sc=new Scanner(System.in);

System.out.println("Enter search element");

String str=sc.nextLine();

String line=br.readLine();

int i;

boolean flag=false;

long sttime=System.nanoTime();

for(i=0;line!=null;i++,line=br.readLine())

{

if(str.equals(line))

{

flag=true;

break;

}

}

if(flag==true)

System.out.println("Found at "+i);

else

System.out.println("Not found");

long endTime=System.nanoTime();

long time\_ell=endTime-sttime;

System.out.println("Time: "+time\_ell);

}

}

Bubble Sort:

import java.io.BufferedReader;

import java.io.DataInputStream;

import java.io.FileInputStream;

import java.io.InputStreamReader;

import java.io.PrintStream;

public class BubbleSort {

public static void main(String[] args) {

int arr[]=new int[1024];

int i=0;

try {

FileInputStream fstream=new FileInputStream("File 1.txt");

DataInputStream in=new DataInputStream(fstream);

BufferedReader br=new BufferedReader(new InputStreamReader(in));

String s;

while((s=br.readLine())!=null)

{

arr[i]=Integer.parseInt(s);

i++;

}

int n = arr.length;

int temp;

long start=System.currentTimeMillis();

for (int k = 0; k < n-1; k++)

for (int j = 0; j < n-k-1; j++)

if (arr[j] > arr[j+1])

{

temp = arr[j];

arr[j] = arr[j+1];

arr[j+1] = temp;

}

long end=System.currentTimeMillis();

System.out.println("Time Taken :"+(end-start));

}

catch(Exception e) {

System.out.println("error:"+e.getMessage());

}

}

}

Selection Sort:

import java.io.BufferedReader;

import java.io.DataInputStream;

import java.io.FileInputStream;

import java.io.InputStreamReader;

import java.io.PrintStream;

public class SelectionSort {

public static void main(String[] args)

{

int arr[]=new int[1024];

int i=0;

try {

FileInputStream fstream=new FileInputStream("File 1.txt");

DataInputStream in=new DataInputStream(fstream);

BufferedReader br=new BufferedReader(new InputStreamReader(in));

String s;

while((s=br.readLine())!=null)

{

arr[i]=Integer.parseInt(s);

i++;

}

int min\_idx ;

int n = arr.length;

long start=System.currentTimeMillis();

for (int t = 0; t < n-1; t++)

{

min\_idx = t;

for (int j = t+1; j < n; j++)

if (arr[j] < arr[min\_idx])

min\_idx = j;

int temp = arr[min\_idx];

arr[min\_idx] = arr[t];

arr[t] = temp;

}

long end=System.currentTimeMillis();

System.out.println("Time Taken :"+(end-start));

}

catch(Exception e) {

System.out.println("error:"+e.getMessage());

}

}

}

Insertion Sort:

import java.io.BufferedReader;

import java.io.DataInputStream;

import java.io.FileInputStream;

import java.io.InputStreamReader;

import java.io.PrintStream;

public class InsertionSort {

public static void main(String[] args)

{

int arr[]=new int[1024];

int i=0;

try {

FileInputStream fstream=new FileInputStream("1024.txt");

DataInputStream in=new DataInputStream(fstream);

BufferedReader br=new BufferedReader(new InputStreamReader(in));

String s;

while((s=br.readLine())!=null)

{

arr[i]=Integer.parseInt(s);

i++;

}

int key,j;

int n = arr.length;

long start=System.currentTimeMillis();

for (int t = 1; t < n; ++t) {

key = arr[t];

j = t - 1;

while (j >= 0 && arr[j] > key) {

arr[j + 1] = arr[j];

j = j - 1;

}

arr[j + 1] = key;

}

long end=System.currentTimeMillis();

System.out.println("Time Taken :"+(end-start));

}

catch(Exception e)

{

System.out.println("error:"+e.getMessage());

}

}

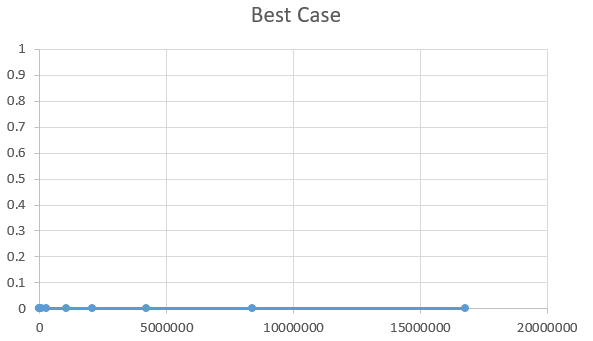
}

**1.5. Measure the best-case time and worst-case time of linear search for all ten files.**

**Plot a graph.**

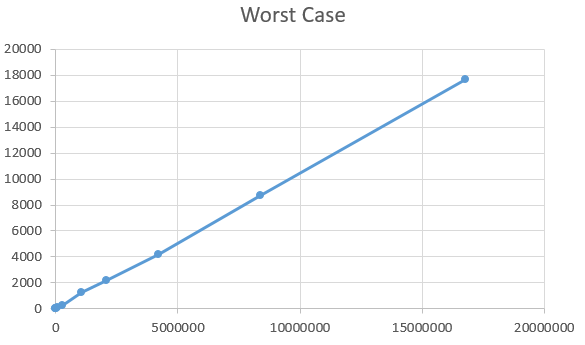
Linear Search (Best Case):

|  |  |
| --- | --- |
| Input from file | Time Taken |
| 1024 | 0 |
| 4096 | 0 |
| 16384 | 0 |
| 65536 | 0 |
| 262144 | 0 |
| 1048576 | 0 |
| 2097152 | 0 |
| 4194304 | 0 |
| 8388608 | 0 |
| 16777216 | 0 |



Linear Search (Worst Case):

|  |  |
| --- | --- |
| Input from file | Time Taken |
| 1024 | 7 |
| 4096 | 14 |
| 16384 | 25 |
| 65536 | 85 |
| 262144 | 277 |
| 1048576 | 1265 |
| 2097152 | 2198 |
| 4194304 | 4203 |
| 8388608 | 8713 |
| 16777216 | 17604 |

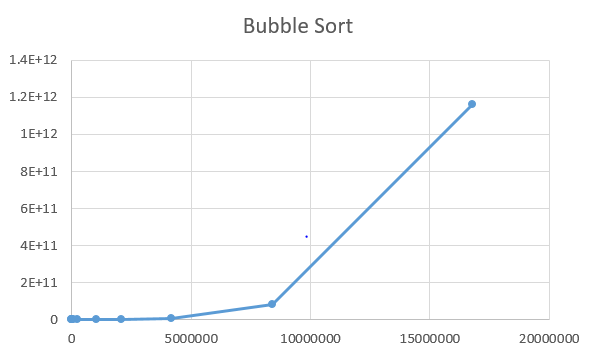


**1.6. Measure the average-case time (considering current data of ten files) of bubble**

**sort, selection sort, and insertion sort for all ten files. Plot a graph.**

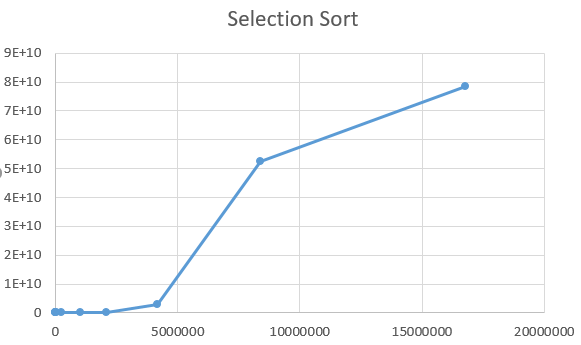
Bubble Sort:

|  |  |
| --- | --- |
| Input from file | Time Taken |
| 1024 | 22 |
| 4096 | 302 |
| 16384 | 5346 |
| 65536 | 112046 |
| 262144 | 1379052 |
| 1048576 | 23065168 |
| 2097152 | 353042688 |
| 4194304 | 5.24E+09 |
| 8388608 | 8.47E+10 |
| 16777216 | 1.16E+12 |



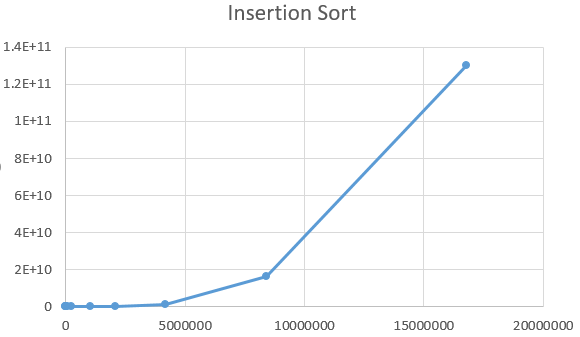
Selection Sort:

|  |  |
| --- | --- |
| Input from file | Time Taken |
| 1024 | 15 |
| 4096 | 175 |
| 16384 | 1948 |
| 65536 | 43864 |
| 262144 | 822412 |
| 1048576 | 10280900 |
| 2097152 | 153932600 |
| 4194304 | 29222105661 |
| 8388608 | 5.25E+10 |
| 16777216 | 7.86E+10 |



Insertion Sort:

|  |  |
| --- | --- |
| Input from file | Time Taken |
| 1024 | 35 |
| 4096 | 184 |
| 16384 | 6554 |
| 65536 | 48440 |
| 262144 | 721946 |
| 1048576 | 1082988 |
| 2097152 | 12958560 |
| 4194304 | 1.16E+09 |
| 8388608 | 1.63E+10 |
| 16777216 | 1.30E+11 |

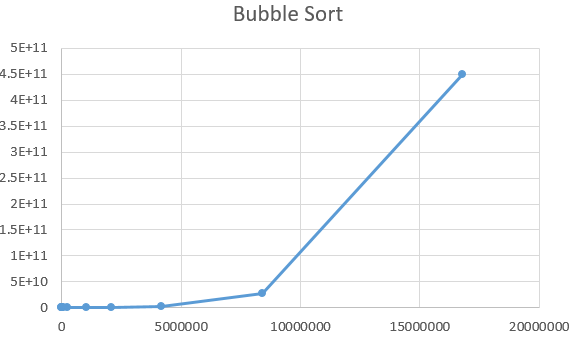


**1.7. Measure the best-case time of bubble sort, selection sort, and insertion sort for**

**all ten files. Plot a graph.**

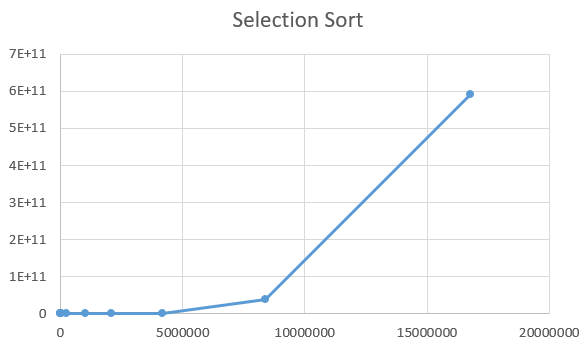
Bubble Sort:

|  |  |
| --- | --- |
| Input from file | Time Taken |
| 1024 | 9 |
| 4096 | 138 |
| 16384 | 2114 |
| 65536 | 36416 |
| 262144 | 1130650 |
| 1048576 | 19689728 |
| 2097152 | 27476736 |
| 4194304 | 2.01E+09 |
| 8388608 | 2.80E+10 |
| 16777216 | 4.49E+12 |



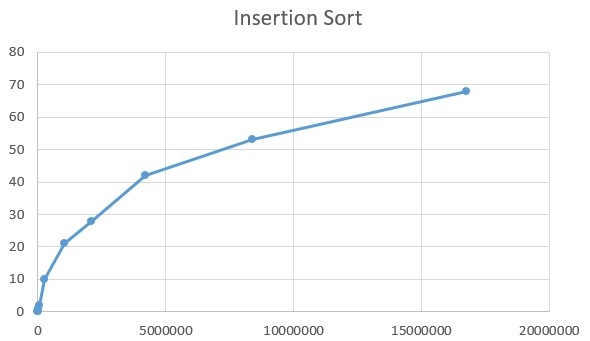
Selection Sort:

|  |  |
| --- | --- |
| Input from file | Time Taken(ms) |
| 1024 | 18 |
| 4096 | 145 |
| 16384 | 1871 |
| 65536 | 38164 |
| 262144 | 686694 |
| 1048576 | 8240184 |
| 2097152 | 115362576 |
| 4194304 | 2.19E+09 |
| 8388608 | 3.94E+10 |
| 16777216 | 5.91E+10 |



Insertion Sort:

|  |  |
| --- | --- |
| Input from file | Time Taken(ms) |
| 1024 | 0 |
| 4096 | 0 |
| 16384 | 1 |
| 65536 | 2 |
| 262144 | 10 |
| 1048576 | 21 |
| 2097152 | 28 |
| 4194304 | 42 |
| 8388608 | 53 |
| 16777216 | 68 |

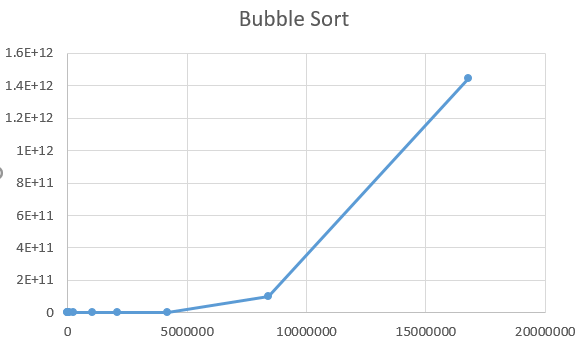


**1.8. Measure the worst-case time of bubble sort, selection sort, and insertion sort for**

**all ten files. Plot a graph.**

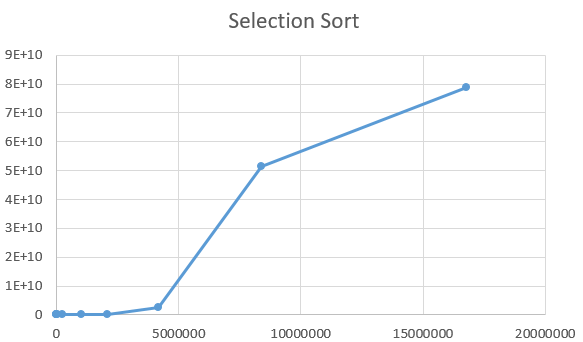
Bubble Sort:

|  |  |
| --- | --- |
| Input from file | Time Taken |
| 1024 | 22 |
| 4096 | 450 |
| 16384 | 7062 |
| 65536 | 159384 |
| 262144 | 2171825 |
| 1048576 | 32833462 |
| 2097152 | 537501938 |
| 4194304 | 6.45E+04 |
| 8388608 | 1.03E+11 |
| 16777216 | 1.44E+12 |



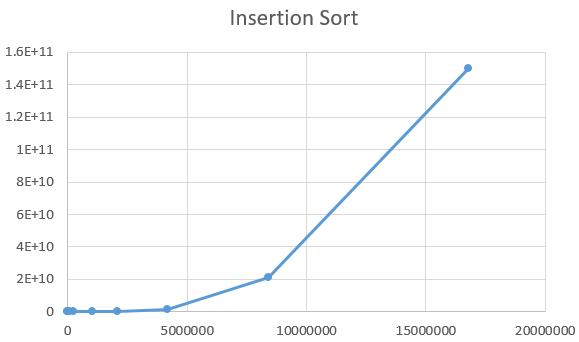
Selection Sort:

|  |  |
| --- | --- |
| Input from file | Time Taken(ms) |
| 1024 | 18 |
| 4096 | 245 |
| 16384 | 3357 |
| 65536 | 70745 |
| 262144 | 913564 |
| 1048576 | 13623200 |
| 2097152 | 221594400 |
| 4194304 | 2.71E+09 |
| 8388608 | 5.14E+10 |
| 16777216 | 7.91E+10 |



Insertion Sort:

|  |  |
| --- | --- |
| Input from file | Time Taken(ms) |
| 1024 | 45 |
| 4096 | 248 |
| 16384 | 22954 |
| 65536 | 77657 |
| 262144 | 1625612 |
| 1048576 | 6302528 |
| 2097152 | 113445504 |
| 4194304 | 1.18E+09 |
| 8388608 | 2.11E+10 |
| 16777216 | 1.50E+11 |



**1.9. 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?**

Ans: Yes, you can by executing some of the files and after that we draw graph using MS Excel. Using trend line we can predict the behavior of graph and we can predict the time complexity of the algorithm.

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

Ans: Yes

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

**the difference.**

Ans:

|  |  |  |  |
| --- | --- | --- | --- |
| **Algorithm** | **Best Case** | **Average Case** | **Worst Case** |
| Linear Search | O(1) | O(n) | O(n) |
| Bubble Sort | O(n) | O(n^2) | O(n^2) |
| Selection Sort | O(n^2) | O(n^2) | O(n^2) |
| Insertion Sort | O(n) | O(n^2) | O(n^2) |

Name: Ayush Yadav

ID No: 201851032

Section: 1

Lab Group: A