Design and Analysis of Algorithm

PROJECT

NAME: Abdullah Saqib, Rayed Saeed, Muhammad Hashir

ROLL NO: i20-0458, i20-1822, i20-0440

SECTION: F

DEGREE: BS-CS

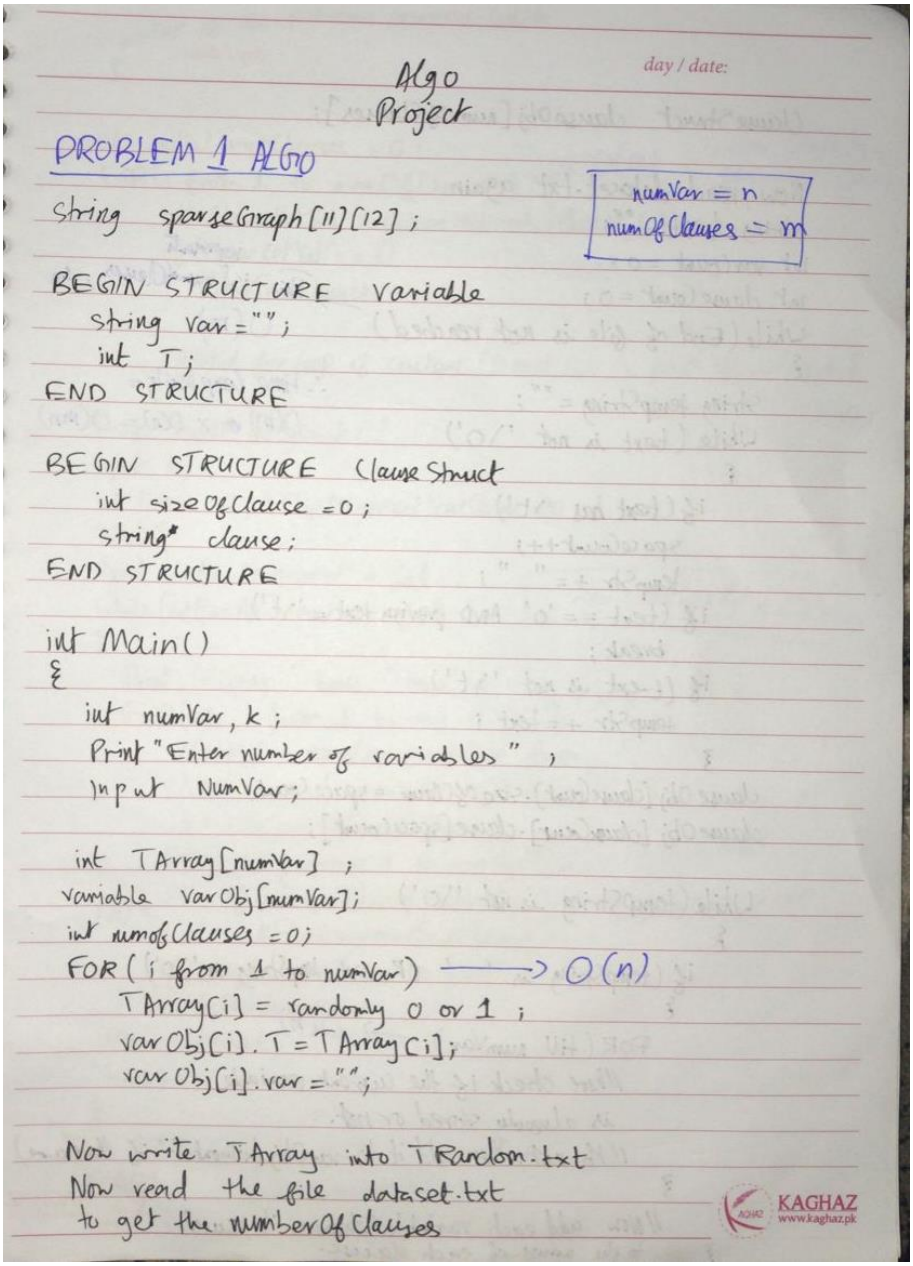
SUBJECT: Design and Analysis of Algorithms,

SUBMITTED TO: Sir Rohail Gulbaz

SUBMISSION DATE: 23 / 5 / 2022

PROBLEM 1

Done by: Abdullah Saqib i20-0458



Algo
Project

day / date:

PROBLEM 1 ALGO

numVar = n
numOfClauses = m

```
string sparseGraph[11][12];

BEGIN STRUCTURE Variable
   string var = "";
   int T;
END STRUCTURE

BEGIN STRUCTURE ClauseStruct
   int sizeOfClause = 0;
   string* clause;
END STRUCTURE

int Main()
{
   int numVar, k;
   Print "Enter number of variables",
   Input NumVar;

   int TArray[numVar];
  Variable varObj[numVar];
  int numofClauses = 0;
  FOR (i from 1 to numVar) ----> O(n)
      TArray[i] = randomly 0 or 1;
      varObj[i].T = TArray[i];
      varObj[i].var = "";

  Now write TArray into TRandom.txt
  Now read the file dataset.txt
  to get the numberOfClauses
```

KAGHAZ
www.kaghaz.pk

day / date:

```
clauseStruct clauseObj[numofClauses];

Now, read dataset.txt again
string text = "";
int varCount = 0;
int clauseCount = 0;
While (End of file is not reached)        -> This is numofClauses, so represents O(m)
{
    string tempString = "";               ∴ time complexity =
    While (text is not '\0')              O(m) x O(n) = O(mn)
    {
        if (text has '\t')
            spaceCount++;
            tempStr += " ";
        if (text == '0' AND previous text = '\t')
            break;
        if (text is not '\t')
            tempStr += text;
    }
clauseObj[clauseCount].sizeofClause = spaceCount;
clauseObj[clauseCount].clause[spaceCount];

While (tempString is not '\0')
    {
        if (tempString is ' ' OR next tempString is '\0')
            {
                FOR (till numVar) -> O(n)
                //here check if the current variable
                is already stored or not.
                // If not, then add it to varObj (without '-' if there are)
            }
        // Now add each variable (whenever it occurs)
        to the arrays of each clause.
}
```

```
} → end of the previous previous while
```

day / date:

```
int initialTrueClauses = 0;                                   → O(m)   ∴ time Complexity = O(m) x O(n) = O(mn)
FOR(i from 1 to numOfClauses)
    int ClauseRetVal = clauseNetwork(clauseObj[i], varObj, numVar);
    if(clauseRetVal == 1)                                     → O(n)
        initialTrueClauses++;

    //Nested for loop of constant (11 and 12) to print sparse Graph
                                                              → O(1)
Print "Enter integer k:"
Input k;
int times = 0, kFlipAssignmentTrueClause = 0;
bool betterAssignment = false;
While(betterAssignment == false)
{
    Print "Fliping " times "times";
    FOR(int i from 1 to k)                                    → O(k)
        //select a random variable from
        //objVar and invert its truth value
        kFlipAssignmentTrueClauses = 0;
        FOR(int j from 1 to numOfClauses)                     → O(m)
            if(clauseNetwork(clauseObj[i], varObj, numVar) == 1)   ← O(n)
                kFlipAssignmentTrueClauses++;
        if(kFlipAssiTrueClauses > initialTrueClauses)
            betterAssignment = true;
            break;
    times++;
}
int TDashArray[numVar];
FOR(int i from 1 to numVar)
    TDashArray[i] = varObj[i].T;
```

time Complexity = O(nmk)

time Complexity =>
O(nmk)

day / date:

```
//Now write this TDashArray to TDash.txt file
return 0;
}→ end of main
```

time complexity = O(n) + O(n) = O(n)

```
int clauseNetwork (clauseObj, varObj[], numVar)
{
//initialize sparse graph with '---' on each index
if(clauseObj.sizeOfClause == 1)
{
   FOR(int i from 1 to numVar) ——→ O(n)
      //check the T value of that variable in
      // the clause
      //return 0 if T = 0 } vice versa if variable
      //return 1 if T = 1 } in clause is '-' negative
      # //when variable found:
         sparseGraph[0][6] = obj clauseObj.clause[0];
}
else ——→ O(1) x O(1) x O(n) = O(n)
{  int count = 0 , orCount=0 ; bool nodesFilled = false;
   FOR(i 1 to 11) ——→ O(1)
      FOR(j 1 to 12) ——→ O(1)
         if (count >= clauseObj.sizeOfClause)
            nodesFilled = true
         if (nodesFilled == false)
            if (count, i and j values in code ) //For [redacted] greater than N2
               sparseGraph[i][j] = obj clauseObj[count];
               count++;
            else if (i is 0 and j is 6) //for N1
               sparseGraph[i][j] = clauseObj[count];
               count++;
```

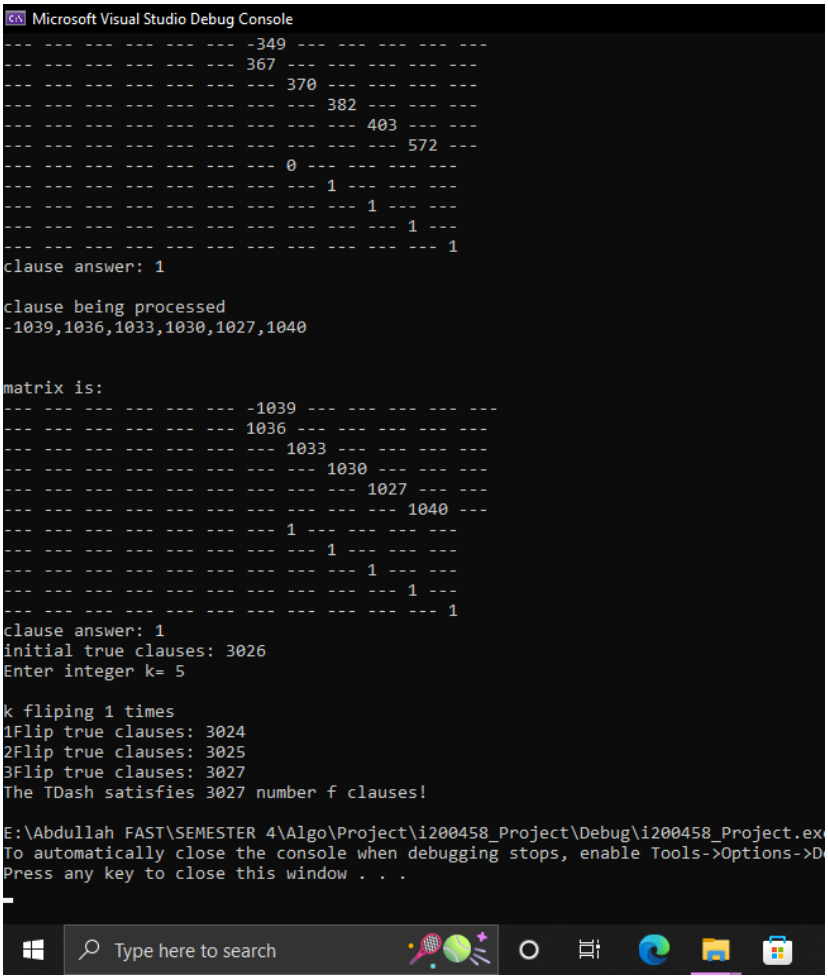
```
else if (i=1 and j=6) for N2                    day / date:
    sparseGraph[i][j] = clauseObj.clause[count];
    count++
else // Performing OR
    if (i=6+orCount and j = i+1)
        if (orCount ==0)                                  → O(n)
            sparseGraph[i][j] = performORop(sparseGraph[][],
                                            sparseGraph[][],
                                            varObj,
                                            numVar)
        else
            sparseGraph[i][j] = performORop(sparseGraph[][],
                                            sparseGraph[][],
                                            varObj, numVar)
                                  ↓ O(n)
    orCount++;
    if(orCount+1 == count)
        return stoi(sparseGraph[i][j]);
    } -> else
} -> function

string performORop(s1, s2, varObj[], numVar)   ∴ O(n)
{
    if (s1 and s2 are not 1 or 0)   ##D
        FOR(int i 1 to numVar) ——————→ O(n)
            // check the Tvalue1 of s1 (considering NOT)
    else
        // Tvalue1 of s1 is s1 itself

    // Do the same for s2 ——————→ O(n)

    if (Tvalue1 ==0 AND Tvalue2 ==0)
        return "0"
    else
        return "1"
}
```

∴ time complexity = O(n) + O(n) = O(n)

day / date:

Time Complexity in asymtotic notation is ~~O(mn)~~ O(nmk)
This was where TDash was being calculated.

Cpp sample output:



PROBLEM 2

Done by: Rayed Saeed i20-1822

1. **Top down without memoization Algorithm:**

Start

Plot_divide(price,size)

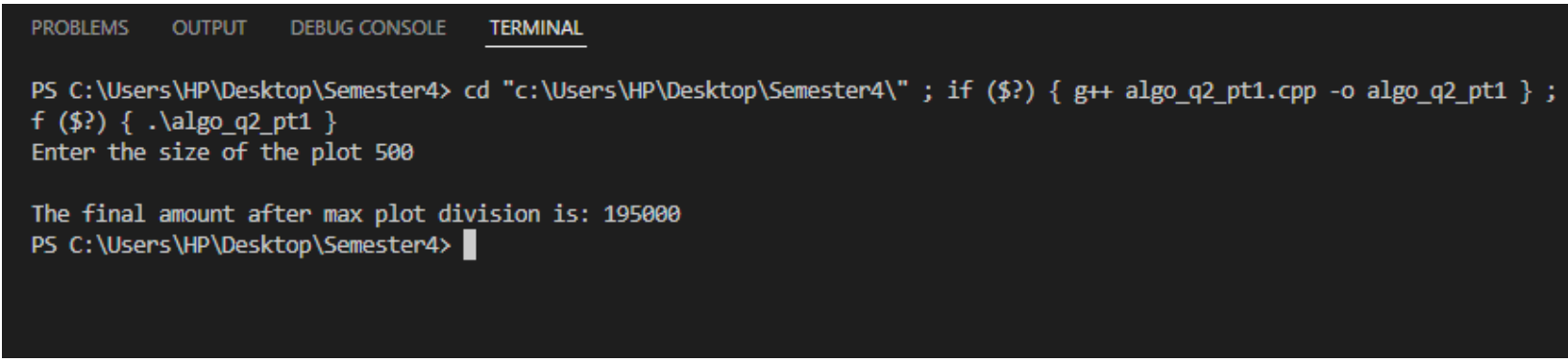
    If size == 0

        Return 0

    Else

```
        q = - infinity
        for i = 1 to size
            q = max(q, price [ i ] + plot_divide(price, size – i))
        return q
End
```

Program code output for the first part:



2. **Top down with memoization Algorithm:**

```
Start
Memorized_plot_divide(price, size)
        R[0 till size] // new array
        For i = 0 to size
            R [ i ] = - infinity
        Return memorized_plot_divide2(price, size, R)
End
Start
Memorized_plot_divide2(price, size, R)
        If R[size] >= 0
            Return R[size]
```

If size == 0

q=0

else

q= - infinity

for i =1 to size

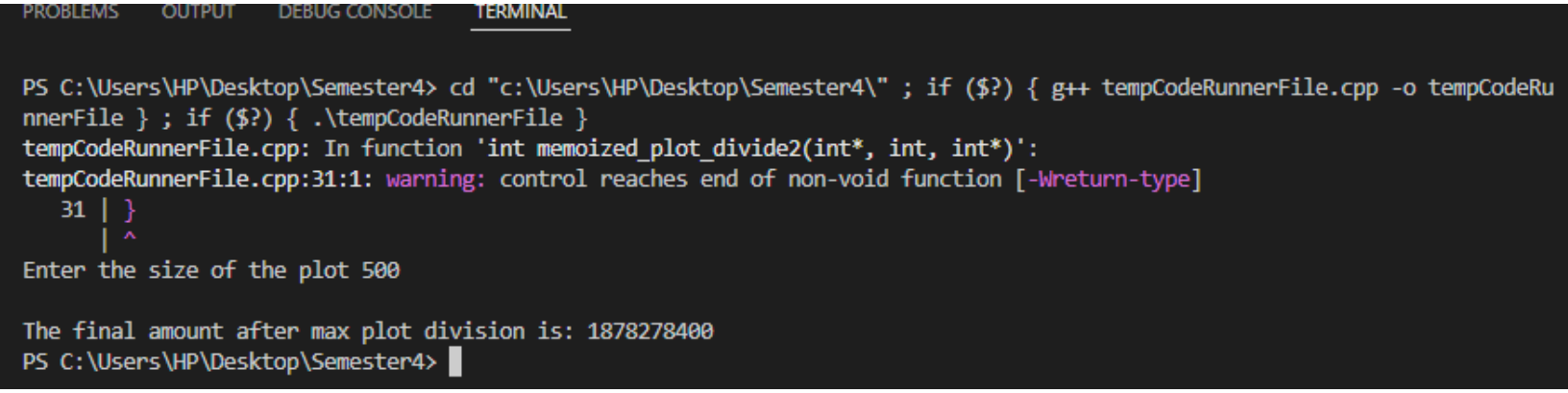
q = max(q, price[i] + memorized_plot_divide2(price,size-i,R)

R[size]=q

Return q

End

Program code output for the second part:



3. **Bottom up Iterative Algorithm:**

Start

Bottom_up_plot_divide (price, size)

R[0 till size] //creating a new array

R[0] = 0

For i =1 to size

q = - infinity

for j =1 to i

```
            q = max(q,price[ j ] + R[ i – j ] )
        R[ i ] = q
        Return R[size]
End
```

Program code output for the third part:

```
PS C:\Users\HP\Desktop\Semester4> cd "c:\Users\HP\Desktop\Semester4\" ; if ($?) { g++ algo_q2_pt3.cpp -o algo_q2_pt3 } ; i
f ($?) { .\algo_q2_pt3 }
Enter the size of the plot 500

The final amount after max plot division is: 4199500
PS C:\Users\HP\Desktop\Semester4>
```

4. Optimization was not applicable for this program

5. **Time complexity analysis of the iterative algorithm**

Loop 1 runs till the size of array hence ‘n’

Loop 2 runs till the end of loop 1 for i iterations, the i iterations are till the end of size of array. Hence, loop 2 also runs till the size of array and hence has the running time ‘n’

n*n = n^2

the running time complexity of the iterative program for this plot cutting problem is

O (n^2)

Big-oh n square

PROBLEM 3

Done by: Muhammad Hashir

M. Hashir 201-0440 Section H F

Design and Analysis of Algorithm
Q#3

Q3) Text Array = Array[n][n]
Pattern Array = arr[m][m]

4 nested Loops

Outer loop = (n-m)
Inner loop = (n-m+1)
Inner loop = m
Inner most loop = m

$= (n-m)(n-m+1)(m)^2$

Time Complexity = $O((n-m)(n-m+1)m^2)$

Logic:

Traverse the **Text matrix** to check if it contains the element at the first index of **Pattern matrix**. If it exists, then traverse using for loop to check if the matrix exists in the Text matrix, if it exists, then display it. Start traversing the matrix again but from new index, where the first occurrence of pattern ended. This logic is repeated to check for diagonals.

Sample Output: