King Saud University College of Computer and Information Sciences Computer Science Department	
CSC311 The Design and Analysis of Algorithms	Second Semester 1443 - 2022

CSC311 Project
Phase 1

Student Name	Student ID
Aljawharh Alotaibi	441200846
Aljwhra Almakhdoub	441201225
Raghad Aljuhaimi	441201212

1.1. Description of the Brute force algorithm (pseudo-code with explanation):

The basic idea is to merge both sorted arrays and get the median by taking an average of both middle elements in the merged array. And we used an idea like the merging procedure of the *merge sort*. using int x[]={1,3,4,6,9}; int y[]={2,5,7,8,10}; as an input, and getting 5.5 as an output.

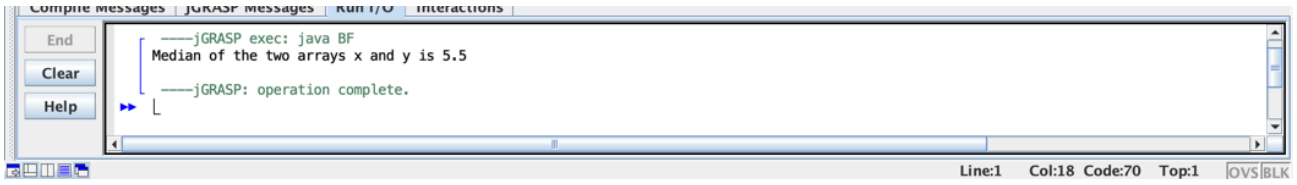
- **Pseudo-code:**

```
FUNCTION GETMEDIAN(arr1, arr2, n):
//Sorting both arrays to get its median, similar to merge sort.
//input: two arrays.
//output: double number presenting a median.
while i < n and j < n
      if  arr1[i] <= arr2[j]
      merge[k] ← arr1[i]
    end If
end while

 else
merge[k] ← arr2[j]
j← j+1
k← k+1

 while i < n
   merge[k] ← arr1[i]
        j← j+1
        k← k+1
  end while

while j < n
merge[k] ← arr2[j]
j← j+1
k← k+1
  end while

return (double)(merge[n-1]+merge[n])/2
END  FUNCTION
```

- **The time and space complexity of the Brute Force solution:**

We need to traverse each element to merge both sorted arrays. With placing one value to the merge[]
So, time complexity of merging = O(n).
Time complexity = Time complexity of merging + Time complexity of finding median = O(n)+ O(1) = O(n).
We are using 2n size extra space for merging, so space complexity = ***O(n)***.

Sample run of the Brute Force Algorithm:



```
 ----jGRASP exec: java BF
Median of the two arrays x and y is 5.5

 ----jGRASP: operation complete.
```

1.2. Enhancement of the Brute Force algorithm:

We cannot enhance the time complexity but to enhance the space complexity of the previous Brute Force approach while comparing the elements of two sorted arrays during the merging process, we can track the count of elements in the merged array using variable k. We stop when the count becomes n + 1 and calculate the median by taking an average of the n^{th} and $(n+1)^{th}$ element of the merged array. So, there is no need to use extra space and perform complete merging.

- **Pseudo-Code:**

1- We initialize two variables i = 0 and j = 0 to move in arr1[] and arr2[].

2- We also initialize three variables: mid1=0 and mid2=0 to keep track of the middle elements and k=0 to track the count of elements in the merged array.

3- We start the merging loop that iterates k less than or equal to the value of n times and move forward by comparing the elements in arr1[] and arr2[] until k becomes equal to n +1. In the process we keep track of the middle elements mid1 and mid2.

4- By end of the loop, we return the median by taking an average of the mid1 and mid2. The average equation: mid1 + mid2 / 2.

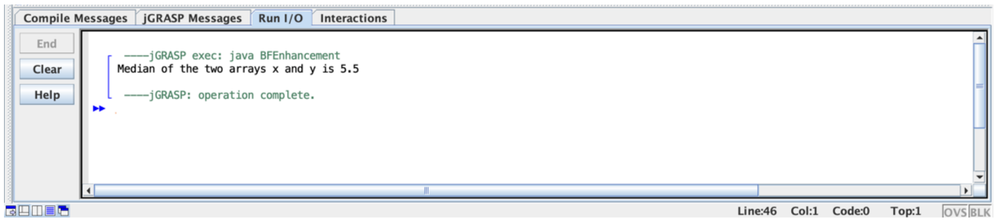
- **Source-Code:**

```
public class BFEnhancement{
   public static void main(String args[]){

      int x[]={1,3,4,6,9};
      int y[]={2,5,7,8,10};

      System.out.println("Median of the two arrays x and y is " + median(x,y,5));

   }

  public static double median(int arr1[], int arr2[], int n)
   {
      int i = 0, j = 0, k = 0, mid1 = 0, mid2 = 0;

      while(k <= n)
      {
         if(arr1[i] <= arr2[j])
         {
            mid1 = mid2;
            mid2 = arr1[i];
            i++;
         }
         else
         {
            mid1 = mid2 ;
            mid2 = arr2[j];
            j++;
         }
         if(i == n)
         {
            mid1 = mid2;
            mid2 = arr2[0];
            break;
         }
         else if(j == n)
         {
            mid1 = mid2;
            mid2 = arr1[0];
            break;
         }
         k++;
      }
      return (double)(mid1 + mid2)/2; }}
```

- **The time and space complexity of the enhancement:**

While loop is running n times and doing O(1) operation at each iteration. Which means the time complexity will be: n * O(1) = O(n).
We are using a constant number of extra variables, which means the space complexity will be: O(1).

- **Sample run of the Brute Force Algorithm Enhancement:**



```
 ----jGRASP exec: java BFEnhancement
Median of the two arrays x and y is 5.5

 ----jGRASP: operation complete.
```

2.1. A description of the Divide and Conquer algorithm (pseudo-code with explanation):

Median explanation: the basic idea is to find the median for each array if the median of the first median bigger than the second the first array will be taken from the most left index to the middle index, and the second array from the middle index to the most right index. Otherwise, if the second median is bigger the first array will be taken from the middle to the most right index, and the second array from the left to the middle index.

If both have the same median, then one of those medians will be returned
At the end we will take the maximum index if the array length has reached two and the minimum from the second index add them, then divide by tow and return the median of the two array.

```
//Sorting both arrays to get its median, similar to merge sort.
//input: two arrays.
//output: double number presenting a median.

Median(arr1[1..n], l1, r1, arr2, l2, r2)
n← r1- l1 + 1
mid1 ← ⌈(r1+ l1)/2⌉
mid2 ← ⌊(r2+ l2)/2⌋

m1← median(arr1[1..n], n)
m2← median(arr2[1..n], n)

if (n=1)
return (arr1[0] + arr2[0])/2
if(n=2)
return (Max(arr1[l1], arr2[l2]) + Min(arr1[r1], arr2[r2]))/2

if (m1=m2)
return m2

else if (m1 > m2)
return median (arr1[1..n], l1, mid1, arr2[1..n], mid2, r2)
else
return median (arr1[1..n], mid1  r1, arr2[1..n], l2, mid2)
```

```
 Median (arr[1..n], int n)
   {
      if (n % 2 = 0)
         return (arr[n / 2] + arr[n / 2 - 1]) / 2;
      else
         return arr[n / 2];
   }
```

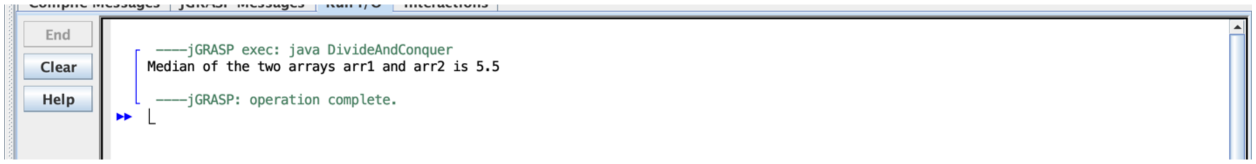
```
Median (arr1[1…n], arr2[1…n])
N ← array length
return median(arr1[1..n], 0, N-1, arr2, 0, N-1)
```

- **The time and space complexity of the Divide and Conquer solution:**

  Time Complexity: O(logn)

  Space Complexity: O(logn)

- **Sample run of the Divide and Conquer Algorithm:**



```
----jGRASP exec: java DivideAndConquer
Median of the two arrays arr1 and arr2 is 5.5

----jGRASP: operation complete.
```

2.2. The recurrence relation for the algorithm's basic operation and the solution:

Basic operation: number of comparisons.

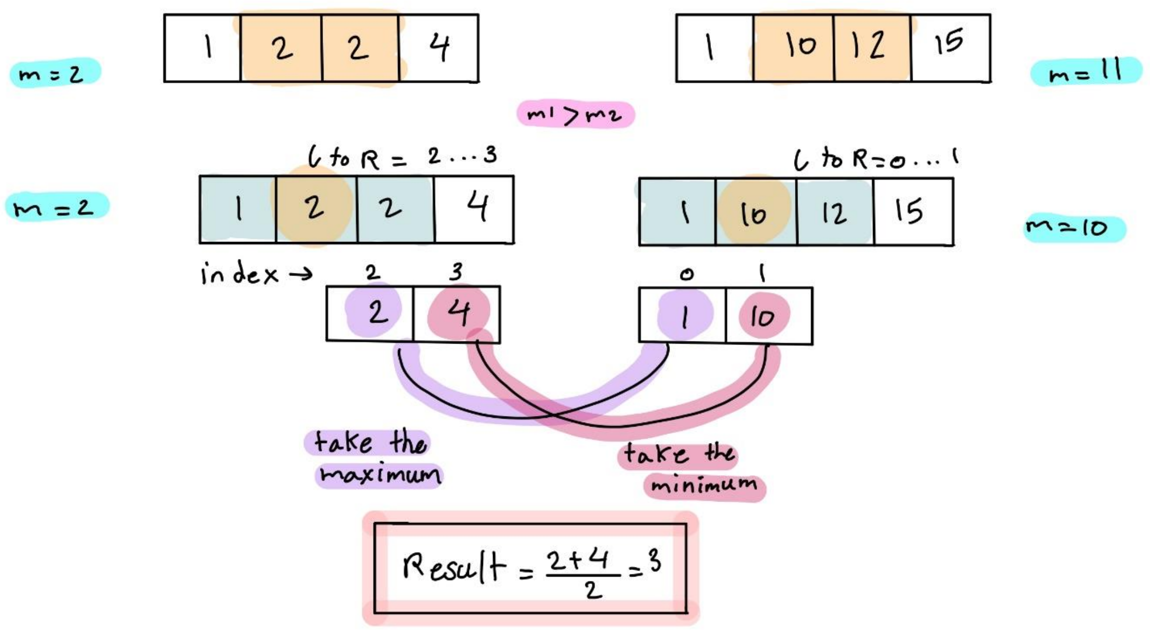
Setting and solving the recurrence relation:

$T(n) = T(n/2) + \Theta(1)$ level 1
$= [T((n/2)/2) + 1] + 1$ $\frac{1}{2k} = n$

$= T(n/4) + 2$ level 2 $n = 2k$
$= [T((n/4)/2) + 1] + 2$ $\log2\ n = k$

$= T(n/8) + 3$ level 3
:
$= T\ n\ 2k + k$
:
$= T\ 1 + \log2\ n$ level log2n (leaves)
$= \Theta(\log2 n)$

2.3. Illustration shows how our algorithm will work on an Example:



*Note the value of left and right depends on the median of each array

3.1. Source code of Brute Force Algorithm with comments:

```
public class BF {

   public static void main(String args[]) {

   //Initializing the two sorted arrays of equal size n (n = 5)

      int x[]={1,3,4,6,9};

      int y[]={2,5,7,8,10};

   //Calling the getmedian method(and sending the two initialized arrays and their n size) in a
print statement that will print the method's output

      System.out.println("Median of the two arrays x and y is " + getmedian(x,y,5));

   }

   //The getmedian method that will merge the two arrays and calculate the merged array median

   static double getmedian(int arr1[], int arr2[], int n)

   {

      int merge[]=new int[2*n];   //Initializing an array with the size of 2n (n being the
size of the two previous arrays, n = 5) to merge the two arrays

      int i=0,j=0,k=0;

      while (i < n && j < n) {    //While loop that iterates as long as i and j values are less
than the n value (i and j values start from 0 and n = 5) and contains if statements that will
sort the merge array



         if (arr1[i] <= arr2[j]) {  //if statement that checks if the value of the number at
index i in the first array is less than or equal to the value of the number at index j in the
second array

            merge[k] = arr1[i];      //if the previous condition is met assigning the value of
the number at index i of the first array in the merge array at index k

            i++;   //Incrementing the value of i if previous condition is met

         }

         else {

            merge[k] = arr2[j];     //if the previous condition is not met assigning the value of
the number at index j of the second array in the merge array at index k

            j++;   //Incrementing the value of j if previous condition is not met

         }

         k++;   //Incrementing the value of k and continuing on with the while loop
      }

      //While loops either of them only get executed if there are any values left in the first
array or the second array

      while (i < n) {

         merge[k] = arr1[i];

         i++;

         k++;

      }

      while (j < n) {

         merge[k] = arr2[j];

         j++;

         k++;

    return (double)(merge[n-1]+merge[n])/2;

   }

}
```

3.2. Source code of Divide and Conquer Algorithm:

```
  public static void main(String[] args) {

       //Initializing the two sorted arrays of equal size n (n = 5)

        int[] arr1= {1,3,4,6,9};

        int[] arr2={2,5,7,8,10};

      //Calling the Median method that receives two parameters (and sending the two initialized
arrays) in a print statement that will print the method's output

        System.out.println("Median of the two arrays arr1 and arr2 is " + Median(arr1,arr2));

    }

       //The Median method that receives two arrays, initializes N and calls The Median recursive
method

    public static double Median(int[] arr1, int[] arr2)

    {

        int N = arr1.length;            //Initializing variable N and assigning it arr1's length

        return Median(arr1, 0, N -1 , arr2, 0, N - 1); //Calling the median recursive method
    }

     /* The Median method that receives the first array, l1, N-1, second array, l2, r2 . And
calculates the median by calling the Median method

     that receives two parameters the array and n value(n is calculated by the equation n= r1 -
l1 +1) and then returning the median value or executing

     certain equations based on the value of n or the value that the Median method that receives
two parameters calling returns */

    public static double Median(int[] arr1, int l1, int r1, int[] arr2, int l2, int r2)

    {

        int n=r1-l1+1; //Initializing the value of n

        int mid1 = (int)Math.ceil((double) (r1 + l1 ) / 2); //Initializing the value of mid1
which is the index that depends on m we used the celling method to take the furthest possible
index

        int mid2 = (r2 + l2 ) / 2; //Initializing the value of mid2 which will be floored
directly by java language

        double m1=Median(arr1,n); //Calling the Median method and sending the first array and the
value of n

        double m2=Median(arr2,n); //Calling the Median method and sending the second array and
the value of n

        //Conditions that execute based on the value of n or the value of m1 and m2

        if (n==1)

            return (double) (arr1[0]+arr2[0])/2;  //If the merge of the two arrays length is one

        if (n==2)

            return (double) (Math.max(arr1[l1] , arr2[l2]) + Math.min(arr1[r1] , arr2[r2]))/2;
//If the merge of the two arrays length is two

        if (m1==m2)

            return m2;  //If the median of the first array equals the median of the second array

        else if (m1 > m2)

            return Median(arr1, l1, mid1 , arr2, mid2 , r2);  //Recursion if the median of the
first array is larger than the median of the second array

        else

            return Median(arr1, mid1, r1, arr2, l2, mid2);  //Recursion if the median of the
second array is larger than the median of the first array

    }

    //The Median method that receives two parameters an array and n value
```

```
        static double Median(int[] arr, int n)

        {

            if (n % 2 == 0) //If the length of the received array is even

                return (double) (arr[n / 2] + arr[n / 2 - 1]) / 2;

            else // If the length of the received array is odd

                return arr[n / 2];

        }

    }
```

4. Challenges faced and how we tackled them:

- We faced a challenge when determining the strategy to use to solve the Brute Force Algorithm, but then we used merge sort in executing the problem and it was quite simple and easy to understand.
- We also faced a challenge when dividing the array in the Divide and Conquer approach because when dividing we used indexes, but then we solved the problem by ceiling.

CSC311 Project
Phase 2

5. Text Formatting explanation:

To solve the problem, we need to calculate extra space by the provided formula we can calculate the extra space for each line
**Note: the extra space can be a negative number.*

$$M - j + i - \sum_{k=i}^{j} L$$

contains words i through j, where $i \le j$

Then we compute costs of all possible lines in a 2D table cost[][], The value cost [i][j] indicates the cost to put words from i to j in a single line where i and j are indexes of words in the input sequences.

cost[i , j] $= \infty$	If extra[i , j] < 0	Word dose not fit
cost[i , j] = 0	If j=n and extra[i , j] $\ge$ 0	Last line cost 0
cost[i , j] = extra[i, j]^3		

We need to try all the different possibilities after calculating the cost to find the minimum cost from 1 to j, thus we can define total where total is an optimized solution:

Total[0] = 0	If j = 0
Total[j] = Minimum (total[i-1]+cost[i , j]) where $1 < I \le j$	If j > 0

The method printSolution() uses p[] to print the solution, its keep track of what words go on what line, so we can keep a parallel p array that points to where each total value came from. The last line starts at word p[n] and goes through word n. The previous line starts at word p[p[n]] and goes through word p[n] – 1, etc.

- **Pseudocode:**

```
//calculate extra spaces in a single line.
for  i = 1 to n;
      extra[ i , i ]= M – l [i-1]

      for j = i+1 to n
        extra[ I , j ] = extra[ i  , j-1] - l[ j-1] - 1

// Calculate  cost
    for i = 1 to  n

      for j = i to n
        if  extra[ i , j ] < 0
          cost[ i , j ] = ∞
        else if j == n && extra[i][j] >= 0
          cost[ i , j ] = 0
        else
          cost[ i , j ] = $extra[ i ,j ]^3$

// Calculate minimum cost and find minimum cost arrangement.

    total[0] = 0;
    for j = 1 to n
      total[j] = ∞
      for i = 1 to j
        if total[i-1] != ∞ AND cost[ i , j ]  != ∞ AND (total[i-1] + cost[ i , j ] < total[j])
          total[j] = total[i-1] + cost[ i , j ]
          p[j] = i

return total and p
```

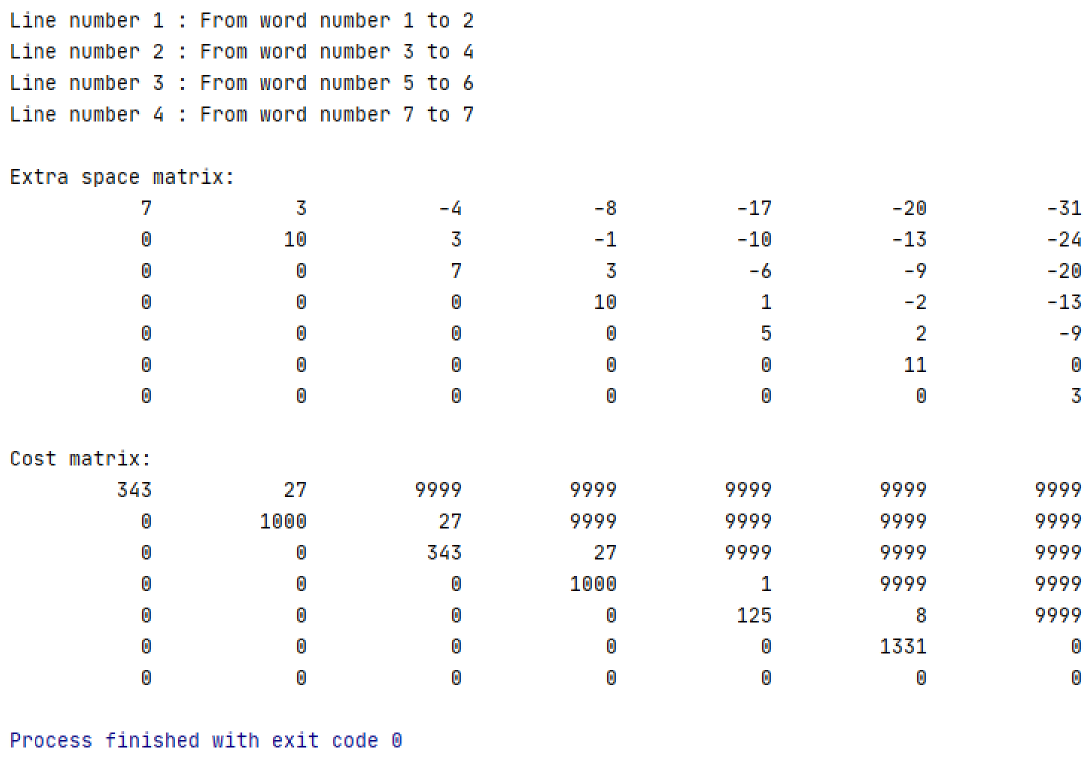
- **Time and space complexity:**

Time complexity: O(n^2)
space complexity: O(n^2)

- **Source code:**

```
public class TextFormating{

    int INF = 9999;

    int[][] extra;
    int[][] cost;
    int[] total;
    int[] p;

    //To print the solution
    int print(int[] arr, int n){
        int s;

        if(arr[n] == 1)
            s=1;
        else
            s=print(arr, arr[n]-1) + 1;

        System.out.println("Line number" + " " + s + " : " + "From word number" +" "+ arr[n] + "
" + "to" + " " + n);
        return s;}


    // l[] represents lengths of different words in input sequence. For example, int l[] =
{6,3,6,3,8,2,10}; represents the given sentence "CSC311 The Design and Analysis of Algorithms".
// n is size of l[], and M is line width: (maximum no. of characters that can fit in a line)
    void solution(int[] l, int n, int M){

        extra =   new int[n+1][n+1]; // extra[i][j] will have number of extra spaces if words
from i to j are put in a single line
        cost =   new int[n+1][n+1]; // cost[i][j] will have cost of a line which has words from i
to j
        total =   new int[n+1];      // total[i] will have total cost of optimal arrangement of
words from 1 to i
        p =   new int[n+1];      // p[] is used to print the solution.

//calculate extra spaces in a single line.
        for (int i = 1; i <= n; i++){
            extra[i][i]= M - l[i-1];

            for (int j = i+1; j <= n; j++)
                extra[i][j] = extra[i][j-1] - l[j-1] - 1;}

// Calculate  cost
        for (int i = 1; i <= n; i++){

            for (int j = i; j <= n; j++){
                if (extra[i][j] < 0)
                    cost[i][j] = INF;
                else if (j == n && extra[i][j] >= 0)
                    cost[i][j] = 0;
                else
                    cost[i][j] = extra[i][j] * extra[i][j] * extra[i][j];
            }

        }

// Calculate minimum cost and find minimum cost arrangement.
        total[0] = 0;
        for (int j = 1; j <= n; j++){
            total[j] = INF;
            for (int i = 1; i <= j; i++){
                if (total[i-1] != INF && cost[i][j] != INF && (total[i-1] + cost[i][j] <
total[j])){
                    total[j] = total[i-1] + cost[i][j];
                    p[j] = i;}}}
        print(p, n);}


    public static void main(String args[]){
        TextFormating t = new TextFormating();
        int[] l = {6, 3, 6,3,8,2,10};
        int n = l.length;
        int M = 13;
        t.solution (l, n, M);
        System.out.println();
        System.out.println("Extra space matrix:");
        t.printMatrix(t.extra);
        System.out.println();
        System.out.println("Cost matrix:");
        t.printMatrix(t.cost);
    }

    public void printMatrix(int[][] matrix){
        for (int i = 1; i < matrix.length; i++) {

            for (int j = 1; j < matrix[i].length; j++) {
                System.out.printf("%12d ",matrix[i][j] );

            }
            System.out.println();
        }
    }
    }
```

- **Sample run:**



```
Line number 1 : From word number 1 to 2
Line number 2 : From word number 3 to 4
Line number 3 : From word number 5 to 6
Line number 4 : From word number 7 to 7

Extra space matrix:
           7            3           -4           -8          -17          -20          -31 
           0           10            3           -1          -10          -13          -24 
           0            0            7            3           -6           -9          -20 
           0            0            0           10            1           -2          -13 
           0            0            0            0            5            2           -9 
           0            0            0            0            0           11            0 
           0            0            0            0            0            0            3 

Cost matrix:
         343           27         9999         9999         9999         9999         9999 
           0         1000           27         9999         9999         9999         9999 
           0            0          343           27         9999         9999         9999 
           0            0            0         1000            1         9999         9999 
           0            0            0            0          125            8         9999 
           0            0            0            0            0         1331            0 
           0            0            0            0            0            0            0 

Process finished with exit code 0
```

- **Challenges:**

We believe that we faced a little problem in the beginning, on how to decide to implement the code. and then using an analysis skill we figure out how to do it in the right way, we also retuned to several books that describe the algorithm which was hard to understand at first.

- **references**:

Cormen, T., Leiserson, C., Rivest, R. and Stein, C., 2022. *Introduction to Algorithms, Fourth Edition*. Cambridge: MIT Press.

Winston, W. and Roe, A., 1997. *User's guide for LINDO and LINGO, Windows versions to accompany Operations research : applications and algorithms, third edition and Introduction to mathematical programming : applications and algorithms, second edition*. Belmont, CA [etc.]: Duxbury Press.