Fatih Selim Yakar
161044054

QUESTION 1

a) Prove by induction.

$k = i + 1 \text{ is given.}$
$\text{For inductive step assume } k = i + n$
$\text{Then we should } k + 1 = i + n + 1 \text{ prove.}$

$\text{All above these then we get:}$

Assumption:
$A[i, j] + A[k, j + 1] \leq A[i, j + 1] + A[k, j]$

Given:
$A[k, j] + A[k + 1, j + 1] \leq A[k, j + 1] + A[k + 1, j]$

$\text{Then we convert above to these:}$

$$A[i, j] + A[k, j + 1] + A[k, j] + A[k + 1, j + 1] \leq A[i, j + 1] + A[k, j] + A[k, j + 1] + A[k + 1, j]$$

We achieve:
$A[i, j] + A[k + 1, j + 1] \leq A[i, j + 1] + A[k + 1, j]$

b) I use the $A[i, j] + A[i + 1, j + 1] \leq A[i, j + 1] + A[i + 1, j]$ rule. I check the all appropriate element for this rule, if does not fit the element then it takes the difference between them after adds and completes the missing part.

```
function oneChangeToSpecial(matrix)
begin function
        for i = 0 to length of matrix-1
        begin for
                for j = 0 to length of matrix[0]-1
                begin for
                        leftside = matrix[i][j] + matrix[i+1][j+1]
                        rightside = matrix[i][j+1] + matrix[i+1][j]
                        if leftside > rightside
                        begin if
                                diff=leftside-rightside
                                matrix[i][j+1]+=diff
                        end if
                end for
        end for
end function
```

c) I implement merge-sort for all of the rows and creates new array appending 0^{th} elements of sorted rows.

d) MergeSort(arr[], l, r)
If r > l
1. Find the middle point to divide the array into two halves:
middle m = (l+r)/2
2. Call mergeSort for first half:
Call mergeSort(arr, l, m)
3. Call mergeSort for second half:
Call mergeSort(arr, m+1, r)
4. Merge the two halves sorted in step 2 and 3:
Call merge(arr, l, m, r)

$T(n/2)$ come from 2^{nd} part.
$T(n/2)$ come from 3^{rd} part
$\vartheta(n)$ come from 4^{th} part

so the recurrence relation for merge-sort is: $\boldsymbol{2T(n/2) + \vartheta(n)}$

the for loop cycle is: number of row(r)
n = number of column(c)

after that the total recurrence relation is: $\boldsymbol{r(2T(c/2) + \vartheta(c))}$

QUESTION 2

I find the elements in the k/2th indexes in two arrays and evaluate them according to these elements. Let's say k/2th element calls current, arr1's size size1 arr2's size size2 and these array's indexes are index1 and index2. Then I make a recursive call depending on whether the current-1>= size1-index1 and current-1 >= size2-index2 and so on. Thus, in each recursive call, the sub problems are reduced to half in sum of two arrays.

So if the first array's size m and second array's size n, then the maximum value of traverse can take m+n. Thus the worst running time case is worst case running time is $O(\log_2(m+n))$ because of the continuously dividing by 2 the main problem .

QUESTION 3

I used an approach similar to merge sort to find the maximum contiguous subset. The approach as follows:

First: Divide array into two parts.
Second: Return the maximum sum of these:
Recursively found maximum sum in left part.
Recursively found maximum sum in right part.
Sum of subarray constructed to exceed the midpoint.

The main approach is this:

Find the maximum sum middle point to any point on left of mid, then find the maximum sum mid+1 to any point on right of mid+1. And then merge two and return this. Then apply this sum as recursive for the whole array and find the result.

So I can calculate the complexity like bottom:

- Finding recursively sum of left half of the array's worst case is $T(n/2)$
- Finding recursively sum of right half of the array's worst case is $T(n/2)$
- Merge these two: $\Theta(\mathrm{n})$

Total recurrence is $2T\left(\frac{n}{2}\right) + \Theta(\mathrm{n})$ so according to master theorem $a = 2, b = 2, d = 1$
The complexity is $\boldsymbol{W(n) = \theta(nlogn)}$

QUESTION 4

I used to DFS algorithm and graph coloring in this part. DFS is a decrease by constant algorithm. So the sub problem's size reduced by some constant on each loop or recursive calls. Graph coloring is a way of marking graph nodes. Nodes connected directly to each other are marked with a different color so that we can control 2 clusters with different colors. In terms of these I combined the two approach like this: Mark with the coloring approach while traversing DFS. The graph is not bipartite if there exists an edge connecting current node to a other colored node with the same color. In contrast, the condition is bipartite.

The worst complexity should be normally sum of $m + n$ because of the dfs search (n is number of node and m is number of edges). But I used to adjacency matrix(so m is near n^2) in code because of this the worst case complexity is $\boldsymbol{O(n^2)}$

- For four element number of iteration:

```
Is bipartite?:True
16
```

- For five element number of iteration:

```
Is bipartite?:True
25
```

QUESTION 5

I used the merge sort algorithm in this part. First I find linearly the gains of given cost and price in days. I constructed the 2d gain array first dimension is normally gains and second dimension is the day that it belongs in gain. And then I merge sorted according to first dimension (gains). Last I return the last element's 2th element in gain array. Thus I provide return the maximum gain's day.

Fatih Selim Yakar
161044054

I found the recurrence relation of merge sort in the first part d: $2T(n/2) + \vartheta(n)$

So if I apply the master theorem in this relation($a = 2, b = 2, d = 1$)

The complexity is $\boldsymbol{W(n) = \theta(nlogn)}$

The algorithm includes merge sort + for loop 0 to n-1

So the total complexity is: $\boldsymbol{\theta(nlogn + n)}$