Introduction to Algorithm Design
Hamza Yoğurtcuoğlu
171044086 - Hw4

Question 1 :

Part a)

if A[i,j] + A[i+1,j+1] <= A[i,j+1] + A[i+1,j] is provided with **special array**.

A[i,j] + A[k,j+1] <= A[i,j+1] + A[k,j] The base case -> **k = i + 1** is given. As for the inductive step, we assume it holds for **k = i + n** and I will prove for **k + 1 = i + n + 1**.

Additivity

1) A[i,j] + A[i+1,j+1] <= A[i,j+1] + A[i+1,j]

2) + A[i,j] + A[k+1,j+1] <= A[i,j+1] + A[k+1,j]

A[i,j] + A[i+1,j+1] + A[i,j] + A[k+1,j+1] <= A[i,j+1] + A[i+1,j] + A[i,j+1] + A[k+1,j]

So, A[i,j] + A[k+1,j+1] <= A[i,j+1] + A[k+1,j]

Part b)

Note : Python implementation exists in question1b.py

Pseudo code :

```
function changeSinglePoint2DArray(array):
    distance = 0
    points = []  # that keeps 2 point in order to fixing special array
    while i=1 to array_row -1 do
        while  j=1 to array_column -1 do
            if array(i,j) + array(i+1,j+1) > array(i,j+1) + array(i+1,j) then
                right side coordinate assigned to points array
                distance=array(i,j)+array(i+1,j+1)-array(i,j+1)- array(i+1,j)
                breaks all loops
            end if
        end while
    end while
    array[point[0]] += distance
    special_Array = True # When first point is changed, already array is not
             special so, second point must be changed
    while i=1 to array_row -1 do
        while  j=1 to array_column -1 do
            if array(i,j) + array(i+1,j+1) > array(i,j+1) + array(i+1,j) then
                special_Array = False
            end if
        end while
    end while
    if not specialArray
        array[point[1]] += distance
    end if
end
```

Part c)

I separate 2 matrix our special matrix recursively. Then I am finding leftmost minimum elements in each even rows. If height/step is 1 that mean is row of matrix is one . I directly calculate the minimum element in matrix. If there is more rows in one matrix. I divide two according to even matrix and odd matrix. So I calculate minimum elements divide even rows and odds rows.

Part d)

As I explained above we divide the matrix by even and odd lines.
So that means is recurrence → **T(m/2) Note: m is row**
Then I merged divided steps that takes : **O(m+n)**

Recurrence Relation : $T(m) = T(m/2) + c_1.n + c_2.m$
$T(m/2) = T(m/4) + c_1.n + (c_2.m)/2$
....
....

If we write T(m/2) in T(m).

$T(m) = T(m/4) + c_1.n + c_2.m + c_1.n + (c_2.m)/2$

If I continue

T(m) = $\sum_{i=0}^{logm-1} (c_1 * n + \frac{(c_2 * m)}{2^i})$

$T(m) = c_1.n.logm + c_2.m.2$

$$T(m) \in O(nlogm+m)$$

Question 2:

We compare the middle elements of arrays A and B,I said middle elements of array which are midA and midB . If begin and end is equal that means cannot be kth in A. As the same for B

```
if (beginA == endA):
    return B[k]
if (beginB == endB):
    return A[k]
```

If A[midA] is k, we understand that midB cannot be the kth element. Then we set the last element of A to be B[mid2]. So , we divide 2 sub arrays according to which mid element is bigger than others.

Else side if midA+midB is bigger than kth point. We undertand that we reduce array A or B because kth is smaller than mids of sum.

```
if (mid1 + mid2 < k):
    if (A[mid1] > B[mid2]):
        return kthElement(A,B, beginA,beginB + mid2 + 1,endA,endB,k - mid2 - 1)
    else:
        return kthElement(A,B, beginA+mid1+1,beginB,endA,endB,k - mid1 - 1)
else:
    if (A[mid1] > B[mid2]):
        return kthElement(A,B, beginA,beginB,beginA+mid1,endB,k)
    else:
        return kthElement(A,B, beginA,beginB,endA,beginB+mid2,k)
```

Note : Firstly ,I found kth elements witout sorted array.

```
First Sorted Array :  [4, 9, 11, 15, 100]

Second Sorted Array :  [1, 2, 8, 16, 990, 50000]

kth element :  4

[1, 2, 4, 8, 9, 11, 15, 16, 100, 990, 50000]
```

Worst Case :

Kth elements can be last value for two sorted arrays that meas is (m+n). I divide subarray with 2 case m and n. That takes logm and logn.

Let's think we select 1th. I will sum middle points. Then I will check **mid1+mid2<k** that is bigger than **k.**

So I will take subarray every time. I will rid of the right side of middle of the array. But that can be A array or B array. That can be n.m subarray. I will take logarithms

$$T = \log n.m$$
$$= \log n + \log m$$

$$T \in O(\log n + \log m)$$

Question 3 : I divided 2 arrays which are left and right array according to middle of array. Then I call recursively in order to find maximum subarray sum in left half , maximum subarray sum in right half and maximum subarray sum such that the subarray crosses the midpoint. My algorithm is finding the maximum sum starting from mid point and ending at some point on left of mid, then find the maximum sum starting from mid + 1 and ending with sum point on right of mid + 1. Finally, combine the two and return.

Worst Case :

Note : That works like mergesort :
The given list is divided into two part. In the integration, the sum of left and right subsets are calculated separately with recursive. **2T(n/2)**
The complexity of merge operation is O(n) since the sum of both side is calculated and the total element number is n. **O(n)**

$$T(n) = 2T(n/2) + \Theta(n)$$

Using Master Theorem:

$$a = 2 \ , b = 2 \ , d = 1$$

$$a = b^d$$

$$T \in O(nlogn)$$

Question 4 :

I created a U and V array to store all vertices knowledge according to U or V. Vertex number is used as index in this array. Value -1 of setUV array is used to indicate that no in any set(U or V) is assigned to vertex i. The value 1 is used to indicate V set is assigned and value 0 indicates V set is assigned. I used bread first search for traversal in graph. If any U vertices has U neighbor that is said. The Graph is not bipartite graph. If each U and V sets of vertices has just different set of vertices. It can be bipartite graph.

Worst Case :

My algorithm is decrease and conquer (**Decrease by a Constant**)
The size of an graph is reduced by the same constant on each iteration of the algorithm. This constant is equal to one.

If V is the number of vertices and E is the number of edges of a graph, then the time complexity for my algorithm can be expressed as **O (|V|+|E|)**. Having said this, it also depends on the data structure that we use to represent the graph.

But I used adjacency matrix if we use the adjacency matrix, then the time complexity is

$$O(Vertex^2) == O((U+V)^2)$$

Bipartite Graph

```
[0, 1, 0, 1, 0, 1]
[1, 0, 1, 0, 1, 0]
[0, 1, 0, 1, 0, 1]
[1, 0, 1, 0, 1, 0]
[0, 1, 0, 1, 0, 1]
[1, 0, 1, 0, 1, 0]

Yes

[0, 0, 0, 0]
[0, 1, 1, 0]
[0, 1, 1, 0]
[0, 0, 0, 0]

Yes
```

NOT Bipartite Graph

```
[0, 1, 1, 0]
[1, 0, 0, 1]
[1, 1, 1, 1]
[0, 0, 0, 0]

No
```

Question 5:

I divide 2 part of array according to middle point in my algorithm. I divided until one day remaining. Then I returned the day. Then I checked which day has bigger gain. Then I returned its index of day. If there is no a day make money. I print its day on the screen.

Worst Case :

Every time I divide 2 part the array. So takes recurrence relation→
2T(n/2)
Then I calculate a gain for a day.
O(1)

T(n) = 2T(n/2) + O(1)
Master Theorem Using
a = 2 , b = 2 , d = 0

$$a>b^d$$

$$T \in O(n)$$

```
Cost :  [5, 11, 2, 21, 5, 7, 8, 12, 13, '-']
Price :  ['-', 1, 9, 5, 21, 7, 13, 10, 14, 20]

REPORT! ZERO MAKE MONEY DAY -> 4 .day

REPORT! ZERO MAKE MONEY DAY -> 1 .day

REPORT! ZERO MAKE MONEY DAY -> 2 .day

('The Best Day To Buy Goods -> 9', 'Gain -> 7')
---------------------------------
---------------------------------
Cost :  [1, 11, 5, 21, 5, 7, 8, 12, 13, '-']
Price :  ['-', 7, 9, 5, 21, 17, 13, 10, 14, 20]

REPORT! ZERO MAKE MONEY DAY -> 4 .day

REPORT! ZERO MAKE MONEY DAY -> 2 .day

('The Best Day To Buy Goods -> 5', 'Gain -> 12')
---------------------------------
```