CSE 321 Homework 4

Question 1:

a) $A[i,j] + A[i+1,j+1] \leq A[i,j+1] + A[i+1,j]$ (first equality)

$$\Downarrow$$

$A[i,j] + A[k,j+1] \leq A[i,j+1] + A[k,j]$ (second equality)

where $i<k$.

We can use induction rules for the prove. If we think the two equality, we can see $k=i+1$ case. The inductive steps, we assume it holds for $k=i+n$ and we want to prove it for $k+1=i+n+1$. If we add the

$A[i,j] + A[k,j+1] \leq A[i,j+1] + A[k,j]$ (after equality for first)

$A[k,j] + A[k+1,j+1] \leq A[k,j+1] + A[k+1,j]$ (after eq. for second)

$A[i,j] + A[k,j+1] + A[k,j] + A[k+1,j+1] \leq$
$A[i,j+1] + A[k,j] + A[k,j+1] + A[k+1,j]$ } I added the equalities.

Then:

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

b)

```
procedure findSpecial ( Arr[] ):
    for i ← 0 to (len(Arr)-1) do
        for j ← 0 to (len(Arr[i])-1) do
            firstEq ← Arr[i][j] + Arr[i+1][j+1]
            secondEq ← Arr[i][j+1] + Arr[i+1][j]
            if ( firstEq > secondEq) then
                Arr[i][j+1] += (firstEq - secondEq)
                findSpecial(Arr)
            end if
        end for
    end for
end
```

I traversed in all array elements for finding not suitable element.

I used the equality for finding : $\underbrace{A[i,j] + A[i+1,j+1]}_{\text{first Eq.}} \leq \underbrace{A[i,j+1] + A[i+1,j]}_{\text{second Eq}}$

Then I checked the equality for adding to the suitable element.

I used recursion because of checking the equality.

I wrote a python code and I test for the code. File name is Q1-b.py //

c) Python code file is in Q1-c.py //

I seperated the array to two parts as even number rows and odd number rows. I didn't use even number rows because I should use odd number rows for finding left most minimum element.
I wrote 2 for loop, and one while loop. for loops for traversing the array and while loop for finding equality.
I used given equality for finding array elements.
I tested my code for two array.

d) $T(m,n) = T(m/2, n) + O(m+n)$

If we think my code, we can obtain the recurrence relation.

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

2) We can think like binary search and we can slice the arrays into two halves at each step. The index for the middle element might be different for each array so we should find different middle indexes.
We should compare both of arrays middle elements.
Binary time complexity is $O(\log k)$. k is array size.
So our algorithm complexity will be:

$T(m,n) \in O(\log m + \log n)$ // (m = first array size, n = second array size) //

Python filename is Q2.py //

3) The problem is about finding maximum crossing subarray so I used the algorithm.

Subarray for $A[low..mid]$ $\Rightarrow$ $low \leq i \leq j \leq mid$

Subarray for $A[mid+1..high]$ $\Rightarrow$ $mid < i \leq j \leq high$

Crossing mid $\Rightarrow$ $low \leq i \leq mid < j \leq high$

FindMaxCrossingSubarray function complexity is $T(n) \in O(n)$.

We can write our recurrence relation as:

$$T(n) = \begin{cases} O(1) & \text{if } n = 1 \\ 2T(n/2) + O(n) & \text{if } n > 1 \end{cases}$$

Python filename is Q3.py //

We can use master's Theorem:

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

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

$c \ ?\ \log_b a \quad 1 = \log_2 2$ $\rightarrow$ $T(n) \in O(n \log n)$ //

4) I used the coloring method for finding bipartite or not. I used array for the storing data. I write some comments for the my algorithm. Also I wrote two test graph for testing my algorithm.

$$T(n) = \underbrace{T(n-1)}_{\text{isBipartiteHelper}} + \underbrace{O(2n)}_{\text{isBipartite}}$$

$T(n) = T(n-1) + 2n$

$T(n) = T(n-2) + 2n + 2(n-1)$

$T(n) = T(n-2) + 2n + 2n - 2$

$T(n) = T(n-3) + 2n + 2n - 2.1 + 2n - 2.2$

$T(n) = T(n-3) + 3.2n - 2.1 - 2.2$

$T(n) = T(n-4) + 3.2n - 2.1 - 2.3 + 2n - 2.3$

$T(n) = T(n-4) + 4.2n - 2.1 - 2.2 - 2.3$

$T(n) = T(n-k) + k.\frac{k}{2}n - \frac{k}{2}(1+2+3..+k-1) \rightarrow \frac{(k-1)(k-2)}{2} \Rightarrow \frac{k^2-3k+2}{2}$

$T(n) = T(1) + n.\frac{n}{2}.n - \frac{n}{2}.\left(\frac{n^2}{2} - \frac{3n}{2} + 1\right)$

$T(n) = 1 + \frac{n^3}{2} - \frac{n^3}{4} + \frac{3n^2}{4} - \frac{n}{2}$ $\quad T(n) \in O(n^3)$ //

Python filename is Q4.py //

5) I though like binary search so firstly I sort [low..mid] then I sort [mid+1..high] elements. I used an array for storing difference between cost and price. The array name is lst and stored gain amount. I found most gain in the lst.

$T(n) = 2T(n/2) + (n-1)$

Master's Theorem;

$a = 2 \quad b = 2 \quad c = 1 \quad c \, ? \, \log_b^a \quad 1 = \log_2^2$

$T(n) \in O(n\log n)$ //

Python filename is Q5.py //