Introduction to Algorithm Design
HW4 Report

Question 1:

a)

First prove the "if" part;

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

for i < k

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

The base case of k = i + 1 is given. As for the inductive step, we assume it holds for k = i + n and we want to prove it for $k + 1 = i + n + 1$. If we add the given to the assumption, we get that;

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

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

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

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

b)

```
convert_special_array(arr)
    temp
    change_num is 0
    for 0 to end of arr length:
        for 0 to end of arr[0] length:
            assin arr[i][j] + arr[i + 1][j + 1]) - (arr[i][j + 1] + arr[i + 1][j] to val
            add val to temp
            if val > 0:
                change_num increase 1
            end if
        end for
    end for
    if change_num equal to 0:
        return arr, 2
    end if
    num is 0
    for 0 to end of temp length:
        for 0 to end of temp[0] length:
            if temp[i][j] > 0:
                if __helper(temp[i][j], i, j, arr) == true:
                    if num == 0:
                        num increase 1
                    end if
                    else:
                        return arr, 0
                    end else
                end if
                else:
                    return arr, 0
                end else
            end if
    return arr, 1
```

```
__helper(value, i, j, arr):

        if __checker(arr, i, j, value * (-1)) == true:

                arr[i][j] decrease with value

                return true

        end if

        if __checker(arr, i, j +1, value ) == true:
                arr[i][j+1] increase with value

                return true

        end if

        if __checker(arr, i+1, j, value * (-1)) == true:
                arr[i+1][j] decrease value

                return true

        end if

        if __checker(arr, i+1, j+1, value ) == true:
                arr[i+1][j+1] decrease with value

                return true

        end if

        return false


__check(self, arr, i, j, value):

        if i - 1 biger or equal 0 and j - 1 bigger or equal 0:

            if ( arr[i - 1][j - 1] + (arr[i][j] + value) ) - ( arr[i - 1][j] + arr[i + 1][j - 1] ) is biger 0:

                return False

             end if

        end if

        if i + 1 smaller than length of arr and j + 1 smaller than length of arr[0]:

            if ( arr[i + 1][j + 1] + (arr[i][j] + value) ) - ( arr[i + 1][j] + arr[i][j + 1] ) is biger 0:

                return False

             end if
        end if


        if i + 1 smaller than length of arr and j - 1 bigger or equal 0:

            if ( arr[i + 1][j - 1]+(arr[i][j] + value) ) - ( arr[i][j - 1] + arr[i + 1][j - 1] ) is smaller than 0:

                return False

            end if
        end if
```

```
if i - 1 bigger or equal 0 and j + 1 smaller than length of arr[0]:
    if ( arr[i - 1][j + 1] + (arr[i][j] + value) ) - ( arr[i - 1][j] + arr[i][j + 1] ) is smaller 0:
        return False
    end if
end if

return True
```

In this alorithm convert_special_array() function takes an array and use this formula $A[i,j] + A[i+1,j+1] \leq A[i,j+1] + A[i+1,j]$ and create list of formulas result. If list element bigger than 0 it calls __helper() function. __helper() function call __check() function for every value of element which bigger than 0. __check() function search for changeable value. If it can find it returns true and __helper() function returns true. If it cant find it returns false and __helper() function returns false. After all of this convert_special_array() function returns result.

c)

In this algorithm find_leftmost_min() function takes an array and find left most minimum value of all rows. Firstly it create a list of index number. After that it calls __find_even_and_odd_rows() function two times. __find_even_and_odd_rows() function find even and odd rows and return list of rows. find_leftmost_min() function calls __find_row_leftmost_min() two times for odd and even rows. It finds left most minimum value of all rows and append to list. After all that find_leftmost_min() function returns list of left most minimum value of rows.

d)

T (n) = T (n/2) + cm + dn

= cm + dn + cm / 2 + dn + cm / 4 + dn + ….

= $\sum_{i=0}^{\log m-1} dn + \sum_{i=0}^{\log m-1} \frac{dm}{2^i}$

= dn logm + cm $\sum_{i=0}^{\log m-1}$ < dn logm +2cm

= O(m + n log m).

Question 2:

In this algorithm kth_element() function takes two list and find k th element of this list combine. Function find element recursivly. Evrey time it check which list element smaller and increase smaller list index after that call itself. And every time it divides k value in half. It does this until it finds the k th element. The maximum value K can receive is m + n. This means its worst case is O(log(m + n)). Because every time it divides the k in half.

Worst Case : O(log(m + n))

Question 3:

In this algorithm given list is divided into two part and problem is solved for each part recursively. After that the results are combined. In combine part, the sum of left and right subsets are calculated separately using built-in sum function at first. If the two subsets are contiguous, then the sum of left and right subsets calculated. If the sum is greater than or equal to both subset sum, the two subsets are merged. If the subsets are not neighbor, the sum of elements are calculated from beginning of the left subset to end of the right subset. If the range sum is greater than or equal to both subset sum, the two subsets are merged. If these two cases are not satisfied, then the subset that has the largest sum returned. The return value of this function is tuple in which includes beginning and end of the subset. The complexity of dividing is O(logn) since the list is divided into two part. The complexity of merge operation is O(n) since the sum of both side is calculated and the total element number is n. Complexity of this problem is O(nlogn).

Question 4:

In this algorithm is_bipartite() function takes a graph and vertex number of graph.
It creates color list of vertexes of graph. It assigns -1 all vertex and call __helper() function. __helper() function check combined vertexes have same color. If they have return false. It calls itself for checks all vertexes and assign new color to vertexes. If there is no combined same color it returns true.

Its time complexity is O(V^2).

best case = average case = worst case because function should control every vertex.

So; Worst case = O(V^2)

Question 5:

In this part I design a divide-and-conquer algorithm that finds the best day to buy the goods. In this algorithm given lists are divided into two part. Algorithm search for best day in each part recursively. Algorithm check every time which part is greater than the other part and return the greater part. Algorithm take 1 extra array which take indexes of the other array. Algorithm return best day and gain of best day. If arrays have 1 element algorithm calculate gain and return result.

So;

$T(1) = O(1)$

Every time arrays divide two and there are two recursive call in algorithm and our base case does only O(1) work.

$T(n) = 2T(n / 2) + 0(1)$

$a = 2$
$b = 2$
$d = 0$

$a > b^d$
$2 > 0$

$T(n) = O (n^{\log_2 2})$
$T(n) = O(n)$