CSE 321 HW4

Elif Akgün
1801042271

① $m \times n$ array

$1 \le i \le k \le m \qquad 1 \le j \le l \le n$

$A[i,j] + A[k,l] \le A[i,l] + A[k,j] \Rightarrow$ special array

a) $A[i,j] + A[i+1,j+1] \overset{?}{\le} A[i,j+1] + A[i+1,j]$

We can prove this by using induction method.

Let $m = j + n$

$n = 1$ is our base case and $m = j + 1$

So our assumption is:

$A[i,j] + A[i+1,m] \le A[i,m] + A[i+1,j]$

We assume $m = j + n$ is true and according to induction method we have to show $m+1 = j+n+1$ is TRUE.

Then:

$$A[i,j] + A[i+1,m] \le A[i,m] + A[i+1,j]$$
$$A[i,m] + A[i+1,m+1] \le A[i,m+1] + A[i+1,m]$$

+

$$A[i,j] + \cancel{A[i+1,m]} + \cancel{A[i,m]} + A[i+1,m+1]$$
$$\le \cancel{A[i,m]} + A[i+1,j] + A[i,m+1] + \cancel{A[i+1,m]}$$

$$A[i,j] + A[i+1,m+1] \le A[i+1,j] + A[i,m+1]$$

b) As I proved in part (a), an array is special if and only if for all $i = 1, 2, \dots, m-1$ and $j = 1, 2, \dots, n-1$, we have:

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

For this disequilibrium, if $A[i,j] + A[i+1, j+1]$ is bigger then $A[i, j+1] + A[i+1, j]$, we can add $\Big(A[i,j] + A[i+1, j+1] - \big(A[i, j+1] + A[i+1, j]\big)\Big)$ to $A[i, j+1]$. And now $A[i,j] + A[i+1, j+1] = A[i, j+1] + A[i+1, j]$

This is my solution, there can be another solutions.

Pseudcode for this algorithm:

```
function SpecialArray (A[0:m-1])
    signal = True
    for i=0 to m-1 do
        for j=0 to n-1 do
            if A[i,j]+A[i+1,j+1] > A[i,j+1] + A[i+1,j] then
                A[i,j+1] = A[i,j+1] + (A[i,j] + A[i+1,j+1] -
                                      (A[i,j+1] + A[i+1,j]) )
                signal=False
            end if
        end for
    end for

    if (signal == False): then
        return SpecialArray(arr)
    else
        return A
```

* if signal = False, this mean that array is not special in the beginning. So there is chance in the array and it need to check the array again. When signal is True this means that array is special in the beginning and returns the array.

For example:

37	23	22	32
21	6	7	10
53	34	30	31
32	13	9	6
43	21	15	8

$i=0$ to $i=4$

$j=0$, to $j=3$

$i=0 \quad j=0$

$A[0,0] + A[1,1] \ ? \ A[0,1] + A[1,0]$

$37 + 6 < 23 + 21$ ✓

$i=0 \quad j=1$

$A[0,1] + A[1,2] \ ? \ A[0,2] + A[1,1]$ X → If we add 2 to 22, problem solve

$23 + 70 > 22 + 6$

$i=0 \quad j=2$

$A[0,2] + A[1,3] \ ? \ A[0,3] + A[1,2]$

$22 + 10 < 32 + 7$ ✓

⋮

After all iterations, the disequilibrium provides.

After this change, we must check new array whether it is still special array.

37	23	24	32
21	6	7	10
53	34	30	31
32	13	9	6
43	21	15	8

$24+10 < 32+7$ ✓

$23+7 = 24+6$ ✓

} It is still special array.

c) To compute leftmost minimum element of each row, we can create seperate subarrays of odd numbered rows and even numbered rows. So we divide two parts of array. Then we can compute the leftmost minimum element in odd numbered rows and even numbered rows.

To find leftmost minimum elements, specify the first element is leftmost minimum element for beginning. Then compare each element in each row, if a element is smaller then leftmost minimum element, our new leftmost min. element is it.

In my algorithm, all leftmost min. elements append an array then function returns this array.

d) $k_{i-1} \leq k_i \leq k_{i+1}$, k_i: index of leftmost minimum of the ith row.

Finding k_i takes $k_{i+1} - k_{i-1} + 1$ steps at most, for $i = 2m+1, m > 0$

$$T(a,b) = \sum_{i=0}^{\frac{a}{2}-1} (k_{2i+2} - k_{2i} + 1) = \sum_{i=0}^{\frac{a}{2}-1} k_{2i+2} - \sum_{i=0}^{\frac{a}{2}-1} k_{2i} + \underbrace{\sum_{i=0}^{\frac{a}{2}-1} 1}_{\frac{a}{2}-1+1=\frac{a}{2}}$$

$$= \underbrace{\sum_{i=1}^{\frac{a}{2}} k_{2i}}_{\to 2\cdot\frac{a}{2}=a} - \sum_{i=0}^{\frac{a}{2}-1} k_{2i} + \frac{a}{2} = \underbrace{k_a - k_0}_{n} + \frac{a}{2} = n + \frac{m}{2}$$

$$= O(\tfrac{m}{2} + n) \in O(m+n)$$

divide $\to O(1)$ conquer $\to m/2$ merge $\to m+n$

$$T(m) = T(m/2) + cn + dm = cn + dm + cn + d\frac{m}{2} + cn + d\frac{m}{4} + cn + d\frac{m}{8} + \ldots$$

$$= \sum_{i=0}^{\log m - 1}\left(cn + \frac{dm}{2^i}\right) = \sum_{i=0}^{\log m-1} cn + \sum_{i=0}^{\log m-1} \frac{dm}{2^i} = cn(\log m - 1 + 1) + dm \sum_{i=0}^{\log m-1} 1/2^i$$

$$\boxed{\in O(n\log m + m)}$$

(2) In my algorithm, arrays divided $k/2$ recursively.

k can not be less then 1 or more then $(m+n)$.

If one of arrays is empty, searching element is $[k-1]^{th}$ element of other array. Because k can not be $\emptyset$.

In this algorithm, firstly we look minimum of number of elements and $k/2$. According to $(temp-1)^{th}$ indexed-elements of arrays, we divide arrays into subarrays.

An example can help to understand:

$arr1 = [1, 12, 19, 26] \rightarrow 4$ elements $\rightarrow m=4$

$arr2 = [18, 27, 28] \rightarrow 3$ elements $\rightarrow n=3$

$temp1 = \min(m, k/2) = \min(4, 5/2) = \min(4, 2) = 2$

$temp2 = \min(n, k/2) = \min(3, 2) = 2$

$$\underbrace{arr[\overset{2}{temp1} - 1]}_{1} \; ? \; \underbrace{arr2[\overset{2}{temp2} - 1]}_{18}$$

$$1 < 18$$

$\Rightarrow$ return findKthElement (newArr2, arr2, k - temp1)

$$\underbrace{newArr2}_{19,26}, \quad \underbrace{arr2}_{18,27,28}, \quad \underbrace{k - temp1}_{5-2=3}$$

— — — — —

$arr1 = 19, 26$ $\quad arr2 = 18, 27, 28$ $\quad k=3$

$temp1 = \min(2, 1) = 1$

$temp2 = \min(3, 1) = 1$

$$\underbrace{arr1[1-1]}_{19} \; ? \; \underbrace{arr2[1-1]}_{18}$$

$$19 > 18$$

$\Rightarrow$ findKthElement (arr1, newArr2, k - temp2)

$$\underbrace{arr1}_{19,26}, \quad \underbrace{newArr2}_{27,29}, \quad \underbrace{k - temp2}_{3-1=2}$$

— — — —

arr1 = 19, 26   arr2 = 27, 29   k = 2

temp1 = min (2, 1) = 1

temp2 = min (2, 1) = 1

$\underbrace{\text{arr1[1-1]}}_{19}$? $\underbrace{\text{arr2 [1-1]}}_{26}$

19 < 26

=> findKthElement (newArr2, arr2, k - temp1)

$\downarrow$ 26 $\downarrow$ 27, 29 $\downarrow$ 2 - 1 = 1

---

arr1 = 26   arr2 = 27, 29   k = 1

k = 1 => return   min (arr1[0], arr2[0])

$\downarrow$ 26 $\downarrow$ 27

=> $\boxed{\text{return 26}}$

We divide the arrays k/2 recursively. Finding kth element takes log k time. And k can take m+n time.
So the worst case is O(logk) = O(log(m+n)).

(3) In this problem, if all elements in the array are positive, find the all contiguous subset and return the subarray that has largest sum. For example:

arr = {3, 5, 1, 7}

subarray that has largest sum is {3, 5, 1, 7} so max sum is 16.

But if all elements in the array are not positive, our subarray which has largest sum is in anywhere.

To find this subarray, we will use divide and conquer method. It follows these steps:

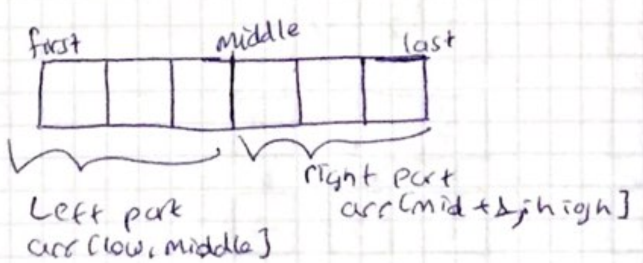
→ Divide the array two part : left part : right part

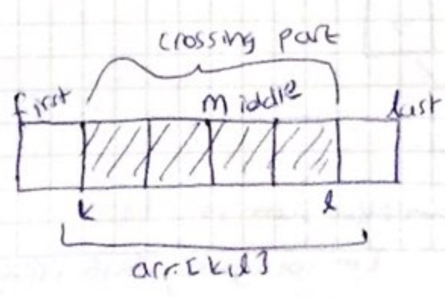
→ Find max. subarray sum in left part.

→ Find max. subarray sum in right part.

→ Find max subarray sum in the middle which crosses the midpoint.

→ Find maximum of these sums.





contigues max. subarray can be three locations:

- in the arr[first, middle]
- in the arr[middle+1, last]
- in the arr[k, l]

An example can help to understand:

arr = [-5, 3, 7, 9, 8, 1, -2, -5]

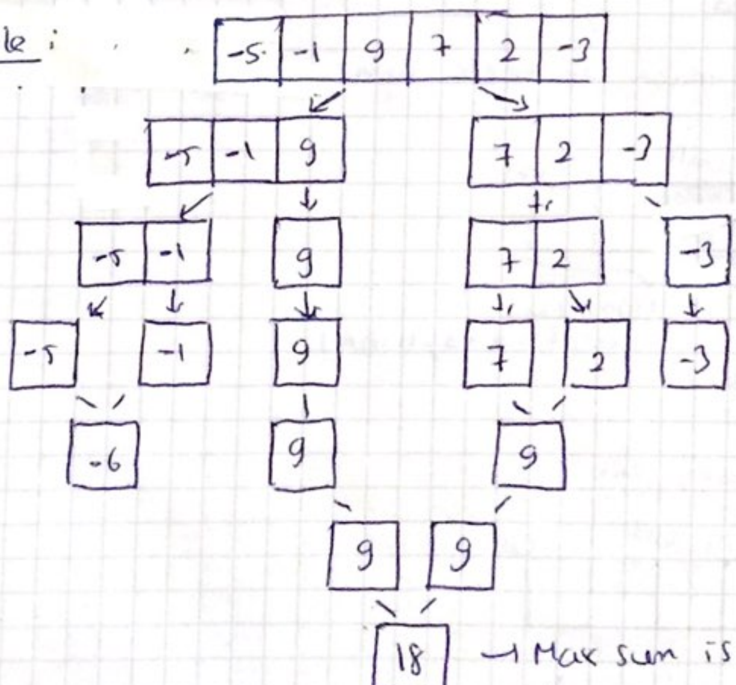
for left part, divide and conquer technique returns <u>19</u> that sum of elements of arr[1,3]

for right part, divide and conquer technique returns <u>9</u> that sum of elements of arr[4,5]

But also arr[1,5] is maximum contiguous subarray which cross the midpoint.

So the answer is [3, 7, 9, 8, 1] and its sum of elements is <u>28</u>.

<u>Example</u>:



→ Max sum is 18.
(in contigues max subarray)

After find the contigues max subarray sum, we will find contigues max subarray. To do this, we will start $\emptyset$ th index and variable sum is $\emptyset$. After each iteration, sum = sum + arr[i] and if sum = max_subarray_sum the signal is false. It means that subarray is found.

(e) **Complexity Analysis**

→ This algorithm divides the array two parts. Since the combine step requires a scan from the middle index of arr to the first and to the last, a linear term is added this step, so recurrence is:

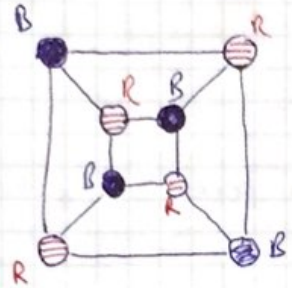
$T(n) = 2T(n/2) + \Theta(n)$

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

$c \stackrel{?}{.} \log_b a \rightarrow 1 = \log_2 2$

$\rightarrow \boxed{T(n) = \Theta(n \log n)}$

(4) A bipartite graph is a set of graph vertices, i.e., points where multiple lines meet, decomposed into two disjoint sets, meaning they have no element in common, such that no two graph vertices within the same set are adjacent.



→ This graph is bipartite because all adjacent vertices color is different

To find wheter given graph is bipartite, first, color the start vertex 1. Then color the all adjacents of start vertex to 0. Check the adjacent vertices wheter their color is same. If it is same, return false; if it is not, return true.

→ Keep going until there is no vertex that is not visited.

* The graph has two most commonly representations:

→ Adjacency Matrix

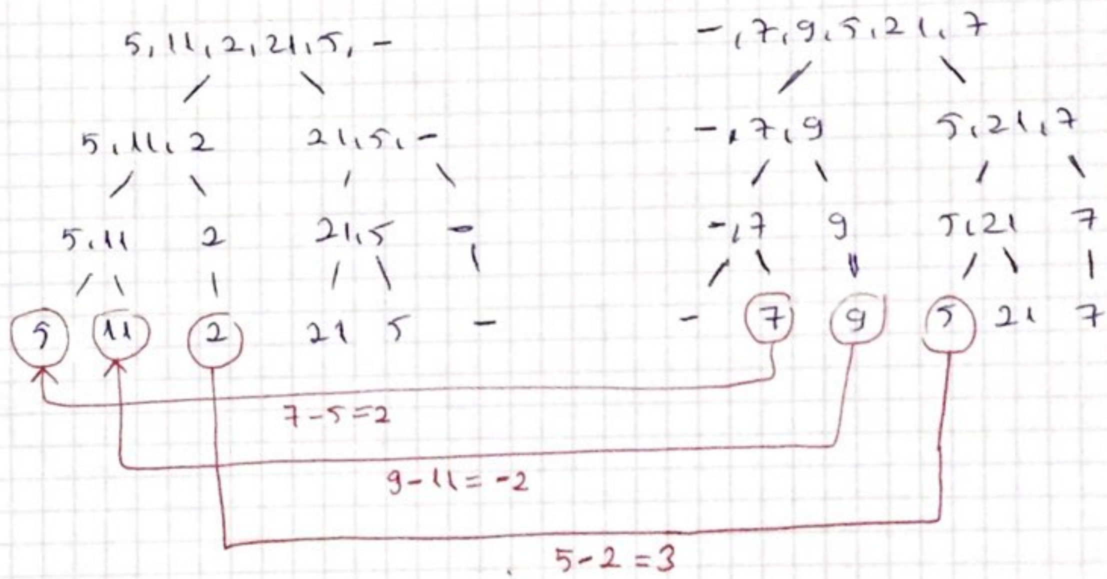
→ Adjacency List

In my algorith, I use adjacency matrix representation. While loop takes $O(V)$ time and for loop (thats in the while loop) takes $O(V)$ time.

$O(V) \cdot O(V) = O(V^2)$ time. So the worst case is: $\boxed{O(V^2)}$

(5) In the managing a warehouse problem, gain of first day is zero.(Also None). Because goods sell next day.
To find the best day to buy the goods, I use divide and conquer technique. I divide both arrays 2 part. First part starts from 0th index to middle index. Second part starts from (middle+1)th index to last index. Like this, after all recurece call, there is one element left. Then I compute the gain from the remaining elements. Then I appends this gains in an array.
An example helps to understand:

cost = [5, 11, 2, 21, 5, −]   price = [−, 7, 9, 5, 21, 7]



gain = [−, 2, −2, $\boxed{3}$, 0, 2] → 4th day is the best day.

↳ This algorithm divides the arrays into two parts. Since the combine step requires a scan from the middle index of arr to the first and to the last, a linear term is adde this step, so recurrence is:

$T(n) = 2T(n/2) + \Theta(n) \rightarrow \left.\begin{array}{l} a=2 \\ b=2 \\ c=1 \end{array}\right\} 1 = \log_2 2 \rightarrow \boxed{T(n) = \Theta(n\log n)}$