CSE 321 HOMEWORK 4

FURKAN OZEV

161044036

1) Special Array Problem:

a. Prove that an array is special:

- The "only if" part is unimportant, it follows definition of Monge array.
- As for the "if" part, let's first prove that
- Assume that array A is Monge.
- $A[i, j] + A[i+1, j+1] \leq A[i, j+1]+A[i+1, j]$
- $\Rightarrow A[i, j]+A[k, j+1] \leq A[i, j+1]+A[k, j]$, where $i < k$
- Let's prove it by induction. Set $k = i + 1$.
- Assume it holds for $k = i + n$ and prove it for $k + 1 = i + n + 1$.
- Add the given to the assumption,
- Trying to prove that for all i, j and k>i
- $A[i, j] + A[k, j+1] \leq A[i, j+1]+A[k, j]$
- The base case $k= i+1$ is true by our assumption.
- Then, $A[k, j] + A[k+1, j+1] \leq A[k, j+1]+A[k+1, j]$
- $\Rightarrow A[i, j] + A[k, j+1] + A[k, j] + A[k+1, j+1] \leq A[i, j+1] + A[k, j] + A[k, j+1] + A[k+1, j]$
- $\Rightarrow A[i, j] + A[k+1, j+1] \leq A[i, j+1] + A[k+1, j]$

b. Convert Special Array:

- PSEUDO CODE:
  1- def convert(arr):
  2- Calculate row and column
  3- While True:
     a. For i = 0 to row:
        i. For k = i+1 to row:
           1. For j = 0 to column:
              a. For l = j+1 to column
                 i. Find if the condition of the custom array is satisfied: arr[i][j] + arr[k][l] <= arr[i][l] + arr[k][j]
                 ii. If the condition is not met:
                 iii. Keep indexes of the case that breaks the rule.
                 iv. The minimum value is calculated to meet the condition.
                 v. Amount of changing single element is increased by 1
                 vi. Break the loop if there are 1 elements that do not meet the condition
     b. If there is only 1 element that does not meet the requirement:
        i. Assign this element a minimum value that is calculated before.
     c. Else:
        i. Return arr

- EXPLANATION ALGORITHM:
  1- Arr is 2D array.
  2- It is checked whether the rule given for the special array is provided for all elements and for all situation.
  3- For this control, all values of the variables in this rule are considered.
  4- Rule: An $m \times n$ array A of real numbers is called a special array if for all i, j , k, and l such that $1 \leq i < k \leq m$ and $1 \leq j < l \leq n$, we have: $A[i, j] + A[k, l] \leq A[i, l] + A[k, j]$
  5- If there are any number of elements that break the rule, the position of this element should be kept.
  6- The minimum value required to satisfy the rule is calculated.
  7- After checking the rule for all variables, the number of states that do not satisfy the rule is checked.
  8- If there are 1 conditions that do not satisfy the rule, the calculated minimum value is assigned to this position
  9- The same process continues until the rules are satisfied for all variables.
  10- The goal here is to find the correct value of that single variable that does not satisfy the rule.

c. Leftmost Minumum Element (Divide and Conquer):

- ALGORITHM:
  1- Calculate row amount of array
  2- If row amount is 1:
     a. Find leftmost minimum number in this row
     b. Append this result in list
     c. Return this list
  3- Else:
     a. Find middle row index
     b. Divide the list into 2 parts
     c. The first part contains the rows before the middle row index. (DIVIDE PART)
     d. The other rows are in the second part. (DIVIDE PART)
     e. Recursively determine the leftmost minimum for each part.
     f. Adds the results to the list in order.
     g. Return this list

- EXPLANATION ALGORITHM:
  1- This algorithm finds the leftmost minumum element in each row.
  2- It does this by dividing the matrix by 2.
  3- Repeat until the number of rows in the matrix is 1.
  4- When the number of rows is 1, leftmost minimum element for that row is found.
  5- Then this result is returned.
  6- The results are kept in the list and the list is returned.

d. **Recurrence Relation:**

- We know that the number of rows of the matrix is m.
- Assume that adding element in list, returning list or element, and comparing elements takes O (1) time.
- Suppose it takes f(n) time to find the leftmost minimum element for a row.
- Then the recurrence relation is like this:

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

- Since all elements will be visited in the minimum leftmost item for a row, this process takes O (n) time. So f(n) = O(n)

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

  $T(m) = O(n\log m)$

2) Find k th Element Of Merged 2 Sorted Arrays (Divide and Conquer):

ALGORITHM:

- function kth (arr1, arr2, k):
  - If size of arr1 is 0:
    - Return the kth element of arr2.
  - If size of arr2 is 0:
    - Return the kth element of arr1.
  - Find the indices of the median elements of arr1 and arr2.
  - If k is bigger than the sum of arr1 and arr2's median indices:
    - If arr1's median is bigger than arr2's:
      - Arr2's first half doesn't include k th element. (Divide Part)
    - Else:
      - Arr1's first half doesn't include k th element. (Divide Part)
  - Else, k is smaller than the sum of arr1 and arr2's indices:
    - If arr1's median is bigger than arr2's:
      - Arr1's second half doesn't include k th element. (Divide Part)
    - Else:
      - Arr2’s second half doesn’t include k th element. (Divide Part)

PSEUDO CODE:

- def kth(arr1, arr2, k):
  - If length of arr1 = 0:
    - return arr2[k]
  - If length of arr2 = 0:
    - return arr1[k]
  - ia <- length of arr1 // 2
  - ib <- length of arr2 // 2
  - If ia + ib < k:
    - If arr1[ia] > arr2[ib]:
      - kth(arr1, arr2[ib + 1:], k - ib - 1)
    - Else:
      - kth(arr1[ia + 1:], arr2, k - ia - 1)
  - Else:

            - If arr1[ia] > arr2[ib]:
                - kth(arr1[:ia], arr2, k)
            - Else:
                - kth(arr1, arr2[:ib], k)

ANALYZE COMPLEXITY:

1- Calculate length of arrays and return operations take O(1) time.
2- Calculate median indices take O(1) time:
3- The function is called recursive by halving the size of one of the array.
4- The Worst case state is the division of different arrays in each function call.
    a. Assume that length of arr1 is n and length of arr2 is m.
    b. Totally for Arr1, function take O(log n)
    c. Totally for Arr2, function take O(log m)
5- Continuing this process until the found kth element take O(logn + logm)

Worst Case:

- The Worst case state is the division of different arrays in each function call.
- Continuing this process until the found kth element take O(logn + logm)

O(logn + logm)

EXPLANATION ALGORITHM:

- Arr1 and Arr2 are sorted arrays.
- We need to find kth element of merged array of sorted arrays.
- We compare the middle elements of arrays arr1 and arr2.
- Let us call these indices ia and ib respectively.
- Let us assume arr1[ia] k, then clearly the elements after ib cannot be the required element.
- We then set the last element of arr2 to be arr2[ib].
- The same situation is valid for arr1.
- In this way, we define a new subproblem with half the size of one of the arrays.
- So, there 2 conditions and 2 sub-conditions for each situations.

3) Find Largest Subset (Divide and Conquer):

ALGORITHM:

- function find (arr):
    - If size of arr is 0:
        - Return None
    - To find low and high indexes of the subset, call find_helper1 function
    - Returns a new subset based on low and high indexes.

- function find_helper1(arr, low, high):
    - If low and high same:
        - Return list of low, high elements
    - Calculate index of middle element
    - To get the left largest sum of contiguous subset, call find_helper1 function

    - To get the right largest sum of contiguous subset, call find_helper1 function
    - To combine left and right subsets, call find_helper2 function
    - Return combined subset

- function find_helper2(arr, low, right):
    - Calculate the sum of left and right part
    - If left and right subsets are contiguous:
        - Calculate the both subset sum
        - If it is greater or equal than right and left subset sum:
            - return low from left subset, high from right subset
    - Else :
        - Calculate the sum of range
        - If range sum is greater than left and right subset sum:
            - Return low and high indexes.
    - If sum of left part smallar than sum of right part:
        - Return right
    - Else :
        - Return left

ANALYZE COMPLEXITY:

1- Since the list is divided into two parts, dividing operations takes O(logn) time.
2- Let assume that the sum function takes O(n) time.
3- Since the sum of both sides is calculated and the total number of elements is n, the join operation takes O(n) time.
4- Therefore, this problem takes O(nlogn) time.

O(nlogn)

EXPLANATION ALGORITHM:

- The list is divided into two parts.
- The problem is solved separately for each partition.
- Finally, the results are combined.
- The partition is simple.
- In the combine section, the sum of the left and right subsets is firstly calculated separately using the sum function.
- If the two subsets are contiguous, then the sum of left and right subsets calculated.
- If the sum is greater than or equal to the sum of both subsets, the two subsets are combined.
- If subsets are not contiguous, the sum of the items is calculated from the beginning of the left subset to the end of the right subset.
- If the range sum is greater than or equal to the sum of both subsets, the two subsets are combined.
- If these two conditions are not met, the subset that returns the largest sum is returned.
- The return value of this function is a bundle containing the start and end of the subset.

4) Check Is A Bipartite Graph (Decrease and Conquer):

ALGORITHM:

- function isBipartite (graph, src = 0):
  - Calculate vertex amount
  - Create color array to store colors consisting of elements -1. The value '-1' to specify no color is assigned.
  - Assign first color to source. The value 1 specify Red and value 0 specify Blue.
  - Create a queue of vertex numbers
  - Append src in queue
  - Run while there are vertices in queue:
    - Return false if there is a self-loop
    - For each vertex:
      - If an edge from u to v exists and destination v is not colored:
        - Assign alternate color to this adjacent v of u.
        - Append vertex in queue
      - If an edge from u to v exists and destination v is colored with same color as u:
        - Return False
  - Return True

ANALYZE COMPLEXITY:

1- Calculate vertex amount operation takes O(1) time.
2- Create color array, initialize and assign operations take O(1) time.
3- Create queue and append operations take O(1) time.
4- For worst case, while loop turns for each vertex, this operation takes O(V) time.
   a. Traversing all vertex takes O(V) time.
      i. Control, Access, initialize, append operations takes O(1) time.
      ii. Return operation takes O(1) time.
5- Return operation takes O(1) time.

Worst Case:

T(n) = max(O(1), O(1), O(1), O(V)*[O(V) * O(1)], O(1))

T(n) = max(O(1), O(1), O(1), O(V^2), O(1))

T(n) = O(V^2)

- The complexity is O(V^2) where V is number of vertices.

O(V^2)

EXPLANATION ALGORITHM:

- This algortihm basically check whether the graph is 2-colorable.
- Assign RED color to the source vertex.
- Color all the neighbors with BLUE color.
- Color all neighbor’s neighbor with RED color.
- In this way, assign color to all corners to meet all the limitations of the coloring problem.
- While assigning colors:
- If we find a neighbor which is colored with same color as current vertex, then the graph cannot be colored with 2 vertices.

5) Gain The Best Day To Buy (Divide and Conquer):

ALGORITHM:

- function part5 (cost, price):
  - Remove the last element(because None) from the cost list. (cost <- cost[: 1])
  - Remove the first element(because None) from the price list. (price <- price [: 1])
  - Assign day as 1
  - To find the best day, call part5_helper function, and the result of this call is assigned to the res variable.
  - If the result gain is not negative:
    - Add True to the beginning of the list.
  - Else:
    - Add False to beginning of the list.
  - Return res

- Function part5_helper(cost, price, day):
  - If length of cost is 1:
    - Calculate gain
    - Return gain and day
  - The list will divide 2 part. DIVIDE PART AND RECURSIVE CALL
  - First part run with only the first elements of the lists. Day will be same. The result of this call is assigned to res1 variable.
  - Second part run with the exception of the first element of the lists. Day will be increased by 1. The result of this call is assigned to res2 variable.
  - If gain of res1 bigger than gain of res2:
    - Return res1
  - Else:
    - Return res2

ANALYZE COMPLEXITY:

1- Removing element from list takes $O(1)$ time.
2- Assignment takes $O(1)$ time.
3- Call part5_helper function takes $T_1(n)$ time.
   a. Comparison takes $O(1)$ time.
   b. If true:
      i. Calculate gain operation takes $O(1)$ time.
      ii. Return gain and day as list operation takes $O(1)$ time.
   c. Else:
      i. For first part, lists has 1 element, So takes $O(1)$ time.
      ii. For second part, lists (n-1) element, So takes $T_1(n-1)$ time.
   d. Comparison and return operation take $O(1)$ time.
4- Comparison, Adding a element to list and return operation take $O(1)$ time.

Worst Case:

$$T(n) = \max(O(1), O(1), T_1(n), O(1))$$

$$T_1(n) = \max(O(1), O(1), T_1(n-1), O(1))$$

$$T_1(n) = T_1(n-1) + O(1)$$

$T_1(n) = O(n)$

$T(n) = max(O(1), O(1), O(n), O(1))$

$T(n) = O(n)$

- The complexity is $O(n)$ where n is number of days.

$O(n)$

EXPLANATION ALGORITHM:

- In order to make the indexes of the Cost and Price lists equal, "None" elements are deleted from both lists.
- To keep track of the days, the algorithm will run in the helper function with the extra day parameter.
- The algorithm simply compares the gain of that day with the result of the next day.
- This process will continue until the last day.
- This benchmarking process will be done from the last day to the first day since it is a recursive call.
- As a result of the comparison, the day which has a big gain will be given for before day as a result.
- Finally, the result from the next day is compared with the first day's gains and returned as the result with the biggest gain.
- If the resulting return is positive, the True flag is added to the result. If not, the False flag is added.
- True flag specify that the best day to buy are found.
- False flag specify that there is no day to make money.