

Assignment 4: Algorithms using Divide and Conquer

Competitive Programming

Question 1

Aim: You are given an array of k linked-lists lists, each linked-list is sorted in ascending order. Using divide and conquer, Merge all the linked-lists into one sorted linked-list and return it.

Algorithm

```
Merge Klists (L [0..k-1])
  if k=0 return null
  while size(L) > 1
    temp ← empty list
    for i ← 0 to size(L) - 1 step 2
      list1 ← L[i]
      if i + 1 < size(L)
        list2 ← L[i+1]
      else
        list2 ← null
      merged ← MergeTwoLists(list1, list2)
      Add merged to temp
    L ← temp
  return L[0]

MergeTwoLists (list1, list2)
  dum ← new Node()
  current ← dum
  while list1 is not null and list2 is not null
    if list1.val ≤ list2.val
      current.next ← list1
      list1 ← list1.next
    else
      current.next ← list2
      list2 ← list2.next
    current ← current.next
  if list1 is not null current.next ← list1
  else current.next ← list2
  return dum.next
```

Code

```
class Solution:
    def mergeKLists(self, lists: List[Optional[ListNode]]) -> Optional[ListNode]:
        if not lists or len(lists) == 0:
            return None

        while len(lists) > 1:
            tmp = []
            for i in range(0, len(lists), 2):
                list_1 = lists[i]
                list_2 = lists[i+1] if i + 1 < len(lists) else None
                tmp.append(self.merge_lists(list_1, list_2))
            lists = tmp

        return lists[0]

    def merge_lists(self, list_1, list_2):
        node = ListNode()
        ans = node

        while list_1 and list_2:
            if list_1.val < list_2.val:
                node.next = list_1
                list_1 = list_1.next
            else:
                node.next = list_2
                list_2 = list_2.next
            node = node.next

        if list_1:
            node.next = list_1
        else:
            node.next = list_2

        return ans.next
```

Time complexity

Recurrence relation:

$$T(k) = 2T(k/2) + f(k)$$

Applying Master's Theorem

$$A = 2, b = 2,$$

$$f(k) = \Theta(n)$$

Comparing a with b^d ,

$$a = 2$$

$$b^d = 2^0 = 1$$

Since $a > b^d$, standard Master Theorem suggests

$$\Theta(k^{\log_b a}) = \Theta(k^1).$$

However, because the work at each level is actually n , there are $\log(k)$ levels.

$$T(n, k) = \Theta(n \log k)$$

Screenshot of Output

The screenshot displays a coding platform interface. On the left, the 'Problem List' shows '21. Merge Two Sorted Lists' and '264. Ugly Number II'. The main area shows a solution for 'Merge Two Sorted Lists' in Python3. The code is as follows:

```
1 class Solution:
2     def mergeKLists(self, lists: List[Optional[ListNode]]) -> Optional[ListNode]:
3         if not lists or len(lists) == 0:
4             return None
5
6         while len(lists) > 1:
7             tmp = []
8             for i in range(0, len(lists), 2):
9                 list_1 = lists[i]
10                list_2 = lists[i+1] if i + 1 < len(lists) else None
11                tmp.append(self.merge_lists(list_1, list_2))
12            lists = tmp
13
14        return lists[0]
15
16    def merge_lists(self, list_1, list_2):
17        node = ListNode()
18        ans = node
19
20        while list_1 and list_2:
21            if list_1.val < list_2.val:
22                node.next = list_1
23                list_1 = list_1.next
24            else:
25                node.next = list_2
26                list_2 = list_2.next
27        node.next = list_1 if list_1 else list_2
28        return node.next
```

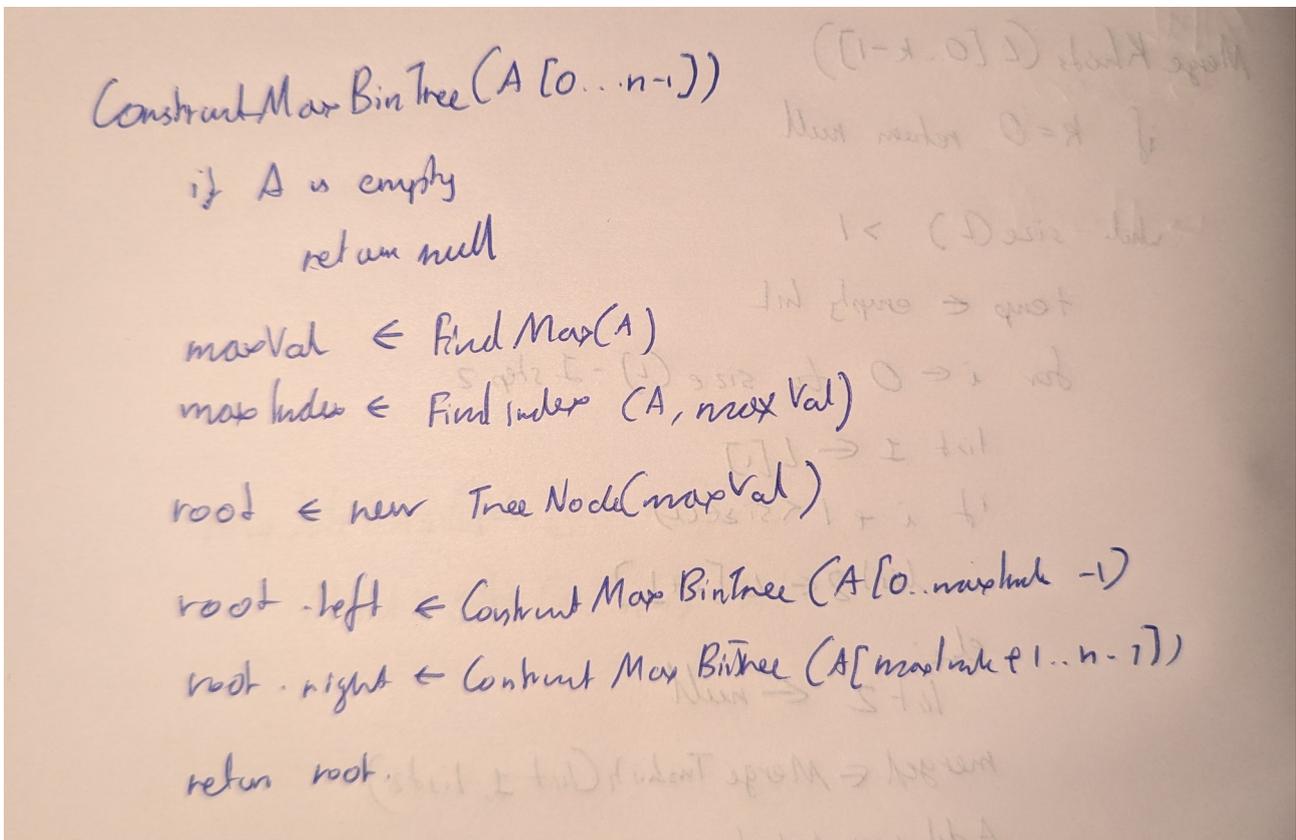
The runtime graph shows a single data point at 19ms, indicating a very fast execution. The test case result for Case 1 shows the input lists `[[1,4,5], [1,3,4], [2,6]]` and the output `[1, 1, 2, 3, 4, 4, 5, 6]`.

Question 2

Aim: You are given an integer array `nums` with no duplicates. A maximum binary tree can be built recursively from `nums` using the following algorithm:

- Create a root node whose value is the maximum value in `nums`.
- Recursively build the left subtree on the subarray prefix to the left of the maximum value.
- Recursively build the right subtree on the subarray suffix to the right of the maximum value.
- Return the maximum binary tree built from `nums`.

Algorithm



Code

```
class Solution(object):  
    def constructMaximumBinaryTree(self, nums):  
        if not nums:  
            return
```

```
element = max(nums)
index = nums.index(element)

node = TreeNode(element)

node.left = self.constructMaximumBinaryTree(nums[0:index])
node.right = self.constructMaximumBinaryTree(nums[index + 1:])

return node
```

Time complexity

Recurrence relation:

$$T(n) = T(i) + T(n - i - 1) + \Theta(n)$$

Applying Master's Theorem

A = 2, b = 2,

$$f(n) = n^1 \implies d = 1$$

Since $a = b^d$ ($2 = 2^1$), this is case 2 of Master's Theorem,

$$T(n) \in \Theta(n^d \log n) = \Theta(n \log n)$$

In the worst case,

$$T(n) = T(n - 1) + \Theta(n)$$

This is recursive summation and results in,

$$T(n) \in \Theta(n^2)$$

Screenshot of Output

The screenshot displays a coding environment with the following components:

- Problem List:** Shows '998. Maximum Binary Tree II'.
- Submission Details:** 'Accepted' status, 107/107 testcases passed, submitted by Varghese K James on Feb 10, 2026 at 23:03. Performance metrics: Runtime 31 ms (Beats 54.62%), Memory 19.53 MB (Beats 58.29%).
- Performance Graph:** A bar chart showing runtime performance across various test cases.
- Code Editor:** Contains the following Python code:

```
1 class Solution(object):
2     def constructMaximumBinaryTree(self, nums):
3         if not nums:
4             return
5
6         element = max(nums)
7         index = nums.index(element)
8
```
- Test Result:** Shows 'Accepted' status with a runtime of 0 ms. Two test cases are listed, both passed. Input: [3,2,1,6,0,5], Output: [6,3,5,null,2,0,null,null,1].
- User Profile:** A dropdown menu for 'Varghese K James' is visible, showing options like 'My Lists', 'Notebook', 'Progress', 'Points', 'Try New Features', 'Orders', 'My Playgrounds', 'Settings', 'Appearance', and 'Sign Out'.

Question 3

Aim: A city's skyline is the outer contour of the silhouette formed by all the buildings in that city when viewed from a distance. Given the locations and heights of all the buildings, return the skyline formed by these buildings collectively.

Algorithm

```
Skyscraper (B[0..n-1])
n ← length (B)
if n=0 return []
if n=1
    [L, R, H] ← B[0]
    return [[L, H], [R, 0]]
mid ← ⌊n/2⌋
leftSkyscraper ← skyscraper (B[0..mid-1])
rightSkyscraper ← skyscraper (B[mid..n-1])
return MergeSkyscrapers (leftSkyscraper, rightSkyscraper)

MergeSkyscrapers (left, right)
h1 ← 0, h2 ← 0, i ← 0, j ← 0
result ← empty list
while i < length (left) and j < length (right)
    if left[i].x < right[j].x
        x ← left[i].x
        h1 ← left[i].h
        i ← i + 1
    else if left[i].x > right[j].x
        x ← right[j].x
        h2 ← right[j].h
        j ← j + 1
    else:
        x ← left[i].x
        h1 ← left[i].h
        h2 ← right[j].h
        i ← i + 1
        j ← j + 1
maxH ← max (h1, h2)
if result is empty or result.last.h ≠ maxH
    Append [x, maxH] to result
```

Code

```
class Solution:
    def getSkyline(self, buildings: List[List[int]]) -> List[List[int]]:
        n = len(buildings)
        if n == 0: return []
        if n == 1:
            L, R, H = buildings[0]
            return [[L, H], [R, 0]]

        mid = n // 2
        left = self.getSkyline(buildings[:mid])
        right = self.getSkyline(buildings[mid:])
        return self.merge(left, right)

    def merge(self, left: List[List[int]], right: List[List[int]]) -> List[List[int]]:
        h1, h2 = 0, 0
        i, j = 0, 0
        res = []
        n_l, n_r = len(left), len(right)

        while i < n_l and j < n_r:
            if left[i][0] < right[j][0]:
                x, h1 = left[i]
                i += 1
            elif left[i][0] > right[j][0]:
                x, h2 = right[j]
                j += 1
            else:
                x, h1 = left[i]
                h2 = right[j][1]
                i += 1
                j += 1

            h = max(h1, h2)
            if not res or res[-1][1] != h:
                res.append([x, h])

        res.extend(left[i:] or right[j:])
        return res
```

Time complexity

Recurrence relation:

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

Applying Master's Theorem:

$a = 2, b = 2,$

$$f(n) = \Theta(n^1) \implies d = 1$$

Since, $a = b^d$ ($2 = 2$), it is case 2 of Master's Theorem

$$T(n) \in \Theta(n^d \log n)$$

Substituting $d = 1,$

$$T(n) \in \Theta(n \log n)$$

Screenshot of Output

The screenshot displays a coding platform interface. On the left, the submission status is 'Accepted' with 44/44 testcases passed. The runtime is 62ms (18.56% faster than others) and memory usage is 25.45 MB (59.67% less than others). A bar chart shows the distribution of runtimes for other submissions. The code is in Python3 and implements a divide-and-conquer algorithm for finding the skyline of buildings. The test result shows the input buildings and the expected output skyline.

```
class Solution:
    def getSkyline(self, buildings: List[List[int]]) -> List[List[int]]:
        n = len(buildings)
        if n == 0: return []
        if n == 1:
            L, R, H = buildings[0]
            return [[L, H], [R, 0]]

        mid = n // 2
        left = self.getSkyline(buildings[:mid])
        right = self.getSkyline(buildings[mid:])
        return self.merge(left, right)

    def merge(self, left: List[List[int]], right: List[List[int]]:
        h1, h2 = 0, 0
        i, j = 0, 0
        res = []
```

Testcase: Accepted Runtime: 0 ms

Case 1 Case 2

Input: buildings = [[2,9,10], [3,7,15], [5,12,12], [15,20,10], [19,24,8]]

Output: [[2,10], [3,15], [7,12], [12,0], [15,10], [20,8], [24,0]]

Expected: [[2,10], [3,15], [7,12], [12,0], [15,10], [20,8], [24,0]]

Learning Outcomes

- Learned about using divide and conquer algorithms to solve questions.
- Learned about optimisations done to solve competitive programming questions.