Parul University

NAAC A++ ACCREDITED UNIVERSITY

Faculty of Engineering & Technology
Batchelor of Technology

Design & Analysis of Algorithms
(303105218)

Semester – 5th

Computer Science & Engineering

Laboratory Manual

CERTIFICATE

This is to certify that

Mr. **PIYUSH BAGADI** with enrolment no. **2203031050081** has successfully completed his laboratory experiments in the **Design & Analysis of Algorithms** Laboratory (303105218) from the department of **Computer Science & Engineering** During the academic year **2024-25**

Date of Submission: .....

Staff In charge: .....

Head Of Department: .....

Content of Table

NAME –PIYUSH BAGADI
ENROLLMENT NO. – 2203031050081

6.	There are n servers numbered from 0 to n – 1 connected by undirected server-to-server connections forming a network where connections[i] = [ai, bi] represents a connection between servers ai and bi. Any server can reach other servers directly or indirectly throughthe network. A critical connection is a connection that, if removed, will make some servers unable to reach some other servers. Return all critical connections in the network in any order.						
7.	Given a grid of size NxM (N is the number of rows and M is the number of columns in the grid) consisting of '0's (Water) and '1's(Land). Find the number of islands.						
8.	Given a grid of dimension N x M where each cell in the grid can have values 0, 1, or 2 which has e following meaning: 0: Empty cell 1: Cells have fres 2. Cells have oranrotten oranges We have to determine what is the minimum time required to rot all oranges. A rotten orange at index [i,j] can rot other fresh oranges at indexes [i- 1,j], [i+1,j], [i,j-1], [i,j+1] (up, down, left and right) in unit time'						
9.	Given two strings str1 and str2 and below operations that can be performed on str1. Find minimum number of edits (operations) required to convert 'str1' into 'str2'. Insert Remove Replace, All of the above operations are of equal cost.						

10.	Minimum Path Sum" says that givena n x m grid consisting of non- negative integers and we need to find a path from top left to bottom right, which minimizes the sum of all numbers along the path.						
11.	Given string num representing a non-negative integer num, and an integer k, return the smallest possible integer after removing k digits from num.						
12.	There is a robot on an m x n grid. The robot is initially located at the top-left corner (i.e., grid[0][0]).The robot tries to move to the bottom- right corner (i.e., grid[m - 1][n - 1]). The robot can only move either down or right at any point in time. Given the two integers m and n, return the number of possible unique paths that the robot can take to reach the bottom-right corner.						

Practical – 1

Problem Statement: Write a program to determine whether the given numberis Prime or not.

Code:

```
[3] def check_prime(n: int) -> bool:
        if n < 2:
            return False
        for i in range(2, int(n**0.5) + 1):
            if n % i == 0:
                return False
        return True



n = int(input("Enter a number"))
if(check_prime(n)):
  print(f"{n} is a prime number")
else:
  print(f"{n} is not a prime number")
```

```
Enter a number31
31 is a prime number
```

Algorithm Explanation:

1. **Base Case Check**:
   - If nis less than 2, it immediately returns False, because numbers less than 2 are not prime.
2. **Prime Checking Loop**:
   - The function iterates through numbers from 2 up to the square root of n (int(n**0.5) + 1).
   - For each integer iin this range, it checks if nmodulo iequals zero (n % i ==0):
     - If true, nis divisible by iand hence not prime, so it returns False.
     - If no divisors are found in this range, it concludes that nis prime and returns True.
3. **Efficiency**:
   - The loop runs approximately up to sqrt(n)times, which significantly reducesthe number of checks compared to iterating up to n.
   - This makes the function efficient for large values of n.

Time Complexity:

1) Base case check →

* The function starts with a check if n<2. This check is O(1) a constant time operation

2) Loop Analysis →

* The loop runs from i=2 to i= int(sqrt(n))+1
* The number of iteration of loop is approx $\sqrt{n}$ because it iterates up to $\sqrt{n}$

3) Inside the loop →

* Each iteration involves a constant-time operation

Total Time complexity →

$$\text{Total} = O(1) + O(\sqrt{n}) \times O(1) = O(\sqrt{n})$$

∴ The time complexity of check-prime function is $O(\sqrt{n})$

Space Complexity:

- **O(1)**: The function uses a constant amount of space.

Practical –2

Problem Statement: Given a sorted array and a target value, return the index if the target is found. If not, return the index where it would be if it were inserted in order.

Code:

```
[13] def search_insert(nums, target):
         left, right = 0, len(nums) - 1

         while left <= right:
             mid = (left + right) // 2
             if nums[mid] == target:
                 return mid
             elif nums[mid] < target:
                 left = mid + 1
             else:
                 right = mid - 1

         return left
```

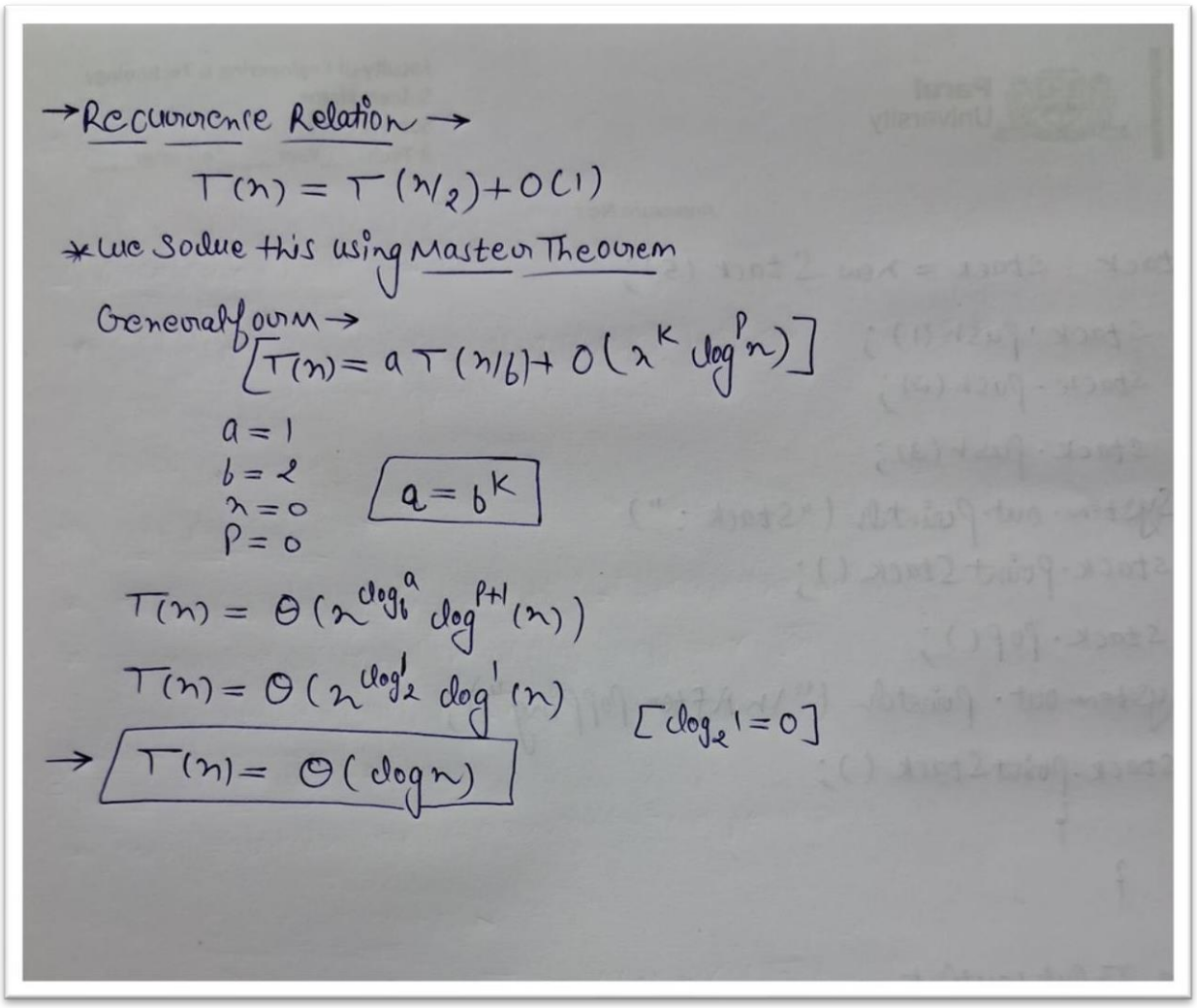
```
[17] nums = [1, 3, 5, 6]
     target = 5
     print(f"The index of {target} is {search_insert(nums, target)}")
     target = 4
     print(f"The index of {target} is {search_insert(nums, target)}")
```

```
The index of 5 is 2
The index of 4 is 2
```

Algorithm Explanation:

- Initialize two pointers, **left** and **right**, to the start and end of the array respectively (left = 0, right = len(nums) - 1).
- While left is less than or equal to right:
  - Calculate the middle index mid as mid = (left + right) // 2.
  - Compare the middle element nums[mid] with target:
    - If nums[mid] == target, return mid (target found).
    - If nums[mid] < target, move the left pointer to mid + 1 (search in the right half).
    - If nums[mid] > target, move the right pointer to mid - 1 (search in the left half).
- If the loop exits without finding the target (left > right), return the left pointer which indicates the insertion point.

Time Complexity:



Time Complexity = O(log n)

Space Complexity: O(1) since we use only a constant amount of extra space

Practical – 3

Problem Statement: There are n children standing in a line. Each child is assignerating value given in the integer array ratings.
You are giving candies to these children subjected to the

- Each child must have at least one candy
- Children with a higher rati

Return the minim
the

Code:

```
def min_candies(ratings):
    n = len(ratings)
    if n == 0:
        return 0

    candies = [1] * n


    for i in range(1, n):
        if ratings[i] > ratings[i-1]:
            candies[i] = candies[i-1] + 1


    for i in range(n-2, -1, -1):
        if ratings[i] > ratings[i+1]:
            candies[i] = max(candies[i], candies[i+1] + 1)


    total_candies = sum(candies)

    return total_candies
```

```
ratings = [1, 0, 2]
print(f"The total number of candies required is {min_candies(ratings)}")
```

```
The total number of candies required is 5
```

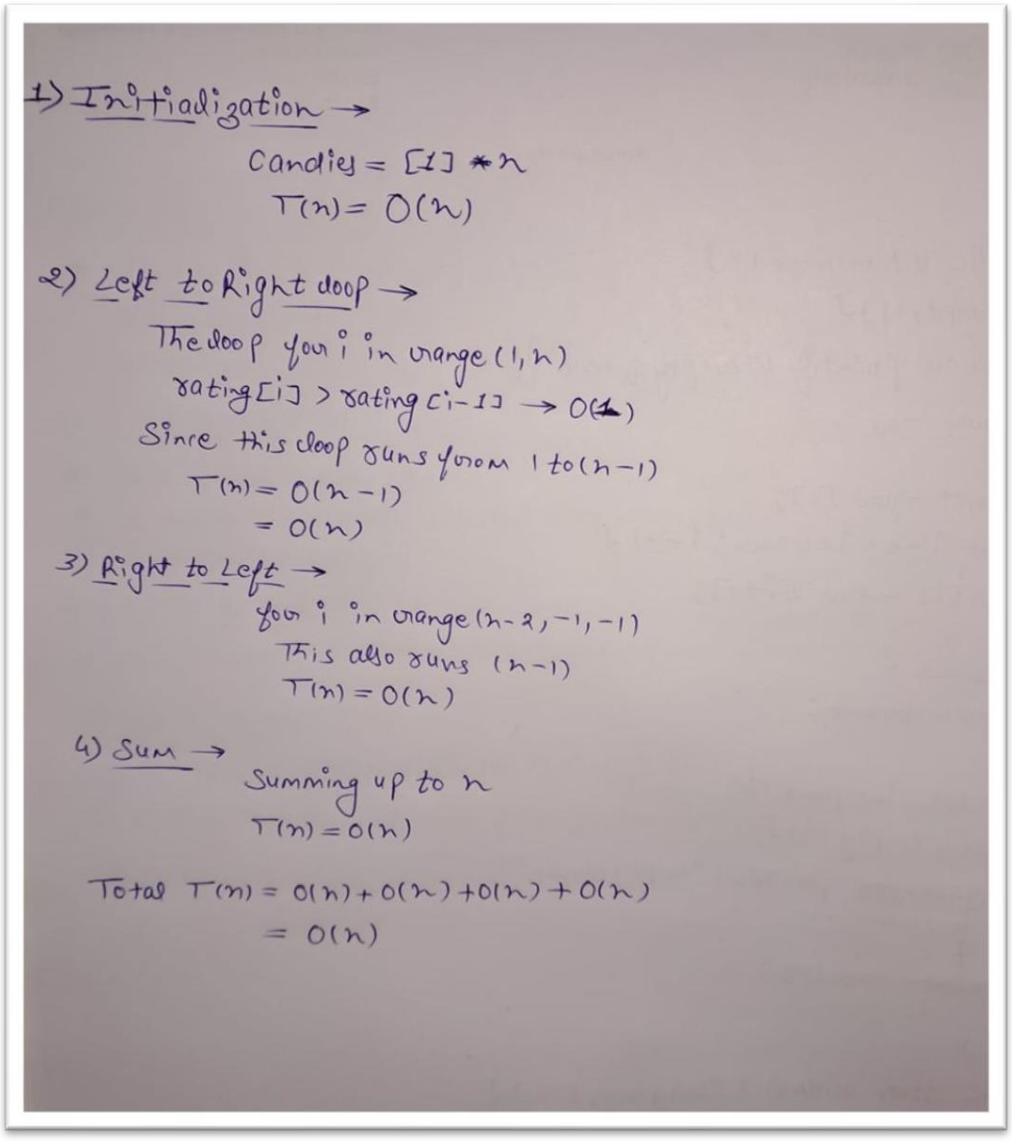
Algorithm Explanation:

1. **Initialization**:
   - Create an array candies initialized to all ones, as each child must receive at least one candy initially.
2. **First Pass (Left to Right)**:
   - Traverse the ratings array from left to right.
   - If a child's rating is greater than the previous child's rating (ratings[i] > ratings[i-1]), assign candies[i] = candies[i-1] + 1. This ensures that a child with a higher rating gets more candies than the previous child.
3. **Second Pass (Right to Left)**:
   - Traverse the ratings array from right to left.

- Adjust the candies count for each child again:
- If a child's rating is greater than the next child's rating (ratings[i] > ratings[i+1]) and the current candies count (candies[i] <= candies[i+1]), update candies[i] to ensure that the child with higher rating gets more candies than the next child.

4. **Compute Total Candies**:
   - Sum up all the values in the candies array to get the total minimum number of candies needed.

Time Complexity:



Space Complexity:

- **Input Size**: The ratings array of size n requires O(n) space.
- **Candies Array**: An additional array candies of size n is used totrack candies given to each child, also requiring O(n) space.
- **Additional Variables**: Variables like n, loop counters (i), and temporary variables (ratings[i], ratings[i-1], ratings[i+1]) occupy constant O(1) space.
- **Total Space Complexity**: Combining the above, the total space complexity is **O(n),** where n is the number of children (length of the ratings array).