# Assignment Solutions

PW SKILLS

**1. Given a sorted array of n elements and a target 'x'. Find the last occurrence of 'x' in the array. If 'x' does not exist return -1.**

- **Input 1: arr[] = {1,2,3,3,4,4,4,5} , x = 4**
- **Output 1: 6**

```
def find_last_occurrence(arr, x):
    low = 0
    high = len(arr) - 1
    last_occurrence = -1

    while low ⩽ high:
        mid = low + (high - low) // 2

        if arr[mid] == x:
            # Update last occurrence and continue searching in
the right half
            last_occurrence = mid
            low = mid + 1
        elif arr[mid] > x:
            # Search in the left half
            high = mid - 1
        else:
            # Search in the right half
            low = mid + 1

    return last_occurrence

# Example usage
arr = [1, 2, 3, 3, 4, 4, 4, 5]
x = 4
last_occurrence_index = find_last_occurrence(arr, x)

if last_occurrence_index ≠ -1:
    print(f"Last occurrence of {x} is at index:
{last_occurrence_index}")
else:
    print(f"{x} does not exist in the array.")
```

**Last occurrence of 4 is at index: 6**

**2. Given a sorted binary array, efficiently count the total number of 1's in it.**

- **Input 1: a = [0,0,0,0,1,1]**
- **Output 1: 2**

```
def count_ones(arr):
    low = 0
    high = len(arr) - 1

    # Binary search to find the first occurrence of 1
    while low <= high:
        mid = low + (high - low) // 2

        if arr[mid] == 1:
            # Move to the left subarray to find the first
occurrence
            high = mid - 1
        else:
            # Move to the right subarray
            low = mid + 1

    # The count is calculated as the remaining elements in the
array
    return len(arr) - low

# Example usage
arr = [0, 0, 0, 0, 1, 1]
ones_count = count_ones(arr)
print(f"Total number of 1's in the array: {ones_count}")
```

**Total number of 1's in the array: 2**

**3. Given a matrix having 0-1 only where each row is sorted in increasing order, find the row with the maximum number of 1's.**

**Input matrix:**

- **0 1 1 1**
- **0 0 1 1**
- **1 1 1 1 this row has maximum 1s**
- **0 0 0 0**

**Output: 2**

```
def find_max_ones_row(matrix):
    max_ones_count = 0
    max_ones_row = -1

    for i in range(len(matrix)):
        ones_count = count_ones_in_row(matrix[i])

        if ones_count > max_ones_count:
            max_ones_count = ones_count
            max_ones_row = i

    return max_ones_row + 1  # Adjusting to 1-based indexing

def count_ones_in_row(row):
    low, high = 0, len(row) - 1

    # Binary search to find the index of the first occurrence of
1
    while low <= high:
        mid = low + (high - low) // 2

        if row[mid] == 1:
            # Move to the left subarray to find the first
occurrence
            high = mid - 1
        else:
            # Move to the right subarray
            low = mid + 1

    # The count is calculated as the remaining elements in the
row
    return len(row) - low

# Example usage
matrix = [
    [0, 1, 1, 1],
    [0, 0, 1, 1],
    [1, 1, 1, 1],
    [0, 0, 0, 0]
]
max_ones_row = find_max_ones_row(matrix)
print("Row with the maximum number of 1's:", max_ones_row)
```

**Row with the maximum number of 1's: 3**

**4. Given an array of integers nums containing n + 1 integers where each integer is in the range [1, n] inclusive in sorted order. There is only one repeated number in nums, return this repeated number.**

- **Input 1: arr[] = {1,2,3,3,4}**
- **Output 1: 3**
- **Input 2: arr[] = {1,2,2,3,4,5}**
- **Output 2: 2**

```
def find_duplicate(nums):
    tortoise = nums[0]
    hare = nums[0]

    # Phase 1: Detect if there's a cycle
    while True:
        tortoise = nums[tortoise]
        hare = nums[nums[hare]]
        if tortoise == hare:
            break

    # Phase 2: Find the entrance to the cycle (the repeated
number)
    tortoise = nums[0]
    while tortoise != hare:
        tortoise = nums[tortoise]
        hare = nums[hare]

    return tortoise

# Example usage
nums1 = [1, 2, 3, 3, 4]
result1 = find_duplicate(nums1)
print(f"Output 1: {result1}")

nums2 = [1, 2, 2, 3, 4, 5]
result2 = find_duplicate(nums2)
print(f"Output 2: {result2}")
```

**Output 1: 3**
**Output 2: 2**

**5. Given a number 'n'. Predict whether 'n' is a valid perfect square or not.**

- **Input 1: n = 36**
- **Output 1: yes**
- **Input 2: n = 45**
- **Output 2: no**

```
import math

def is_perfect_square(n):
    if n < 0:
        return False  # Negative numbers are not perfect squares

    sqrt = int(math.sqrt(n))
    return sqrt * sqrt == n

# Example usage
n1 = 36
result1 = is_perfect_square(n1)
print(f"Output 1: {'yes' if result1 else 'no'}")

n2 = 45
result2 = is_perfect_square(n2)
print(f"Output 2: {'yes' if result2 else 'no'}")
```

**Output 1: yes**
**Output 2: no**

**6. You have n coins and you want to build a staircase with these coins. The staircase consists of k rows where the ith row has exactly i coins. The last row of the staircase may be incomplete.**

**Given the integer n, return the number of complete rows of the staircase you will build.**
**Example 1:**

- **Input: n = 5**
- **Output: 2**

**Explanation: Because the 3rd row is incomplete, we return 2.**
**Example 2:**

- **Input: n = 8**
- **Output: 3**

**Explanation: Because the 4th row is incomplete, we return 3.**

```
def arrange_coins(n):
    left = 0
    right = n

    while left <= right:
        mid = left + (right - left) // 2
        coins_in_mid_rows = mid * (mid + 1) // 2

        if coins_in_mid_rows == n:
            return mid
        elif coins_in_mid_rows < n:
            left = mid + 1
        else:
            right = mid - 1

    return right  # Adjust to 0-based indexing

# Example usage
n1 = 5
result1 = arrange_coins(n1)
print(f"Output 1: {result1}")

n2 = 8
result2 = arrange_coins(n2)
print(f"Output 2: {result2}")
```

**Output 1: 2**
**Output 2: 3**

**7. Write a program to apply binary search in an array sorted in decreasing order.**

```
def binary_search(arr, target):
    left = 0
    right = len(arr) - 1

    while left ⩽ right:
        mid = left + (right - left) // 2

        if arr[mid] == target:
            return mid
        elif arr[mid] > target:
            # Adjust the search to the left (decreasing order)
          left = mid + 1
        else:
        # Adjust the search to the right (decreasing order)
          right = mid - 1
    return -1  # Target not found

arr = [10, 8, 6, 4, 2, 0, -2, -4]
target1 = 6
result1 = binary_search(arr, target1)
print(f"Index of {target1}: {result1}")
target2 = -4
result2 = binary_search(arr, target2)
print(f"Index of {target2}: {result2}")
target3 = 5
result3 = binary_search(arr, target3)
print(f"Index of {target3}: {result3}")
```

**Index of 6: 2**
**Index of -4: 7**
**Index of 5: -1**

**8. You have a sorted array of infinite numbers, how would you search an element in the array?**

```
def search_infinite_array(arr, target):
    left, right = 0, 1

    while arr[right] < target:
        left = right
        right *= 2

    while left ⩽ right:
        mid = left + (right - left) // 2

        if arr[mid] == target:
            return mid
        elif arr[mid] < target:
            left = mid + 1
        else:
            right = mid - 1

    return -1  # Target not found

# Example usage
arr = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]  #
Example sorted array of infinite numbers
target = 8
result = search_infinite_array(arr, target)
if result ≠ -1:
    print("Element {} found at index {}".format(target, result))
else:
    print("Element {} not found in the array.".format(target))
```

**Element 8 found at index 7**

**9. You are given an m x n integer matrix matrix with the following two properties:**
**Each row is sorted in non-decreasing order.**
**The first integer of each row is greater than the last integer of the previous row.**
**Given an integer target, return true if target is in matrix or false otherwise.**

**You must write a solution in O(log(m * n)) time complexity.**
**Example 1:**

- **Input: matrix = [[1,3,5,7],[10,11,16,20],[23,30,34,60]], target = 3**
- **Output: true**

**Example 2:**

- **Input: matrix = [[1,3,5,7],[10,11,16,20],[23,30,34,60]], target = 13**
- **Output: false**

```
def search_matrix(matrix, target):
    if not matrix or not matrix[0]:
        return False

    m, n = len(matrix), len(matrix[0])
    left, right = 0, m * n - 1

    while left ⩽ right:
        mid = left + (right - left) // 2
        mid_element = matrix[mid // n][mid % n]

        if mid_element == target:
            return True
        elif mid_element < target:
            left = mid + 1
        else:
            right = mid - 1

    return False

# Example usage
matrix = [[1, 3, 5, 7], [10, 11, 16, 20], [23, 30, 34, 60]]
targets = [3, 13]
for target in targets:
    result = search_matrix(matrix, target)
    print("Target {} found: {}".format(target, result))
```

**Target 3 found: True**
**Target 13 found: False**

**10. There is an integer array nums sorted in non-decreasing order (not necessarily with distinct values). Before being passed to your function, nums is rotated at an unknown pivot index k (0 <= k < nums.length) such that the resulting array is [nums[k], nums[k+1], ..., nums[n-1], nums[0], nums[1], ..., nums[k-1]] (0-indexed). For example, [0,1,2,4,4,4,5,6,6,7] might be rotated at pivot index 5 and become [4,5,6,6,7,0,1,2,4,4]. Given the array nums after the rotation and an integer target, return true if target is in nums, or false if it is not in nums.**

**You must decrease the overall operation steps as much as possible.**
**Example 1:**

- **Input: nums = [2,5,6,0,0,1,2], target = 0**
- **Output: true**

**Example 2:**

- **Input: nums = [2,5,6,0,0,1,2], target = 3**
- **Output: false**

```
def search(nums, target):
    left, right = 0, len(nums) - 1

    while left ⩽ right:
        mid = left + (right - left) // 2

        if nums[mid] == target:
            return True

        if nums[left] < nums[mid]:  # Left half is sorted
            if nums[left] ⩽ target < nums[mid]:
                right = mid - 1  # Search in the left half
            else:
                left = mid + 1  # Search in the right half
        elif nums[left] > nums[mid]:  # Right half is sorted
            if nums[mid] < target ⩽ nums[right]:
                left = mid + 1  # Search in the right half
            else:
                right = mid - 1  # Search in the left half
        else:  # Handle duplicates
            left += 1

    return False  # Target not found

# Example usage
nums = [2, 5, 6, 0, 0, 1, 2]
targets = [0, 3]
for target in targets:
    result = search(nums, target)
    print("Target {} found: {}".format(target, result))
```

**Target 0 found: True**
**Target 3 found: False**