Sorting Assignment

December 26, 2023

```
[3]: #Sorting Assignment



     #Problem 1:
     #To find the element appearing the maximum number of times in an array, you can␣
      ↪use a hash table to keep track of the frequency of each element. Here's a␣
      ↪Python algorithm:


     def find_max_occurrence(arr):
         frequency = {}
         max_element = None
         max_count = 0

         for num in arr:
             if num in frequency:
                 frequency[num] += 1
             else:
                 frequency[num] = 1

             if frequency[num] > max_count:
                 max_count = frequency[num]
                 max_element = num

         return max_element

     # Example usage
     arr = [1, 2, 3, 2, 2, 1, 4, 5, 2]
     result = find_max_occurrence(arr)
     print("Element with maximum occurrences:", result)

     #Problem 2:
     #To find the missing element in a list of integers, you can calculate the␣
      ↪expected sum of the integers in the range and subtract the sum of the given␣
      ↪list from it. Here's a Python algorithm:

     def find_missing_element(arr):
```

```
    n = len(arr) + 1
    expected_sum = n * (n + 1) // 2
    actual_sum = sum(arr)
    missing_element = expected_sum - actual_sum

    return missing_element

# Example usage
arr = [1, 2, 4, 6, 3, 7, 8]
result = find_missing_element(arr)
print("Missing element:", result)



#Problem 3:
#To find the element occurring an odd number of times in an array of positive␣
 ↪numbers, you can use bitwise XOR. XORing all elements cancels out the even␣
 ↪occurrences, leaving only the element occurring an odd number of times.␣
 ↪Here's a Python algorithm:

def find_odd_occurrence(arr):
    result = 0
    for num in arr:
        result ^= num
    return result

# Example usage
arr = [1, 2, 3, 2, 3, 1, 3]
result = find_odd_occurrence(arr)
print("Element occurring odd times:", result)

#Problem 4:
#To find two elements in an array whose sum is equal to a given element K, you␣
 ↪can use a hash table to store the difference between K and each element.␣
 ↪Here's a Python algorithm:

def find_pair_with_sum(arr, K):
    complement = {}
    for num in arr:
        if num in complement:
            return complement[num], num
        complement[K - num] = num
    return None

# Example usage
arr = [1, 4, 2, 7, 11, 15]
K = 9
```

```
result = find_pair_with_sum(arr, K)
print("Pair with sum", K, ":", result)

#Problem 5:
#To find two numbers in an array such that their sum is closest to 0, you can␣
↪sort the array and then find the pair with the smallest absolute sum. Here's␣
↪a Python algorithm:

def closest_sum_to_zero(arr):
    arr.sort()
    closest_sum = float('inf')
    result = None

    left, right = 0, len(arr) - 1

    while left < right:
        current_sum = arr[left] + arr[right]

        if abs(current_sum) < abs(closest_sum):
            closest_sum = current_sum
            result = (arr[left], arr[right])

        if current_sum < 0:
            left += 1
        else:
            right -= 1

    return result

# Example usage
arr = [1, 60, -10, 70, -80, 85]
result = closest_sum_to_zero(arr)
print("Pair with closest sum to zero:", result)


#Problem 6:
#To find three elements in an array such that their sum is equal to a given␣
↪number, you can sort the array and use a two-pointer approach. Here's a␣
↪Python algorithm:


def find_triplet_with_sum(arr, target_sum):
    arr.sort()
    n = len(arr)

    for i in range(n - 2):
        left, right = i + 1, n - 1
        while left < right:
            current_sum = arr[i] + arr[left] + arr[right]

            if current_sum == target_sum:
                return arr[i], arr[left], arr[right]
            elif current_sum < target_sum:
                left += 1
            else:
                right -= 1

    return None

# Example usage
arr = [1, 4, 2, 7, 11, 15]
target_sum = 18
result = find_triplet_with_sum(arr, target_sum)
print("Triplet with sum", target_sum, ":", result)

#Problem 7:
#To find three elements i, j, k in an array such that i * i + j * j = k * k,␣
 ↪you can use a hash table to store the squares of elements. Here's a Python␣
 ↪algorithm:

def find_pythagorean_triplet(arr):
    squares = {num * num: num for num in arr}

    for i in range(len(arr)):
        for j in range(i + 1, len(arr)):
            sum_of_squares = arr[i] * arr[i] + arr[j] * arr[j]

            if sum_of_squares in squares:
                return arr[i], arr[j], squares[sum_of_squares]

    return None

# Example usage
arr = [3, 1, 4, 6, 5]
result = find_pythagorean_triplet(arr)
print("Pythagorean triplet:", result)


#Problem 8:
#An element is a majority if it appears more than n/2 times. Give an algorithm␣
 ↪takes an array of n
#element as argument and identifies a majority (if it exists).
```

```
def find_majority_element(arr):
    candidate = None
    count = 0

    for num in arr:
        if count == 0:
            candidate = num
            count = 1
        elif num == candidate:
            count += 1
        else:
            count -= 1

    # Validate if the candidate is a majority element
    if arr.count(candidate) > len(arr) // 2:
        return candidate
    else:
        return None

# Example usage
arr = [2, 2, 3, 5, 2, 2, 6]
result = find_majority_element(arr)
print("Majority element:", result)

#Problem 9:
#To find the row with the maximum number of 0's in an n × n matrix where in␣
  ↪each row, all 1's are followed by 0's, you can start from the top-right␣
  ↪corner and move left or down based on the value encountered. Here's a Python␣
  ↪algorithm:


def find_max_zeros_row(matrix):
    rows, cols = len(matrix), len(matrix[0])
    row, col = 0, cols - 1
    max_zeros_row = 0

    while row < rows and col >= 0:
        if matrix[row][col] == 0:
            max_zeros_row = row
            col -= 1
        else:
            row += 1

    return max_zeros_row

# Example usage
matrix = [
    [1, 1, 1, 0],
    [1, 1, 0, 0],
    [1, 0, 0, 0],
    [1, 1, 1, 1]
]
result = find_max_zeros_row(matrix)
print("Row with maximum 0's:", result)


#Problem 10:
#To sort an array of 0’s, 1’s, and 2’s (or R’s, G’s, and B’s) in linear time,␣
 ↪you can use the Dutch National Flag algorithm. Here's a Python algorithm:


def sort_colors(arr):
    low, mid, high = 0, 0, len(arr) - 1

    while mid <= high:
        if arr[mid] == 0:
            arr[low], arr[mid] = arr[mid], arr[low]
            low += 1
            mid += 1
        elif arr[mid] == 1:
            mid += 1
        else:
            arr[mid], arr[high] = arr[high], arr[mid]
            high -= 1

# Example usage
arr = [0, 1, 1, 0, 1, 2, 1, 2, 0, 0, 0, 1]
sort_colors(arr)
print("Sorted array:", arr)
```

```
Element with maximum occurrences: 2
Missing element: 5
Element occurring odd times: 3
Pair with sum 9 : (2, 7)
Pair with closest sum to zero: (-80, 85)
Triplet with sum 18 : (1, 2, 15)
Pythagorean triplet: (3, 4, 5)
Majority element: 2
Row with maximum 0's: 2
Sorted array: [0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 2, 2]
```

[ ]: