sorting

October 3, 2024

[ ]:
```
Sorting
```

[1]:
```
# Problem 1.Given an array of n numbers, give an algorithm which gives the␣
 ↪element appearing maximum number of times?
def find_max_occurrence(arr):
    count_map = {}

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

    max_count = 0
    max_element = None

    for num, count in count_map.items():
        if count > max_count:
            max_count = count
            max_element = num

    return max_element

arr = [1, 3, 2, 1, 2, 1, 3, 3]
result = find_max_occurrence(arr)
print(f"The element appearing maximum number of times is: {result}")
```

```
The element appearing maximum number of times is: 1
```

[3]:
```
# Problem 2 : We are given a list of n-1 integers and these integers are in the␣
 ↪range of 1 to n . There are no duplicates in the list. One of the integers␣
 ↪is missing in the list. Give an algorithm to find that element Ex:␣
 ↪[1,2,4,6,3,7,8] 5 is the missing num.
def find_missing_number(arr):
    n = len(arr) + 1
    expected_sum = n * (n + 1) // 2
    actual_sum = sum(arr)
```

```
    missing_number = expected_sum - actual_sum
    return missing_number

arr = [1, 2, 4, 6, 3, 7, 8]
missing_num = find_missing_number(arr)
print(f"The missing number is: {missing_num}")
```

```
The missing number is: 5
```

[5]:
```
# Problem 3 : Given an array of n positive numbers. All numbers occurs even␣
 ↪number of times except 1 which occurs odd number of times. Find that number␣
 ↪in O(n) time and O(1) space. Ex: [1,2,3,2,3,1,3]. 3 is repeats odd times
def find_odd_occurrence(arr):
    result = 0

    for num in arr:
        result ^= num

    return result

arr = [1, 2, 3, 2, 3, 1, 3]
odd_occurrence_number = find_odd_occurrence(arr)
print(f"The number that occurs an odd number of times is:␣
 ↪{odd_occurrence_number}")
```

```
The number that occurs an odd number of times is: 3
```

[7]:
```
# Problem 4 : Given an array of n elements. Find two elements in the array such␣
 ↪that their sum is equal to given element K.
def find_two_elements_with_sum(arr, K):
    seen = set()

    for num in arr:
        complement = K - num
        if complement in seen:
            return (num, complement)
        seen.add(num)

    return None

arr = [10, 15, 3, 7]
K = 17
result = find_two_elements_with_sum(arr, K)
if result:
    print(f"The two elements that sum to {K} are: {result[0]} and {result[1]}")
else:
    print("No two elements found that sum to the given value.")
```

```
The two elements that sum to 17 are: 7 and 10
```

[9]:
```python
# Problem 5 : Given an array of both positive and negative numbers, find two numbers such that their sum is closest to 0. Ex: [ 1 ,60 ,-10, 70, -80,85]. Ans : -80,85.
def closest_sum_to_zero(arr):
    arr.sort()

    left = 0
    right = len(arr) - 1
    closest_sum = float('inf')
    best_pair = (None, None)

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

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

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

    return best_pair

arr = [1, 60, -10, 70, -80, 85]
result = closest_sum_to_zero(arr)
print(f"The two numbers whose sum is closest to 0 are: {result[0]} and {result[1]}")
```

```
The two numbers whose sum is closest to 0 are: -80 and 85
```

[11]:
```python
# Problem 6 : Given an array of n elements . Find three elements such that their sum is equal to the given number.
def three_sum(arr, target):
    arr.sort()
    triplets = []
    n = len(arr)

    for i in range(n):
        if i > 0 and arr[i] == arr[i - 1]:
            continue

        left, right = i + 1, n - 1

        while left < right:
```

```
            current_sum = arr[i] + arr[left] + arr[right]

            if current_sum == target:
                triplets.append((arr[i], arr[left], arr[right]))

                while left < right and arr[left] == arr[left + 1]:
                    left += 1
                while left < right and arr[right] == arr[right - 1]:
                    right -= 1

                left += 1
                right -= 1

            elif current_sum < target:
                left += 1
            else:
                right -= 1

    return triplets

arr = [1, 2, -2, 0, -1, 1]
target = 0
result = three_sum(arr, target)
print(f"The triplets that sum to {target} are: {result}")
```

```
The triplets that sum to 0 are: [(-2, 0, 2), (-2, 1, 1), (-1, 0, 1)]
```

[13]:
```
# Problem 7 : Given an array of n elements . Find three elements i, j, k in the␣
 ↪array such that i * i + j * j = k*k
def find_pythagorean_triplet(arr):
    squares = {x * x for x in arr}

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

            if sum_of_squares in squares:
                k = int(sum_of_squares ** 0.5)
                if k in arr:
                    return (arr[i], arr[j], k)

    return None

arr = [3, 1, 4, 6, 5]
result = find_pythagorean_triplet(arr)
if result:
    print(f"The Pythagorean triplet is: {result}")
```

```
else:
    print("No Pythagorean triplet found.")
```

```
The Pythagorean triplet is: (3, 4, 5)
```

[15]:
```
# 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

    if candidate is not None:
        count = 0
        for num in arr:
            if num == candidate:
                count += 1

        if count > len(arr) // 2:
            return candidate

    return None

arr = [3, 3, 4, 2, 4, 4, 2, 4, 4]
result = find_majority_element(arr)
if result is not None:
    print(f"The majority element is: {result}")
else:
    print("No majority element found.")
```

```
The majority element is: 4
```

[19]:
```
# Problem 9 : Given n × n matrix, and in each row all 1's are followed by 0's.␣
 ↪Find the row with the maximum number of 0's.
def find_row_with_max_zeros(matrix):
    n = len(matrix)
    max_zeros = -1
    row_index = -1
```

```
    col = n - 1
    for row in range(n):
        while col >= 0 and matrix[row][col] == 1:
            col -= 1

        num_zeros = n - (col + 1)
        if num_zeros > max_zeros:
            max_zeros = num_zeros
            row_index = row

    return row_index, max_zeros

matrix = [
    [1, 1, 1, 0, 0],
    [1, 1, 0, 0, 0],
    [1, 0, 0, 0, 0],
    [0, 0, 0, 0, 0]
]

result = find_row_with_max_zeros(matrix)
print(f"The row with the maximum number of zeros is: Row {result[0]} with␣
 ↪{result[1]} zeros.")
```

```
The row with the maximum number of zeros is: Row 0 with 0 zeros.
```

[21]:
```
# Problem 10 : Sort an array of 0's, 1's and 2's [or R's, G's and B's]: Given␣
 ↪an array A[] consisting of 0's, 1's and 2's, give an algorithm for sorting␣
 ↪A[].The algorithm should put all 0's first, then all 1's and finally all 2's␣
 ↪at the end. Example Input = {0,1,1,0,1,2,1,2,0,0,0,1}, Output =␣
 ↪{0,0,0,0,0,1,1,1,1,1,2,}
def sort_array(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] == 2
            arr[mid], arr[high] = arr[high], arr[mid]
            high -= 1

arr = [0, 1, 1, 0, 1, 2, 1, 2, 0, 0, 0, 1]
sort_array(arr)
```

```
print("Sorted array:", arr)
```

```
Sorted array: [0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 2, 2]
```

[ ]: