Assignment_03

July 19, 2024

1 Assignment 02

1.1 Question 1

Given an integer array nums of length n and an integer target, find three integers in nums such that the sum is closest to the target. Return the sum of the three integers.

You may assume that each input would have exactly one solution.

Example 1: Input: nums = [-1,2,1,-4], target = 1 Output: 2

Explanation: The sum that is closest to the target is 2. (-1 + 2 + 1 = 2).

[ ]:
```
public static int closestSum(int[] nums, int target) {
    int closestSum = Integer.MAX_VALUE;
    for (int i = 0; i < nums.length; i++) {
        for (int j = i + 1; j < nums.length; j++) {
            for (int k = j + 1; k < nums.length; k++) {
                int currentSum = nums[i] + nums[j] + nums[k];
                if (Math.abs(currentSum - target) < Math.abs(closestSum -␣
 ↪target)) {
                    closestSum = currentSum;
                }
            }
        }
    }
    return closestSum;
}
```

[ ]:
```
int[] nums = {-1,2,1,-4};
int target = 1;
System.out.println(closestSum(nums, target));
```

2

1.2 Question 2

Given an array nums of n integers, return an array of all the unique quadruplets [nums[a], nums[b], nums[c], nums[d]] such that: 0 <= a, b, c, d < n a, b, c, and d are distinct. nums[a] + nums[b] + nums[c] + nums[d] == target

You may return the answer in any order.

Example 1: Input: nums = [1,0,-1,0,-2,2], target = 0 Output: [[-2,-1,1,2],[-2,0,0,2],[-1,0,0,1]]

[ ]:
```
public static List<int[]> fourSum(int[] nums, int target) {
    List<int[]> quadruplets = new ArrayList<>();
    Arrays.sort(nums);
    for (int i = 0; i < nums.length - 3; i++) {
        for (int j = i + 1; j < nums.length - 2; j++) {
            int target2 = target - nums[i] - nums[j];
            int left = j + 1;
            int right = nums.length - 1;
            while (left < right) {
                int sum = nums[left] + nums[right];
                if (sum == target2) {
                    quadruplets.add(new int[]{nums[i], nums[j], nums[left],␣
 ↪nums[right]});
                    left++;
                    right--;
                } else if (sum < target2) {
                    left++;
                } else {
                    right--;
                }
            }
        }
    }
    return quadruplets;
}
```

[ ]:
```
int[] nums = {1,0,-1,0,-2,2};
int target = 0;
List<int[]> quadruplets = fourSum(nums, target);
for (int[] quadruplet : quadruplets) {
    System.out.println(Arrays.toString(quadruplet));
}
```

```
[-2, -1, 1, 2]
[-2, 0, 0, 2]
[-1, 0, 0, 1]
```

1.3 Question 3

A permutation of an array of integers is an arrangement of its members into a sequence or linear order.

For example, for arr = [1,2,3], the following are all the permutations of arr: [1,2,3], [1,3,2], [2, 1, 3], [2, 3, 1], [3,1,2], [3,2,1].

The next permutation of an array of integers is the next lexicographically greater permutation of

its integer. More formally, if all the permutations of the array are sorted in one container according to their lexicographical order, then the next permutation of that array is the permutation that follows it in the sorted container.

If such an arrangement is not possible, the array must be rearranged as the lowest possible order (i.e., sorted in ascending order).

For example, the next permutation of arr = [1,2,3] is [1,3,2]. Similarly, the next permutation of arr = [2,3,1] is [3,1,2]. While the next permutation of arr = [3,2,1] is [1,2,3] because [3,2,1] does not have a lexicographical larger rearrangement.

Given an array of integers nums, find the next permutation of nums. The replacement must be in place and use only constant extra memory.

Example 1: Input: nums = [1,2,3] Output: [1,3,2]

[ ]:
```java
public static int[] swap(int data[], int left, int right) {
        // Swap the data
        int temp = data[left];
        data[left] = data[right];
        data[right] = temp;
        // Return the updated array
        return data;
    }
    public static int[] reverse(int data[], int left, int right) {
        // Reverse the sub-array
        while (left < right) {
            int temp = data[left];
            data[left++] = data[right];
            data[right--] = temp;
        }
        // Return the updated array
        return data;
    }
    public static boolean findNextPermutation(int data[]) {
        if (data.length <= 1) return false;
        int last = data.length - 2;
         while (last >= 0) {
            if (data[last] < data[last + 1]) {
                break;
            }
            last--;
        }
        if (last < 0) return false;
        int nextGreater = data.length - 1;
       // Find the rightmost successor to the pivot
        for (int i = data.length - 1; i > last; i--) {
            if (data[i] > data[last]) {
                nextGreater = i;
```

```
                break;
            }
        }
        data = swap(data, nextGreater, last);
        data = reverse(data, last + 1, data.length - 1);
        return true;
    }
```

[ ]:
```
int data[] = {1, 2, 3};
        if (!findNextPermutation(data))
            System.out.println("There is no higher" + " order permutation " +
                               "for the given data.");
        else {
            System.out.println(Arrays.toString(data));
        }
```

```
[1, 3, 2]
```

1.4 Question 4

Given a sorted array of distinct integers 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.

You must write an algorithm with O(log n) runtime complexity.

Example 1: Input: nums = [1,3,5,6], target = 5 Output: 2

[ ]:
```
public static int searchInsertPosition(int[] nums, int target) {
        int low = 0;
        int high = nums.length - 1;
        while (low <= high) {
            int mid = low + (high - low) / 2;
            if (nums[mid] == target) {
                return mid;
            } else if (nums[mid] < target) {
                low = mid + 1;
            } else {
                high = mid - 1;
            }
        }
        return low;
    }
```

[ ]:
```
int[] nums = {1,  3, 5,6};
int target = 5;
int index = searchInsertPosition(nums, target);
System.out.println(index); // 2
```

```
2
```

1.5 Question 5

You are given a large integer represented as an integer array digits, where each digits[i] is the ith digit of the integer. The digits are ordered from most significant to least significant in left-to-right order. The large integer does not contain any leading 0's.

Increment the large integer by one and return the resulting array of digits.

Example 1: Input: digits = [1,2,3] Output: [1,2,4]

Explanation: The array represents the integer 123. Incrementing by one gives 123 + 1 = 124. Thus, the result should be [1,2,4].

[ ]:
```
public static int[] incrementInteger(int[] digits) {
    int n = digits.length;
    int carry = 1;
    for (int i = n - 1; i >= 0; i--) {
        int sum = digits[i] + carry;
        digits[i] = sum % 10;
        carry = sum / 10;
    }
    if (carry > 0) {
        int[] newDigits = new int[n + 1];
        newDigits[0] = carry;
        for (int i = 1; i < n + 1; i++) {
            newDigits[i] = digits[i - 1];
        }
        return newDigits;
    } else {
        return digits;
    }
}
```

[ ]:
```
int[] digits = {1,2,3};
int[] newDigits = incrementInteger(digits);
System.out.println(Arrays.toString(newDigits));
```

```
[1, 2, 4]
```

1.6 Question 6

Given a non-empty array of integers nums, every element appears twice except for one. Find that single one.

You must implement a solution with a linear runtime complexity and use only constant extra space.

Example 1: Input: nums = [2,2,1] Output: 1

[ ]:
```
public static int singleNumber(int[] nums) {
    int result = 0;
    for (int i = 0; i < nums.length; i++) {
```

```
        result ^= nums[i];
    }
    return result;
}
```

[ ]:
```
int[] nums = {2,2,1};
int singleNumber = singleNumber(nums);
System.out.println(singleNumber);
```

```
1
```

1.7 Question 7

You are given an inclusive range [lower, upper] and a sorted unique integer array nums, where all elements are within the inclusive range.

A number x is considered missing if x is in the range [lower, upper] and x is not in nums.

Return the shortest sorted list of ranges that exactly covers all the missing numbers. That is, no element of nums is included in any of the ranges, and each missing number is covered by one of the ranges.

Example 1: Input: nums = [0,1,3,50,75], lower = 0, upper = 99 Output: [[2,2],[4,49],[51,74],[76,99]]

Explanation: The ranges are: [2,2] [4,49] [51,74] [76,99]

[ ]:
```
import java.util.*;

public class MissingRanges {

    public static List<Range> findMissingRanges(int[] nums, int lower, int␣
 ↪upper) {
        List<Range> ranges = new ArrayList<>();
        int prev = lower - 1;
        for (int i = 0; i < nums.length; i++) {
            if (nums[i] < lower) {
                continue;
            }
            if (nums[i] == prev + 1) {
                prev = nums[i];
            } else {
                if (prev + 1 != lower) {
                    ranges.add(new Range(prev + 1, nums[i] - 1));
                }
                prev = nums[i];
            }
        }
        if (prev != upper) {
            ranges.add(new Range(prev + 1, upper));
        }
```

```
        return ranges;
    }

    public static class Range {

        int start;
        int end;

        public Range(int start, int end) {
            this.start = start;
            this.end = end;
        }

        @Override
        public String toString() {
            return "[" + start + "," + end + "]";
        }
    }

    public static void main(String[] args) {
        int[] nums = {0, 1, 3, 50, 75};
        int lower = 0;
        int upper = 99;
        List<Range> ranges = findMissingRanges(nums, lower, upper);
        System.out.println(ranges); // [[2,2],[4,49],[51,74],[76,99]]
    }
}
```

1.8 Question 8

Given an array of meeting time intervals where intervals[i] = [starti, endi], determine if a person could attend all meetings.

Example 1: Input: intervals = [[0,30],[5,10],[15,20]] Output: false

[ ]:
```
import java.util.*;

public class MeetingAttendance {

    public static boolean canAttendAllMeetings(Interval[] intervals) {
        Arrays.sort(intervals, (a, b) -> a.start - b.start);
        for (int i = 1; i < intervals.length; i++) {
            if (intervals[i].start < intervals[i - 1].end) {
                return false;
            }
        }
        return true;
    }
```

```
    public static class Interval {

        int start;
        int end;

        public Interval(int start, int end) {
            this.start = start;
            this.end = end;
        }
    }

    public static void main(String[] args) {
        Interval[] intervals = new Interval[] {
            new Interval(0, 30),
            new Interval(5, 10),
            new Interval(15, 20),
            new Interval(30, 40)
        };
        boolean canAttend = canAttendAllMeetings(intervals);
        System.out.println(canAttend); // true
    }
}
```

2 Thank You!