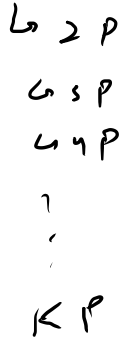
Starts @ 8:40pm

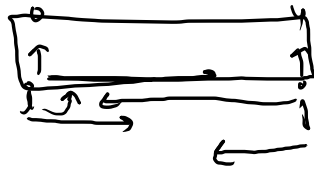
Two pointers technique



pointers → index

i , j

* Physical significance of the pointer → range / index



* How to initialize the pointers.

* How to change the pointer

* When to stop

two-sum-ii-input-array-is-sorted

Problem | Submissions | Leaderboard | Discussions

Given a 1-indexed array of integers numbers that is already sorted in non-decreasing order, find two numbers such that they add up to a specific target number. Let these two numbers be numbers[index1] and numbers[index2] where 1 <= index1 < index2 < numbers.length.

Return the indices of the two numbers, index1 and index2, added by one as an integer array [index1, index2] of length 2.

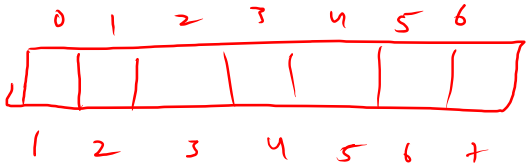
The tests are generated such that there is exactly one solution. You may not use the same element twice.

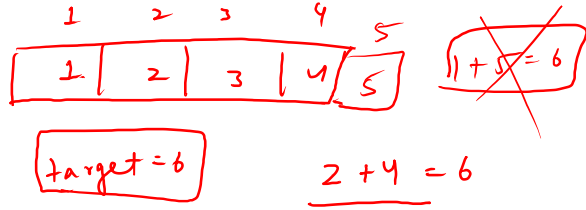
Sample Input 0

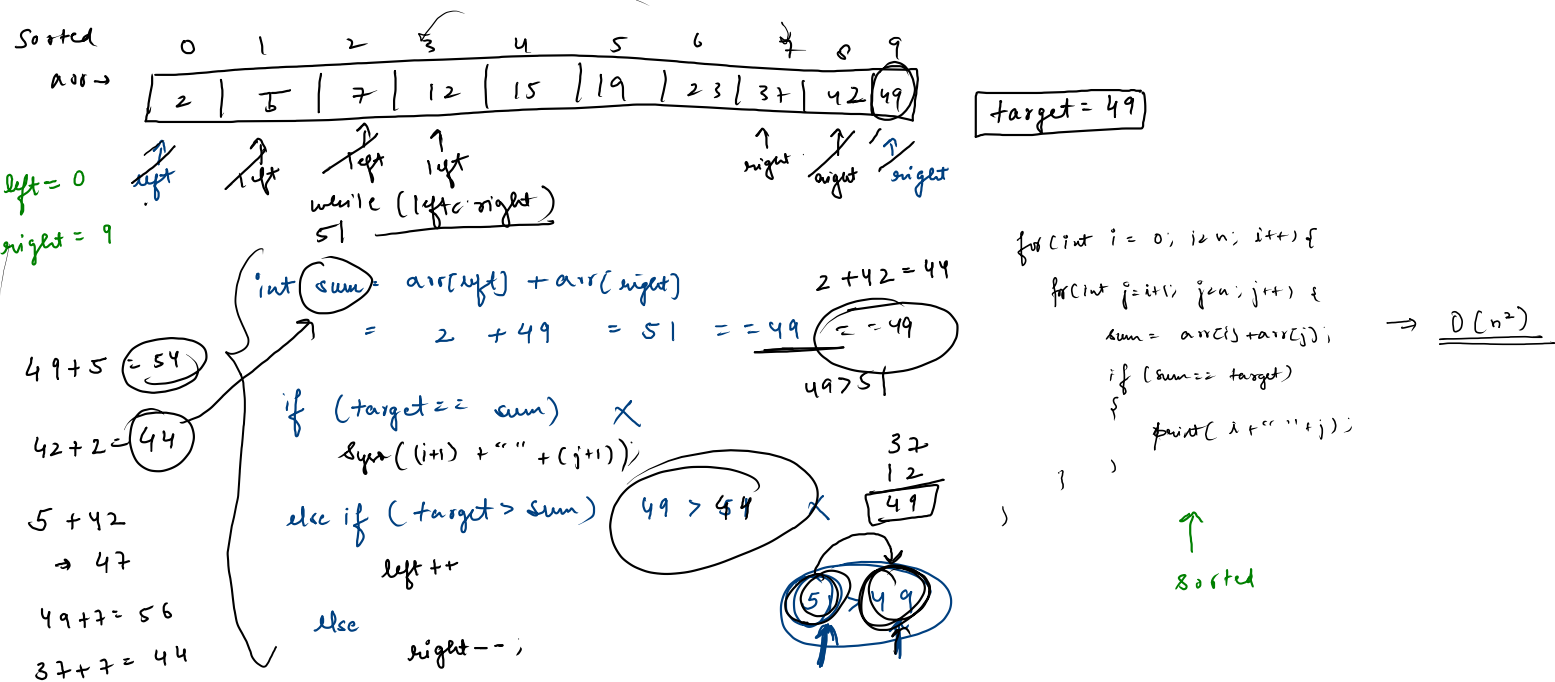
```
4
1
2
3
4
6
```

Sample Output 0

```
2 4
```






Sorted arr →
0 1 2 3 4 5 6 7 8 9
2 5 7 12 15 19 23 37 42 49
target = 49
left = 0
right = 9
left
left
left
left
right
right
right
while (left < right)
51
int sum = arr[left] + arr[right]
= 2 + 49 = 51
2 + 42 = 44
49 + 5 = 54
42 + 2 = 44
5 + 42 → 47
49 + 7 = 56
37 + 7 = 44
if (target == sum)
Syso((i+1) + " " + (j+1));
else if (target > sum)
left++
else
right--;
49 > 44
37
12
49
51
49
for(int i = 0; i < n; i++) {
for(int j = i+1; j < n; j++) {
sum = arr[i] + arr[j];
if (sum == target)
{
print(i + " " + j);
}
}
⇒ O(n²)
Sorted

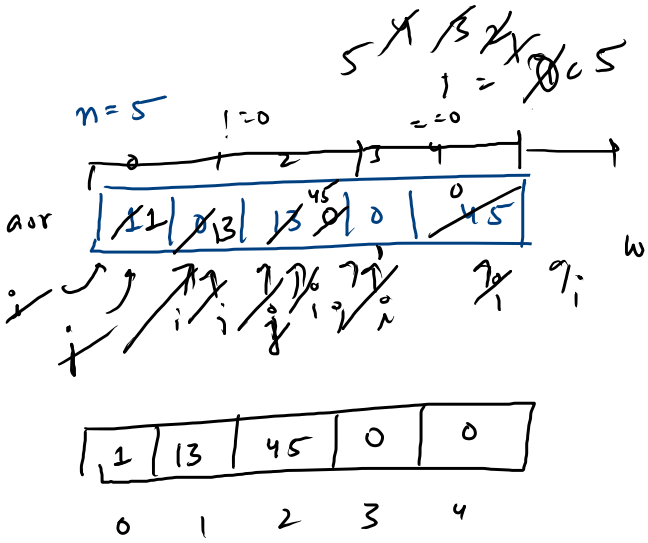
```java
public class Solution {
    public static void findTarget(int arr[],int target) {
        int left = 0;
        int right = arr.length-1;

        while(left < right) {
            int sum = arr[left] + arr[right];

            if (sum == target) {
                System.out.println((left+1) + " " + (right+1));
                return;
            } else if (sum <target) {
                left++;
            } else {
                right--;
            }
        }
    }
```

move-zeroes/

Problem | Submissions | Leaderboard | Discussions

Given an integer array nums, move all 0's to the end of it while maintaining the relative order of the non-zero elements.

Note that you must do this in-place without making a copy of the array.



i = 0, j = 0

```
while (i < arr.length) {
    if (arr[i] != 0) {
        swap(i, j)
        i++;
        j++;
    }
    else
        i++;
}
```

Sort → 0, 1

Sample Input 0

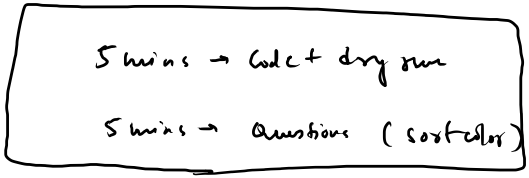
```
5
1
0
13
0
45
```

Sample Output 0

```
1 13 45 0 0
```

```java
public static void swap(int arr[], int i,int j) {
    int temp = arr[i];
    arr[i] = arr[j];
    arr[j] = temp;
}

public static void moveZeros(int arr[]) {
    int i = 0,j = 0;
    while(i<arr.length) {
        if (arr[i] != 0) {
            swap(arr,i,j);
            i++;
            j++;
        } else {
            i++;
        }
    }
}
```

```java
public static void main(String[] args) {
    /* Enter your code here. Read input from STDIN. Pri
    Scanner scn = new Scanner(System.in);
    int n = scn.nextInt();
    int arr[] = new int[n];
    for(int i=0;i<n;i++) {
        arr[i] = scn.nextInt();
    }

    moveZeros(arr);

    for(int i=0;i<n;i++) {
        System.out.print(arr[i] + " ");
    }
}
```



sort-colors/

Problem | Submissions | Leaderboard | Discussions

Given an array nums with n objects colored red, white, or blue, sort them in-place so that objects of the same color are adjacent, with the colors in the order red, white, and blue.

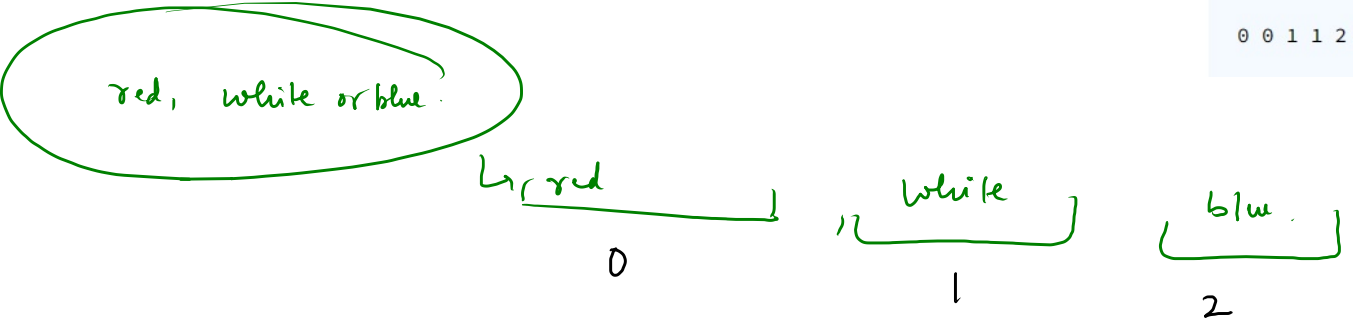
We will use the integers 0, 1, and 2 to represent the color red, white, and blue, respectively.

Sample Input 0

```
6
2
0
2
1
1
0
```

Sample Output 0

```
0 0 1 1 2 2
```

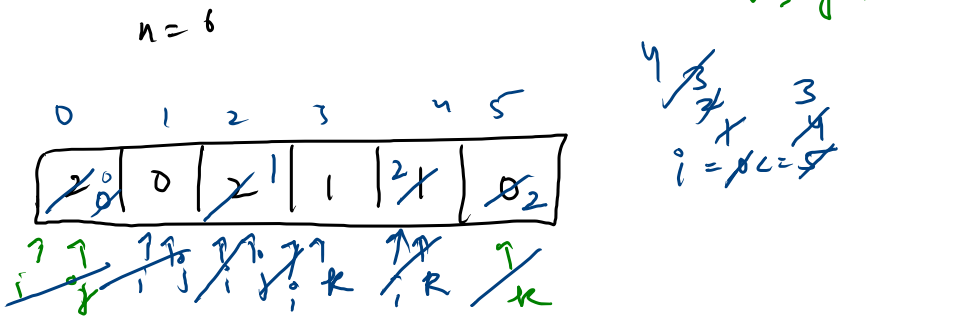


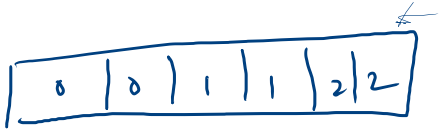
Sample Input 0

```
6
2
0
2
1
1
0
```

Sample Output 0

```
0 0 1 1 2 2
```





```
public class Solution {
    public static void swap (int arr[], int i, int j) {
        int temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
    }

    public static void sort012(int arr[]) {
        int i=0,j=0,k=arr.length-1;
        while(i <= k) {
            if (arr[i] == 0) {
                swap(arr,i,j);
                i++;
                j++;
            } else if (arr[i] == 1) {
                i++;
            } else {
                swap(arr, i,k);
                k--;
            }
        }
    }

    public static void main(String[] args) {
        /* Enter your code here. Read input from STDIN.
        Scanner scn = new Scanner(System.in);
        int n = scn.nextInt();
        int arr[] = new int[n];
        for(int i=0;i<n;i++) {
            arr[i] = scn.nextInt();
        }

        sort012(arr);

        for(int i=0;i<n;i++) {
            System.out.print(arr[i] + " ");
        }
    }
}
```

remove-duplicates-from-sorted-array/

Problem | Submissions | Leaderboard | Discussions

Given an integer array nums sorted in non-decreasing order, remove the duplicates in-place such that each unique element appears only once. The relative order of the elements should be kept the same. Then return the number of unique elements in nums.

Consider the number of unique elements of nums to be k, to get accepted, you need to do the following things:

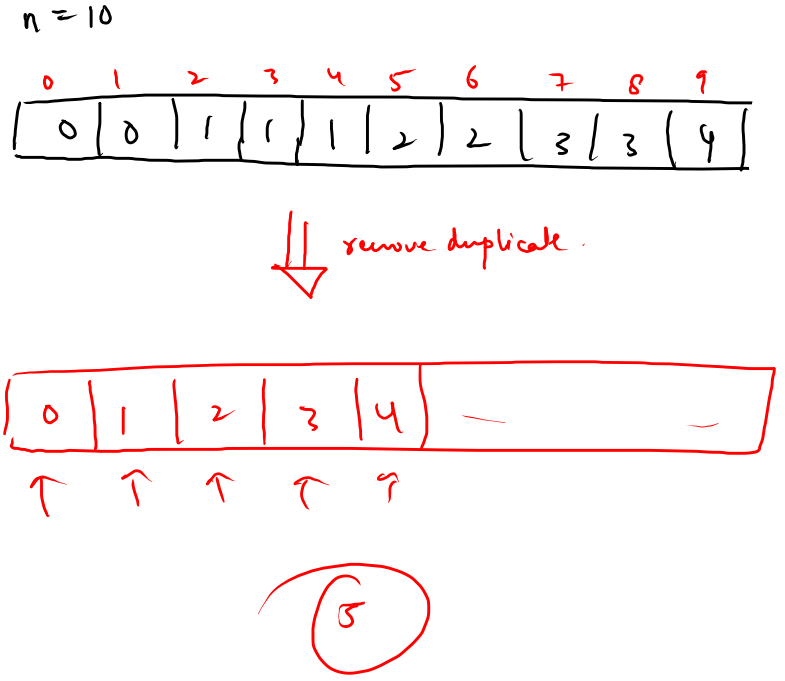
Change the array nums such that the first k elements of nums contain the unique elements in the order they were present in nums initially. The remaining elements of nums are not important as well as the size of nums. Return k.

Sample Input 0

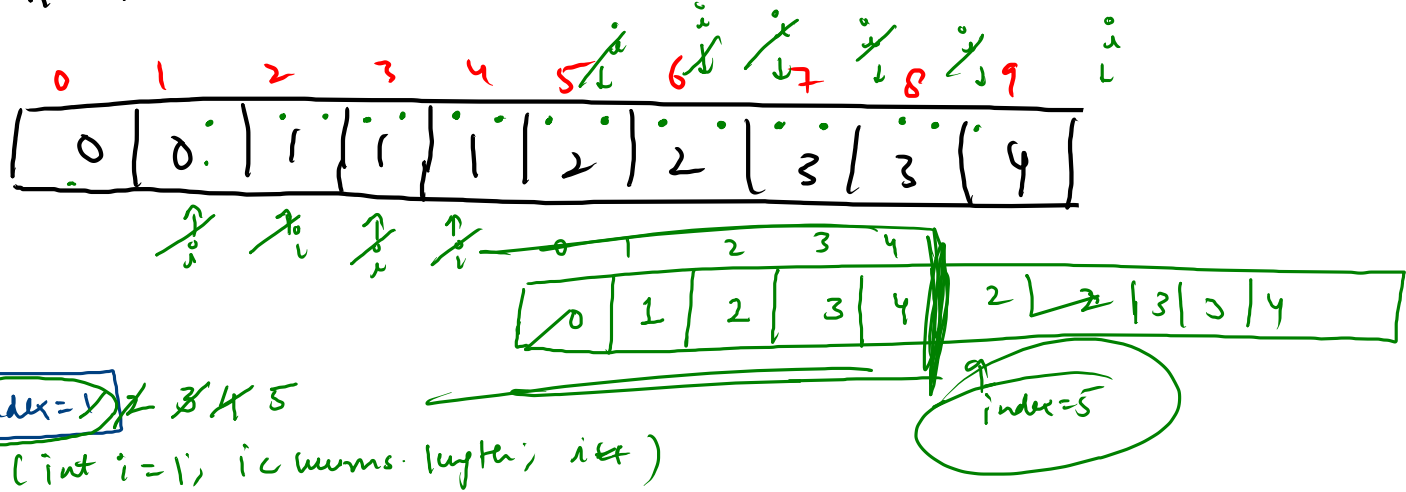
```
10
0
0
1
1
1
2
2
3
3
4
```

Sample Output 0

```
5
```



n = 10



index=1 2 3 4 5

```
for(int i=1; i<nums.length; i++)
{
    if(nums[i] != nums[i-1])
    {
        nums[index] = nums[i];
        index++;
    }
}
```

```java
public class Solution {
    public static int removeDuplicates(int arr[]) {
        int index = 1;
        for(int i=1; i<arr.length;i++) {
            if(arr[i] != arr[i-1]) {
                arr[index] = arr[i];
                index++;
            }
        }
        return index;
    }

    public static void main(String[] args) {
        /* Enter your code here. Read input from STDIN. Print out
        Scanner scn = new Scanner(System.in);
        int n = scn.nextInt();
        int arr[] = new int[n];
        for(int i=0;i<n;i++) {
            arr[i] = scn.nextInt();
        }

        System.out.println(removeDuplicates(arr));
    }
}
```

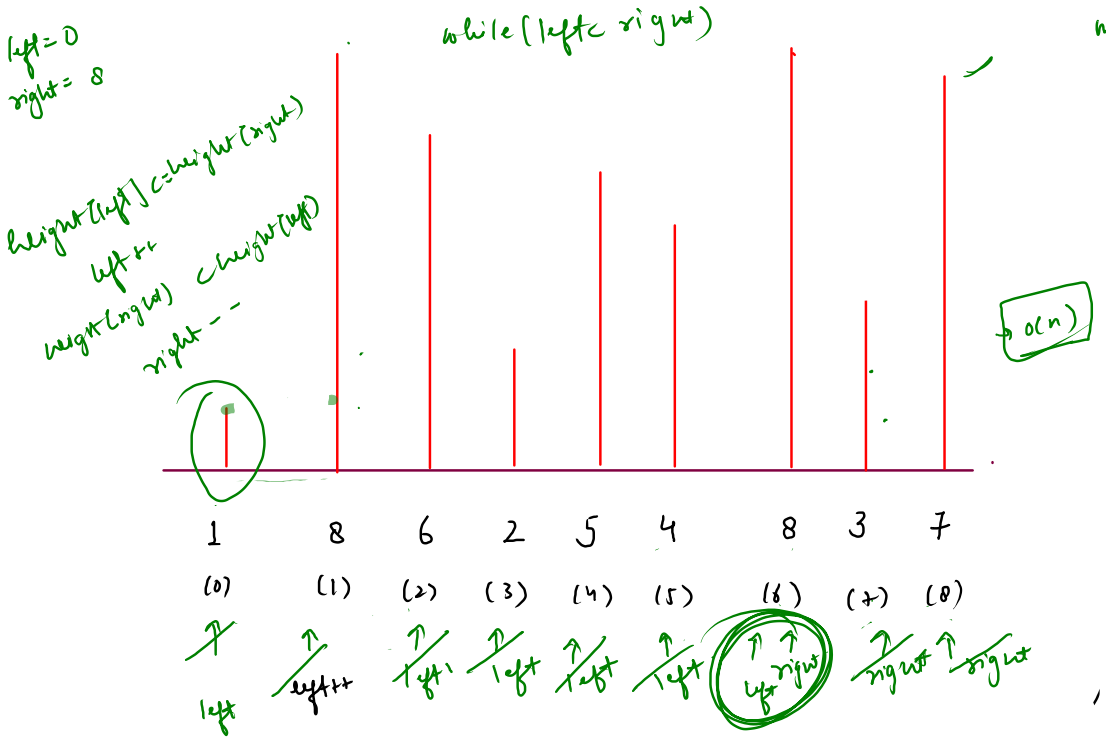
container-with-most-water

Problem | Submissions | Leaderboard | Discussions

You are given an integer array height of length n. There are n vertical lines drawn such that the two endpoints of the ith line are (i, 0) and (i, height[i]).

Find two lines that together with the x-axis form a container, such that the container contains the most water.

Return the maximum amount of water a container can store.



Sample Input 0

```
9
1
8
6
2
5
4
8
3
7
```

Sample Output 0

```
49
```

```
public class Solution {
    public static int containerWithMostWater(int arr[]) {
        int left = 0, right = arr.length-1;
        int maxArea = 0;

        while(left<right) {
            int leftHt = arr[left];
            int rightHt = arr[right];

            int currArea = Math.min(leftHt,rightHt)*(right-left);

            maxArea = Math.max(maxArea,currArea);

            if(leftHt <= rightHt) left++;
            else right--;
        }

        return maxArea;
    }

    public static void main(String[] args) {
        /* Enter your code here. Read input from STDIN. Print output to S
        Scanner scn = new Scanner(System.in);
        int n = scn.nextInt();
        int arr[] = new int[n];

        for(int i=0;i<n;i++) {
            arr[i] = scn.nextInt();
        }

        System.out.println(containerWithMostWater(arr));
    }
}
```

Code + Dry Run

(10 mins)

Sunday: 11:30 - 2:30 pm