Kadane's Algorithm

Medium Accuracy: 36.28% Submissions: 647K+ Points: 4

Given an array **Arr[]** of **N** integers. Find the contiguous sub-array(containing at least one number) which has the maximum sum and return its sum.

Example 1:

Input:
N = 5
Arr[] = {1,2,3,-2,5}
Output:
9
Explanation:
Max subarray sum is 9
of elements (1, 2, 3, -2, 5) which
is a contiguous subarray.

C++ (g++ 5.4) Average Time: 20m Start Timer

```cpp
// } Driver Code Ends
class Solution{
    public:
    // arr: input array
    // n: size of array
    //Function to find the sum of contiguous subarray with maximum sum.
    long long maxSubarraySum(int arr[], int n){

        long sum = 0;
        long max_sum = arr[0];
        for(int i = 0 ; i < n ; i++){
            sum = sum+arr[i];
            max_sum = max(sum , max_sum);
            if(sum < 0){
                sum = 0;
            }

        }
        return max_sum;
    }
};
// } Driver Code Ends
```

Custom Input | Compile & Run | Submit

Description | Discussion (11) | Solutions (2.5K) | Submissions

C++ | Auto

229. Majority Element II

Hint

Medium | 6.8K | 328

Companies

Given an integer array of size `n`, find all elements that appear more than `⌊ n/3 ⌋` times.

Example 1:

```
Input: nums = [3,2,3]
Output: [3]
```

Example 2:

```
Input: nums = [1]
Output: [1]
```

Example 3:

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

```cpp
class Solution {
public:
    vector<int> majorityElement(vector<int>& nums) {
        map <int,int> mp;
        vector <int> v;
        int n = nums.size();
        for(int i :nums){
            mp[i]++;
        }
        for(auto &it:mp){
            if((it.second)>(n/3))
                v.push_back(it.first);
        }
        return v;
    }
};
```

Console | Run | Submit

Three way partitioning

Easy Accuracy: 41.58% Submissions: 104K+ Points: 2

Given an array of size n and a range [**a**, **b**]. The task is to partition the array around the range such that array is divided into three parts.

1) All elements smaller than **a** come first.

2) All elements in range **a** to **b** come next.

3) All elements greater than **b** appear in the end.

The individual elements of three sets can appear in any order. You are required to return the modified array.

Note: The generated output is 1 if you modify the given array successfully.

Example 1:

Input:

n = 5


```
// } Driver Code Ends
//User function template for C++

class Solution{
public:
    //Function to partition the array around the range such
    //that array is divided into three parts.
    void threeWayPartition(vector<int>& arr,int a, int b)
    {
        // code here
        int n = arr.size();
        int left = 0 ;
        int right = n-1;
        for(int i = 0 ; i <= right ; i++){
            if(arr[i] < a){
                swap(arr[i], arr[left]);
                left++;
            }else if(arr[i]>b){
                swap(arr[i] , arr[right]);
                right--;
                i--;
            }
        }
    }
};
// } Driver Code Ends
```

Description | Discussion (4) | Solutions (133) | Submissions

C++ | Auto

2193. Minimum Number of Moves to Make Palindrome

Hint

Hard | 666 | 63

Companies

You are given a string `s` consisting only of lowercase English letters.

In one **move**, you can select any two **adjacent** characters of `s` and swap them.

Return *the **minimum number of moves** needed to make `s` a palindrome.*

Note that the input will be generated such that `s` can always be converted to a palindrome.

Example 1:

```
Input: s = "aabb"
Output: 2
Explanation:
We can obtain two palindromes from s, "abba" and "baab".
- We can obtain "abba" from s in 2 moves: "aabb" -> "abab" -> "abba".
- We can obtain "baab" from s in 2 moves: "aabb" -> "abab" -> "baab".
Thus, the minimum number of moves needed to make s a palindrome is 2.
```

Example 2:

```
class Solution {
public:
    int minMovesToMakePalindrome(string s) {
     int n=s.size(),i=0,j=n-1,ans=0;
        int cnt=-1;
        while(i<j){
            if(s[i]!=s[j]){
                int temp=j-1;
                while(s[temp]!=s[i])
                    temp--;
                if(i!=temp){
                    for(int k=temp+1;k<=j;k++){
                        swap(s[k-1],s[k]);
                        ans++;
                    }
                }else{
                    cnt=i;
                    j++;
                }
            }
            i++;
            j--;
        }if(cnt!=-1){
            ans+=(n/2-cnt);
        }
        return ans;
    }
};
```

Console | Run | Submit

Minimum swaps and K together

Medium Accuracy: 26.0% Submissions: 106K+ Points: 4

Given an array **arr** of **n** positive integers and a number **k**. One can apply a swap operation on the array any number of times, i.e choose any two index i and j (i < j) and swap arr[i] , arr[j] . Find the **minimum** number of swaps required to bring all the numbers less than or equal to **k** together, i.e. make them a contiguous subarray.

Example 1:

Input :
arr[ ] = {2, 1, 5, 6, 3}
K = 3
Output :
1
Explanation:
To bring elements 2, 1, 3 together,
swap index 2 with 4 (0-based indexing)

C++ (g++ 5.4) Start Timer

```
// } Driver Code Ends

class Solution
{
public:
    int minSwap(int arr[], int n, int k) {
        // Complet the function
        int cnt = 0 , maxi = 0 , tmp = 0;
        for(int i = 0 ; i<n;i++){
            if(arr[i] <=k){
                cnt++;
            }
        }
        for(int i = 0 ; i<cnt ; i++){
            if(arr[i] <= k){
                tmp++;
            }
        }
        maxi = tmp;
        for(int i = 0 , j = cnt ;j<n;j++,i++){
            if(arr[i] <= k) tmp--;
            if(arr[j] <= k) tmp++;
            maxi = max(maxi , tmp);
        }
        return cnt-maxi;
    }
};

// } Driver Code Ends
```

Custom Input | Compile & Run | Submit