

---

## Algorithms

---

# 20. Miscellaneous

Handwritten [by pankaj kumar](#)

# Miscellaneous

Date

Page No. 281.

① Kadane's Algorithm

② Dutch National Flag (Three pointers)

- ① Kadane's Algorithm: maximum subarray sum in an Array  
 Given an integer array arr, find the contiguous subarray (containing at least one number) which has the largest sum & return its sum and print the subarray.

Ex:- ① arr = [-2, 4, -3, 4, -1, 2, 1, -5, 4], o/p = 6  
 ② arr = [1]; o/p = 1

Naive Approach:- using two loop :- by generating all subarray with their sum and take max sum.  $O(n^2)$

Optimal solution:- Kadane's Algorithm  $O(n)$ :-

- Iterate over the loop and find the sum by adding every element when sum will become less than '0', then there is no need to keep these subarray in our answer bcz it will ~~be zero~~ reduce the sum so, ignore them and start new subarray after that element
- at every step compare the sum to previous max sum and accordingly store the max <sup>sum</sup> subarray in answer and also mark the ending of subarray where sum is maximum.

code:-

```
int maxSubArray(vector<int> & nums, vector<int> & subarray) {
    int ans = INT_MIN, sum = 0;
    int start = 0;
    for (int i = 0; i < nums.size(); i++) {
        sum += nums[i];
        if (sum > ans) {
            subarray.clear();
            ans = sum;
            subarray.push_back(i);
            subarray.push_back(i);
        }
        if (sum < 0) {
            sum = 0;
            i = i + 1;
        }
    }
    return ans;
}
```

$T.C: O(n)$

$S.C: O(1)$



② Dutch National Flag algorithm: (Three pointer Approach)  
:- (Sort an array of 0s, 1s and 2s): —

Ex:- ①  $nums = [2, 0, 2, 1, 1, 0]$  | ②  $nums = [4, 0, 1]$   
op:-  $[0, 0, 1, 1, 2, 2]$  | op:-  $[0, 1, 2]$

Approach 1:- by sorting ( $O(N \log N)$ )

Approach 2:- keeping count of values (0s, 1s & 2s) and then  
Update the array by overwriting.  $O(N) + O(N)$

Approach 3:- 3-pointers Approach (Dutch National Flag Alg)

In this approach, we will be using 3 pointers named, low, mid and high. we will be using these 3 pointers to move around the values. The primary goal there is to move 0s to the left and 2s to the right and at the same time all the 1s will be in the middle.

Step:- ① initialize, low = 0, mid = 0, high = n-1  
② ③ if  $arr[mid] = 0$ ; then swap ( $arr[low], arr[mid]$ ). low++, mid++  
④ if  $arr[mid] = 1$ ; then mid++  
⑤ if  $arr[mid] = 2$ ; swap ( $arr[mid], arr[high]$ ) high--

Code:-

```
void sortColors (vector<int> & nums) {
    int lo = 0, hi = nums.size() - 1, mid = 0;
    while (mid <= hi) {
        switch (nums[mid]) {
            case 0:
                swap (nums[lo++], nums[mid++]);
                break;
            case 1:
                mid++;
                break;
            case 2:
                swap (nums[mid], nums[hi--]);
                break;
        }
    }
}
```

TC:  $O(N)$   
SC:  $O(1)$