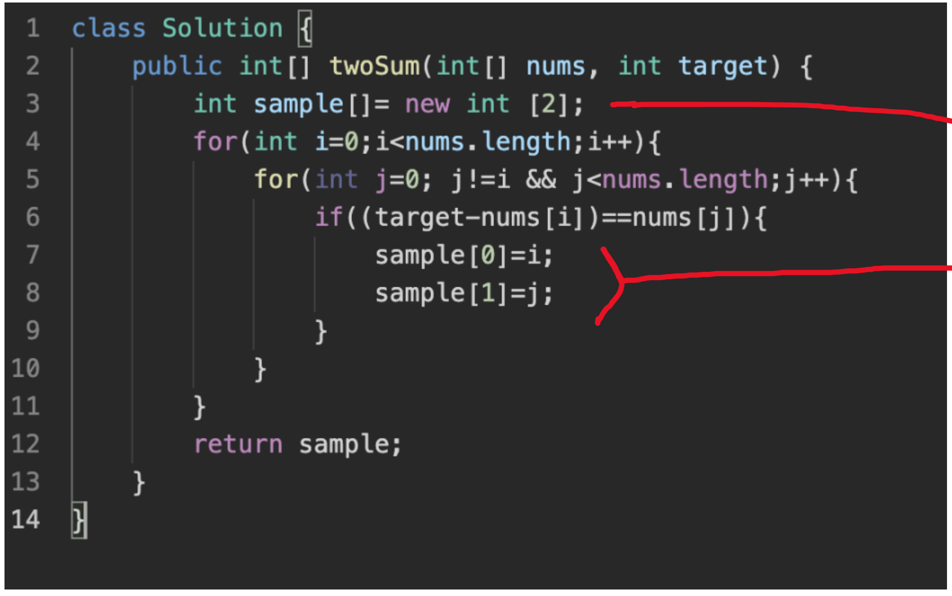
1. **Two Sum**

Input: nums = [2,7,11,15], target = 9
Output: [0,1]
Explanation: Because nums[0] + nums[1] == 9, we return [0, 1].

Brute Force: By Using a 2 nested ForLoop



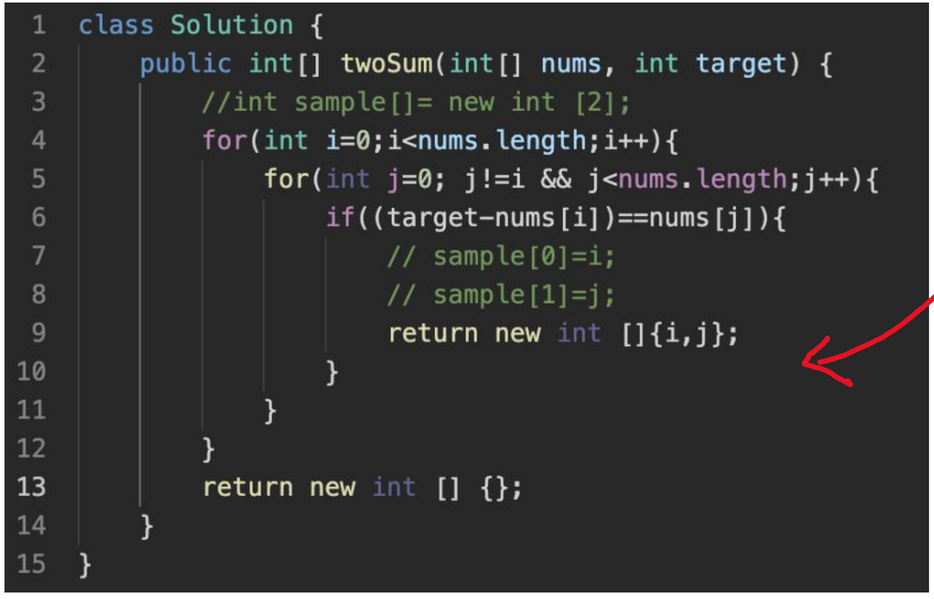
```
class Solution {
    public int[] twoSum(int[] nums, int target) {
        int sample[]= new int [2];
        for(int i=0;i<nums.length;i++){
            for(int j=0; j!=i && j<nums.length;j++){
                if((target-nums[i])==nums[j]){
                    sample[0]=i;
                    sample[1]=j;
                }
            }
        }
        return sample;
    }
}
```

Complexity
Time complexity: O(N^2);
Space Complexity: O(1);

Problem with this code is: Above Code will waits until it has checked all possible pairs before returning the result.

Complexity:
TC : O(N^2) : Because of 2 nested For Loop
SC: O(1) : because we are using array of size 1.

Bit improvement: I have removed the array and it might be faster in cases where a valid pair is found early in the iteration.

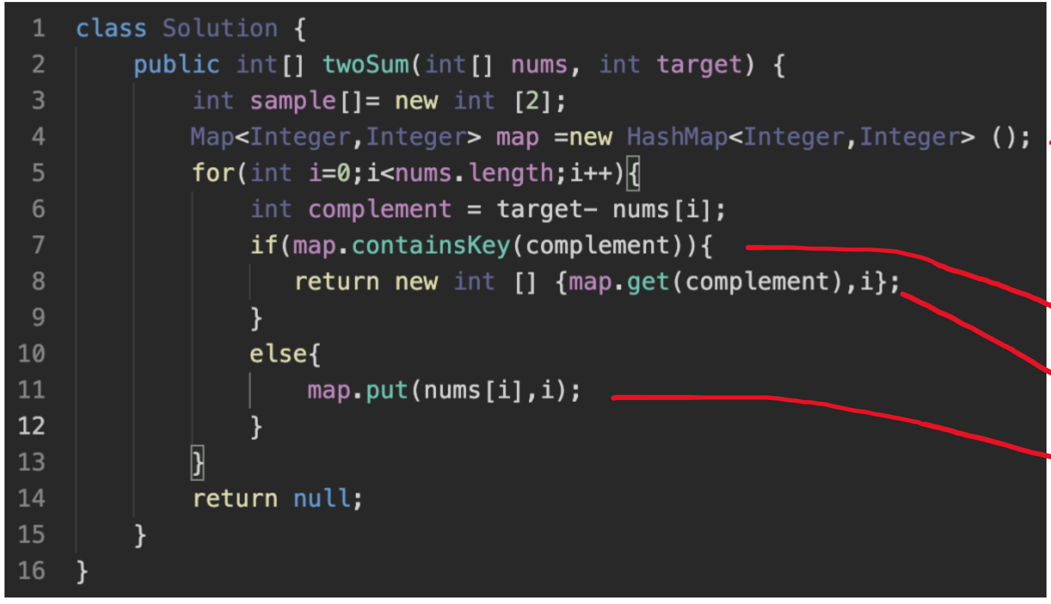


```
class Solution {
    public int[] twoSum(int[] nums, int target) {
        //int sample[]= new int [2];
        for(int i=0;i<nums.length;i++){
            for(int j=0; j!=i && j<nums.length;j++){
                if((target-nums[i])==nums[j]){
                    // sample[0]=i;
                    // sample[1]=j;
                    return new int []{i,j};
                }
            }
        }
        return new int [] {};
    }
}
```

	Mistakes

Complexity: same as above.

Optimize Approach : Using a HashMap because is save as key value pairs.



```
class Solution {
    public int[] twoSum(int[] nums, int target) {
        int sample[]= new int [2];
        Map<Integer,Integer> map =new HashMap<Integer,Integer> ();
        for(int i=0;i<nums.length;i++){
            int complement = target- nums[i];
            if(map.containsKey(complement)){
               return new int [] {map.get(complement),i};
            }
            else{
                map.put(nums[i],i);
            }
        }
        return null;
    }
}
```

	Mistakes
	I forgot how to use Map , HashMap . and it should be I key, value pairs

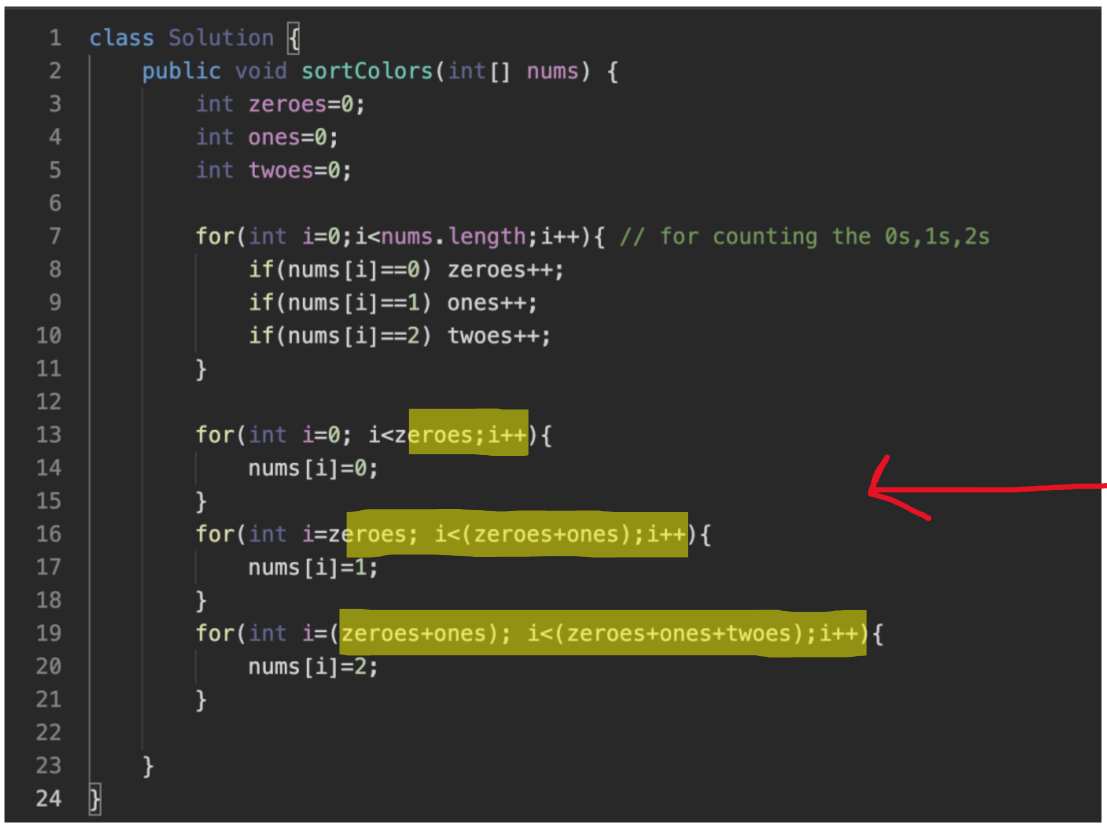
	I forgot the hashmap functions :
	containsKey()
	map.get()
	map.put()

Complexity:
TC : O(N) : Because of 1 For Loop
SC: O(N) : because we are using HashMap of size N(means size of the array).

2. Sort Colors (0,1,2)

Input: nums = [2,0,2,1,1,0]
Output: [0,0,1,1,2,2]
Brute Force: Just use a in-built sort function.

Better Approach: using 2 for loop (one loop split into 3 parts).
Note: Make sure when you are working on the 2nd loop and its sub-parts check the limit otherwise it will show a different output.

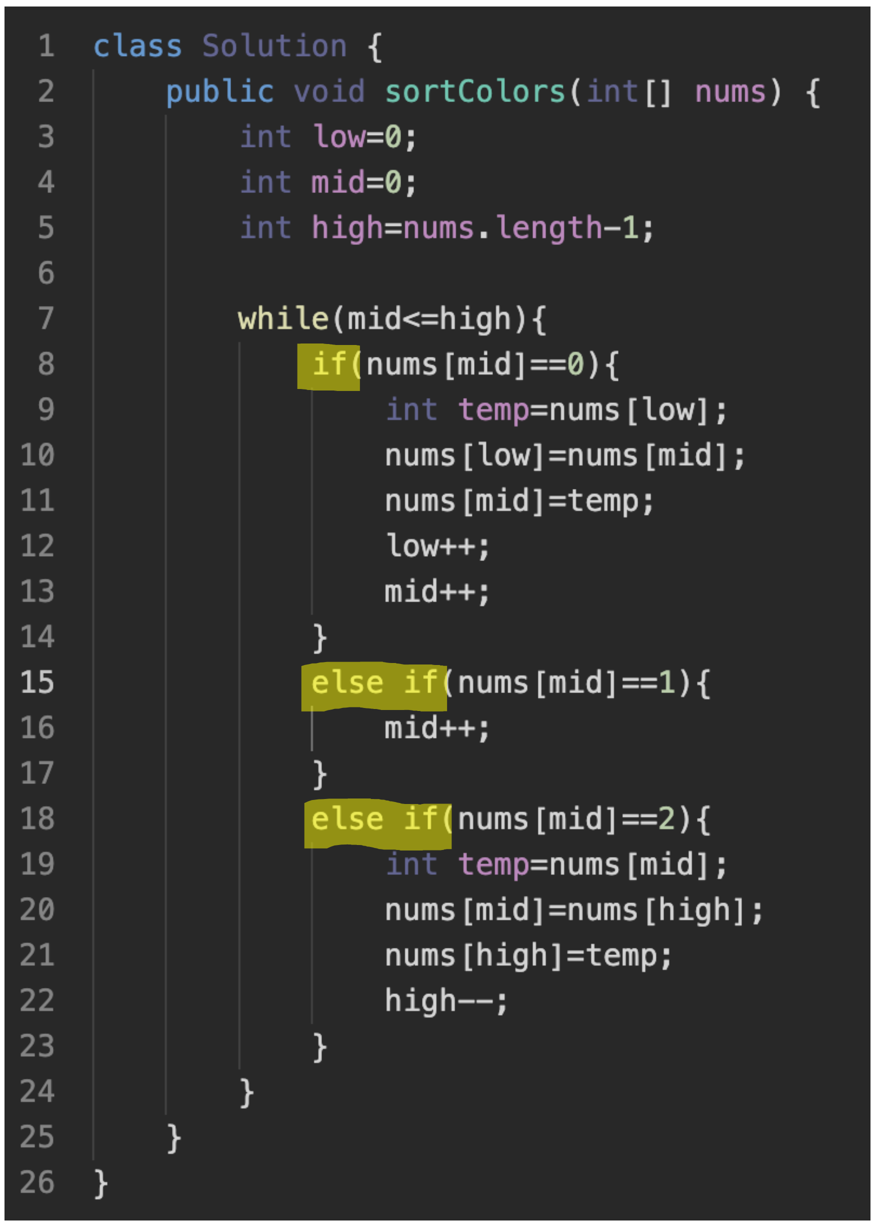


```
class Solution {
    public void sortColors(int[] nums) {
        int zeroes=0;
        int ones=0;
        int twoes=0;

        for(int i=0;i<nums.length;i++){ // for counting the 0s,1s,2s
            if(nums[i]==0) zeroes++;
            if(nums[i]==1) ones++;
            if(nums[i]==2) twoes++;
        }

        for(int i=0; i<zeroes;i++){
            nums[i]=0;
        }
        for(int i=zeroes; i<(zeroes+ones);i++){
            nums[i]=1;
        }
        for(int i=(zeroes+ones); i<(zeroes+ones+twoes);i++){
            nums[i]=2;
        }

    }
}
```

	Mistakes
1	I forgot the for loop Condition eg: i< (zeroes + ones)

Complexity	Reason
TC : O(2N)	2 for loop
SC : O(1)	As we are not using any extra space.

Optimize Approach: Using 3 Pointers(Low, Mid, High) for 0,1,2
Point to Remember: Low and Mid starts with 0th location



```
class Solution {
    public void sortColors(int[] nums) {
        int low=0;
        int mid=0;
        int high=nums.length-1;

        while(mid<=high){
            if(nums[mid]==0){
                int temp=nums[low];
                nums[low]=nums[mid];
                nums[mid]=temp;
                low++;
                mid++;
            }
            else if(nums[mid]==1){
                mid++;
            }
            else if(nums[mid]==2){
                int temp=nums[mid];
                nums[mid]=nums[high];
                nums[high]=temp;
                high--;
            }
        }
    }
}
```

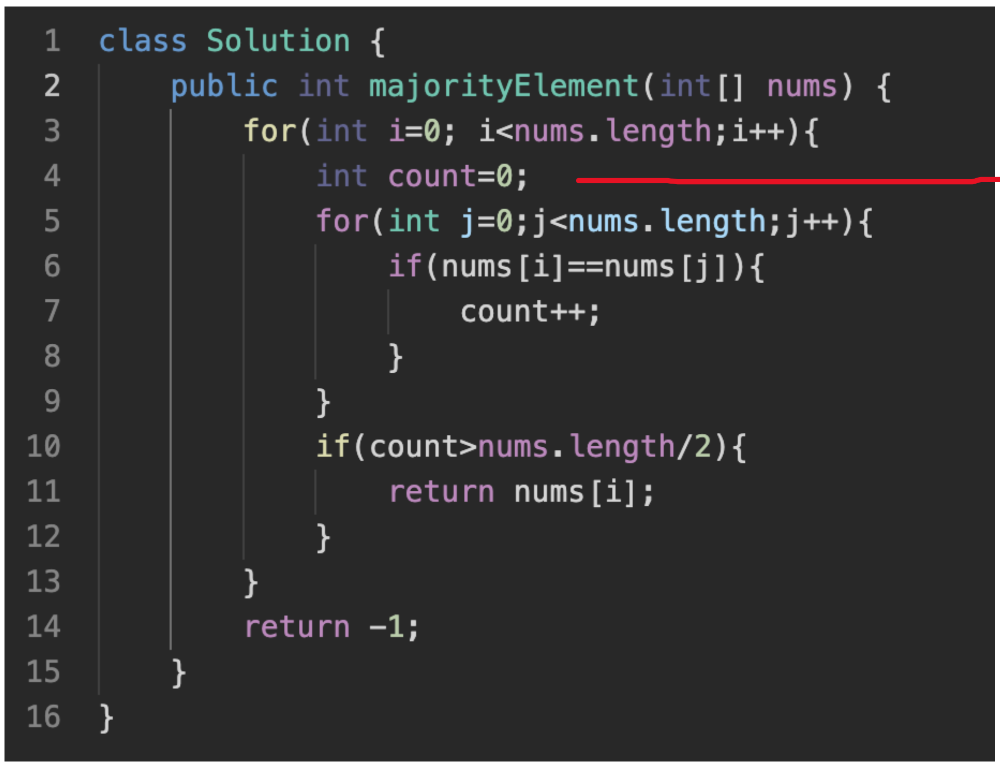
	Mistakes
1	I forgot to use “else-if “

Complexity	Reason
TC : O(N)	We are using a single loop that can run at most N times.
SC : O(1)	As we are not using any extra space.

Key Notes:
We can also use a swap function for swapping the values. It will reduce the code length.

3. Majority Element
Input: nums = [2,2,1,1,1,2,2]
Output: 2

Brute Force : using nested loop

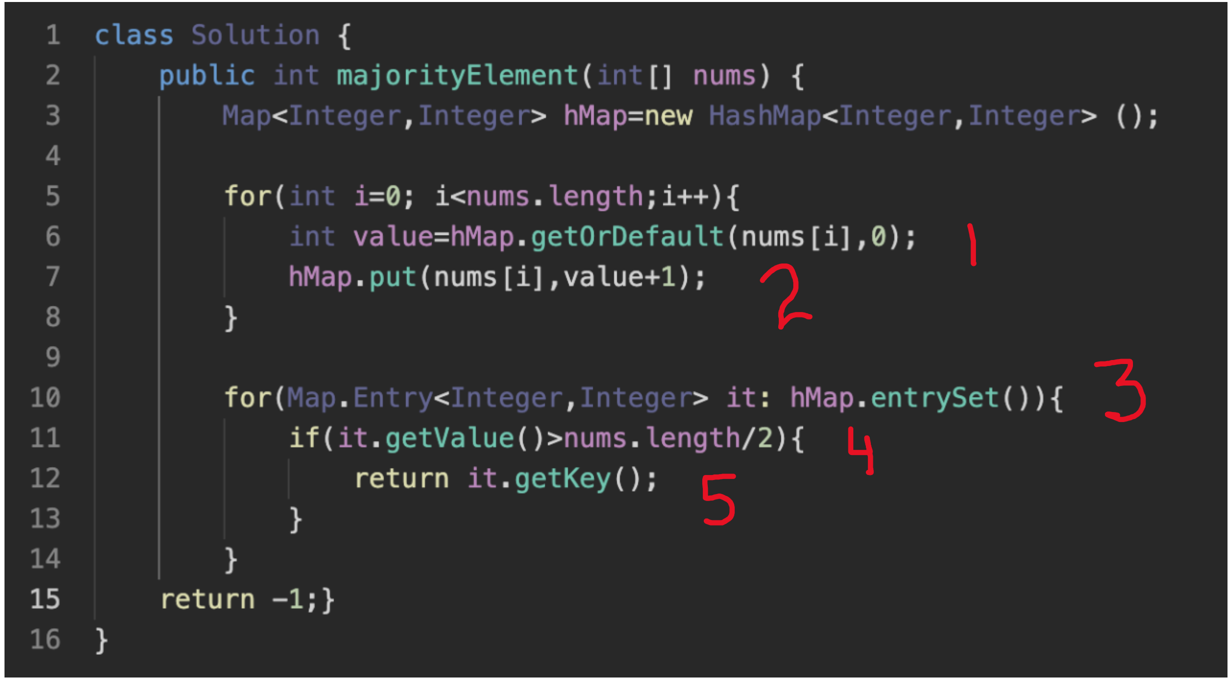


```java
class Solution {
    public int majorityElement(int[] nums) {
        for(int i=0; i<nums.length;i++){
            int count=0;
            for(int j=0;j<nums.length;j++){
                if(nums[i]==nums[j]){
                    count++;
                }
            }
            if(count>nums.length/2){
                return nums[i];
            }
        }
        return -1;
    }
}
```

	Mistakes
1	I forgot to keep "count=0" inside 1st for loop. because "count" should be equal to 0 when we exit from the 2nd loop.

Complexity	Reason
TC : O(N^2)	Nested for loop
SC : O(1)	As we are not using any extra space.

Better Approach : Using HashMap.

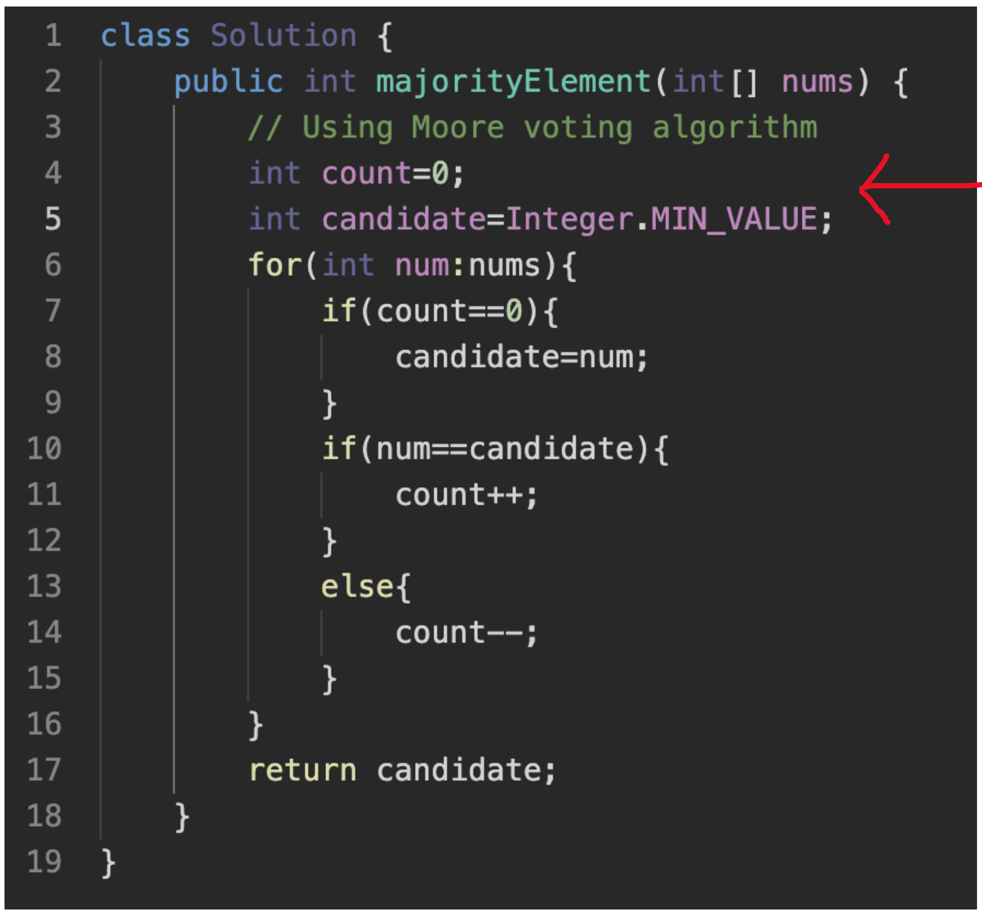


```java
class Solution {
    public int majorityElement(int[] nums) {
        Map<Integer,Integer> hMap=new HashMap<Integer,Integer> ();

        for(int i=0; i<nums.length;i++){
            int value=hMap.getOrDefault(nums[i],0);
            hMap.put(nums[i],value+1);
        }

        for(Map.Entry<Integer,Integer> it: hMap.entrySet()){
            if(it.getValue()>nums.length/2){
                return it.getKey();
            }
        }
    return -1;}
}
```

	Key Note
1	
2	As we are not using any extra space.

Complexity	Reason
TC : O(N*logN) + O(N)	We are using a map data structure. Insertion in the map takes logN time. And we are doing it for N elements. So, it results in the first term O(N*logN). The second O(N) is for checking which element occurs more than floor(N/2) times. If we use unordered_map instead, the first term will be O(N) for the best and average case and for the worst case, it will be O(N2).
SC : O(N)	As we are using a map data structure.

Optimize: Using Moore Voting algorithm

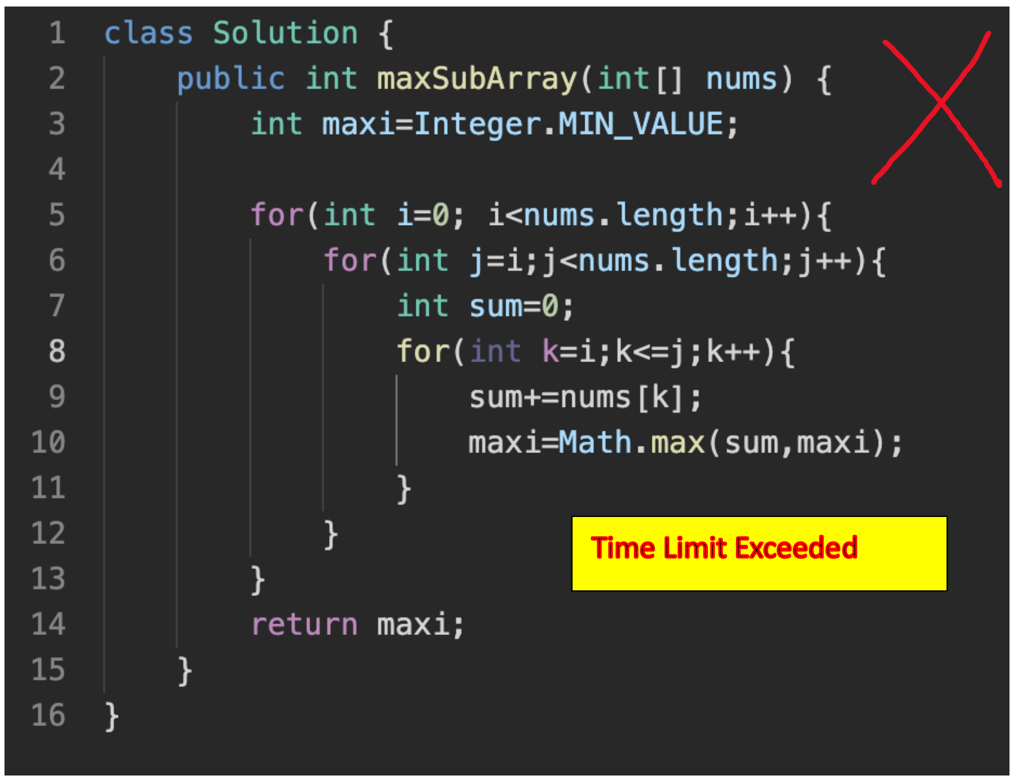


```java
class Solution {
    public int majorityElement(int[] nums) {
        // Using Moore voting algorithm
        int count=0;
        int candidate=Integer.MIN_VALUE;
        for(int num:nums){
            if(count==0){
                candidate=num;
            }
            if(num==candidate){
                count++;
            }
            else{
                count--;
            }
        }
        return candidate;
    }
}
```

Key Notes:
Make sure that check for constraints otherwise it will create a problem when you put any value while initializing the "candidate".

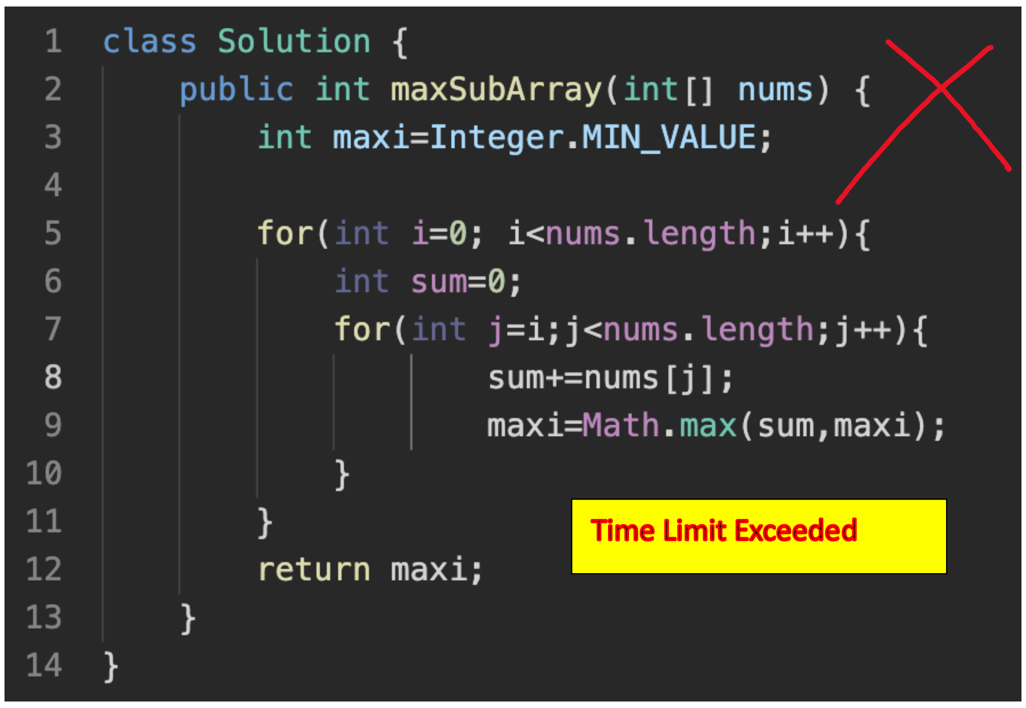
Complexity	Reason
TC : O(N) + O(N), where N = size of the given array.	The first O(N) is to calculate the count and find the expected majority element. The second one is to check if the expected element is the majority one or not. Note: If the question states that the array must contain a majority element, in that case, we do not need the second check. Then the time complexity will boil down to O(N).
SC : O(1)	As we are not using any extra space.

4. Maximum SubArray



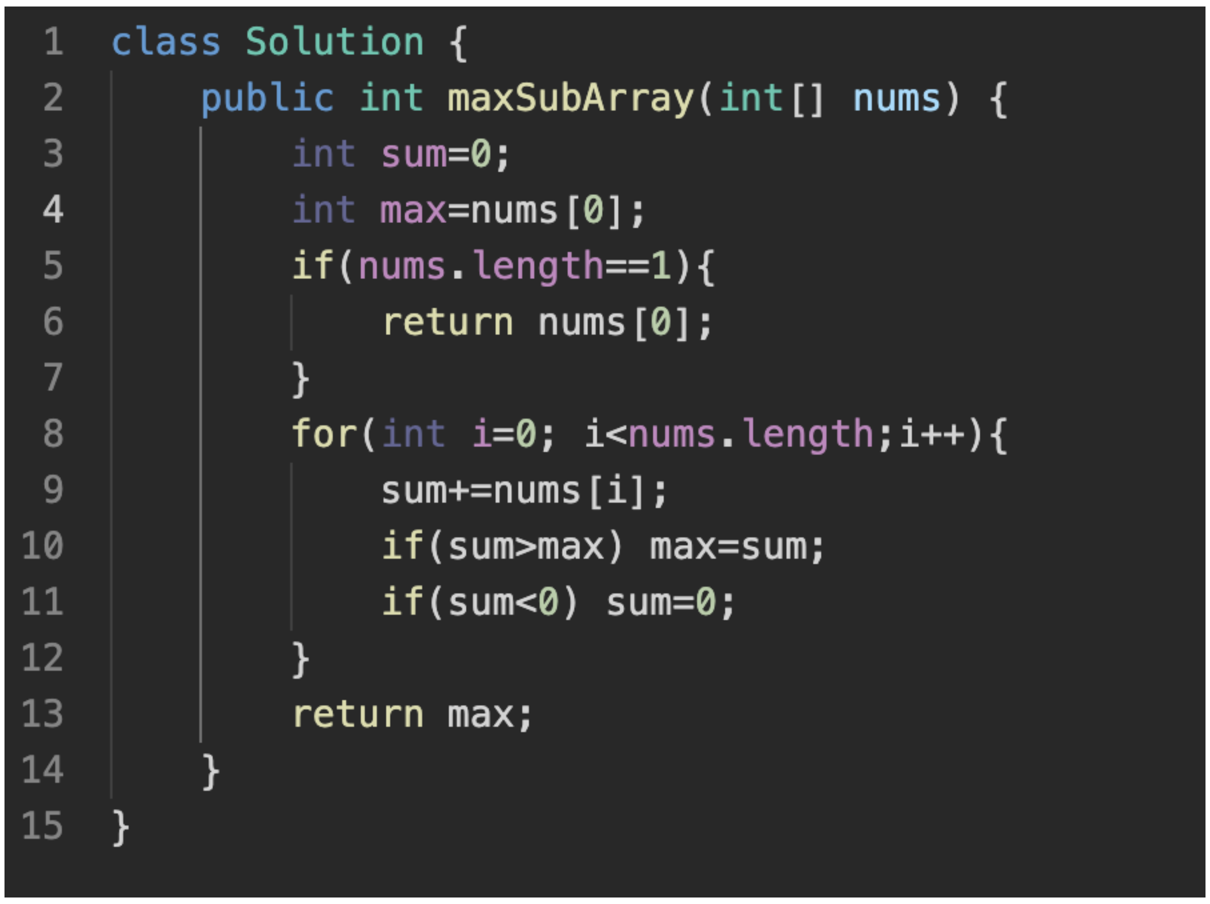
```java
class Solution {
    public int maxSubArray(int[] nums) {
        int maxi=Integer.MIN_VALUE;

        for(int i=0; i<nums.length;i++){
            for(int j=i;j<nums.length;j++){
                int sum=0;
                for(int k=i;k<=j;k++){
                    sum+=nums[k];
                    maxi=Math.max(sum,maxi);
                }
            }
        }
        return maxi;
    }
}
```

Time Limit Exceeded



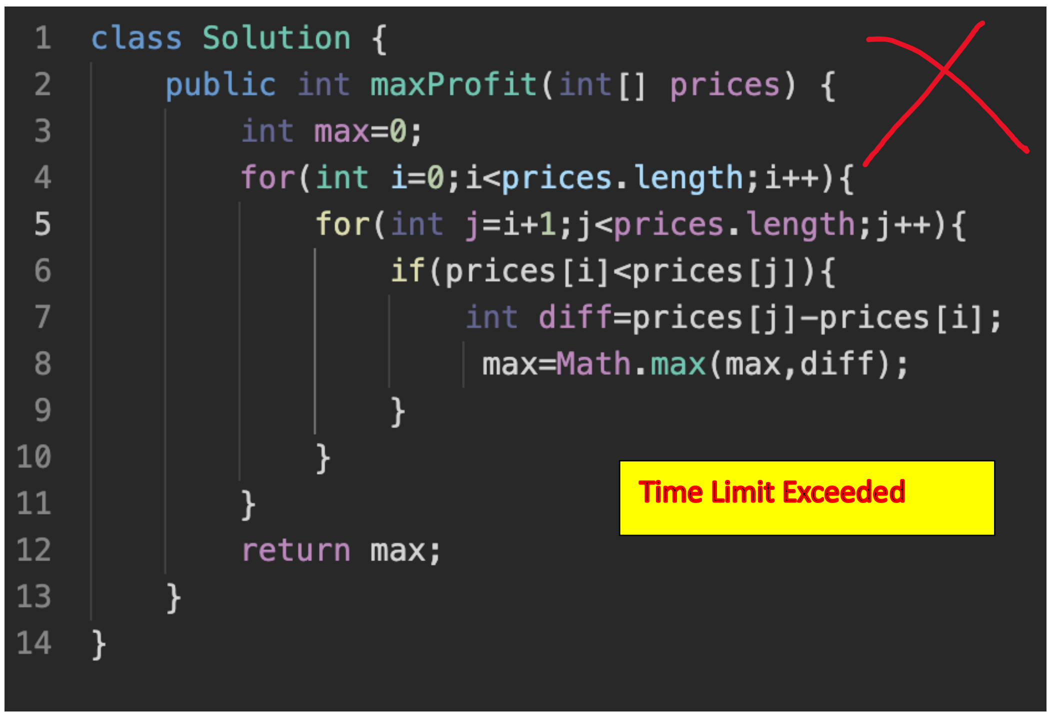
```java
class Solution {
    public int maxSubArray(int[] nums) {
        int maxi=Integer.MIN_VALUE;

        for(int i=0; i<nums.length;i++){
            int sum=0;
            for(int j=i;j<nums.length;j++){
                    sum+=nums[j];
                    maxi=Math.max(sum,maxi);
            }
        }
        return maxi;
    }
}
```

Time Limit Exceeded



```java
class Solution {
    public int maxSubArray(int[] nums) {
        int sum=0;
        int max=nums[0];
        if(nums.length==1){
            return nums[0];
        }
        for(int i=0; i<nums.length;i++){
            sum+=nums[i];
            if(sum>max) max=sum;
            if(sum<0) sum=0;
        }
        return max;
    }
}
```

5. Best time to buy and sell Stock



```
class Solution {
    public int maxProfit(int[] prices) {
        int max=0;
        for(int i=0;i<prices.length;i++){
            for(int j=i+1;j<prices.length;j++){
                if(prices[i]<prices[j]){
                    int diff=prices[j]-prices[i];
                     max=Math.max(max,diff);
                }
            }
        }
        return max;
    }
}
```

Time Limit Exceeded

Things I learned in 1D- Arrays Problem:

1. Using Nested for-loop
   - Usually start with nested for loop
   - 1st loop where int i=0;
   - 2nd loop where j=0 and it will check for all the element w.r.t "i" where "i" remains at it position.
2. Using 2 pointer approach with Binary Search,
   - 1st pointer is at 0th location 2nd pointer it at end location.
   - us can also do a swap.
3. Use of Moore Voting algorithm.
4. Learn Map and its Functions.
5.