ARRAYS - SET 3 SOLUTIONS

Note: All solutions are written in C++.

Question 1:

- **Problem link**: https://leetcode.com/problems/best-time-to-buy-and-sell-stock/
- **Difficulty level**: Easy

Explanation:

Traverse the array and keep track of the lowest price. For the current price, check if the difference between the current price and the current minimum price gives the maximum profit. If so, update the max profit recorder till now.

Solution:

```
class Solution {
public:
    int maxProfit(vector<int>& prices) {

        int maxProfit = INT_MIN;
        int minPrice = INT_MAX;

        for(int i=0; i<prices.size(); i++)
        {
            minPrice = min(minPrice, prices[i]);

            if(prices[i]-minPrice > maxProfit)
            {
                maxProfit = prices[i] - minPrice;
            }
        }
```

```
        return maxProfit;
    }
};
```

Complexity:

- Time: O(n)
- Space: O(1)

Question 2:

- **Problem link**: https://leetcode.com/problems/check-if-n-and-its-double-exist/
- **Difficulty level**: Easy

Explanation:

Maintain a map to store the count of each element present in the given array. Then, traverse the array again. For each element, check if the count of its double is non zero or not. The number '0' is a boundary case which has to be taken care of.

Solution:

```
class Solution {
public:
    bool checkIfExist(vector<int>& arr) {

        map<int,int> count;
        int n = arr.size();

        for(int i=0; i<n; i++)
        {
            count[arr[i]]++;
        }
```

```
        for(int i=0; i<n; i++)
        {
            if(arr[i] == 0 && count[0] >= 2)
            {
                return true;
            }
            else if(arr[i]!=0 && count[2*arr[i]] > 0)
            {
                return true;
            }
        }

        return false;
    }
};
```

Complexity:

- Time: O(n)
- Extra Space: O(n)

Question 3:

- **Problem link**: https://leetcode.com/problems/minimum-distance-to-the-target-element/
- **Difficulty level**: Easy

Explanation:

Simply find the occurrence of the target in 'nums' which is closest to the starting index in linear time. Note the start index can itself contain the target element.

Solution:

```
class Solution {
public:
    int getMinDistance(vector<int>& nums, int target, int start) {
        int m=INT_MAX;
        for(int i=0;i<nums.size();i++)
        {
            if(nums[i]==target and abs(i-start)<m) m=abs(i-start);
        }
        return m;
    }
};
```

Complexity:

- Time: O(n)
- Space: O(1)

Question 4:

- **Problem link**: https://leetcode.com/problems/distance-between-bus-stops/
- **Difficulty level**: Medium

Explanation:

Calculate the total distance, i.e. the sum of all given distances. Then, find the distance between start and destination along any one path. The length of the other path will be the difference between the two calculated distances. Compare the length of both paths and return the smaller one.

Solution:

```
class Solution {
public:
    int distanceBetweenBusStops(vector<int>& distance, int start, int
destination) {

        int n = distance.size();

        int d1=0, totalDistance = 0;

        for(int i=0; i<n; i++)
        {
            totalDistance += distance[i];
        }

        int i=start;

        while(i != destination)
        {
            d1 += distance[i];
            i = (i+1)%n;
        }

        return min(d1, totalDistance-d1);

    }
};
```

Complexity:

- Time: O(n)
- Space: O(1)

Question 5:

- **Problem link**: https://leetcode.com/problems/teemo-attacking/
- **Difficulty level**: Medium

Explanation:

Initiate total time in poisoned condition 'total' = 0. Iterate over 'timeSeries' list. At each step add to the total time the minimum between interval length and the poisoning time duration duration. Return 'total' + 'duration' to take the last attack into account.

Solution:

```
class Solution {
public:
    int findPoisonedDuration(vector<int>& timeSeries, int duration) {
        int total=0,n=timeSeries.size();
        if(n==0) return 0;
        for(int i=0;i<n-1;i++)
        {
            total+=min(timeSeries[i+1]-timeSeries[i],duration);
        }
        total+=duration;
        return total;
    }
};
```

Complexity:

- Time: O(n)
- Space: O(1)

Question 6:

- **Problem link**: https://leetcode.com/problems/find-all-duplicates-in-an-array/
- **Difficulty level**: Medium

Explanation:

Traverse the array. Suppose you come across an element 4, then, make arr[4-1] = arr[3] negative (if it is positive till now). The next time you come across the same element 4 again, check the element arr[4-1] = arr[3]. Since arr[3] is already negative, this means that we have encountered 4 before this. Do this for each element of array.

Solution:

```
class Solution {
public:
    vector<int> findDuplicates(vector<int>& nums) {

        int n = nums.size();
        vector<int> output;

        for(int i=0; i<n; i++)
        {
            int num = nums[i];

            if(num < 0) num = num*(-1);

            if(nums[num-1] > 0)
            {
                nums[num-1] = nums[num-1]*(-1);
            }
            else
            {
                output.push_back(num);
            }
        }

        return output;
    }
};
```

Complexity:

- Time: O(n)
- Space: O(1)

Question 7:

- **Problem link**: https://leetcode.com/problems/grumpy-bookstore-owner/
- **Difficulty level**: Medium

Explanation:

This is an example of a sliding window problem. We first compute the maximum number of customers satisfied if the owner uses his power at the start itself. Then we slide our 'window', i.e. the duration when he is not grumpy, by one unit at each step and calculate the maximum number of customers satisfied in linear time.

Solution:

```
class Solution {
public:
    int maxSatisfied(vector<int>& customers, vector<int>& grumpy, int X) {
        int idx1=0,idx2=X-1,max;
        int n=customers.size(),sum=0;
        for(int i=0;i<n;i++)
        {
            if(i<X) sum+=customers[i];
            else if(grumpy[i]==0) sum+=customers[i];
        }
        max=sum;
        idx1++;
        idx2++;
        while(idx2<n)
        {
            if(grumpy[idx1-1]==1) sum-=customers[idx1-1];
            if(grumpy[idx2]==1) sum+=customers[idx2];
```

```
            if(sum>max) max=sum;
            idx1++;
            idx2++;
        }
        return max;
    }
};
```

Complexity:

- Time: O(n)
- Space: O(1)

Question 8:

- **Problem link**: https://leetcode.com/problems/trapping-rain-water/
- **Difficulty level**: Hard

Explanation:

Here we use brute force to get an O(n^2) time complexity solution. The detailed solution can be found here. The question can be also solved by dynamic programming to get an optimized solution, but we won't discuss that here.

Solution:

```
class Solution {
public:
    int trap(vector<int>& height) {
        int result=0,left_max=0,right_max=0;
        int n=height.size();
        for(int i=0;i<n;i++)
        {
            left_max=0;
```

```
            right_max=0;
            for(int j=0;j<n;j++)
            {
                if(j<i) left_max=max(left_max,height[j]);
                else if(j>i) right_max=max(right_max,height[j]);
                else
                {
                    left_max=max(left_max,height[j]);
                    right_max=max(right_max,height[j]);
                }
            }
            result+=min(left_max,right_max)-height[i];
        }
        return result;
    }
};
```

Complexity:

- Time: O(n^2)
- Space: O(1)