ARRAYS CONTEST SOLUTIONS

Note: All solutions are written in C++.

Question 1:

- **Problem link**: https://www.hackerrank.com/challenges/the-birthday-bar
- **Difficulty level**: Easy

Explanation:

Direct application of window sliding technique.

Sum of the first m elements is found initially. Later the first element in the window is popped and the next element in the array is added to the sum.

At any instance if the sum is the same as d, result is incremented.

Solution:

```
int birthday(vector<int> s, int d, int m)
{
    int result = 0, sum = 0;

    for(int i=0; i<m; i++)
    {
        sum += s[i];
    }

    if(sum == d)
    {
        result += 1;
    }

    for(int i=m; i<s.size(); i++)
    {
        sum -= s[i-m];
        sum += s[i];
```

```
        if(sum == d)
        {
            result += 1;
        }
    }

    return result;
}
```

Complexity:

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

Question 2:

- **Problem link**: https://www.hackerrank.com/challenges/non-divisible-subset
- **Difficulty level**: Medium

Explanation:

A hashmap count of size k is used to count the number of elements in the array that give a given remainder when divided by k.

If a number having remainder r, is added in the subset array, then a number having a remainder k-r cannot be added as their sum will be divisible by k.

Hence for each value of remainder, the remainder or the mirror value having the maximum value of numbers in the array is added in the subset array.

Remainders k/2 (for even numbered k) and 0 are special cases.

Solution:

```
int nonDivisibleSubset(int k, vector<int> s)
{
    vector<int> count(k);
```

```
    for(int i=0; i<s.size(); i++)
    {
        count[s[i] % k] += 1;
    }

    int result = 0;

    if(count[0] > 0)
    {
        result += 1;
    }

    if(k%2 == 0)
    {
        if(count[k/2] > 0)
        {
            result += 1;
        }

        for(int i=1; i<(k/2); i++)
        {
            result += max(count[i], count[k-i]);
        }
    }

    else
    {
        for(int i=1; i<(k/2)+1; i++)
        {
            result += max(count[i], count[k-i]);
        }
    }

    return result;

}
```

Complexity:

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

Question 3:

- **Problem link**: https://www.hackerrank.com/challenges/between-two-sets
- **Difficulty level**: Medium

Explanation:

Find the greatest common divisor of the elements in b and the lowest common multiple of the elements in a.

The number of multiples of the LCM of a that divide the GCD of b is the result.

Solution:

```
int gcd(int a, int b)
{
    if(a == 0)
    {
        return b;
    }

    return gcd(b%a, a);
}

int getTotalX(vector<int> a, vector<int> b)
{
    int x, l = a[0], r = b[0];

    for(int i=1; i<b.size(); i++)
    {
        r = gcd(min(r,b[i]), max(r,b[i]));
    }

    for(int i=1; i<a.size(); i++)
    {
        x = l * a[i];
        l = x / gcd(min(l,a[i]), max(l,a[i]));
    }

    int result = 0;
    x = l;

    while(x<=r)
```

```
    {
        if (r%x == 0)
            result += 1;

        x += 1;
    }

    return result;
}
```

Complexity:

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

Question 4:

- **Problem link**: https://www.hackerrank.com/challenges/two-arrays
- **Difficulty level**: Medium

Explanation:

Two hashmaps are used to store the values of A and B that lie in the range [0, k]. Any value greater than k is stored as k.

Iterate both the hashmaps in parallel, A from the starting index and B from the ending index. If the value of A at an index is less than the surplus and value of B at the mirror index, then continue the iteration. Else the permutation is not possible and return NO.

If the entire array has been iterated then return YES.

Solution:

```
string twoArrays(int k, vector<int> A, vector<int> B)
{
    vector<int> count1(k+1);
    vector<int> count2(k+1);

    for(int i=0; i<A.size(); i++)
    {
```

```
        if(A[i] > k)
        {
            count1[k] += 1;
        }

        else
        {
            count1[A[i]] += 1;
        }
    }

    for(int i=0; i<B.size(); i++)
    {
        if(B[i] > k)
        {
            count2[k] += 1;
        }

        else
        {
            count2[B[i]] += 1;
        }
    }

    for(int i=0, surplus = 0; i<k+1; i++)
    {
        if(count1[i] <= count2[k-i] + surplus)
        {
            surplus += (count2[k-i] - count1[i]);
        }

        else
        {
            return "NO";
        }
    }

    return "YES";
}
```

Complexity:

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

Question 5:

- **Problem link**: https://www.hackerrank.com/challenges/crush
- **Difficulty level**: Hard

Explanation:

For each query, the starting index of the query is incremented by the value in the query and the index after the ending index is decremented by the same value.

Each element in the array is changed to the sum of the element and all previous values.

The maximum value is then returned.

Solution:

```
long arrayManipulation(int n, vector<vector<int>> queries)
{
    vector<long> arr(n);

    for(int i=0; i<queries.size(); i++)
    {
        arr[queries[i][0] - 1] += queries[i][2];

        if(queries[i][1] < n)
            arr[queries[i][1]] -= queries[i][2];
    }

    long long int result = arr[0];

    for(int i=1; i<arr.size(); i++)
    {
        arr[i] += arr[i-1];

        if(result < arr[i])
            result = arr[i];
    }

    return result;
}
```

Complexity:

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