1. Assignment 1

Hexaware Assignment 1.pdf

1. Find a pair with the given sum in an array.

```
#include<bits/stdc++.h>
using namespace std;

int main(){



    vector<int> nums = { 1,3,3,4,4,5,6,7};
    set<int> sec;
    int target = 10;
    int n = nums.size();



    for(int i=0; i<n; i++){
        int complement = target - nums[i];
         if(sec.find(complement) !=sec.end());
         {
            cout<< nums[i] << " " <<complement<<endl;
            break;
         }
         sec.insert(nums[i]);

    }
    return 0;

}
```

```
#include<bits/stdc++.h>
using namespace std;
```

```
int main() {
    vector<int> arr = {8, 7, 2, 5, 3, 1};
    int target = 10;

    int n = arr.size();

    // Sort the array
    sort(arr.begin(), arr.end());

    // Initialize pointers for the two ends of the sorted 
    int left = 0;
    int right = n - 1;

    // Traverse the array to find the pair
    while (left < right) {
        int current_sum = arr[left] + arr[right];

        if (current_sum == target) {
            cout << arr[left] << ", " << arr[right] << end
            return 0; // Found the pair, exit the program
        } else if (current_sum < target) {
            // Increment left pointer if sum is less than 
            left++;
        } else {
            // Decrement right pointer if sum is greater t
            right--;
        }
    }

    cout << "Pair not found with the given sum." << endl;
    return 0;
}
```

💡 using

2. Given an integer array, replace each element with the product of every other element without using the division operator.

```
#include<bits/stdc++.h>
using namespace std;

int main() {d
    // Given array
    vector<int> arr = {1, 2, 3, 4, 5};

    int n = arr.size();

    // Create vectors to store left and right products of
    vector<int> leftProducts(n, 1);
    vector<int> rightProducts(n, 1);
    vector<int> result(n, 1);

    // Compute the product of all elements to the left of
    for (int i = 1; i < n; ++i) {
        leftProducts[i] = leftProducts[i - 1] * arr[i - 1]
    }

    // Compute the product of all elements to the right of
    for (int i = n - 2; i >= 0; --i) {
        rightProducts[i] = rightProducts[i + 1] * arr[i +
    }

    // Multiply the left and right products to get the fin
    for (int i = 0; i < n; ++i) {
        result[i] = leftProducts[i] * rightProducts[i];
    }

    // Output the resultant array
    cout << "Input: ";
    for (int num : arr) {
        cout << num << " ";
    }
    cout << "Output: ";
    for (int num : result) {
        cout << num << " ";
    }
    cout << endl;

    return 0;
}
```

3. Maximum Sum Circular Subarray
   Given a circular integer array, find a subarray with the largest sum in it.

    a.

```
#include <bits/stdc++.h>
using namespace std;

int kadane(const vector<int>& arr) {
    int maxSum = INT_MIN, currentSum = 0;
    for (int i = 0; i < arr.size(); ++i) {
        currentSum = max(arr[i], currentSum + arr[i]);
        maxSum = max(maxSum, currentSum);
    }
    return maxSum;
}

int maxSubarraySumCircular(const vector<int>& A) {
    int totalSum = 0;
    int maxKadane = kadane(A);
    int minKadane = INT_MAX, currentSum = 0;

    for (int i = 0; i < A.size(); ++i) {
        totalSum += A[i];
        currentSum = min(A[i], currentSum + A[i]);
        minKadane = min(minKadane, currentSum);
    }

    if (totalSum == minKadane) {
        return maxKadane;
    }

    return max(maxKadane, totalSum - minKadane);
}

int main() {
    vector<int> arr = {2, 1, -5, 4, -3, 1, -3, 4, -1};
    int result = maxSubarraySumCircular(arr);

    cout << "Subarray with the largest sum is: " << result

    return 0;
}
```

4. Find the maximum difference between two array elements that satisfies the given constraints

   a.

```
Q4.

#include <bits/stdc++.h>
using namespace std;

int main() {
    vector<int> nums = { 2, 7, 9, 5, 1, 3, 5 };
    vector<int> vec;

    int minElement = INT_MAX;
    int maxElement = INT_MIN;

    for(int i = 0; i < nums.size(); i++) {
        if(nums[i] < minElement) {
            minElement = nums[i];
        } else {
            int diff = nums[i] - minElement;
            if(diff > maxElement) {
                maxElement = diff;
            }
        }
    }
    cout << "Max Difference: " << maxElement << endl;

    return 0;
}


/* Logic:

* Firstly intialize two variables
  miniEle = INT_MAX;
  maxDifference = INT_MIN;

secondly iterate through the complete array and if the ele
the minimum element make it the current the minimum elemen


else subtract current element from the smallest element

if(the difference is greater then maxDifference)
maxDifference = difference

return the maxdifference


*/
```

5. Given an array of integers of size N, the task is to find the first non-repeating element in this array.

    a.

```
#include<bits/stdc++.h>
using namespace std;

int firstNonRepeatingElement(const vector<int>& arr) {
    unordered_map<int, int> frequency;
    // Count frequency of each element in the array
    for (int num : arr) {
        frequency[num]++;
    }

    // Find the first element with frequency 1
    for (int num : arr) {
        if (frequency[num] == 1) {
            return num; // Return the first non-repeating
        }
    }

    return -1; // If no non-repeating element found, retur
}

int main() {
    vector<int> arr1 = {-1, 2, -1, 3, 0};

    int firstNonRepeat1 = firstNonRepeatingElement(arr1);
    cout << "First non-repeating element in arr1: " << fir

    return 0;
}
```

6. Minimize the maximum difference between the heights.

```
#include<bits/stdc++.h>
using namespace std;

int main() {
    vector<int> nums = {1, 15, 10};
    int k = 6;
    vector<int> vec;

    // Modify heights based on conditions
    for (int i = 0; i < nums.size(); i++) {
        if (nums[i] > k) {
            vec.push_back(nums[i] - k);
        } else if (nums[i] < k) {
            vec.push_back(nums[i] + k);
        }
    }

    // Find minimum and maximum values in modified vector
    int min_height = *min_element(vec.begin(), vec.end());
    int max_height = *max_element(vec.begin(), vec.end());

    // Calculate and output minimum difference between tow
    int min_diff = max_height - min_height;
    cout << "Minimum difference after modification: " << m

    return 0;
}
```