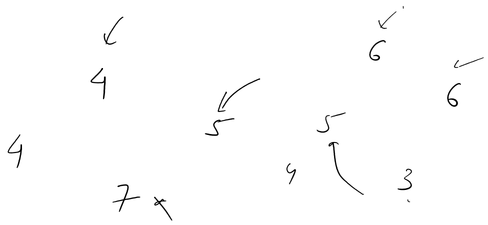
Gas Station



Gas Station

Difficulty: **Medium** Accuracy: **34.79%** Submissions: **193K+** Points: **4** Average Time: **20m**

There are some gas stations along a circular route. You are given two integer arrays **gas[]** denoted as the amount of gas present at each station and **cost[]** denoted as the cost of travelling to the next station. You have a car with an unlimited gas tank. You begin the journey with an empty tank from one of the gas stations. Return the index of the starting gas station if it's possible to travel around the circuit without running out of gas at any station in a **clockwise direction**. If there is no such starting station exists, **return -1**.
Note: If a solution exists, it is guaranteed to be unique.

Examples:

Input: gas[] = [4, 5, 7, 4], cost[]= [6, 6, 3, 5]
Output: 2
Explanation: It is possible to travel around the circuit from station at index 2. Amount of gas at station 2 is (0 + 7) = 7.
Travel to station 3. Available gas = (7 − 3 + 4) = 8.
Travel to station 0. Available gas = (8 − 5 + 4) = 7.
Travel to station 1. Available gas = (7 − 6 + 5) = 6.
Return to station 2. Available gas = (6 − 6) = 0.

Input: gas[] = [1, 2, 3, 4, 5], cost[] = [3, 4, 5, 1, 2]
Output: 3
Explanation: It is possible to travel around the circuit from station at index 3. Amount of gas at station 3 is (0 + 4) = 4.
Travel to station 4. Available gas = 4 − 1 + 5 = 8.
Travel to station 0. Available gas = 8 − 2 + 1 = 7.
Travel to station 1. Available gas= 7 − 3 + 2 = 6.
Travel to station 2. Available gas = 6 − 4 + 3 = 5.
Travel to station 3. The cost is 5. The gas is just enough to travel back to station 3.

Input: gas[] = [3, 9], cost[] = [7, 6]
Output: -1
Explanation: There is no gas station to start with such that you can complete the circuit.

Constraints:
$1 \le$ gas.size(), cost.size() $\le 10^6$
$1 \le$ gas[i], cost[i] $\le 10^3$

Try more examples

currGas + gas[i] > req Gas[i]

0 .... i-1 i i+1 .... n

(4 − 6) + (5 − 6) + (7 − 3) + (4 − 5)

−2 −1 + 4 −1

0

15 15

Start at index i.

$g_{i+1} = (g + g_i) - c_i$ $i = 0, 1, 2, \ldots n-2, n-1$

$g_{i+1} \ge 0 \quad \forall\, i = 0, 1,$

if $g_{i+1} < 0$

[Naive Approach] Considering Every Index as Starting Point – O(n^2) Time & O(1) Space

The simplest approach is to consider each index as a starting point & check if a car can complete the circular tour starting from that index. If we find a valid starting point, we will return it.

```cpp
class Solution {
  public:
    int startStation(vector<int> &gas, vector<int> &cost) {
        // Your code here
        int n=gas.size();
        int startIdx=-1;
        for(int i=0;i<n;i++){
            //Initially car tank is empty
            int currGas=0;
            bool flag=true;
            for(int j=0;j<n;j++){
                //Circular index
                int idx=(i+j)%n;
                currGas=currGas+gas[idx]-cost[idx];
                //If currGas is less than zero, then it isn't
                //possible to proceed further with this starting point
                if(currGas<0){
                    flag=false;
                    break;
                }
            }

            //If flag is true, then we have found
            //the valid starting point
            if(flag){
                startIdx=i;
                break;
            }
        }
        return startIdx;
    }
};
```

T.C. $O(n^2)$

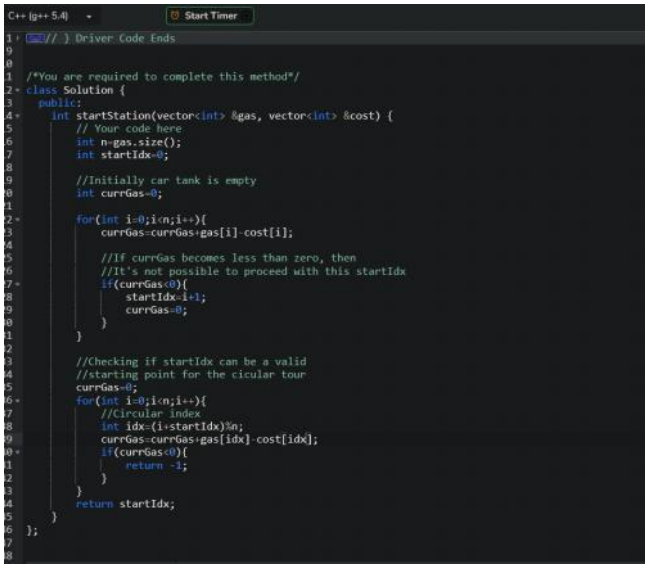
S.C. $O(1)$

[Expected Approach 1] Greedy Approach – O(n) Time & O(1) Space

we start by assuming the 0th index as the starting point for the circular tour. As we traverse the array, we calculate the available gas at each station, which is the previously available gas + gas[i] − cost[i]. If, at any station i, the available gas drops below zero, it indicates that a car can not proceed to the next station (i+1) from the current starting point. In such a case, we update the starting point to i+1 & continue the process. After completing the traversal of the array, we check whether the starting point is valid for the circular tour.

If a car starts at gas station A and cannot reach gas station B, then any gas station located between A and B cannot help us reach B either. But why?

If we start at **A** and are unable to reach **B**, but we can reach all the stations up to **B-1**. Let's assume a gas station **C** (**C<=B-1**) located between station **A** and **B**. When we arrive at **C** from **A**, we must have had a positive amount of gas in our tank. Therefore, if we can't reach B starting with positive amount of gas at **C**, it would be impossible to reach **B** from **C** with a zero amount of gas.



T.C. $O(n)$

S.C. $O(1)$

[Expected Approach 2] Greedy Approach in one-pass – O(n) Time &

O(1) space :-

This approach is optimization for the previous one. After completing the entire traversal of the array, instead of checking the validity by circularly traversing from the starting index, we calculate the total gas remaining (net gas & the cost difference). If the difference is greater than or equal to zero, then it's obvious that the starting point is valid; otherwise it is not possible to complete a circular loop.

```
/*You are required to complete this method*/
class Solution {
  public:
    int startStation(vector<int> &gas, vector<int> &cost) {
        // Your code here
        int n=gas.size();

        //Variables to track total and current remaining gas
        int totalGas=0;
        int currGas=0;
        int startIdx=0;

        //Traverse through each station to calculate remaining
        //gas in the tank, and total gas
        for(int i=0;i<n;i++){
            currGas+=gas[i]-cost[i];
            totalGas+=gas[i]-cost[i];

            //If currGas is negative, circular tour can't
            //start with this index, so update it to next one
            if(currGas<0){
                currGas=0;
                startIdx=i+1;
            }
        }

        //No solution exist
        if(totalGas<0)return -1;
        return startIdx;
    }
};
```

T.C. O(n)

S.C. O(1)