warmup problem
https://leetcode.com/problems/running-sum-of-1d-array

replit link
https://replit.com/join/mnstssvoum-hershyz

- problem: Given a string of text, return the most commonly occurring word.
    - approaches:
        - 2 ArrayLists
        - hashmap

Approach 1 - 2 ArrayLists

Main.java

```java
import java.util.Arrays;
import java.util.ArrayList;
import java.util.List;

class Main {
  public static void main(String[] args) {

    /*
      Given a string of text, return the most commonly occurring word.
    */
      String input = "The quick brown fox jumps over the lazy dog";


      input = input.toLowerCase();
      String[] words = input.split(" ");
      List<String> uniqueWords = new ArrayList<>();
      List<Integer> frequencies = new ArrayList<>();

      for (String word : words) {
        if (uniqueWords.contains(word)) {
          for (int i = 0; i < uniqueWords.size(); i++) {
            if (uniqueWords.get(i).equals(word)) {
              int curr = frequencies.get(i);
              curr++;
              frequencies.set(i, curr);
            }
          }
        }
        if (!uniqueWords.contains(word)) {
          uniqueWords.add(word);
          frequencies.add(1);
        }
      }

      int max = 0;
      String mostCommonWord = "";
      for (int i = 0; i < frequencies.size(); i++) {
        if (frequencies.get(i) > max) {
          max = frequencies.get(i);
          mostCommonWord = uniqueWords.get(i);
        }
      }

      System.out.println(mostCommonWord);
  }
}
```

Approach 2 - HashMap

```java
import java.util.*;

class Main {
  public static void main(String[] args) {

    /*
      Given a string of text, return the most commonly occurring word.

      HashMap Methods:
      HashMap.put(key, value)       [void]
      HashMap.get(key)              [(value type)]
      HashMap.replace(key, value)   [void]
      HashMap.containsKey(key)      [boolean]
    */
      String input = "she sells sea shells by the sea shore";



      HashMap<String, Integer> map = new HashMap<String, Integer>();
      String[] words = input.split(" ");

      for (String word : words) {
        if (map.containsKey(word)) {
          int curr = map.get(word);
          curr++;
          map.replace(word, curr);
        }
        if (!map.containsKey(word)) {
          map.put(word, 1);
        }
      }

      int max = 0;
      String mostCommonWord = "";
      for (Map.Entry<String, Integer> entry : map.entrySet()) {
          if (entry.getValue() > max) {
            max = entry.getValue();
            mostCommonWord = entry.getKey();
          }
      }

      System.out.println(mostCommonWord);
  }
}
```

Let's Revisit - https://leetcode.com/problems/two-sum/

Quadratic Time Complexity

```java
class Solution {

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

Linear Time Complexity

```java
class Solution {

    public int[] twoSum(int[] nums, int target) {

        //value, index
        HashMap<Integer, Integer> map = new HashMap<Integer, Integer>();

        for (int i = 0; i < nums.length; i++) {
            if (!map.containsKey(nums[i])) {
                map.put(nums[i], i);
            }
            int complement = target - nums[i];
            if (map.containsKey(complement) && map.get(complement) != i) {
                return new int[]{i, map.get(complement)};
            }
        }

        return new int[]{-1};
    }
}
```