Problem: Two Sum

Level: Easy

Given an array of integers, find a pair of integers that sums to a number Target.

For e.g, if A = [6,3,5,2,1,7]. Target = 4, Result= [3,1]

Questions to Clarify:
Q. How do you want the output?
A. Return a pair of numbers.

Q. What if there are multiple pairs that sum to Target?
A. Return any pair.

Q. What to return if there are no pairs that sum to Target?
A. Return null.

Solution:
The brute force solution is to iterate through every pair in the array. This is done by using a nested for loop. You should know how to write nested for loops. So if this is new to you, please practice the brute force solution as well.

Using a HashTable/HashSet, we can reduce the time complexity to *O(n)*. We loop through the array. Here is the key observation. Let's say we're at a[i]. If we've seen target-a[i] before, we can return the pair (a[i], target-a[i]), because they both sum to *target*.

Keep in mind that the HashSet solution runs in *O(n)* time - which is faster. However, it takes up *O(n)* space as well. This is because the HashSet takes *O(n)* space. The brute force solution took *O(1)* space. So we have traded off space to improve time complexity. This is a pattern you will See very often.

Pseudocode:
```
(Note: Never write pseudocode in an actual interview. Except when you're
writing a few lines quickly to plan out your solution. Your actual solution
should be in a real language and use good syntax.)
```

Brute Force:

```
for i: 0 to a.length - 1
    for j: i+1 to a.length - 1
           if a[i]+a[j] == target:
                 return new pair - (a[i], a[j])

no pair found, return null
```

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

Using Hash Set:

```
initialize empty hash set
for i: 0 to a.length-1
     if target-a[i] is in hash set
         return new pair - (a[i], target-a[i])
     else
         add a[i] to hash set
no pair found, return null
```

Time Complexity: O(n)
Space Complexity: O(n)

Test Cases:
Edge Cases: empty array, null array
Base Cases: single element, two elements (pair-exists/not-exists)
Regular Cases: more than 2 elements (pair-exists/not-exists)

```
// Brute force implementation - O(n^2) time
public static IntPair twoSumBruteForce(int[] a, int target) {
    if (a == null)
        return null;

    for (int i = 0; i < a.length; i++) {
        for (int j = i + 1; j < a.length; j++) {
            if (a[i] + a[j] == target)
                return new IntPair(a[i], a[j]);
        }
    }
    return null;
}

// HashSet implementation - O(n) time
public static IntPair twoSumUsingHashSet(int[] a, int target) {
    if (a == null)
        return null;

    HashSet<Integer> set = new HashSet<>();
    for (int i = 0; i < a.length; i++) {
        if (set.contains(target - a[i])) {
            return new IntPair(a[i], target - a[i]);
        } else {
            set.add(a[i]);
        }
    }
    return null;
}
/*
 * Helper Code: ask the interviewer if they want you to implement this.
 */

public static class IntPair {
    int first;
    int second;

    public IntPair(int first, int second) {
        super();
        this.first = first;
        this.second = second;
    }
    public int getFirst() {
        return first;
    }
    public void setFirst(int first) {
        this.first = first;
    }
    public int getSecond() {
        return second;
    }
    public void setSecond(int second) {
        this.second = second;
    }

    @Override
    public String toString() {
        return "["+first+","+second+"]";
    }
}
```