**1. Problem Discussion**

Here you will be provided an array of size ‘n’ and containing n elements and a target. All you need to do is find a pair in the array whose difference is equal to the target and print it, if such a pair exists else print -1. For example:[5,2,3,80,5,20] , target=78 Sounds easy!

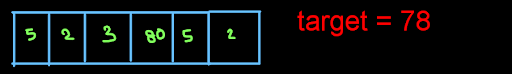
**2. Approach**

Naive Approach

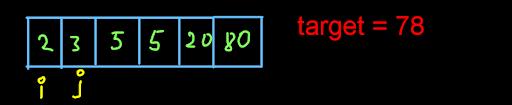
The approach is very simple here.You might think of solving this problem in O(n^2) time , Where you keep a pointer say ‘i’ and then you fix it for an element say (arr[0]) initially and then you check for other array elements from arr[j] (j=i+1 to arr.length-1) if the difference is equal to target you print it , otherwise you increase the value of i to i+1 and again check for arr[j].You continue to increase the value of i till i=arr.length-2. This process is going to take a lot of time.

Optimized Approach

Here You will be working with only two variables i and j .First of all you have to sort the array because it will help you get the answer in less time.

****

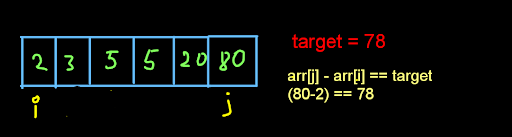
This is the original array and the target is 78. Then we sort the array and take two variables i and j.where i =0 and j=i+1;

****

Now we see that the difference of arr[j] and arr[i] is (3-2) =1 <78. So we increase the value of j to j+1 .So the difference arr[j]-arr[i] increases . Then we again check if(arr[j] - arr[i])>target or equal to target or lesser than target . If the (arr[j] - arr[i]) > target then increase the value of i to i+1 else if the value of (arr[j] - arr[i]) < target then increase the value of j to j+1. So in the previous image you can see that (3-2) =1 < 78 so we increased j to j+1.

****

But here again you can see (5-2) < 78,So again we increase j to j+1.

****

Here we get our difference of arr[j] and arr[i] equal to our target so we print it and return .

**3. Code and Expalination**

Code

ConsoleJava

import java.util.\*;

import java.io.\*;

public class Main {

public static void findPair(int[]arr,int target) {

//write your code here

Arrays.sort(arr);

int i=0;

int j=1;

while(i < arr.length && j < arr.length) {

if(arr[j] - arr[i] == target) {

System.out.println(arr[i] + " " + arr[j]);

return;

}

else if(arr[j] - arr[i] < target) {

j++;

}

else {

i++;

}

}

System.out.println("-1");

}

public static void main(String[]args) {

//input work

Scanner scn = new Scanner(System.in);

int n = scn.nextInt();

int[]arr = new int[n];

for(int i=0; i < n;i++) {

arr[i] = scn.nextInt();

}

int target = scn.nextInt();

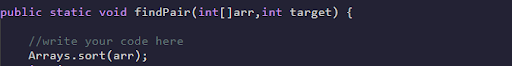
findPair(arr,target);

}

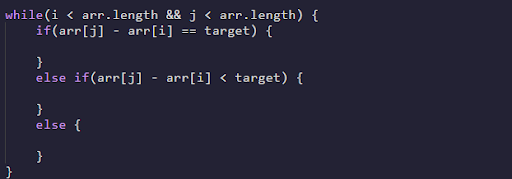
}

Code Explanation

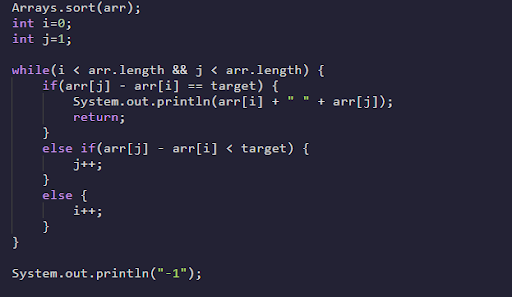
Code to sort the array.

****

Now we take two variables i (initialized to 0) and j(initialized to 1) and then run a while loop while i and j are less than length of the array.and then we handle the case where difference of arr[i] and arr[j] ==target ,< target or > target.

****

Here we increase j to j+1 when target> arr[j] - arr[i] ,increase i to i+1 when target< arr[j] - arr[i] or print the pair when difference arr[j] -arr[i] ==target.

****

**4. Analysis**

Time Complexity:

O(n log(n)) The time complexity for the function is O(n logn) because we are sorting the array in this question and after that it takes constant time .

Space Complexity

O(1) Constant Here we are not using any extra space so space complexity is constant.