OneD fun

In this problem you will implement six different static methods that use `int[]`. The motivation for the first and second problems was asked by Oracle.
The first problem, `getFloor(int[] values, int num)`, given a 1d array, find the floor of a given integer, `num` in the 1d array. The floor of `num` is the highest element in the 1d array less than or equal to the `num`, if the value does not exist, return `-num`.

The following code shows the results of the `getFloor` method.

The following code	Returns
`OneD_fun.getFloor(new int[]{13, 6, 8, 15, 3, 11}, 10)`	`8`
`OneD_fun.getFloor(new int[]{13, 6, 8, 15, 3, 11}, 2)`	`-2`

The second problem, `getCeiling(int[] values, int num)`, given a 1d array, find the ceiling in the 1d array of a given integer, `num`. The ceiling of `num` is the lowest element in the 1d array greater than or equal to `num`, if the value does not exist, return `-num`.

The following code shows the results of the `getCeiling` method.

The following code	Returns
`OneD_fun.getCeiling(new int[]{13, 6, 8, 15, 3, 11}, 12)`	`13`

The motivation for the third and fourth problems was asked by Amazon.
The third method, `makeSum(int[] values, int target)`, given `values`, an array of positive (>0) `int`, and an `int target` (greater than 0), return `true` if a subset of the elements `values` sum to `target`. The array `values` may contain duplicate numbers. For example, given:

- `new int[]{10, 2, 1, 3}` and `target` = 7, return `false`.
- `new int[]{10, 2, 1, 3]` and `target` = 13, return `true`.
- `new int[]{8, 2, 1, 8]` and `target` = 17, return `true`.

You may assume:

- `values.length > 0`
- `values[k] > 0, 0 <= k < values.length`
- `target > 0`.

The following code shows the results of the `makeSum` method.

The following code	Returns
`OneD_fun.makeSum(new int[]{10, 2, 1, 3}, 7)`	`false`
`OneD_fun.makeSum(new int[]{10, 2, 1, 3}, 13)`	`true`
`OneD_fun.makeSum(new int[]{8, 2, 1, 8}, 16)`	`true`

The fourth method, `getMissingSum(int[] values)`, given a 1d array, find the smallest positive integer that is not the sum of a subset of the array. The given array may contain duplicate numbers. For example, given:

- `new int[]{10, 2, 1, 3}`, return `7`.
- `new int[]{2, 3, 1, 2, 10}`, return `9`.
- `new int[]{8, 2, 44, 1, 4}`, return `16`
- `new int[]{1, 2}`, return `4`
- `new int[]{10, 6, 3, 2}`, return `1`

You may assume:

- `values.length > 0`
- `values[k] > 0, 0 <= k < values.length`
- `target > 0`.

The following code shows the results of the `getMissingSum` method.

The following code	Returns
`OneD_fun.getMissingSum(new int[]{10, 2, 1, 3})`	`7`
`OneD_fun.getMissingSum(new int[]{2, 3, 1, 2, 10})`	`9`
`OneD_fun.getMissingSum(new int[]{8, 2, 44, 1, 4})`	`16`
`OneD_fun.getMissingSum(new int[]{1, 2})`	`4`
`OneD_fun.getMissingSum(new int[]{10, 6, 3, 2})`	`1`

The motivation for the fifth and sixth problems was asked by Lyft.

Given a 1d array of (positive or negative) integers and an integer target, return which contiguous elements of the 1d array sum to target.

For example, if the 1d array is [1, 2, 3, 4, 5] and target is 9,
then it should return [2, 3, 4], since 2 + 3 + 4 = 9.

Read the following questions on the following pages carefully as slight modifications have been made to the original question asked by Lyft.

The fifth method, `getClosestNthPartialSum(int[] values, int target, int n)`,
Given `values`, a 1d array of integers (positive, negative, or zero), an `int target`, and an `int n`, return which contiguous `n` elements of `values` that sum **closest** to `target`.
For example:

- If the 1d array is [1, 2, 3, 4, 5], target = 9, and n = 3, then return [2, 3, 4], since 2 + 3 + 4 = 9.
- If the 1d array is [1, 2, 3, 4, 5, 6], target = 15, and n = 4, then return [2, 3, 4, 5], since 2 + 3 + 4 + 5 = 14 which is the closes to 15.
- If the 1d array is [5, -2, -8, 7, -5, 11], target = -2, and n = 2, then return [-8, 7], since -8 + 7 = -1 which is the closes to -2.

You may assume:

- `values.length > 0`
- `n <= values.length`
- there will be exactly one solution
- It is possible a copy `values` will be returned.

The following code shows the results of the `getClosestNthPartialSum` method.

The following code	Returns
`int[] ans = OneD_fun.getClosestNthPartialSum(` `new int[] {1, 2, 3, 4, 5}, 9, 3);`	
`ans.length`	3
`ans[0]`	2
`ans[1]`	3
`ans[2]`	4

Another example showing the results of `getClosestNthPartialSum` method.

The following code	Returns
`int[] ans = OneD_fun.getClosestNthPartialSum(` `new int[] {1, 2, 3, 4, 5, 6}, 15, 4);`	
`ans.length`	4
`ans[0]`	2
`ans[1]`	3
`ans[2]`	4
`ans[3]`	5

The sixth method is on the following page.

The sixth method, `getClosestPartialSum(int[] values, int target)`, Given `values`, a 1d array of integers (positive, negative, or zero), and `int target`, return which contiguous elements of `values` sum **closest** to `target`.
Note, if two solutions exist, return the solution with the fewest elements. You may assume there will exactly one solution with the fewest elements.
For example:

- If the 1d array is [6, -1, -4, 2, 10, -7, 9], target = 12, then return [2, 10], since 2 + 10 = 12.
- If the 1d array is [11, 6, -1, -4, 2, 10, -7, 9], target = 1, then return [6, -1, -4], since 6 – 1 – 4 =1 and contains fewer elements than [-4, 2, 10 , -7].
- If the 1d array is [0, 1, 2, 3, 4, 5, 6], target = 11, then return [5, 6], since 5+6 = 11.

You may assume:

- `values.length > 0`
- there will be exactly one solution.
- It is possible a copy `values` will be returned.

The following code shows the results of the `getClosestPartialSum` method.

The following code	Returns
`int[] ans = OneD_fun.getClosestPartialSum(` `new int[] { 6, -1, -4, 2, 10, -7, 9}, 12);`	
`ans.length`	2
`ans[0]`	2
`ans[1]`	10

Another example showing the results of `getClosestPartialSum` method.

The following code	Returns
`int[] ans = OneD_fun.getClosestPartialSum(` `new int[] { 11, 6, -1, -4, 2, 10, -7, 9}, 1);`	
`ans.length`	3
`ans[0]`	6
`ans[1]`	-1
`ans[2]`	-4

More examples of the `getClosestPartialSum` method are on the next page.

Another example showing the results of `getClosestPartialSum` method.

The following code	Returns
`int[] ans = OneD_fun.getClosestPartialSum(` `  new int[] {0, 1, 2, 3, 4, 5, 6}, 11);`	
`ans.length`	2
`ans[0]`	5
`ans[1]`	6

Another example showing the results of `getClosestPartialSum` method.

The following code	Returns
`int[] ans = OneD_fun.getClosestPartialSum(` `  new int[] { 10, 0, -12, 8, -14, 7, -9, 6, -11}, -20);`	
`ans.length`	5
`ans[0]`	-12
`ans[1]`	8
`ans[2]`	-14
`ans[3]`	7
`ans[4]`	-9

This page left intentionally blank