Given an array of stick lengths, use 3 of them to construct a non-degenerate triangle with the maximum possible perimeter. Return an array of the lengths of its sides as 3 integers in non-decreasing order.

If there are several valid triangles having the maximum perimeter:

1. Choose the one with the longest maximum side.
2. If more than one has that maximum, choose from them the one with the longest minimum side.
3. If more than one has that maximum as well, print any one them.

If no non-degenerate triangle exists, return $[-1]$.

Example

$sticks = [1, 2, 3, 4, 5, 10]$

The triplet $(1, 2, 3)$ will not form a triangle. Neither will $(4, 5, 10)$ or $(2, 3, 5)$, so the problem is reduced to $(2, 3, 4)$ and $(3, 4, 5)$. The longer perimeter is $3 + 4 + 5 = 12$.

Function Description

Complete the maximumPerimeterTriangle function in the editor below.

maximumPerimeterTriangle has the following parameter(s):

- int sticks[n]: the lengths of sticks available

Returns

- int[3] or int[1]: the side lengths of the chosen triangle in non-decreasing order or -1

Input Format

The first line contains single integer n, the size of array $sticks$.
The second line contains n space-separated integers $sticks[i]$, each a stick length.

Constraints

- $3 \leq n \leq 50$
- $1 \leq sticks[i] \leq 10^9$

Explanation

Sample Case 0:

There are 2 possible unique triangles:

1. $(1, 1, 1)$
2. $(1, 3, 3)$

The second triangle has the largest perimeter, so we print its side lengths on a new line in non-decreasing order.

Sample Case 1:

The triangle $(1, 2, 3)$ is degenerate and thus can't be constructed, so we print -1 on a new line.

```
def maximumPerimeterTriangle(sticks):
    # Write your code here
    largest_per = float('-inf')
    triangles = []

    # to find all possible triangles
    for x in range(len(sticks)):
        for y in range(x+1, len(sticks)):
            for z in range(y+1, len(sticks)):
                # current possible triangle
                triangle = [sticks[x], sticks[y], sticks[z]]
                triangle.sort(reverse=False)

                # if current three values does not satisfy conditions for a valid triangle
                if triangle[2] >= triangle[0] + triangle[1]:
                    continue
                else:
                    # current perimeter
                    perimeter = sum(triangle)

                    # if we found two such triangles whose perimeter is maximum
                    # add them to list of possible results
                    if perimeter == largest_per:
                        triangles.append(triangle)

                    # if current perimeter is greater than largest perimeter found so far
                    # all the triangles in the possible results list are obsolete/useless, hence clear the triangles array
                    # and add the new triangle to the list
                    if perimeter > largest_per:
                        triangles.clear()
                        triangles.append(triangle)
                        largest_per = perimeter

    # if there are more than one
    if len(triangles) > 1:
        # start with the first one
        resultant_triangle = triangles[0]

        # compare it with remaining ones to find the best one
        for triangle in triangles[1:]:
            # if max of T1 > T2, result = T1
            if max(triangle) > max(resultant_triangle):
                resultant_triangle = triangle

                                                resultant_triangle: list
            # if not, compare and find the max of min of both Ts
            elif max(triangle) == max(resultant_triangle):
                if min(triangle) > min(resultant_triangle):
                    resultant_triangle = triangle

            else:
                continue

        return resultant_triangle
    else:
        # it could be a possiblity that we didn't find any such triangle
        # hence the len of resultant list will be zero
        # check that
        if len(triangles) != 0:
            # if it's not empty there is only one such triangle
            # return that
            return triangles[0]
        else:
            # if resultant list is empty, return -1
            return [-1]
```

Line: 30 Col: 34

Upload Code as File Test against custom input Run Code Submit Code