

Sort Transformed Array

Given a **sorted** array of integers *nums* and integer values *a*, *b* and *c*. Apply a function of the form $f(x) = ax^2 + bx + c$ to each element *x* in the array.

The returned array must be in **sorted order**.

Expected time complexity: **$O(n)$**

Example:

```
nums = [-4, -2, 2, 4], a = 1, b = 3, c = 5,
```

```
Result: [3, 9, 15, 33]
```

```
nums = [-4, -2, 2, 4], a = -1, b = 3, c = 5
```

```
Result: [-23, -5, 1, 7]
```

Credits:

Special thanks to [@elmirap](#) for adding this problem and creating all test cases.

Solution 1

the problem seems to have many cases $a > 0$, $a = 0$, $a < 0$, (when $a = 0$, $b > 0$, $b < 0$). However, they can be combined into just 2 cases: $a > 0$ or $a < 0$

1. $a > 0$, two ends in original array are bigger than center if you learned middle school math before.

2. $a < 0$, center is bigger than two ends.

so use two pointers i, j and do a merge-sort like process. depending on sign of a , you may want to start from the beginning or end of the transformed array. For $a = 0$ case, it does not matter what b 's sign is. The function is monotonically increasing or decreasing. you can start with either beginning or end.

```
public class Solution {
    public int[] sortTransformedArray(int[] nums, int a, int b, int c) {
        int n = nums.length;
        int[] sorted = new int[n];
        int i = 0, j = n - 1;
        int index = a >= 0 ? n - 1 : 0;
        while (i <= j) {
            if (a >= 0) {
                sorted[index--] = quad(nums[i], a, b, c) >= quad(nums[j], a, b, c)
? quad(nums[i++], a, b, c) : quad(nums[j--], a, b, c);
            } else {
                sorted[index++] = quad(nums[i], a, b, c) >= quad(nums[j], a, b, c)
? quad(nums[j--], a, b, c) : quad(nums[i++], a, b, c);
            }
        }
        return sorted;
    }

    private int quad(int x, int a, int b, int c) {
        return a * x * x + b * x + c;
    }
}
```

written by [xuyirui](#) original link [here](#)

Solution 2

$$ax^2+bx+c = a(x + b/2a)^2 + c - b^2/4a$$

So use offset as **$c - b^2/4a$**

$ax^2+bx+c - \text{offset}$ will be always positive or negative

Then iteratively pop max (or min if a is negative) from both ends and push it into final array

```
function sortTransformedArray(nums, a, b, c) {
  var arr = nums.map(n => a * n * n + b * n + c);
  var offset = a ? c - (b * b) / (4 * a) : Math.min(arr[0], arr.slice(-1)[0]);
  var res = [];

  for (var l = 0, r = arr.length - 1; l <= r;) {
    if (Math.abs(arr[l] - offset) >= Math.abs(arr[r] - offset)) {
      res.push(arr[l++]);
    } else {
      res.push(arr[r--]);
    }
  }

  return res[0] > res[res.length - 1] ? res.reverse() : res;
}
```

written by [linfongi](#) original link [here](#)

Solution 3

```
class Solution {
public:
    vector<int> sortTransformedArray(vector<int>& nums, int a, int b, int c)
    {
        vector<int> res(nums.size());
        if (nums.size() == 0) return res;
        int i = 0, j = nums.size() - 1;
        if (a > 0) {
            int index = nums.size() - 1;
            while (i <= j) {
                if (transform(nums[i], a, b, c) > transform(nums[j], a, b, c))
                {
                    res[index--] = transform(nums[i], a, b, c);
                    i++;
                } else {
                    res[index--] = transform(nums[j], a, b, c);
                    j--;
                }
            }
        } else {
            int index = 0;
            while (i <= j) {
                if (transform(nums[i], a, b, c) < transform(nums[j], a, b, c))
                {
                    res[index++] = transform(nums[i], a, b, c);
                    i++;
                } else {
                    res[index++] = transform(nums[j], a, b, c);
                    j--;
                }
            }
        }
        return res;
    }

    int transform(int num, int a, int b, int c) {
        return a * num * num + b * num + c;
    }
};
```

written by [xiaohui5319](#) original link [here](#)

From [Leetcode](#).