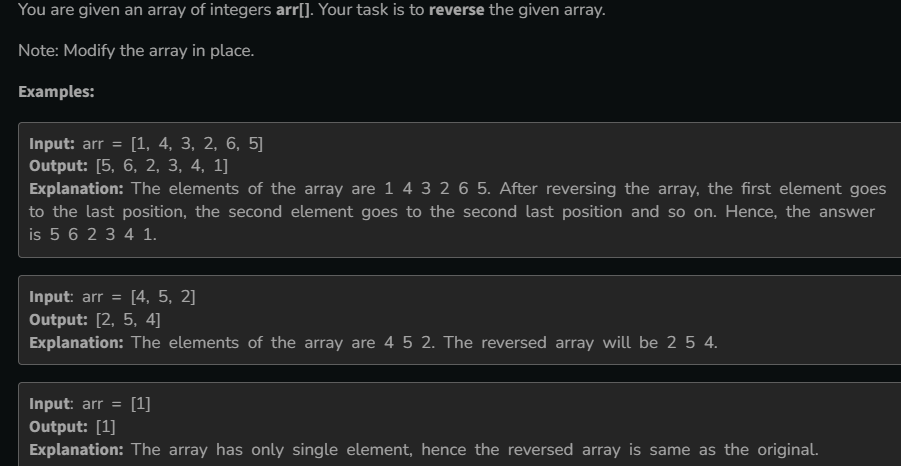
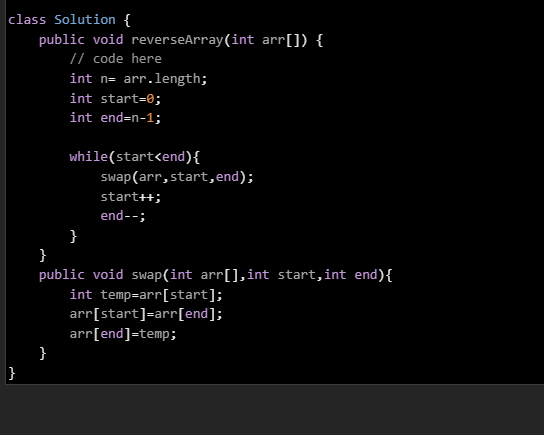
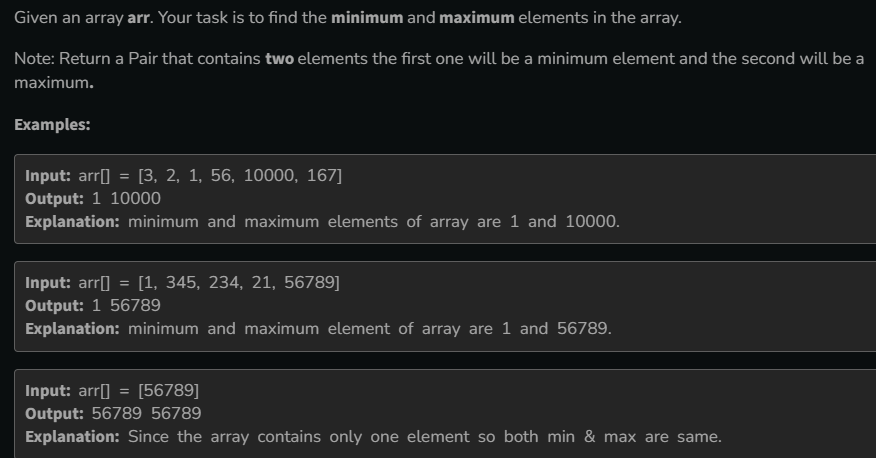
Array’s Problem

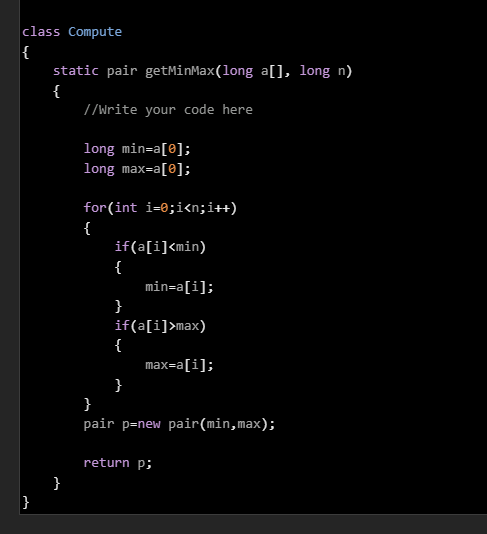
1.Reverse an Array



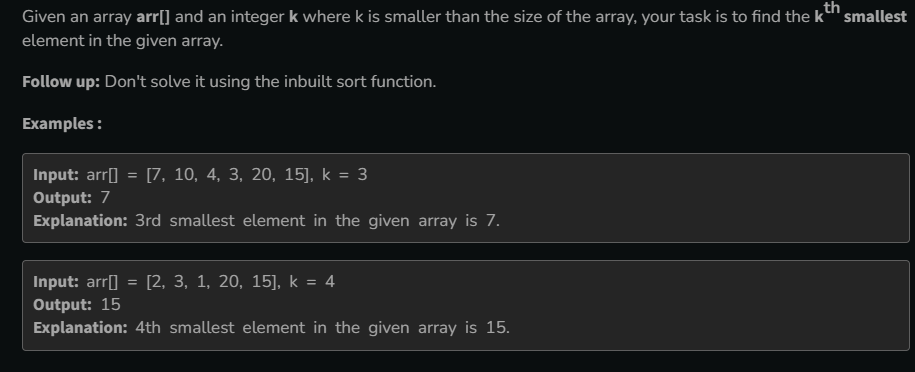


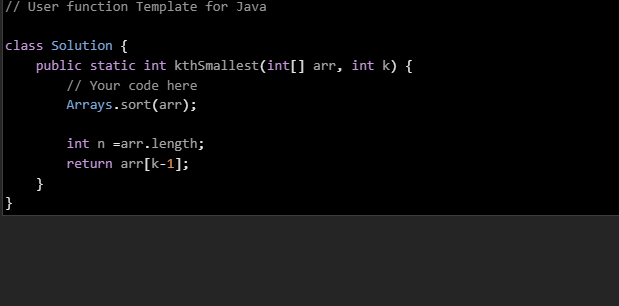
2.Min and max in Array



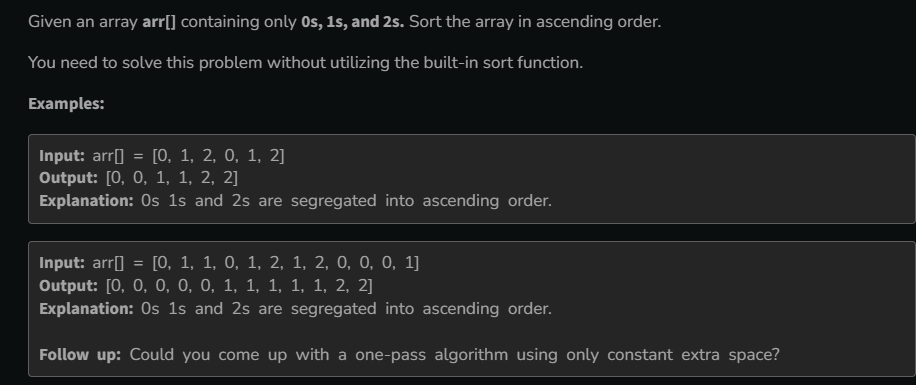


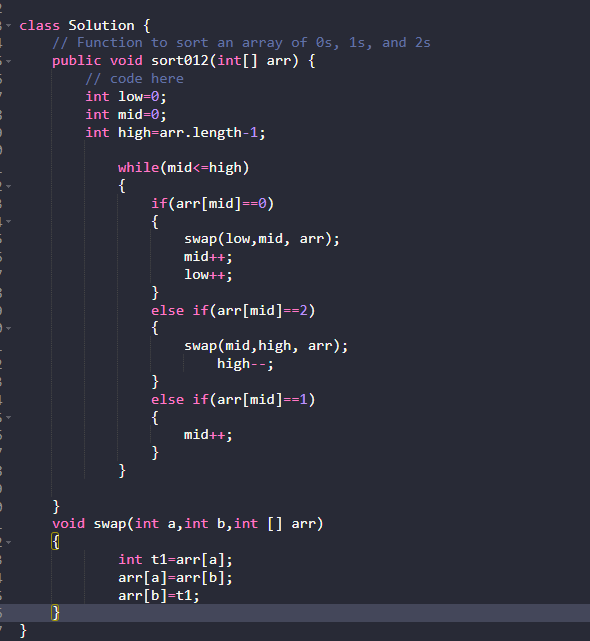
3.Kth Smallest Element



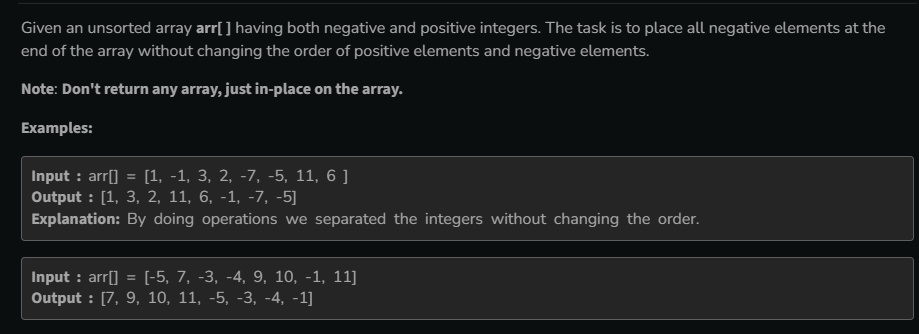


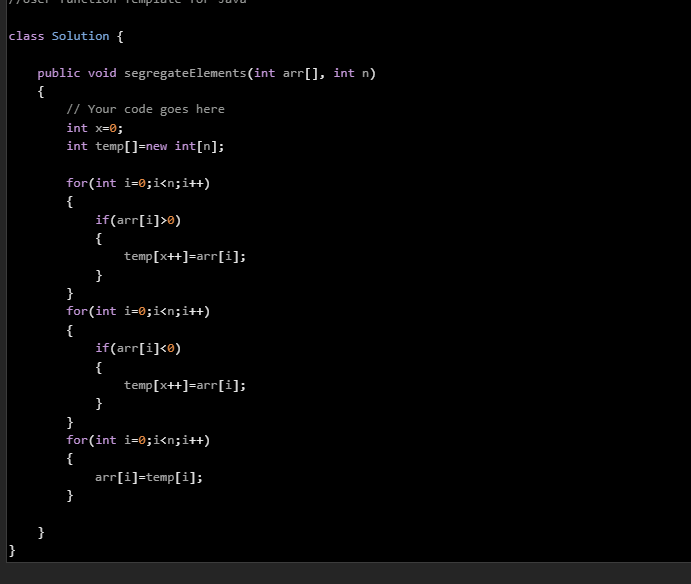
4.Sort 0’s,1’s and 2s



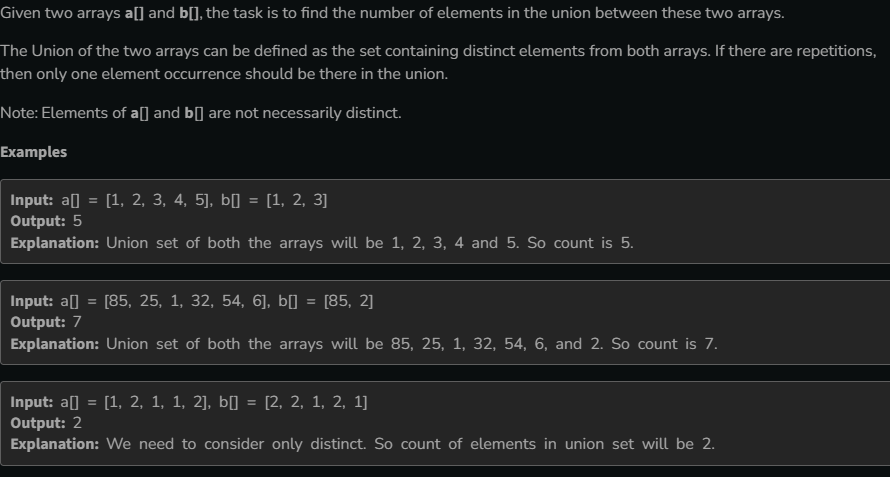


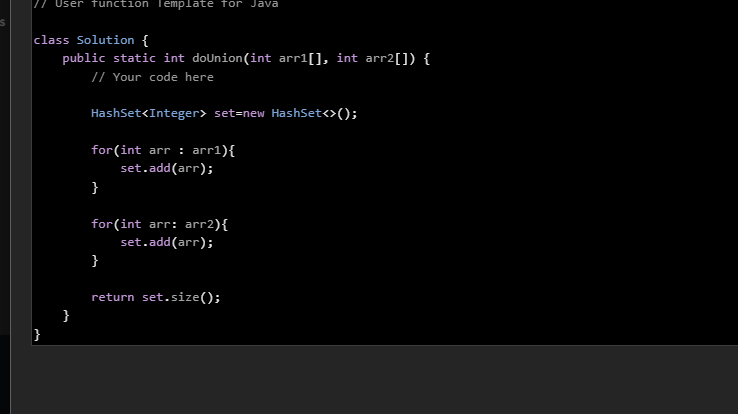
5.Move all Negative element to the End



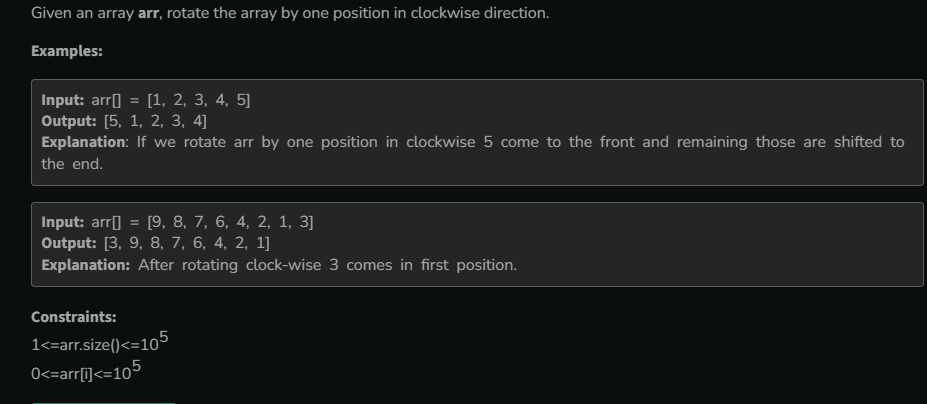


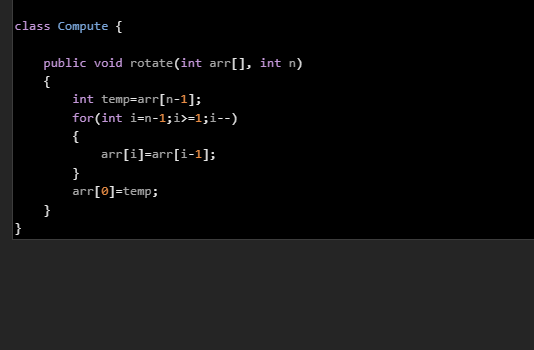
6.Union of array with Duplicates



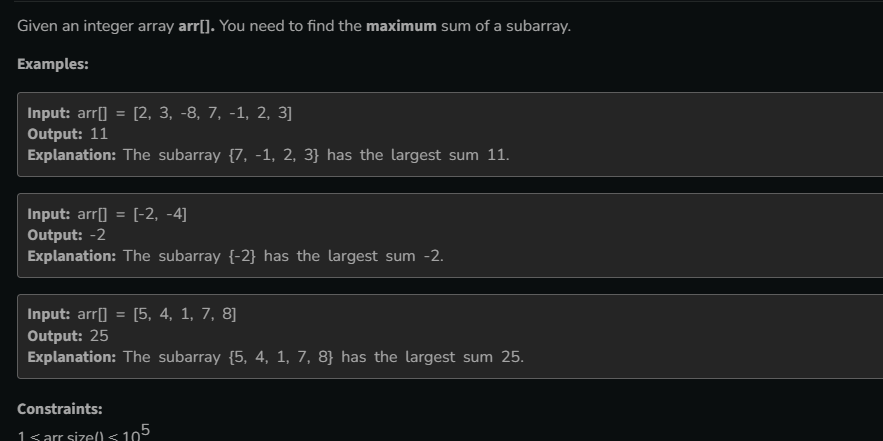


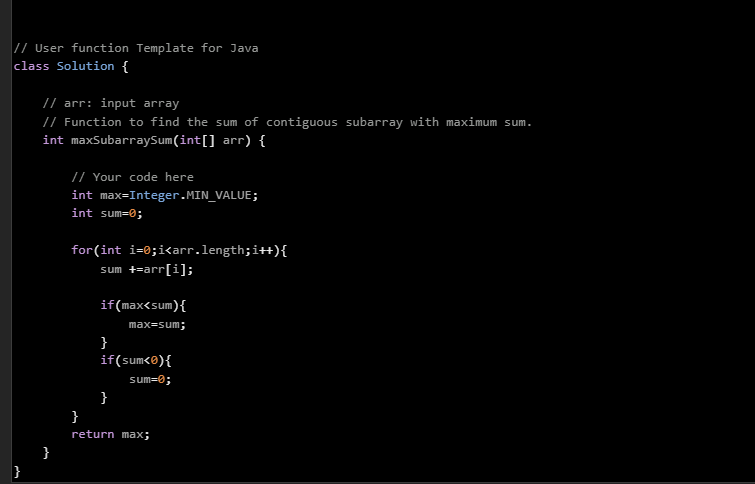
7.Rotate array by one



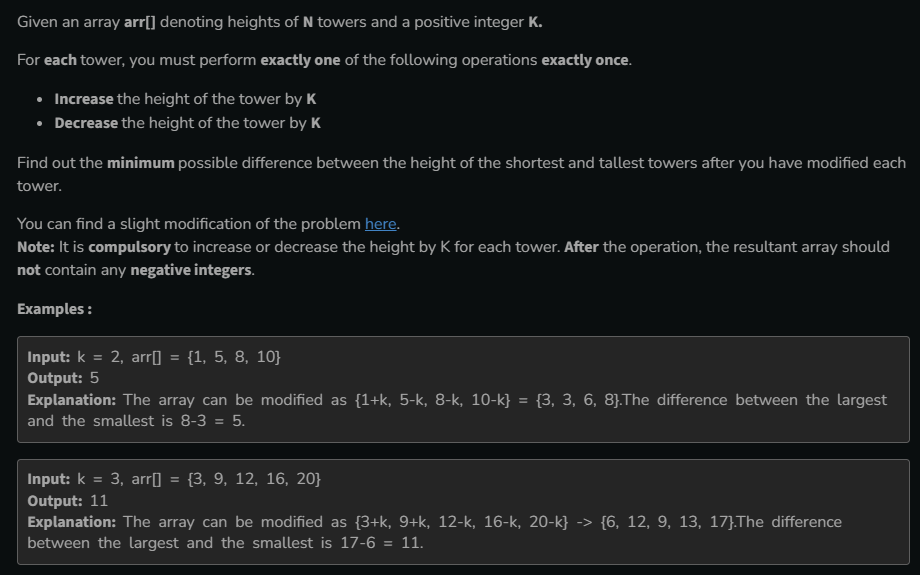


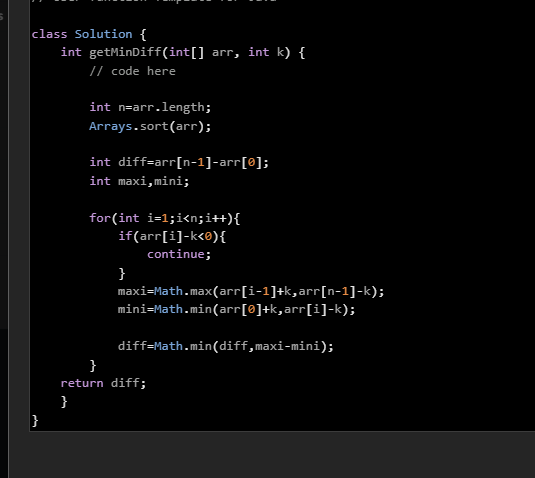
8.Kadane’s Algorithm



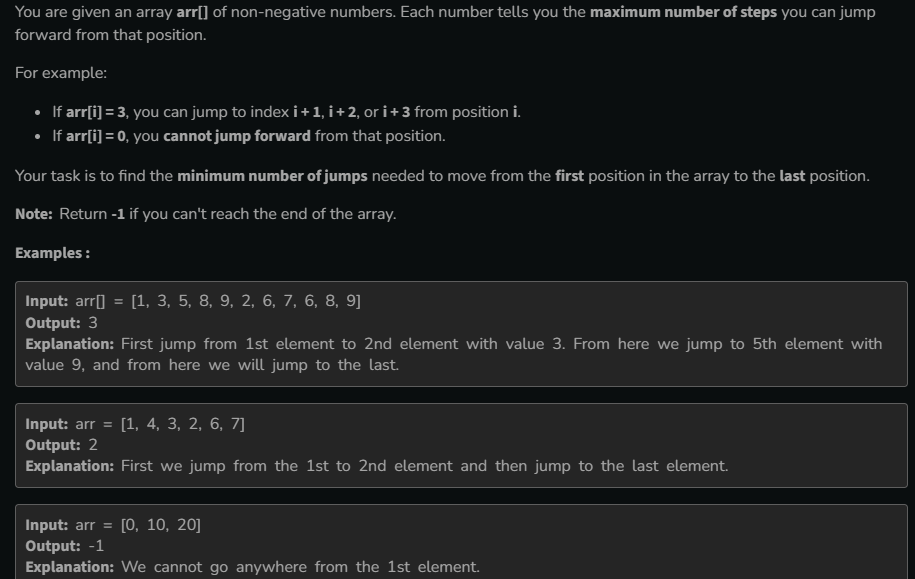


9.Minimise the heights II



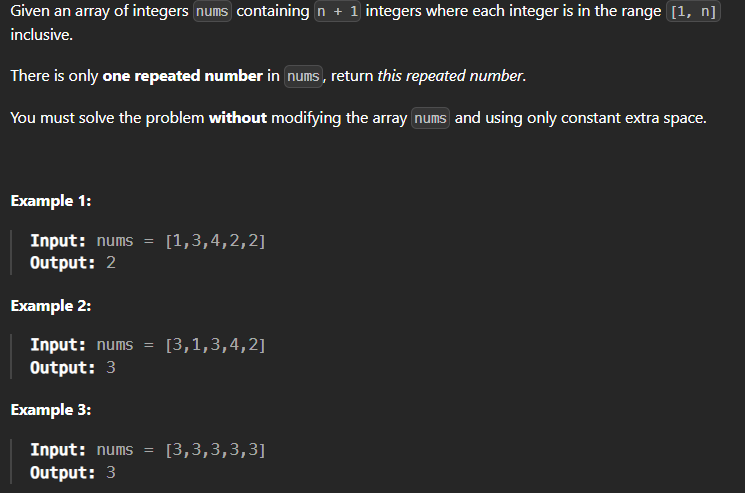


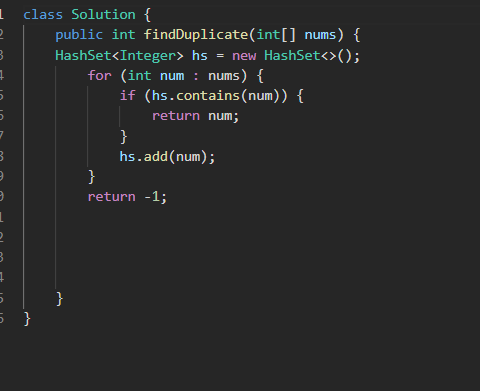
10.Minimum Jumps





11.Find The Duplicate number





// TC: O(n), SC: O(1)

public int findDuplicate\_CyclicSortWay(int[] nums) {

int i = 0;

while (i < nums.length) {

if (nums[i] != nums[nums[i]]) swap(nums, i, nums[i]);

else i++;

}

return nums[0];

}

public void swap(int[] arr, int i, int j) {

int temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

// TC: O(n), SC: O(1)

public int findDuplicate\_FloydCycleDetection\_AnotherWay(int[] nums) {

if (nums.length == 0) return -1;

int slow = nums[0], fast = nums[nums[0]];

while (slow != fast) {

slow = nums[slow];

fast = nums[nums[fast]];

}

slow = 0;

while (slow != fast) {

slow = nums[slow];

fast = nums[fast];

}

return slow;

}

// TC: O(n), SC: O(1)

public int findDuplicate\_FloydCycleDetection(int[] nums) {

int slow = nums[0], fast = nums[0];

do {

slow = nums[slow];

fast = nums[nums[fast]];

} while (slow != fast);

slow = nums[0];

while (slow != fast) {

slow = nums[slow];

fast = nums[fast];

}

return slow;

}

// TC: O(n), SC: O(1)

public int findDuplicate\_ModifyInput(int[] nums) {

for (int i = 0; i < nums.length; i++) {

int val = abs(nums[i]);

if (nums[val] < 0) return val;

nums[val] \*= -1;

}

return -1;

}

// TC: O(n), SC: O(n)

public int findDuplicate\_SetApproach(int[] nums) {

Set<Integer> set = new HashSet<>();

for (int num : nums) {

if (set.contains(num)) return num;

set.add(num);

}

return 0;

}

// TC: O(nlogn), SC: O(1)

public int findDuplicate\_SortApproach(int[] nums) {

Arrays.sort(nums);

for (int i = 1; i < nums.length; i++) {

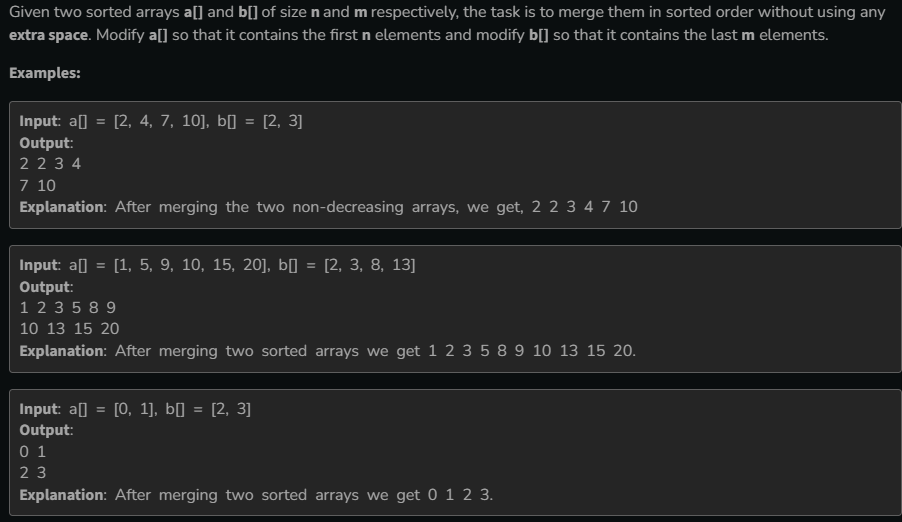
if (nums[i] == nums[i - 1]) return nums[i];

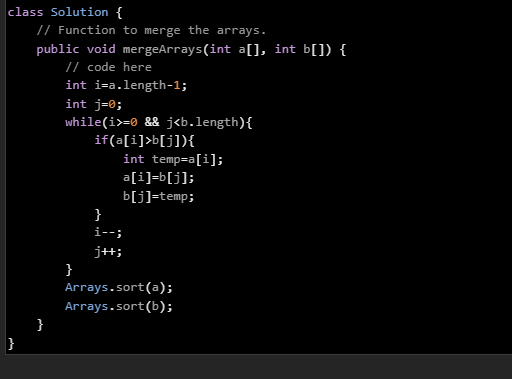
}

return -1;

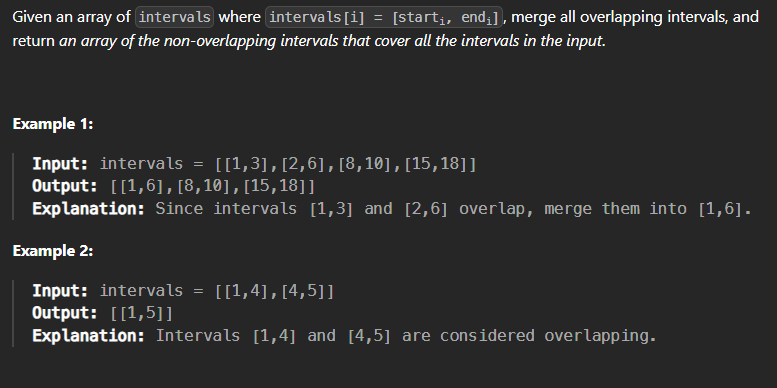
}

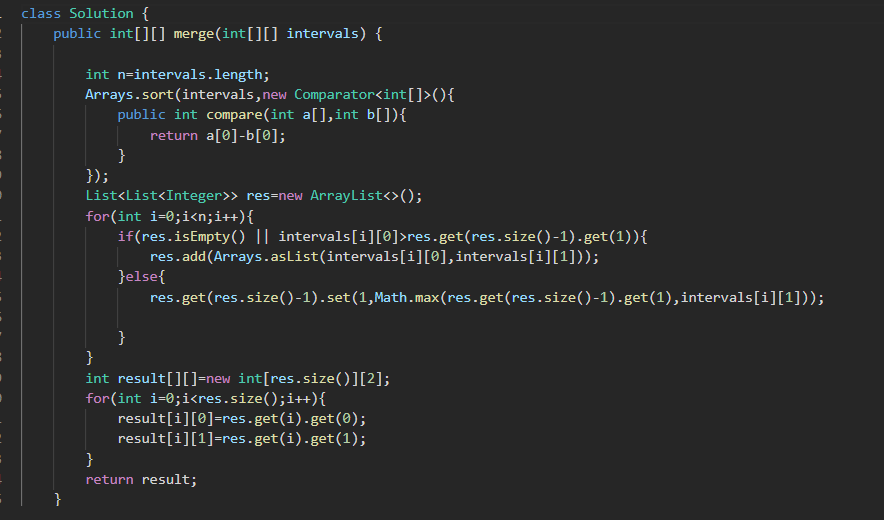
12.Merge Without Extra space



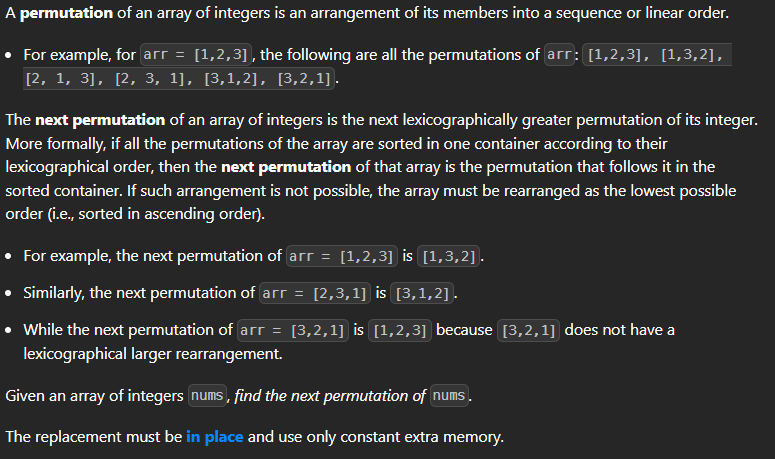


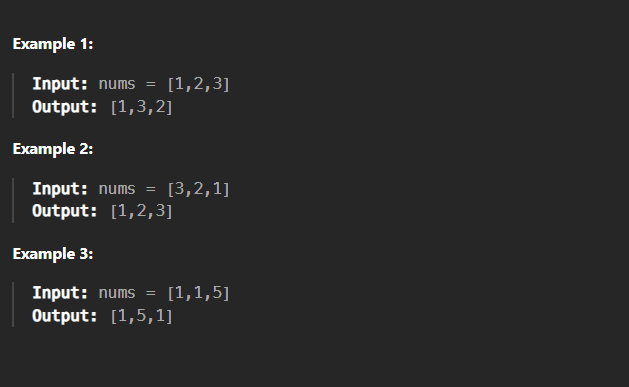
13.Merge Intervals

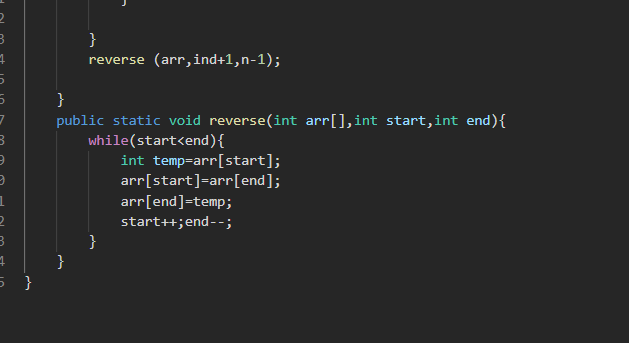
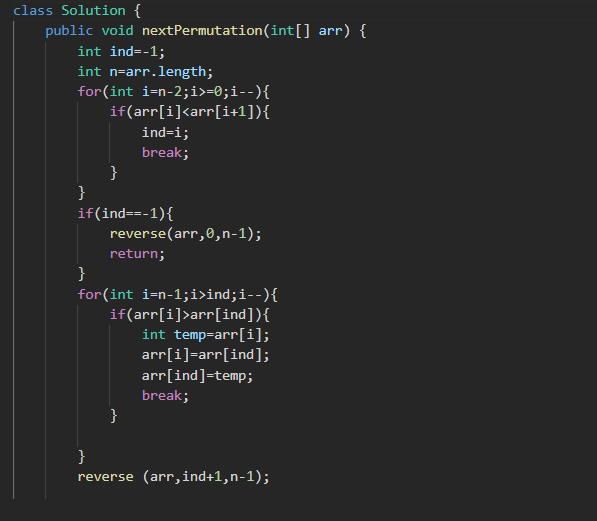




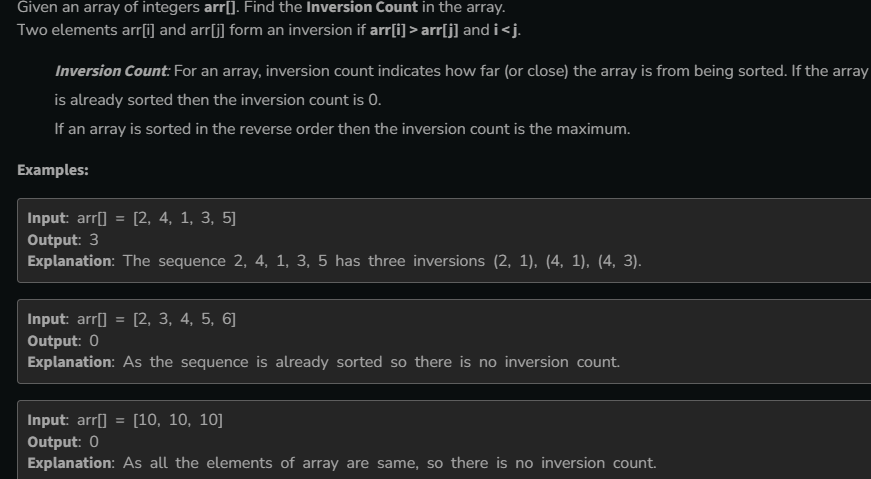
14.Next Permutation

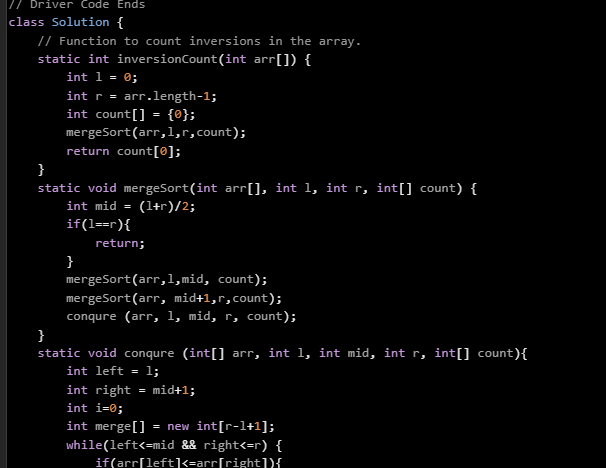


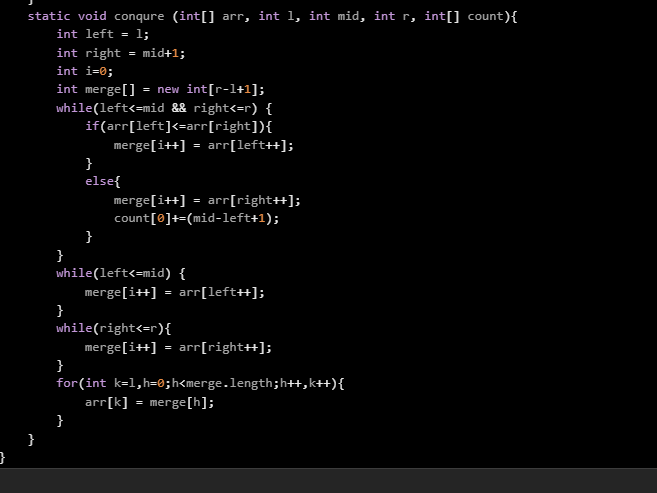




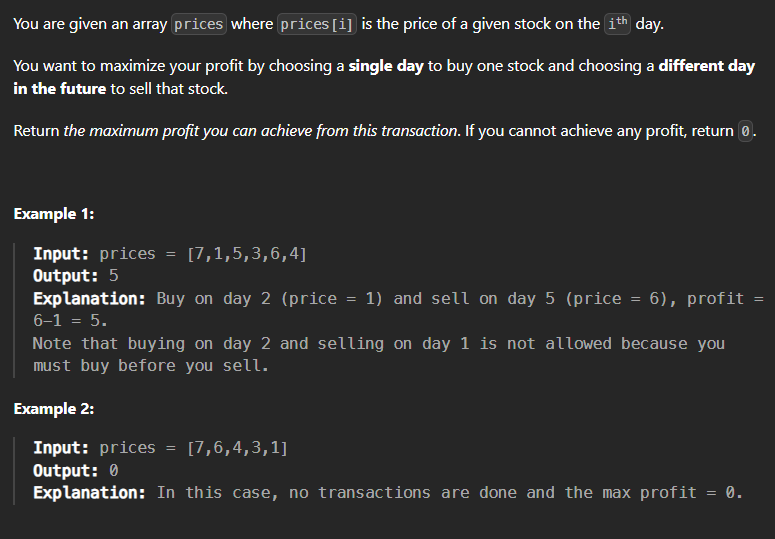
15.Count Inversion

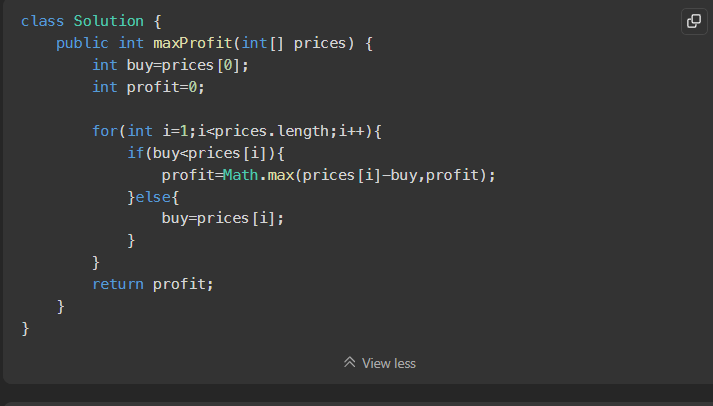




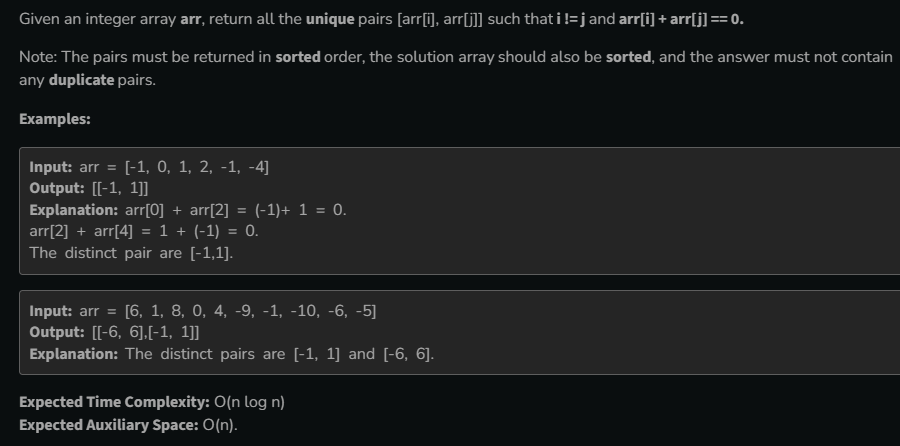


16.Best Time to buy and sell stock



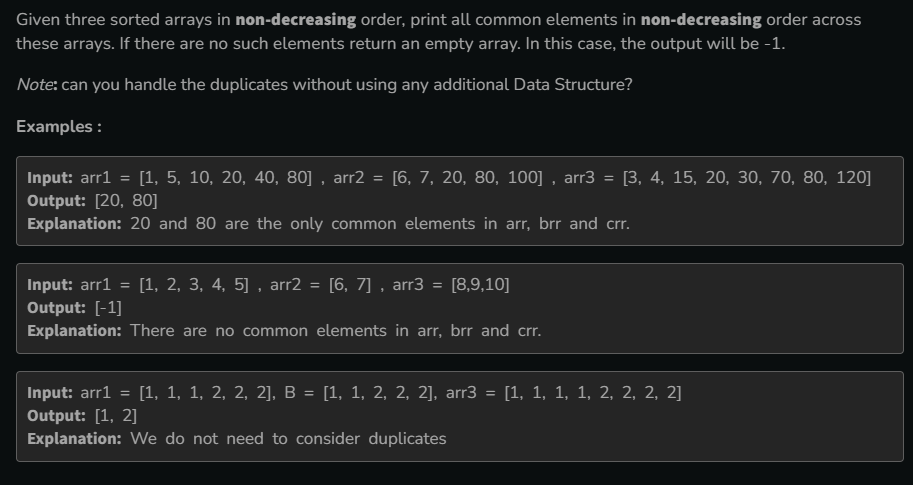


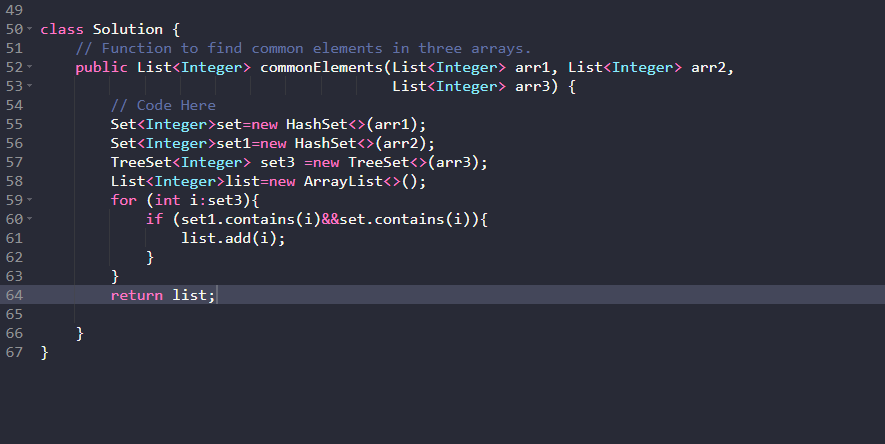
17.Two sum pair with 0 sum



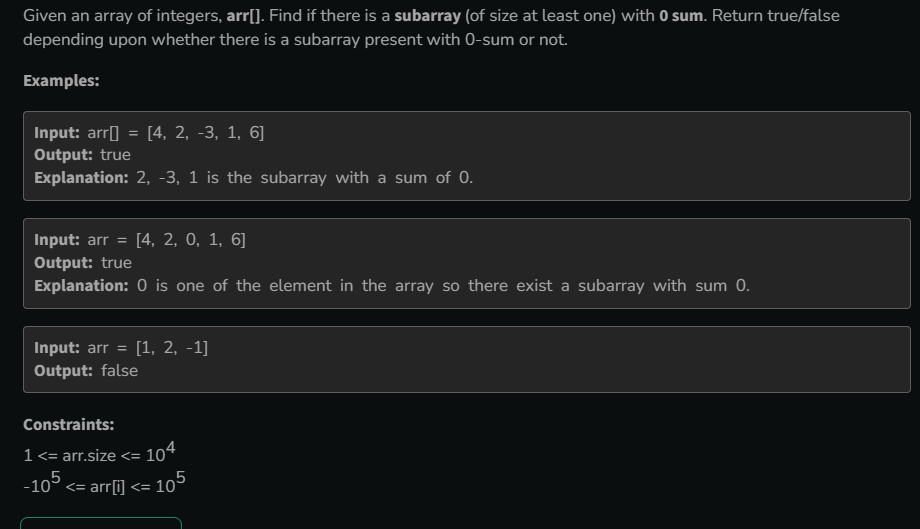


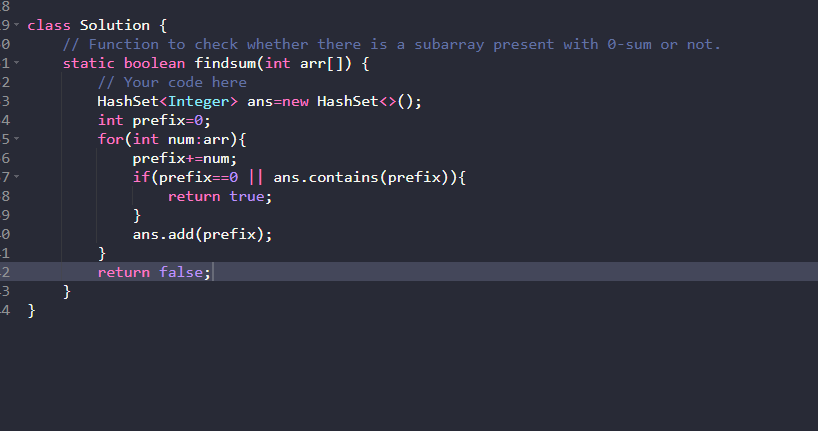
18.Common in 3 sorted array



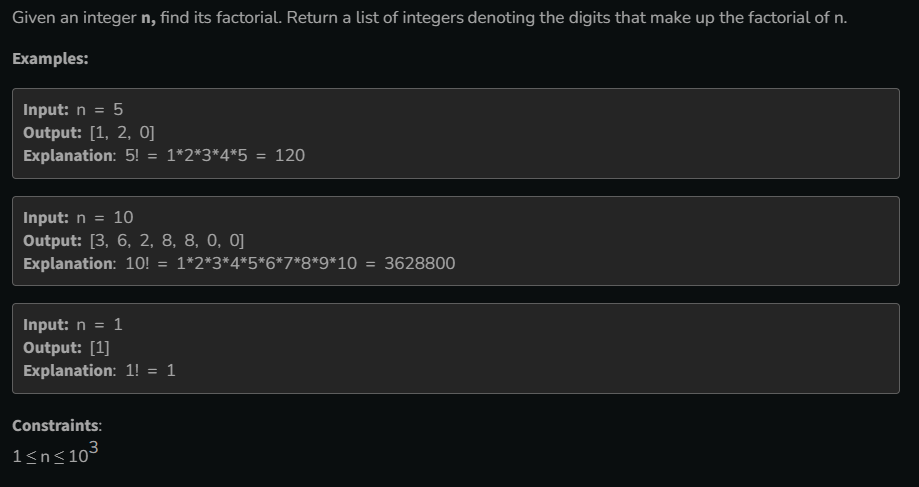


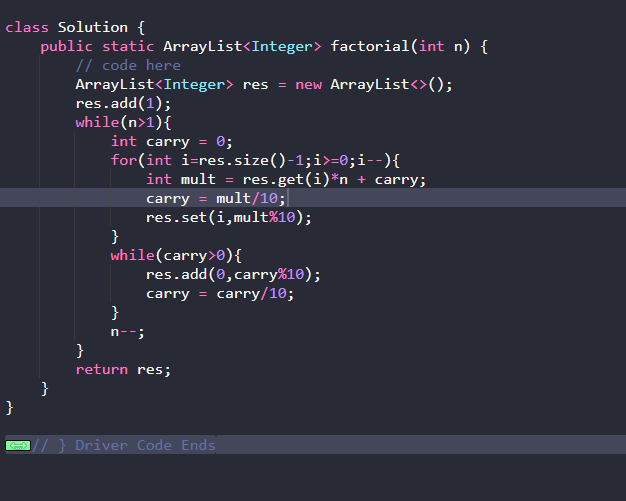
19.Subarray with 0 sum

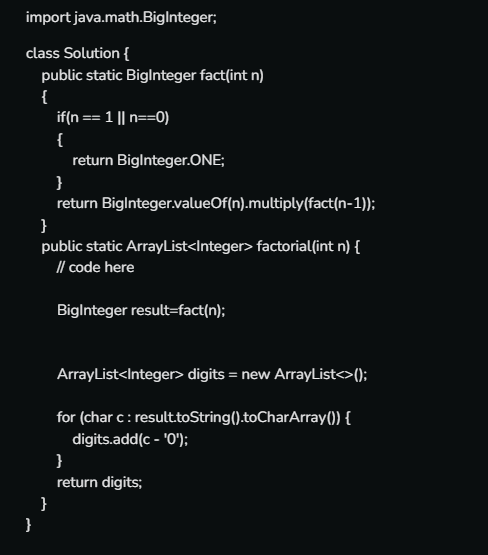




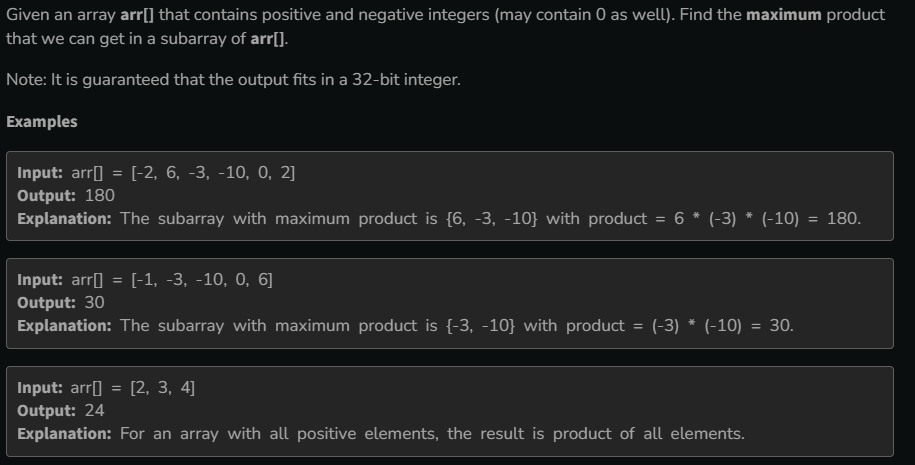
20.Factorial of large number

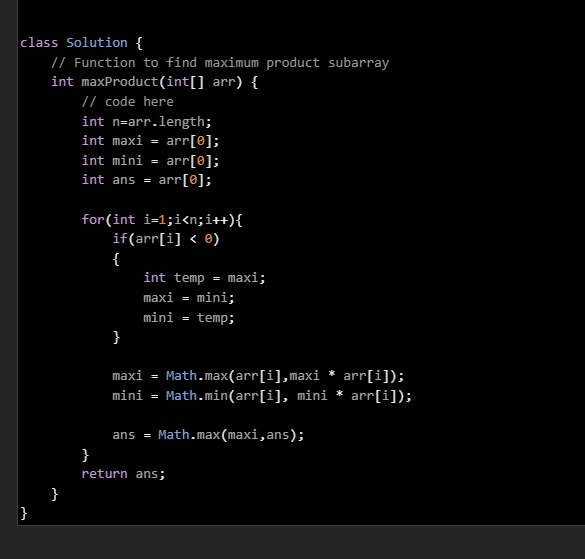




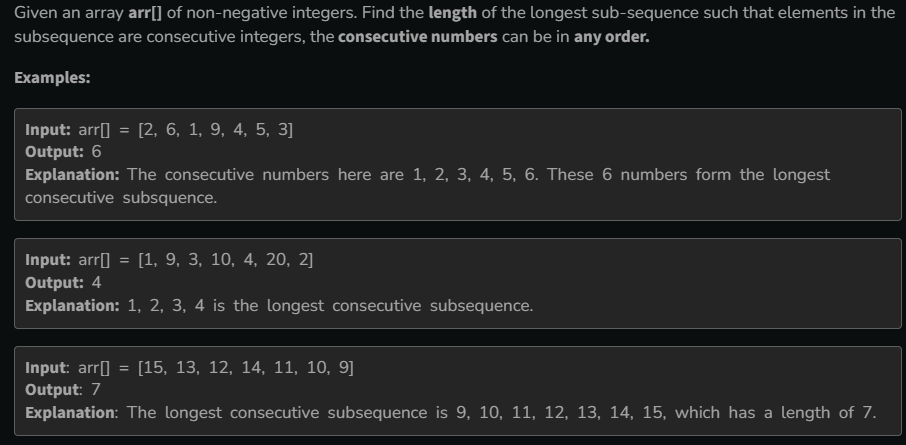


21.Maximum Product subarray



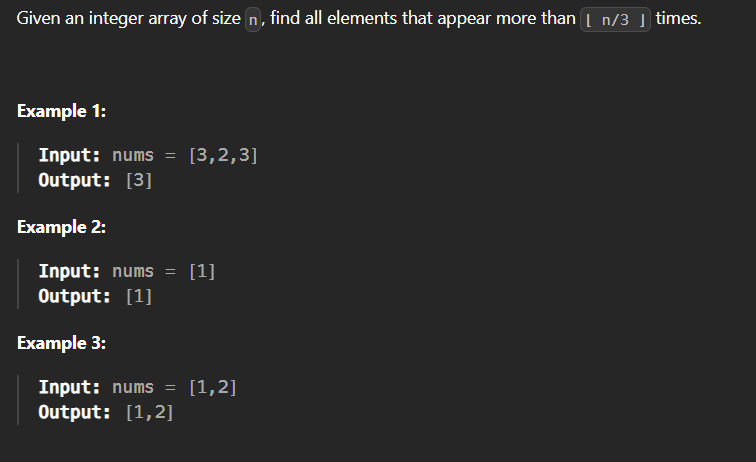


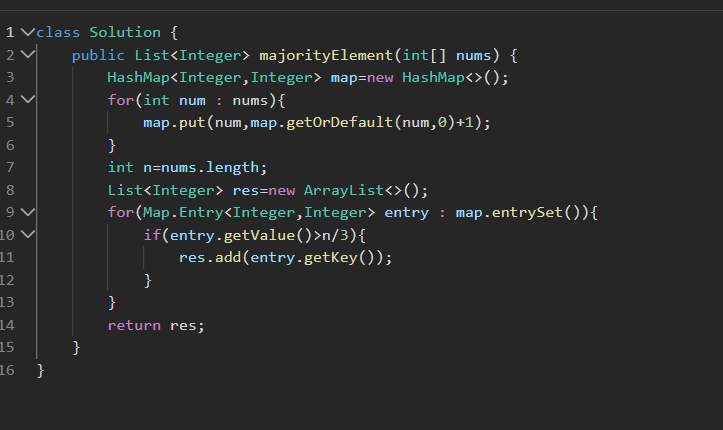
22.Longest Consecutive subsequence



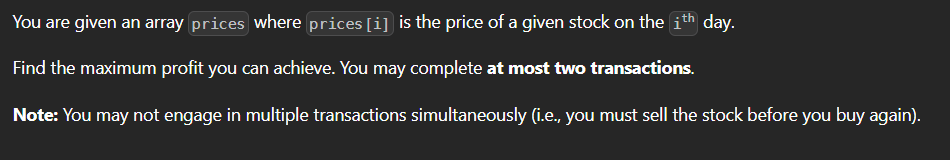


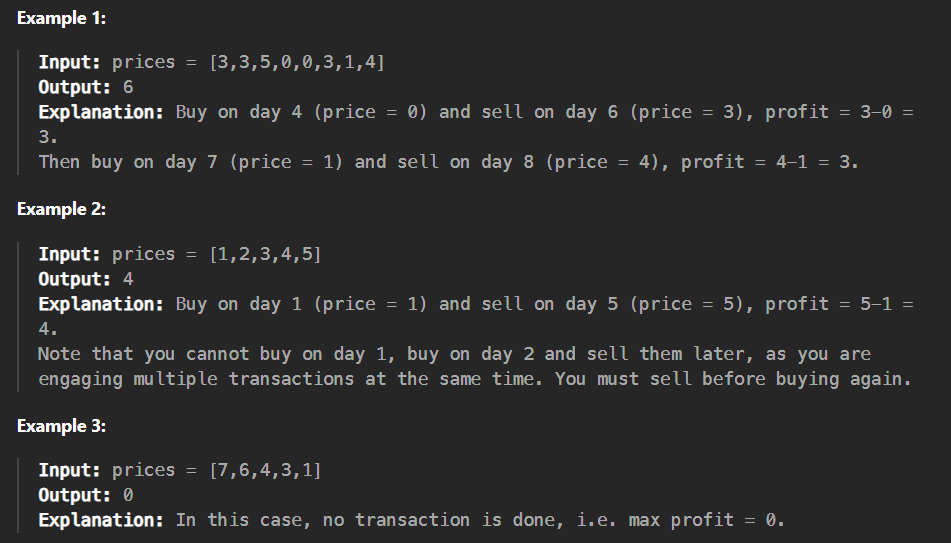
23.Majority Element II

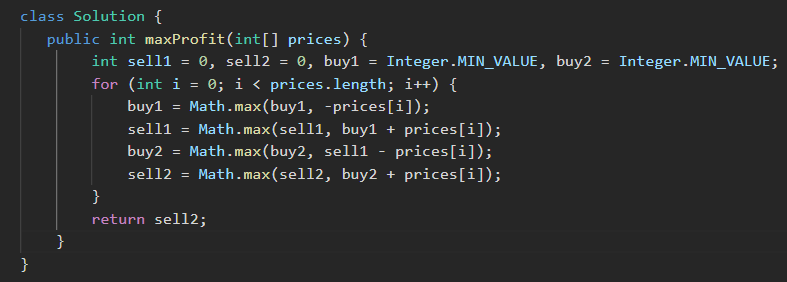




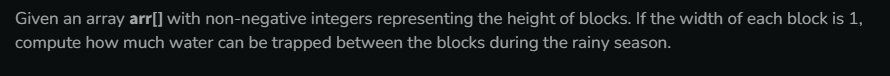
24.Best time to sell and buy stock III

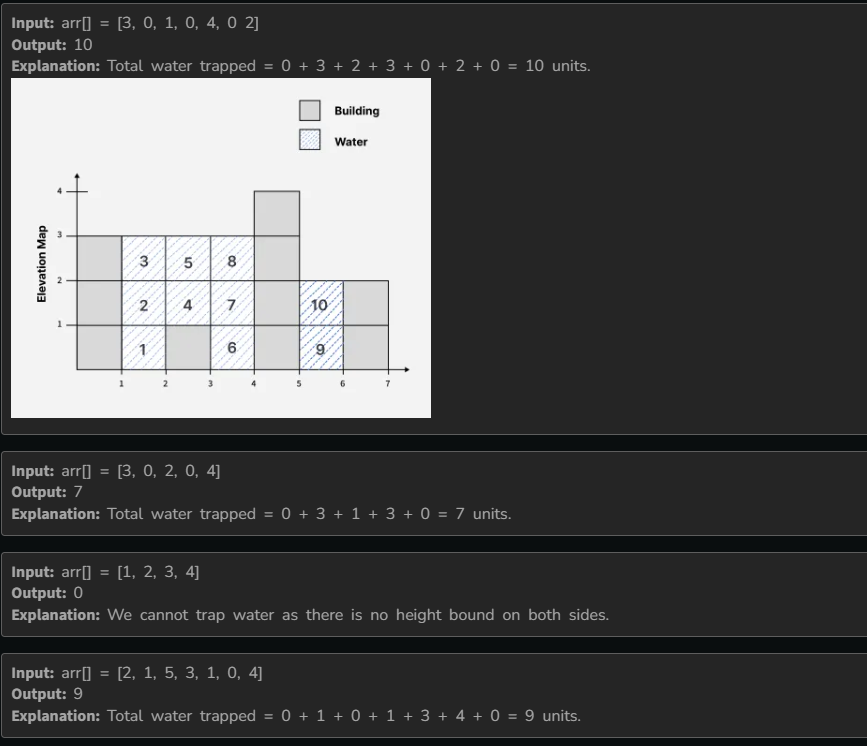


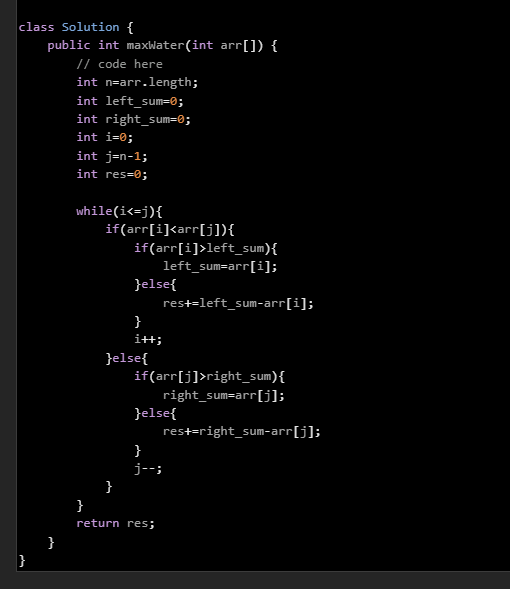




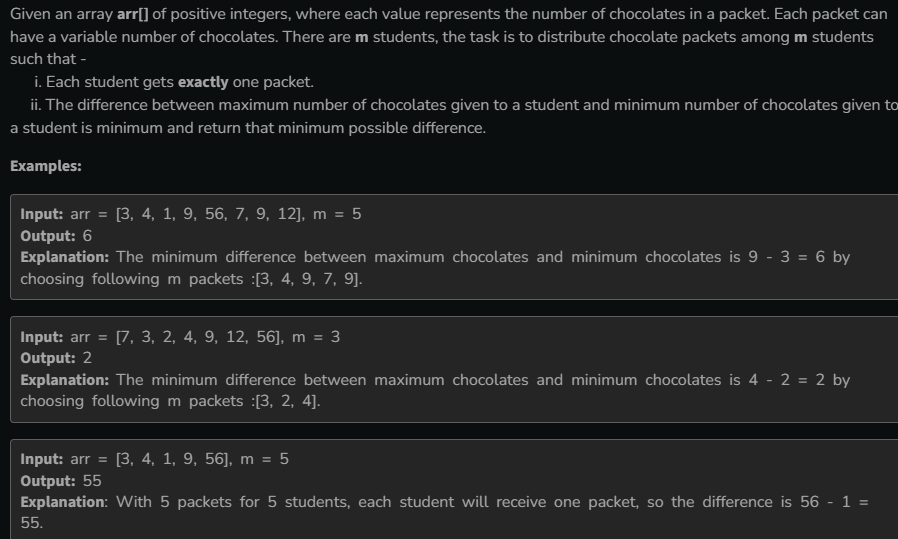
25.Trapping Rain Water

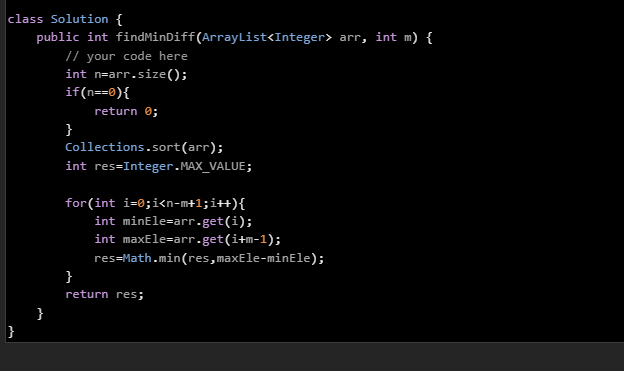




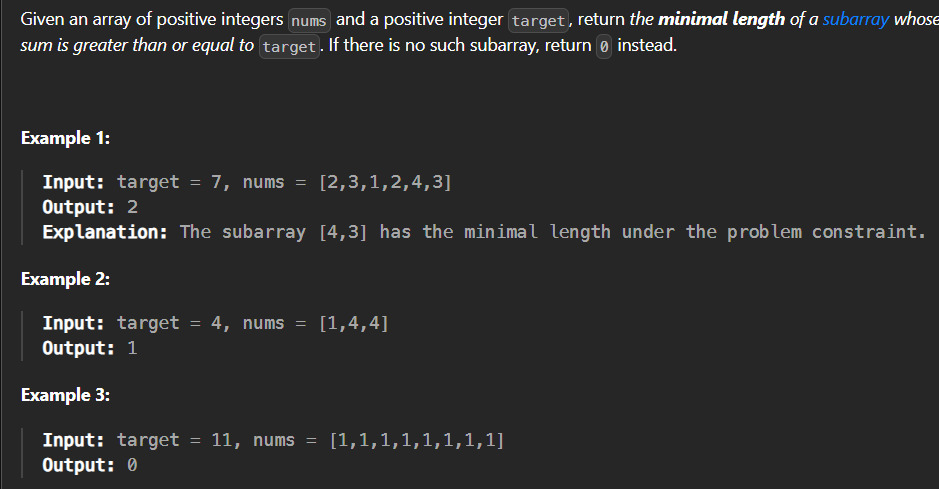


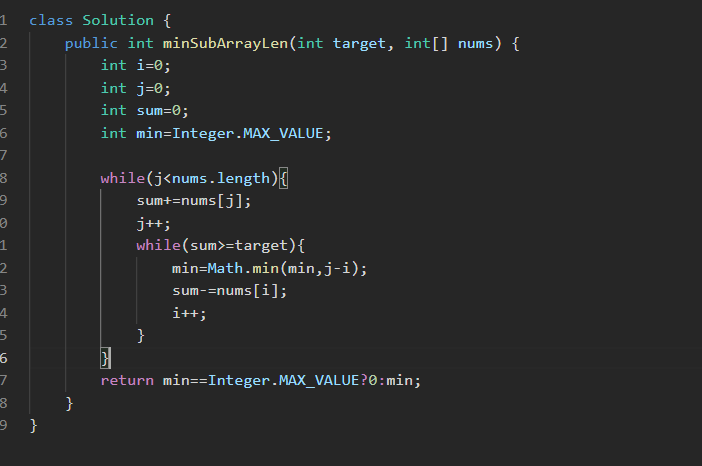
26.Chocolate distribution problem



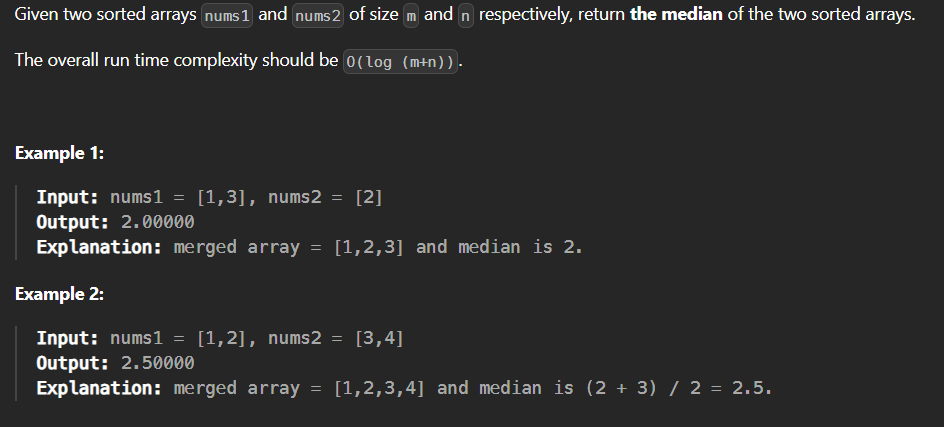


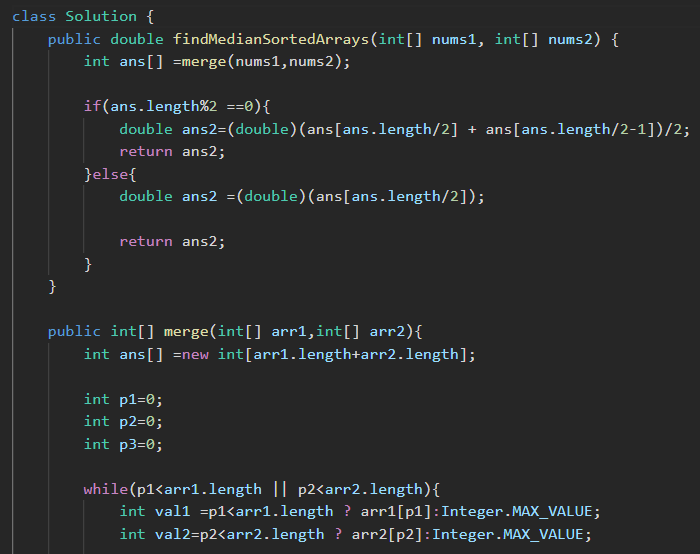
27.Minimum size subarray sum

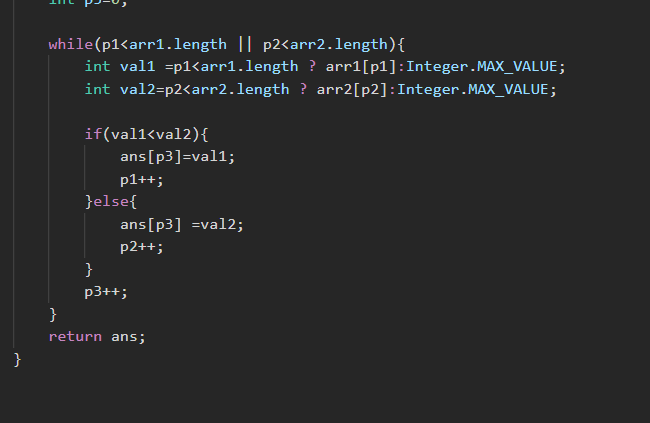




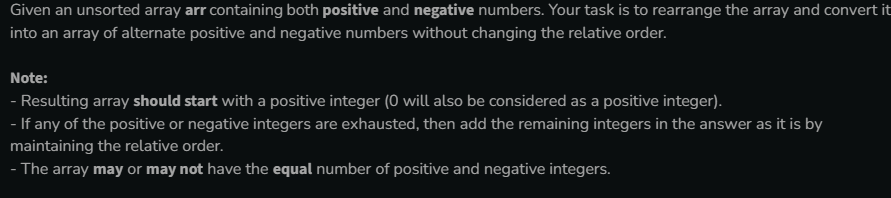
28.Median of two sorted array

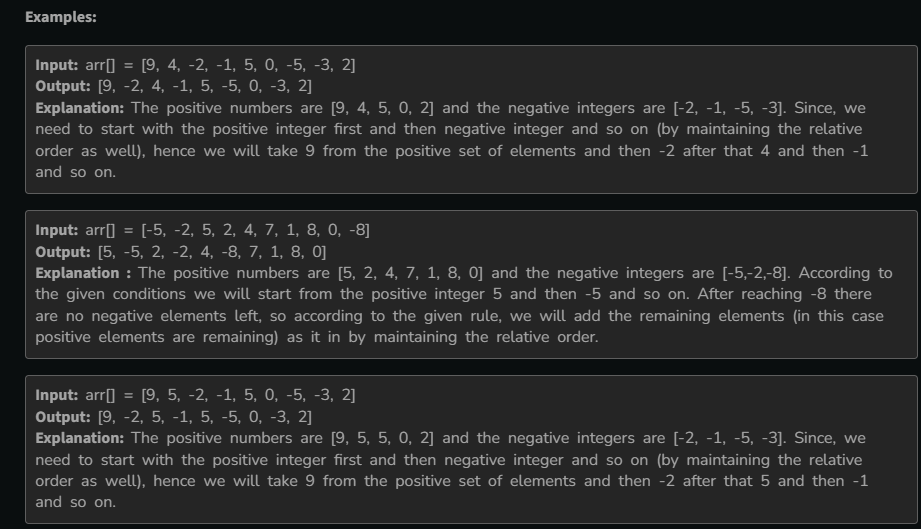


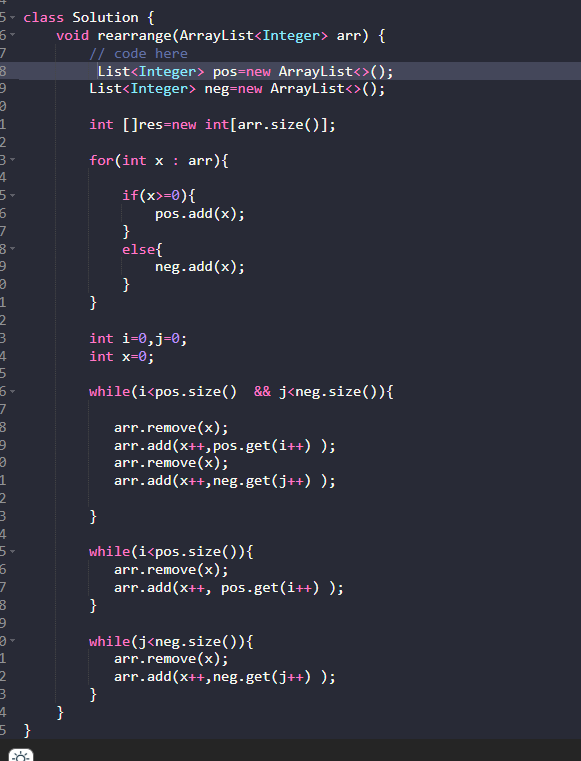




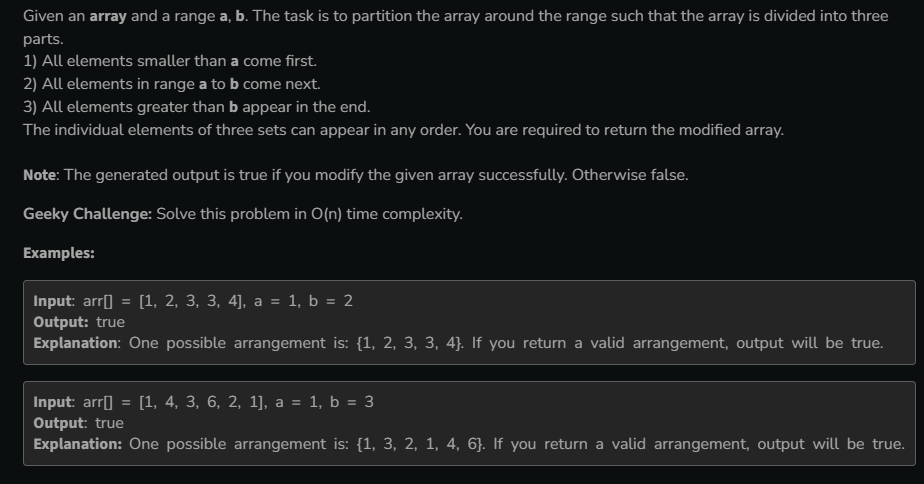
29.Alternate Positive Negative

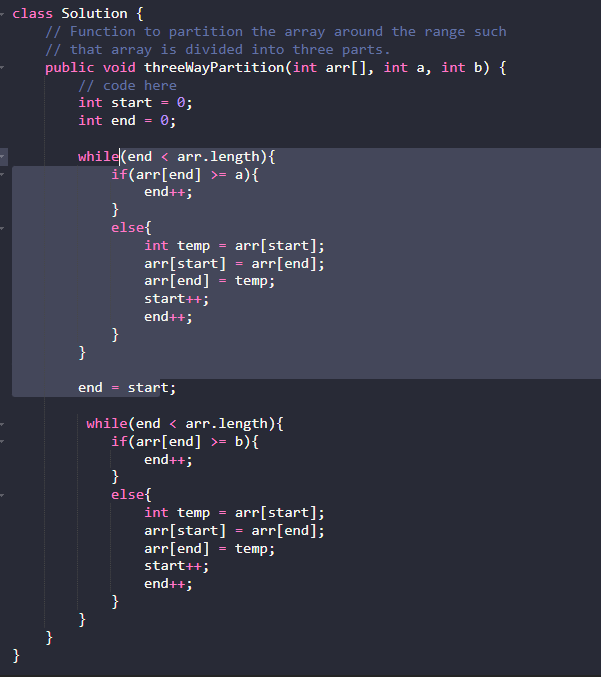




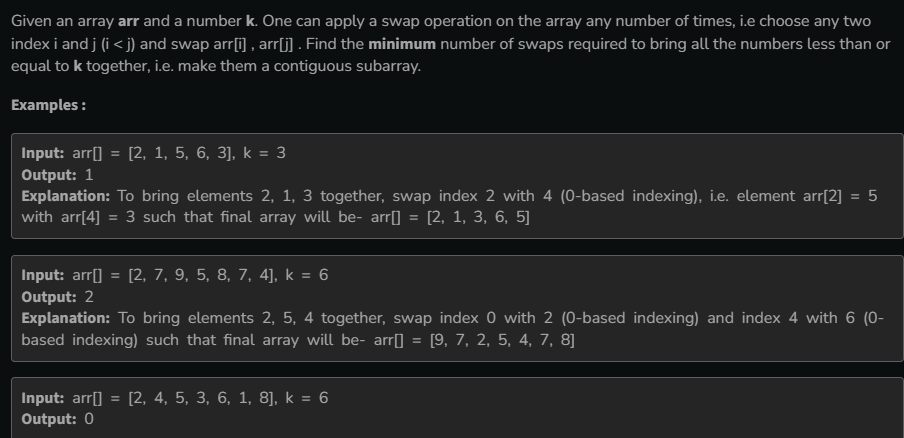


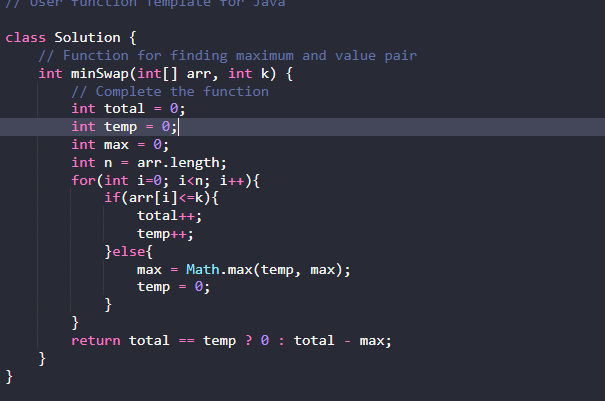
30.Three way partitioning



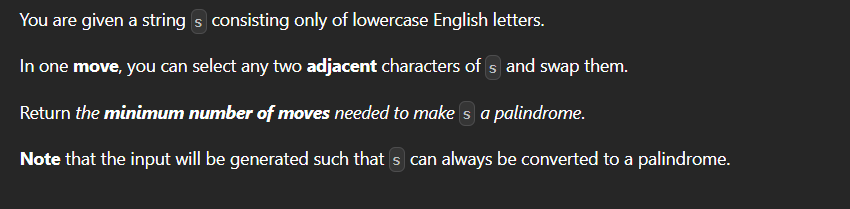


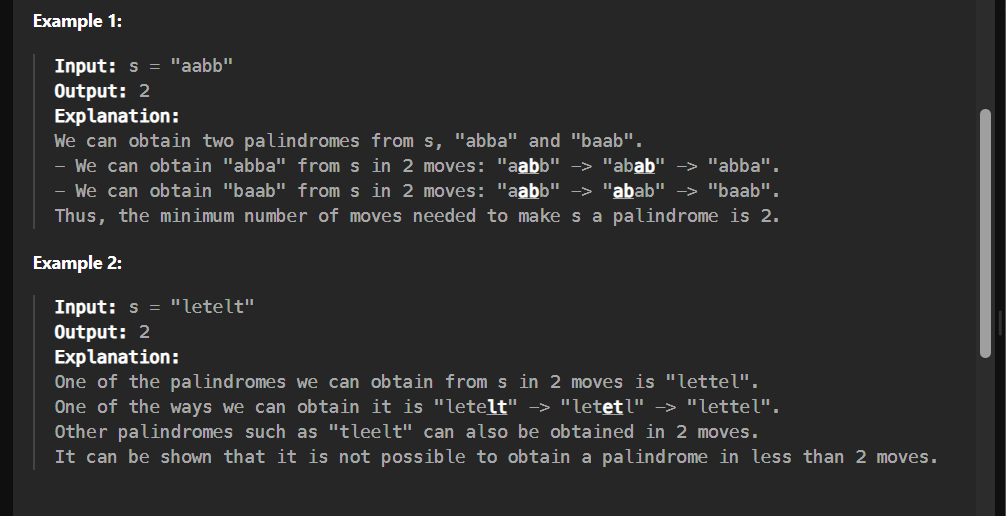
31.Minimum swaps and k Together

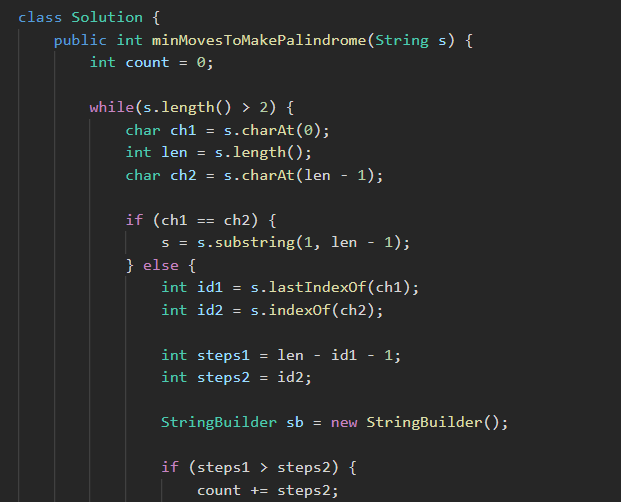


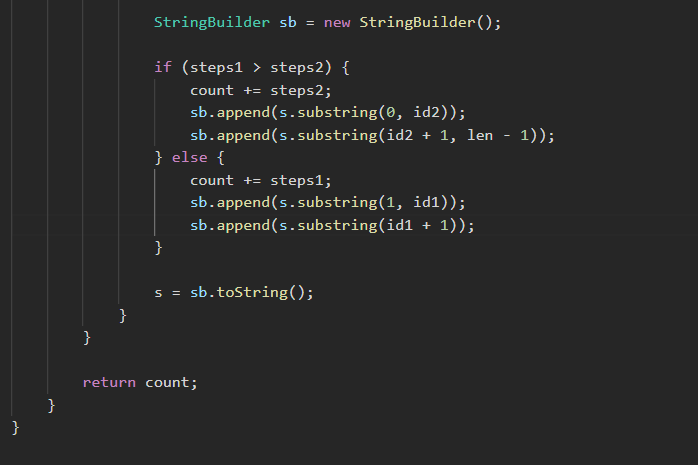


32.Minimum Number of moves to make Palindrome









33.Median of 2 sorted Array of same size

