**Name:\_\_\_\_\_\_\_\_\_\_\_\_Shuqing Ye\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_UCI NET ID:\_\_\_\_\_\_\_shuqiny2\_\_\_\_\_\_\_\_**

|  |  |
| --- | --- |
| Test cases (including the edge cases):  input: [0], output: 0  input: [1, -2, -3], output: 1  input: [-1, -2, 3], output: 3  input: [1,20,-100,40,25,-80,60] output: 65 | time complexity: O(n) / O(nlog n)  space complexity: O(1) / O(log n) |

|  |  |
| --- | --- |
| // there is at least one positive integer in nums[]  // so res > 0  int max\_subArray(int[] nums) {  int res = Integer.MIN\_VALUE;  int sum = 0;  for (int num: nums) {  sum += num;  sum = Math.max(sum, 0);  res = Math.max(res, sum);  }  return res;  } | int max\_subArray(int[] nums) {  return getSubMax(nums, 0, nums.length - 1);  }  int getSubMax(int[] nums, int left, int right) {  if (left < right) {  int mid = (left + right) / 2;  int leftMax = getSubMax(nums, left, mid);  int rightMax = getSubMax(nums, mid + 1, right);  int crossMax = getCrossMax(nums, left, mid, right);  return Math.max(Math.max(leftMax, rightMax), crossMax);  }  return 0;  }  int getCrossMax(int[] nums, int left, int mid, int right) {  int sum = 0;  int leftMax = 0;  for (int i = mid; i >= left; i--) {  sum += nums[i];  leftMax = Math.max(leftMax, sum);  }    int rightMax = 0;  for(int i = mid + 1, sum = 0; i<= right; i++) {  sum += nums[i];  rightMax = Math.max(rightMax, sum);  }  int crossMax = 0;  for (int i = left, sum = 0; i <= right; i++) {  sum += nums[i];  crossMax = Math.max(crossMax, sum);  }  return Math.max(Math.max(leftMax, rightMax), crossMax);  } |