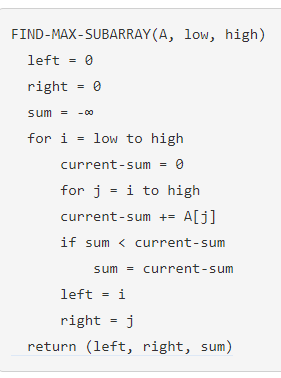


4.1-1，It will return a single-element array with the largest negative integer.

4.1-2，



4.1-4，

The base case needs to return an empty array if A[low] is negative.

The subroutine needs to return an empty array in case the maximum subarray has a negative sum.

#include **<limits.h>**

**typedef** **struct** {

**unsigned** left;

**unsigned** right;

**int** sum;

} max\_subarray;

max\_subarray find\_max\_crossing\_subarray(**int** A[], **unsigned** low, **unsigned** mid, **unsigned** high) {

max\_subarray result = {mid + 1, mid, 0};

**int** sum = 0,

left\_sum = INT\_MIN,

right\_sum = INT\_MIN;

**for** (**int** i = mid - 1; i >= (**int**) low; i--) {

sum += A[i];

**if** (sum > left\_sum) {

left\_sum = sum;

result.left = i;

}

}

sum = 0;

**for** (**int** j = mid; j < high; j++) {

sum += A[j];

**if** (sum > right\_sum) {

right\_sum = sum;

result.right = j + 1;

}

}

**if** (left\_sum + right\_sum < 0) {

max\_subarray empty = { mid, mid, 0 };

**return** empty;

} **else** {

result.sum = left\_sum + right\_sum;

**return** result;

}

}

max\_subarray find\_maximum\_subarray(**int** A[], **unsigned** low, **unsigned** high) {

**if** (high == low + 1) {

**if** (A[low] < 0) {

max\_subarray empty = {low, low, 0};

**return** empty;

} **else** {

max\_subarray result = {low, high, A[low]};

**return** result;

}

} **else** {

**unsigned** mid = (low + high) / 2;

max\_subarray left = find\_maximum\_subarray(A, low, mid);

max\_subarray right = find\_maximum\_subarray(A, mid, high);

max\_subarray cross = find\_max\_crossing\_subarray(A, low, mid, high);

**if** (left.sum >= right.sum && left.sum >= cross.sum) {

**return** left;

} **else** **if** (right.sum >= left.sum && right.sum >= cross.sum) {

**return** right;

} **else** {

**return** cross;

}

}

}