Algorithms

```
BUBBLESORT(A)
for i = 1 to A.length − 1
    for j = A.length down to i + 1
        if A[j] < A[j − 1]
            exchange A[j] with A[j − 1]
```

```
INSERTION-SORT(A)
for j = 2 to A.length
    key = A[j]
    i = j − 1
    while i > 0 and A[i] > key
        A[i + 1] = A[i]
        i = i − 1
    A[i + 1] = key
```

```
MERGE(A, p, q, r)
n1 = q − p + 1
n2 = r − q
let L[1 ··· n1 + 1] and
R = [1 ··· n2 + 1] be new arrays
for i = 1 to n1
    L[i] = A[p + i − 1]
for j = 1 to n2
    R[j] = A[q + j]
L[n1 + 1] = ∞
R[n2 + 1] = ∞
i = 1
j = 1
for k = p to r
    if L[i] ≤ R[j]
        A[k] = L[i]
        i = i + 1
    else A[k] = R[j]
        j = j + 1
```

```
MERGE-SORT(A, p, r)
if p < r
    q = (p + r)/2
    MERGE-SORT(A, p, q)
    MERGE-SORT(A, q + 1, r)
    MERGE(A, p, q, r)
```

```
MAX-HEAPIFY(A, i)
l = left(i)
r = right(i)
if l ≤ A.heap-size and A[l] > A[i]
    largest = l
else largest = i
if r ≤ A.heap-size and A[r] > A[largest]
    largest = r
if largest ≠ i
    exchange A[i] with A[largest]
    MAX-HEAPIFY(A, largest)
```

```
BUILD-MAX-HEAPIFY(A)
A.heap-size = A.length
for i = ⌊A.length/2⌋ down to 1
    MAX-HEAPIFY(A, i)
```

```
HEAPSORT(A)
BUILD-MAX-HEAP(A)
for i = A.length down to 2
    exchange A[1] with A[i]
    A.heap-size = A.heap-size − 1
    MAX-MEAPIFY(A, 1)
```

```
QUICKSORT(A, p, r)
if p < r
    q = PARTITION(A, q, r)
    QUICKSORT(A, q, q − 1)
    QUICKSORT(A, q + 1, r)
```

```
PARTITION(A, q, r)
x = A[r]
i = p − 1
for j = p to r − 1
    if A[j] ≤ x
        i = i + 1
        exchange A[i] with A[j]
exchange A[i + 1] with A[r]
return i + 1
```

```
FIND-MAX-CROSSING-SUBARRAY(A, low, mid, high)
left-sum = -∞
sum = 0
for i = mid down to low
    sum = sum + A[i]
    if sum > left-sum
        left-sum = sum
            max - left = i
right-sum = -∞
sum = 0
for j = mid + 1 to high
    sum = sum + A[j]
    if sum > right-sum
        right-sum = sum
        max-right = j
return(max-left, max-right, left-sum + right-sum)
```

```
FIND-MAXIMUM-SUBARRAY(A, low, high)
if high == low
    return (low, high, A[low]) // base case: only one element
else mid = ⌊(low + high)/2⌋
    (left-low, left-high, left-sum) =
    FIND-MAXIMUM-SUBARRAY(A, low, mid)
(right-low, right-high, right-sum) =
    FIND-MAXIMUM-SUBARRAY(A, mid + 1, high)
(cross-low, cross-high; cross-sum) =
    FIND-MAX-CROSSING-SUBARRAY(A, low, mid, high)
if left-sum ≥ right-sum and left-sum ≥ cross - sum
    return (left-low, left-high, left-sum
elseif right-sum ≥ left-sum and right-sum ≥ cross-sum
    return (right-low, right-high, right-sum)
else return (cross-low, cross-high, cross-sum)
```