Project 1 – Algorithm 4 Pseudocode

```
MAX-SUBARRAY-LINEAR(A)
  n = A.length
  max-sum = −∞
  ending-here-sum = −∞
  for j = 1 to n
      ending-here-high = j
      if ending-here-sum > 0
          ending-here-sum = ending-here-sum + A[j]
      else ending-here-low = j
          ending-here-sum = A[j]
      if ending-here-sum > max-sum
          max-sum = ending-here-sum
          low = ending-here-low
          high = ending-here-high
  return (low, high, max-sum)
```

The variables are intended as follows:

- *low* and *high* demarcate a maximum subarray found so far.
- *max-sum* gives the sum of the values in a maximum subarray found so far.
- *ending-here-low* and *ending-here-high* demarcate a maximum subarray ending at index j. Since the high end of any subarray ending at index j must be j, every iteration of the **for** loop automatically sets *ending-here-high* $= j$.
- *ending-here-sum* gives the sum of the values in a maximum subarray ending at index j.

The first test within the **for** loop determines whether a maximum subarray ending at index j contains just $A[j]$. As we enter an iteration of the loop, *ending-here-sum* has the sum of the values in a maximum subarray ending at $j-1$. If *ending-here-sum* $+ A[j] > A[j]$, then we extend the maximum subarray ending at index $j-1$ to include index j. (The test in the **if** statement just subtracts out $A[j]$ from both sides.) Otherwise, we start a new subarray at index j, so both its low and high ends have the value j and its sum is $A[j]$. Once we know the maximum subarray ending at index j, we test to see whether it has a greater sum than the maximum subarray found so far, ending at any position less than or equal to j. If it does, then we update *low*, *high*, and *max-sum* appropriately.