Section 4.1 – The maximum-subarray problem

4.1-1 What does FIND-MAXIMUM-SUBARRAY return when all elements of A are negative?

A subarray with only the largest negative element of A.

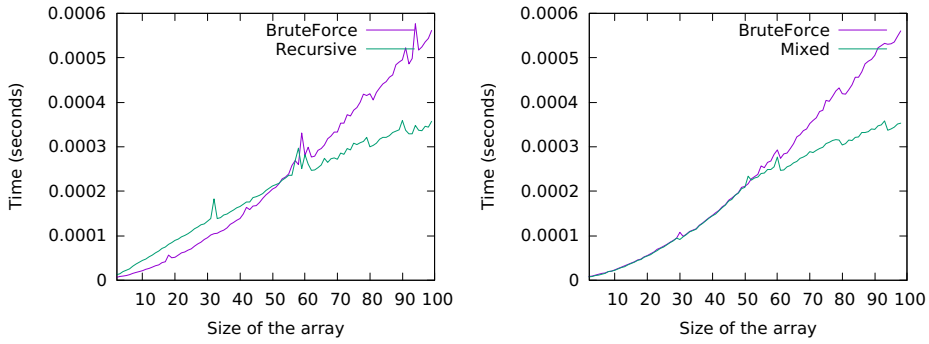
4.1-2 Write pseudocode for the brute-force method of solving the maximum-subarray problem. Your procedure should run in $\Theta(n^2)$ time.

The pseudocode is stated below.

```
   FindMaximumSubarray-BruteForce(A)
 1 | low = 0
 2 | high = 0
 3 | sum = −∞
 4 | for i = 1 to A.length do
 5 | |   cursum = 0
 6 | |   for j = i to A.length do
 7 | |   |   cursum = cursum + A[j]
 8 | |   |   if cursum > sum then
 9 | |   |   |   sum = cursum
10 | |   |   |   low = i
11 | |   |   |   high = j
12 | return low, high, sum
```

4.1-3 Implement both the brute-force and recursive algorithms for the maximum-subarray problem on your own computer. What problem size n_0 gives the crossover point at which the recursive algorithm beats the brute-force algorithm? Then, change the base case of the recursive algorithm to use the brute-force algorithm whenever the problem size is less than n_0. Does that change the crossover point?

Figure below in the lhs ilustrates the crossover point between the BruteForce and Recursive solutions in my machine. In that comparison, $n_0 \approx 52$. Figure below in the rhs ilustrates the crossover point between the BruteForce and Mixed solutions in my machine. The crossover point does not change but the Mixed solution becomes as fast as the BruteForce solution when the problem size is lower than 52.



4.1-4 Suppose we change the definition of the maximum-subarray problem to allow the result to be an empty subarray, where the sum of the values of an empty subarray is 0. How would you change any of the algorithms that do not allow empty subarrays to permit an empty subarray to be the result?

The BruteForce algorithm (stated above in Question 4.1-2) can be updated just by modifying line 3 to $sum = 0$, instead of $sum = -\infty$. In that case, if there is no subarray whose sum is greater than zero, the algorithm will return a invalid subarray ($low = 0, high = 0, sum = 0$), which will denote the empty subarray.

The Recursive algorithm (stated in Section 4.1) can be updated as follows. In the FIND-MAX-CROSSING-SUBARRAY routine, update lines 1 and 8 to initialize $lefturrent\text{-}sum$ and $righturrent\text{-}sum$ to 0, instead of $-\infty$. Also initialize $maxurrent\text{-}left$ (after line 1) and $max\text{-}right$ (after line 8) to 0. In the FIND-MAXIMUM-SUBARRAY routine, surround the return statement of line 2 with a conditional that verifies if $A[low]$ is greater than zero. If it is, return the values as it was before. If it is not, return a invalid subarray (denoted by $low = 0$ and $high = 0$) and the sum as zero.

4.1-5 Use the following ideas to develop a nonrecursive, linear-time algorithm for the maximum-subarray problem. Start at the left end of the array, and progress toward the right, keeping track of the maximum subarray seen so far. Knowing a maximum subarray of $A[1, \ldots, j]$, extend the answer to find a maximum subarray ending at index $j+1$ by using the following observation: a maximum subarray of $A[1, \ldots, j+1]$ is either a maximum subarray of $A[1, \ldots, j]$ or a subarray $A[i, \ldots, j+1]$, for some $1 \leq i \leq j+1$. Determine a maximum subarray of the form $A[i, \ldots, j+1]$ in constant time based on knowing a maximum subarray ending at index j.

The pseudocode is stated below.

```
FindMaximumSubarray-Linear(A)
1  | low = 0
2  | high = 0
3  | sum = 0
4  | current-low = 0
5  | current-sum = 0
6  | for i = 1 to A.length do
7  | |  current-sum = max(A[i], c-sum + A[i])
8  | |  if current-sum == A[i] then
9  | | |  current-low = i
10 | |  if current-sum > sum then
11 | | |  low = current-low
12 | | |  high = i
13 | | |  sum = current-sum
14 | return low, high, sum
```