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.

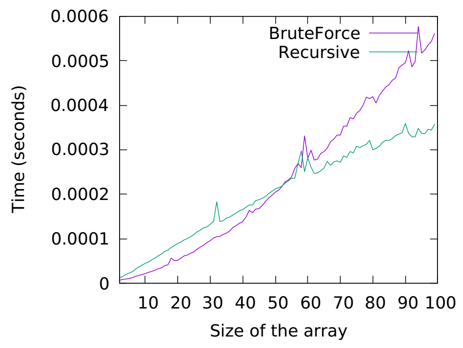
4.1-2 Write pseudocode for the brute-force method of solving the maximum-subarray problem. Your procedure should run in $\Theta(n^2)$ time.

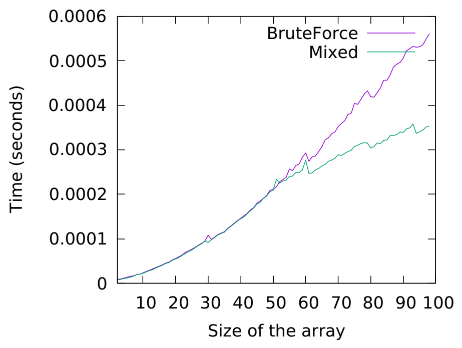
The pseudocode is stated below.

```
   FindMaximumSubarray-BruteForce(A)
1  $low = 0$
2  $high = 0$
3  $sum = -\infty$
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 *left-sum* and *right-sum* to 0, instead of $-\infty$. Also initialize *max-left* (after line 1) and *max-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($A$)
    $low = 0$
    $high = 0$
    $sum = 0$
    $current\text{-}low = 0$
    $current\text{-}sum = 0$
    for $i = 1$ to $A.length$ do
        $current\text{-}sum = \max(A[i], current\text{-}sum + A[i])$
        if $current\text{-}sum == A[i]$ then
            $current\text{-}low = i$
        if $current\text{-}sum > sum$ then
            $low = current\text{-}low$
            $high = i$
            $sum = current\text{-}sum$
    return $low, high, sum$
```

Section 4.2 – Strassen's algorithm for matrix multiplication

4.2-1 Use Strassen's algorithm to compute the matrix product

$$\begin{bmatrix}1 & 3\\7 & 5\end{bmatrix}\begin{bmatrix}6 & 8\\4 & 2\end{bmatrix}.$$

Show your work.

Let

$$A = \begin{bmatrix}1 & 3\\7 & 5\end{bmatrix}, B = \begin{bmatrix}6 & 8\\4 & 2\end{bmatrix},$$

and $C = A \cdot B$. To compute C using Strassen's algorithm, we start by computing the S_i matrices:

$$\begin{aligned}
S_1 &= B_{12} - B_{22} &&= 8 - 2 &&= 6,\\
S_2 &= A_{11} + A_{12} &&= 1 + 3 &&= 4,\\
S_3 &= A_{21} + A_{22} &&= 7 + 5 &&= 12,\\
S_4 &= B_{21} - B_{11} &&= 4 - 6 &&= -2,\\
S_5 &= A_{11} + A_{22} &&= 1 + 5 &&= 6,\\
S_6 &= B_{11} + B_{22} &&= 6 + 2 &&= 8,\\
S_7 &= A_{12} + A_{22} &&= 3 - 5 &&= -2,\\
S_8 &= B_{21} + B_{22} &&= 4 + 2 &&= 6,\\
S_9 &= A_{11} - A_{21} &&= 1 - 7 &&= -6,\\
S_{10} &= B_{11} + B_{12} &&= 6 + 8 &&= 14.
\end{aligned}$$

Then we compute the P_i matrices:

$$\begin{aligned}
P_1 &= A_11 \cdot S_1 &&= 1 \cdot 6 &&= 6,\\
P_2 &= S_2 \cdot B_22 &&= 4 \cdot 2 &&= 8,\\
P_3 &= S_3 \cdot B_11 &&= 12 \cdot 6 &&= 72,\\
P_4 &= A_22 \cdot S_4 &&= 5 \cdot (-2) &&= -10,\\
P_5 &= S_5 \cdot S_6 &&= 6 \cdot 8 &&= 48,\\
P_6 &= S_7 \cdot S_8 &&= (-2) \cdot 6 &&= -12,\\
P_7 &= S_9 \cdot S_10 &&= (-6) \cdot 14 &&= -84.
\end{aligned}$$

Using matrices S_i and P_i, we compute C:

$$C = \begin{bmatrix}(P_5 + P_4 - P_2 + P_6) & (P_2 + P_2)\\ (P_3 + P_4) & (P_5 + P_1 - P_3 - P_7)\end{bmatrix} = \begin{bmatrix}18 & 14\\62 & 66\end{bmatrix}.$$

4.2-2 Write pseudocode for Strassen's algorithm.

The pseudocode is stated below.

```
Square-Matrix-Multiply-Strassen(A, B)
    $n = A.rows$
    let $C$ be a new $n \times n$ matrix
    if $n == 1$ then
        $c_{11} = a_{11} \cdot b_{11}$
    else
        partition $A, B$, and $C$ as into $n/2 \times n/2$ submatrices
        let $S_1, S_2, \ldots, S_{10}$ be new $n/2 \times n/2$ matrices
        let $P_1, P_2, \ldots, P_7$ be new $n/2 \times n/2$ matrices
        $S_1 = B_{12} - B_{22}$
        $S_2 = A_{11} + A_{12}$
        $S_3 = A_{21} + A_{22}$
        $S_4 = B_{21} - B_{11}$
        $S_5 = A_{11} + A_{22}$
        $S_6 = B_{11} + B_{22}$
        $S_7 = A_{12} - A_{22}$
        $S_8 = B_{21} + B_{22}$
        $S_9 = A_{11} - A_{21}$
        $S_{10} = B_{11} - B_{12}$
        $P_1 =$ Square-Matrix-Multiply-Strassen($A_{11}, S_1$)
        $P_2 =$ Square-Matrix-Multiply-Strassen($S_2, B_{22}$)
        $P_3 =$ Square-Matrix-Multiply-Strassen($S_3, B_{11}$)
        $P_4 =$ Square-Matrix-Multiply-Strassen($A_{22}, S_4$)
        $P_5 =$ Square-Matrix-Multiply-Strassen($S_5, S_6$)
        $P_6 =$ Square-Matrix-Multiply-Strassen($S_7, S_8$)
        $P_7 =$ Square-Matrix-Multiply-Strassen($S_9, S_{10}$)
        $C_{11} = P_5 + P_4 - P_2 + P_6$
        $C_{12} = P_1 + P_2$
        $C_{21} = P_3 + P_4$
        $C_{22} = P_5 + P_1 - P_3 - P_7$
    return $C$
```

4.2-3 How would you modify Strassen's algorithm to multiply $n \times n$ matrices in which n is not an exact power of 2? Show that the resulting algorithm runs in time $\Theta(n^{\lg 7})$.

Pad each input $n \times n$ matrix (rows and columns) with $m - n$ zeros, resulting in an $m \times m$ matrix, where $m = 2^{\lceil \lg n \rceil}$. After computing the final matrix, cut down the last $m - n$ rows and $m - n$ columns (which will be zeros).
Padding the matrix with zeros is done once, in the root of the recursion tree, and takes $O(m^2)$. Since we now have an $m \times m$ matrix, the algorithm runs in $\Theta(m^{\lg 7}) + O(m^2) = \Theta(m^{\lg 7})$. We have that $n \leq m < 2^{(\lg n)+1} = 2^{\lg n} \cdot 2 = 2n$. Thus, the algorithm runs in $\Theta((2n)^{\lg 7}) = \Theta(n^{\lg 7})$.

4.2-4 What is the largest k such that if you can multiply 3×3 matrices using k multiplications (not assuming commutativity of multiplication), then you can multiply $n \times n$ matrices in time $o(n^{\lg 7})$? What would the running time of this algorithm be?

If we modify the Square-Matrix-Multiply-Recursive algorithm to partition the matrices into $n/3 \times n/3$ submatrices, we would have the following recurrence:

$$T(n) = \Theta(1) + 27T(n/3) + \Theta(n^2) = 27T(n/3) + \Theta(n^2).$$

Let's proceed to understand a little more about the above recurrence. Let A and B be the two input matrices in each node of the above recursion tree. Like in the original Square-Matrix-Multiply-Recursive algorithm, our modified version will take $\Theta(1)$ to partition A and B into $n/3 \times n/3$ submatrices. In each node of the tree, the product of A and B is recursively computed by the products of their submatrices. Since the number of recursive (submatrices) products to compute $A \cdot B$ in each node of the recurstion tree is 27 and each of these submatrices is 3 times smaller than A and B, the 27 recursive products takes $27T(n/3)$. Finally, the number of summations to compute the final matrix is $\Theta(3 \cdot 9 \cdot n^2/3) = \Theta(n^2)$.

If after partitioning A and B into $n/3 \times n/3$ submatrices we can compute their product with k multiplications (instead of 27), we would have the following recurrence:

$$T(n) = \Theta(1) + kT(n/3) + \Theta(n^2) = kT(n/3) + \Theta(n^2),$$

We can solve the above recurrence using the master method. We have $f(n) = n^2$ and $n^{log_b a} = n^{\log_3 k}$. Using the first case of the master method, we have

$$\forall\, k \mid \log_3 k > 2,\ n^2 = O(n^{(\log_3 k)-\epsilon}),\ 0 \le \epsilon \le \log_3 k - 2,$$

which implies

$$T(n) = \Theta(n^{\log_3 k}).$$

Since $\log_3 21 < \lg 7 < \log_3 22$, the largest value for k is 21. Its running time would be $n^{\log_3 21} \approx n^{2.7712}$.

4.2-5 V. Pan has discovered a way of multiplying 68×68 matrices using 132,464 multiplications, a way of multiplying 70×70 matrices using 143,640 multiplications, and a way of multiplying 72×72 matrices using 155,424 multiplications. Which method yields the best asymptotic running time when used in a divide-and-conquer matrix-multiplication algorithm? How does it compare to Strassen's algorithm?

The algorithms would take:

- $n^{\log_{68} 132{,}464} \approx n^{2.795128}$,
- $n^{\log_{70} 143{,}640} \approx n^{2.795122}$,
- $n^{\log_{72} 155{,}424} \approx n^{2.795147}$.

The fastest is the one that multiplies 70×70 matrices, but all of them are faster then the Strassen's algorithm.

4.2-6 How quickly can you multiply a $kn \times n$ matrix by an $n \times kn$ matrix, using Strassen's algorithm as a subroutine? Answer the same question with the order of the input matrices reversed.

Let A and B be $kn \times n$ and $n \times kn$ matrices, respectivelly. We can compute $A \cdot B$ as follows:

(a) Partition A and B into k submatrices $A_1, \ldots, A_k$ and $B_1, \ldots, B_k$, each one of size $n \times n$.

(b) Compute the desired submatrices C_{ij} of the result matrix C by the product of $A_i \cdot B_j$. Use the Strassen's algorithm to compute each of those products.

Since each of the k^2 products takes $\Theta(n^{\lg 7})$, this algorithm runs in $\Theta(k^2 n^{\lg 7})$.
We can compute $B \cdot A$ as follows:

(a) Partition A and B into k submatrices $A_1, \ldots, A_k$ and $B_1, \ldots, B_k$, each one of size $n \times n$.

(b) Compute the the result matrix $C = \sum_{i=1}^{k} A_i \cdot B_i$. Use the Strassen's algorithm to compute each of those products.

Since each of the k products takes $\Theta(n^{\lg 7})$ and the $k-1$ summations takes $\Theta((k-1)n^2/k) = O(n^2)$, this algorithm runs in $\Theta(kn^{\lg 7}) + O(n^2) = \Theta(kn^{\lg 7})$.

4.2-7 Show how to multiply the complex numbers $a + bi$ and $c + di$ using only three multiplications of real numbers. The algorithm should take a, b, c, and d as input and produce the real component $ac - bd$ and the imaginary component $ad + bc$ separately.

The pseudocode is stated below.

```
Complex-Product(a, b, c, d)
    $x = a \cdot c$
    $y = b \cdot d$
    $real\text{-}part = x - y$
    $imaginary\text{-}part = (a + b) \cdot (c + d) - x - y$
    return $real\text{-}part, imaginary\text{-}part$
```