Chapter 4

November 14, 2022

1 Exercise 4.1

1.1 Question 4.1-1

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

> As all elements are negative, the maximum subarray will contain only a single element, the largest element of A

1.2 Question 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.

Algorithm 1 the brute-force method for maximum-subarray problem

```
procedure MAX SUBARRAY(A, low, high)
    left = low
    right = high
    sum = −∞
    for i = left to high do
        tempsum = 0
        for j = i to high do
            tempsum = tempsum + A[i]
            if tempsum > sum then
                sum = tempsum
                left = i
                right = j
            end if
        end for
    end for
    return(left, right, sum)
end procedure
```

1.3 Question 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?

```
import time
import math
import random

def bruteforce(a,low,high):
    left=low
    right= high
    sum = -math.inf
    for i in range(left,high):
        tempsum = 0
        for j in range(i,high):
            tempsum += a[i]
            if tempsum > sum:
                sum=tempsum
                left = i
                right = j
    return(left,right,sum)

def FindMaxCrossingSubarray(A, low, mid, high):
    left = low
    right = high
    leftSum = -math.inf
    rightSum = -math.inf
    sum = 0
    for i in reversed(range(low, mid)):
        sum += A[i]
        if sum > leftSum:
            leftSum = sum
            left = i

    sum = 0
    for j in range(mid, high):
        sum += A[j]
        if sum > rightSum:
            rightSum = sum
            right = j

    return (left, right, leftSum + rightSum)
```

```
def FindMaxSubarrayRecursive(A, low, high):
    if low + 1 == high:
        return (low, low, A[low])

    mid = (low + high) // 2
    left = FindMaxSubarrayRecursive(A, low, mid)
    right = FindMaxSubarrayRecursive(A, mid, high)
    cross = FindMaxCrossingSubarray(A, low, mid, high)
    if left[2] >= right[2] and left[2] >= cross[2]:
        return left
    elif right[2] >= left[2] and right[2] >= cross[2]:
        return right
    else:
        return cross

Ni =10000
for input_size in range(2, 10):
    arr = [random.randint(-100,100)
           for _ in range(input_size)]

    # Run brute force method and measure time
    start = time.time()
    for _ in range(Ni):
        bf = bruteforce(arr, 0, len(arr))
    bf_time = time.time() - start

    # Run recursive method and measure time
    start = time.time()
    for _ in range(Ni):
        rc = FindMaxSubarrayRecursive(arr, 0, len(arr))
    rc_time = time.time() - start

    if bf_time > rc_time:
        print(f"Cross over point = {input_size}")
        break
```

1.4 Question 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?

We would want to return an empty subarray only if the sum of the maximum subarray is negative. So, we can modify the algorithms to check if the sum it is going to return is negative, we will return the empty subarray instead.

1.5 Question 4.1-5

Use the following ideas to develop a non-recursive, 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...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...j+1] is either a maximum subarray of A[1...j] or a subarray A[i...j+1], for some $1 \leq i \leq j+1$ Determine a maximum subarray of the form A[i...j+1] in constant time based on knowing a maximum subarray ending at index j.

Algorithm 2 Find maximum-subarray problem using linear approach

```
procedure MAX SUBARRAY LINEAR(A, low, high)
    left = 0
    right = 0
    best = −∞
    current = 0
    currentLeft = 0
    for i = left to high do
        current = current + A[i]
        if current > best then
            best = current
            left= currentLeft
            right = i
        end if
        if current < 0 then
            current =0
            currentLeft = i + 1
        end if
    end for
    return(left, right, best)
end procedure
```

2 Exercise 4.2

2.1 Question 4.1-1

Use Strassen's algorithm to compute the matrix product $\begin{pmatrix} 1 & 3 \\ 7 & 5 \end{pmatrix} \begin{pmatrix} 6 & 8 \\ 4 & 2 \end{pmatrix}$ Show your work.

- Sums:

$$S_1 = B_{12} - B_{22} = 8 - 2 = 6$$
$$S_2 = A_{11} + A_{12} = 1 + 3 = 4$$
$$S_3 = A_{21} + A_{22} = 7 + 5 = 12$$
$$S_4 = B_{21} - B_{11} = 4 - 6 = -2$$
$$S_5 = A_{11} + A_{22} = 1 + 5 = 6$$
$$S_6 = B_{11} + B_{22} = 6 + 2 = 8$$
$$S_7 = A_{12} - A_{22} = 3 - 5 = -2$$
$$S_8 = B_{21} + B_{22} = 4 + 2 = 6$$
$$S_9 = A_{11} - A_{21} = 1 - 7 = -6$$
$$S_10 = B_{11} + B_{12} = 6 + 8 = 14$$

- Products

$$P_1 = A_{11} * S_1 = 1 * 6 = 6$$
$$P_2 = S_2 * B_{22} = 4 * 2 = 8$$
$$P_3 = S_3 * B_{11} = 12 * 6 = 72$$
$$P_4 = A_{22} * S_4 = 5 * -2 = -10$$
$$P_5 = S_5 * S_6 = 6 * 8 = 48$$
$$P_6 = S_7 * S_8 = -2 * 6 = -12$$
$$P_7 = S_9 * S_10 = -6 * 14 = -84$$

- The result sub-matrix sums:

$$C_{11} = P_5 + P_4 - P_2 + P_6 = 48 - 10 - 8 - 12 = 18$$

$$C_{12} = P_1 + P_2 = 6 + 8 = 14$$

$$C_{21} = P_3 + P_4 = 72 - 10 = 62$$

$$C_{22} = P_5 + P_1 - P_3 - P_7 = 48 + 6 - 72 + 84 = 66$$

- the finial result:

$$\begin{pmatrix} 18 & 14 \\ 62 & 66 \end{pmatrix}$$