COMP 301 Analysis of Algorithms
Instructor: Zafer Aydın
Lab Assignment 4

Introduction

In this lab you will implement a divide and conquer algorithm for the maximum subarray problem. Submit your answers to the questions below in a text file (e.g. Word document). Name your file in name_surname.docx format. Submit your solution document and Java codes as a zip/rar folder in name_surname format to Canvas.

You can use the code templates in `max_subarray.java` in this lab.

Problem Statement

Given an input array A of n numbers find indices i and j such that $1 \leq i < j \leq n$ and $A[j] - A[i]$ is maximum.

Assignment

1. (a) Implement a method called `brute_force` for solving the maximum subarray problem using brute-force approach. Your method should receive an array A of n integers as input and it should report (i.e. print on screen) two indices i, j such that $1 \leq i < j \leq n$ and $A[j] - A[i]$ is maximum. Your method should also print the maximum possible $A[j] - A[i]$ difference. Note that this notation assumes that the array indices start from 1. In this brute-force approach, you should consider all pairs of i, j. The number of such pairs is equal to $\binom{n}{2} = \frac{n(n-1)}{2}$. Therefore your procedure should run in $\Theta(n^2)$ time.

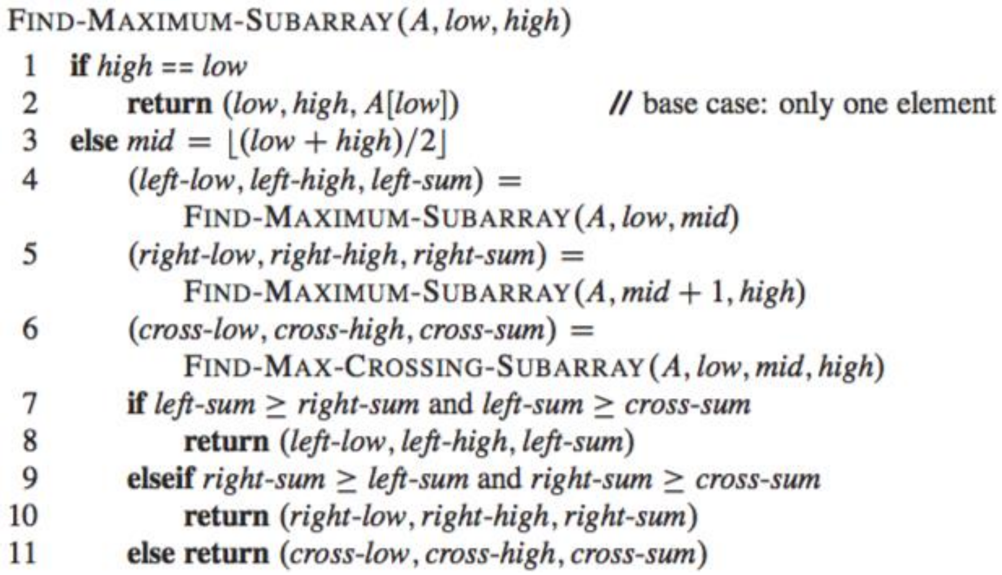
(b) Choose the following array *test_A* as input to the method you implemented in part (a) and verify that your method prints $i = 3, j = 4$, and maximum difference as 3.

$$test\_A = [10, 11, 7, 10, 6]$$

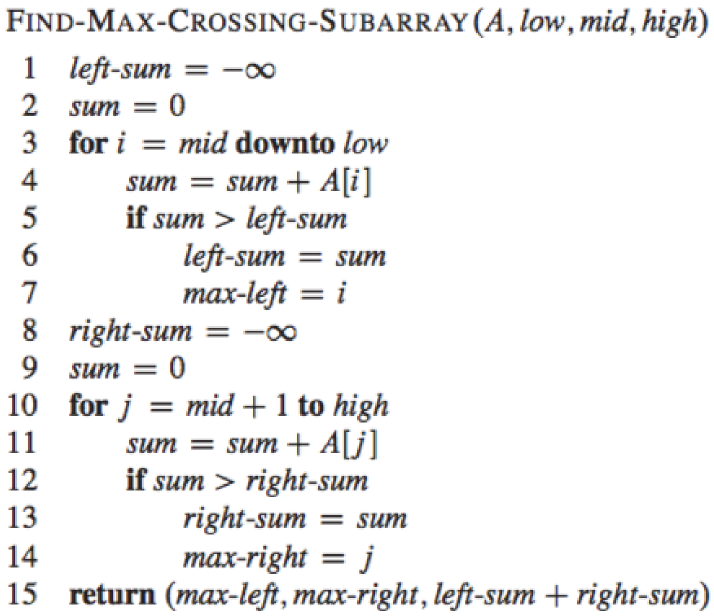
(c) Choose an integer array A of size 65536. Initialize your array by random numbers from 0 to 99. Call the method you implemented in part (a) and find the maximum subarray for this input array. Compute the time it takes to find the maximum subarray in nanoseconds and include this time into your report.

2. (a) Implement the divide and conquer algorithm given below for finding the maximum subarray using recursion. You can use the code templates in `max_subarray.java`. The

outputs of `find_max_crossing_subarray` method can be retrived by the input array named `outputs` (since this method returns multiple outputs).



```
FIND-MAXIMUM-SUBARRAY(A, low, high)
if high == low
    return (low, high, A[low])          // base case: only one element
else mid = $\lfloor (low + high)/2 \rfloor$
    (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 $\geq$ right-sum and left-sum $\geq$ cross-sum
        return (left-low, left-high, left-sum)
    elseif right-sum $\geq$ left-sum and right-sum $\geq$ cross-sum
        return (right-low, right-high, right-sum)
    else return (cross-low, cross-high, cross-sum)
```



```
FIND-MAX-CROSSING-SUBARRAY(A, low, mid, high)
left-sum = $-\infty$
sum = 0
for i = mid downto low
    sum = sum + A[i]
    if sum > left-sum
        left-sum = sum
        max-left = i
right-sum = $-\infty$
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)
```

where A is an array of numbers corresponding to a difference array and the array indices start from 1 in these pseudo-codes.

(b) Test your method using the array $diff\_test\_A$ given below (which is the difference array for $test\_A$ of part 1(b)) and verify that your method prints 3, 3, and 3 as the outputs (the first 2 are left and right indices and the last output is the maximum difference).

$$diff\_test\_A = [1, -4, 3, -4]$$

(c) Start with the same input array A as in question 1 (c). Convert this array to difference array called $diff\_A$. A code template is already given in the `main` method of `max_subarray.java` for this purpose. Find the maximum subarray of A using the difference array called $diff\_A$ as input to the divide and conquer algorithm given above. Report the left and right indices generated by `find_maximum_subarray` method as well as the maximum difference. Then report the i, j indices for the original input array A at which $A[j] - A[i]$ is maximum. Note that to find i you should subtract 1 from the lower index returned by the `find_maximum_subarray` method (i.e. subtract 1 from the first output of this method). The second index returned by `find_maximum_subarray` method can be directly reported as index j. Also report the maximum difference returned by `find_maximum_subarray` method, which should be equal to the maximum $A[j] - A[i]$ difference. Compute the time it takes to find the maximum subarray in nanoseconds and include this time into your report. Do you get better running time as compared to the brute-force approach?

3. (Bonus) What problem size n_0 gives the cross-over point at which the recursive algorithm beats the brute-force algorithm?