COMP 301 Analysis of Algorithms, Fall 2022
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 folder in name_surname format to TA through a USB flash disk.

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 report the three outputs (two indices i, j and the maximum possible $A[j] - A[i]$ difference) for this input array. Compute the time it takes to find the result in nanoseconds and include this time into your report.

COMP 301 Analysis of Algorithms, Fall 2022
Instructor: Zafer Aydın
Lab Assignment 4

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)
 1  if high == low
 2       return (low, high, A[low])          // base case: only one element
 3  else mid = ⌊(low + high)/2⌋
 4       (left-low, left-high, left-sum) =
              FIND-MAXIMUM-SUBARRAY(A, low, mid)
 5       (right-low, right-high, right-sum) =
              FIND-MAXIMUM-SUBARRAY(A, mid + 1, high)
 6       (cross-low, cross-high, cross-sum) =
              FIND-MAX-CROSSING-SUBARRAY(A, low, mid, high)
 7       if left-sum ≥ right-sum and left-sum ≥ cross-sum
 8            return (left-low, left-high, left-sum)
 9       elseif right-sum ≥ left-sum and right-sum ≥ cross-sum
10            return (right-low, right-high, right-sum)
11       else return (cross-low, cross-high, cross-sum)
```

```
FIND-MAX-CROSSING-SUBARRAY(A, low, mid, high)
 1  left-sum = −∞
 2  sum = 0
 3  for i = mid downto low
 4       sum = sum + A[i]
 5       if sum > left-sum
 6            left-sum = sum
 7            max-left = i
 8  right-sum = −∞
 9  sum = 0
10  for j = mid + 1 to high
11       sum = sum + A[j]
12       if sum > right-sum
13            right-sum = sum
14            max-right = j
15  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?