CS146 Data Structures and Algorithms
Fall 2020, Instructor: K. Potika SJSU CS Department

Programming Project 2

Important: Do individually (each student alone is NOT a group assignment), each student has to turn in a java program. Copying from other students or other resources is plagiarism.

The Maximum Subarray problem – Solving with different techniques

In this project we will consider the Maximum Subarray (or Largest Sum Contiguous Subarray) problem discussed in class and presented in our textbook in section 4.1 and lecture 4. You have to implement, optimize and analyze three different algorithms that solve the same problem.

Problem Motivation: Suppose you are a freelancer and that you plan to work at a Mykonos resort for some part of the n-day summer season next year. Unfortunately, there isn't enough work for you to be paid every day and you need to cover your own expenses (but you want to go). Fortunately, you know in advance that if you are at the resort on the ith day of the season, you'll make pi euros where pi could be negative (if expenses are more than earnings) or positive (if expenses are less than earnings). To maximize your earning you should choose carefully which day you arrive and which day you leave; the days you work should be consecutive and you don't need to work all season. For example, if n = 8 and p1 = −9 , p2 = 10 , p3 = −8 , p4 = 10 , p5 = 5 , p6 = −4 , p7 = −2 , p8 = 5 then if you worked from day 2 to day 5, you would earn 10 − 8 + 10 + 5 = 17 euros in total. Assume the resort pays your airtickets.
Facts: When all numbers are positive, then all days is the answer, but when all numbers are negative; no days are selected and the total is 0. Your result should always be 0 or a positive number. Hint: use arrays. The output is the maximum sum, as well as the arriving and departing day.

A. The first algorithm, is a brute force $O(n^2)$ algorithm that considers all possible pairs of arriving and departing dates. The outer loop picks the beginning element, the inner loop finds the maximum possible sum with first element picked by outer loop and compares this maximum with the overall maximum.

B. The second algorithm, is a divide and conquer $O(n \lg n)$ algorithm. The basic idea follows:

Algorithm D&C (pseudocode)

```
Use a Divide and Conquer approach. The idea follows
1) Divide the given array in two halves
2) Return the maximum of the following three
….a) Maximum subarray sum in left half (Make a recursive call)
….b) Maximum subarray sum in right half (Make a recursive call)
….c) Maximum subarray sum such that the subarray crosses the midpoint
```

For 2c) we can easily find the crossing sum as follows: simply combine the left part and the right part.

Basically, that method finds the maximum sum starting from mid point and ending at some point on left of mid, then finds the maximum sum starting from mid + 1 and ending with sum point on right of mid + 1 and adding them together.

C. The third algorithm, is a (dynamic programming) O(n) algorithm, known as Kadane's Algorithm. The basic idea is that you keep only positive contiguous summations that we compute by adding one element each time, kept in maxTemp, and at the end keep the best, and save in maxSum. In the next pseudocode, also the arrive and depart index are computed.

Kadane's Algorithm:

```
//Initialize
  maxSum = 0
  maxTemp= 0
  arrive=-1
  depart=-1
  tempArrive=0
for i=0 to n-1 in array a
  (a) maxTemp = maxTemp + a[i]
  (b) if(maxTemp < 0)
        maxTemp = 0
        arrive=i+1 //next day is the arrival since i-th not included
  (c) if(maxSum < maxTemp)
        maxSum = maxTemp
        depart=i
        tempArrive=arrive
arrive=tempArrive
return (maxSum, arrive, depart)
```

returns depart -1 if sum is <0

Test cases:

1. Consider the tests that are given in **maxSumtest.txt**[1] use them for the JUnits.
The file has one test case per line (10 cases each with 100 entries).
A line corresponding to the examples in the above file would be of the form
`[array], sum, arrive, depart`
2. You can try other simple examples as these two with an array size of 8
```
-2 -3 4 -1 -2 1 5 -3 7 2 6

-3 -4 -5 -6 -7 -8 -9 -10 0 -1 0
```

Additionally, you should test it for various small and different cases.
3. For doing the experimental analysis of the running time, plot the running times as a function of the input size. Run each of your algorithms on random input arrays of size n=100, 200, 500,1000, 2000, 5000, 10000, (for each size/algorithm run 10 times and take average). The first algorithm (a) may be very slow, therefore go up to 1000 only. Remember to have positive and negative numbers.

[1] http://web.engr.oregonstate.edu/~glencora/cs325/mstest.txt

For example,

```
for i = 1 to 10
      create a random array a[] n (size) elements
      start clock
      run maxSubarrayAlgorithm(a)
      stop clock
      add to elapsed time
return (elapsed time/10)
```

Note: Do not include the time to generate the random arrays.

Plots for running time

- Plot the running times as a function of input size for each algorithm in a single plot. Label your plot (axes, title, etc.).
- Include an additional plot of the running times on a log-log axis. See here for an explanation: http://en.wikipedia.org/wiki/Log-log_graph. Note that if the slope of a line in a log-log plot is m, then the line is of the form O(x^m) on a linear plot. Check this video: http://www.khanacademy.org/math/algebra/logarithms/v/logarithmic-scale

Programming Standards (check posted rubric):

- Your header comment must describe what your program does.
- You must include a comment explaining the purpose of every variable or named constant you use in your program.
- You must use meaningful identifier names that suggest the meaning or purpose of the constant, variable, function, etc.
- Precede every major block of your code with a comment explaining its purpose. You don't have to describe how it works unless you do something tricky.
- You must use indentation and blank lines to make control structures more readable.

Deliverables:

- ✓ Your main grade will be based on (a) how well your tests cover your own code, (b) how well your code does on your tests (create for all non-trivial methods), and (c) how well your code does on my tests (which you have to add to your test file). For JUnit tests check canvas.
- ✓ Use cs146F20.<lastname>.project2 as your package, and Test classes should be your main java file, along with your JUnit java tests.
- ✓ What to submit to canvas (Steps):
    - Export your project on eclipse with your entire project (source code).
    - Create a zip file named **LastnameFirstProjectName.zip** that includes the exported zip file + pdf report
    - upload the last zip file to canvas.
- ✓ All projects need to compile. If your program does not compile you will receive 0 points on this project.
- ✓ Do not use any fancy libraries. We should be able to compile it under standard installs. Include a readme file on how to you compile the project.