Lab: Maximum Sum Contiguous Subsequence

Problem: In computer science, the maximum subarray problem is the task of finding the contiguous subarray within a one-dimensional array of numbers (the array contains at least one positive number) which has the largest sum. For example, for the sequence of values −2, 1, −3, 4, −1, 2, 1, −5, 4; the contiguous subarray with the largest sum is 4, −1, 2, 1, with sum 6.

The problem was first posed by Ulf Grenander of Brown University in 1977, as a simplified model for maximum likelihood estimation of patterns in digitized images. A linear time algorithm was found soon afterwards by Jay Kadane of Carnegie-Mellon University (Bentley 1984). (http://en.wikipedia.org/wiki/Maximum_subarray_problem)

Lab, Part 1: Designing an algorithm (follows Bentley 1984)

To start your goal is **not** to develop the most efficient algorithm. Rather it is **to apply the design paradigms you have been learning and develop different solutions**. At the same time you will reinforce your understanding of asymptotic analysis of algorithms by deriving the Θ() bounds for each approach. **As you work through the following, use the attached template to record your results in a document, clearly labeling each improvement and your derivation of its computational complexity.**

Brute force Solutions

Consider a brute force solution to this problem A[1..N] is an array containing a sequence of integers

Initial solution

```
int maxSoFar = 0;
for low = 1 to N
     for high =  low to N
          int ThisSum = 0;
          for k = low to high
               ThisSum = ThisSum + A[ k ];
          if( ThisSum > MaxSum )
               maxSoFar = ThisSum;
return maxSoFar;
```

A. 1) Derive the asymptotic computational complexity of this solution? Work inside out

Improvement 1: By removing redundant calculations find an improvement that gives a quadratic solution

B. Quadratic Solution

1) Clearly write the pseudo code of your algorithm
2) Derive the asymptotic computational complexity of this solution?

Improvement 2

C. Design an n * log n Divide and Conquer Solution

1) Clearly write the pseudo code of your algorithm
2) What is the recurrence relation for the divide and conquer solution?
3) Derive the asymptotic computational complexity of this solution?

Improvement 3:

D. Design a Scanning solution

Try to develop a solution that scans the numbers linearly and keeps track of certain values, so that the best solution can be found.

1) Clearly write the pseudo code of your algorithm
2) Derive the asymptotic computational complexity of this solution?

Lab, Part 2: Design and implement an experiment to test empirical results against the predicted asymptotic results

E. **Continue to record answers to the questions in your file clearly labeling each answer by the question number, e.g. E.1 , convert to a pdf and submit to PolyLearn by 9:00 p.m. on Friday 4/28 .**

Read the program provided, it will be uploaded on Friday 4/28. It implements four of the solutions discussed in part 1. Think about what your hypothesis would be in terms of how they would perform on arrays of various sizes.

1) **What curves would you expect to find that would fit the data?**

Run the program for different sizes of the array to determine what the appropriate sizes for the experiment would be. (Note: Make sure you understand what the duration numbers mean. Also you will want to use different sizes of arrays for the different algorithms?)

2) **Why do you need to run the different size tests for the different algorithms to get some idea of the empirical performance of the different algorithms?**

Before running the final tests. make a few runs with the same sample size. How much do the results vary from run to run on your computer? This a problem, why?

3) **How can you reduce its effect?**

Once you have a set of array sizes to test for each algorithm. Run the tests. Record the results in you write up.

Graph the results to compare your results to the asymptotic predictions.

4) **Do the empirical results match the theoretical curves? Why or why not?**