Computer Science 320SC – (2018)

Programming Assignment 4

Due: Saturday, September 22nd (11:57pm)

Requirements

This fourth assignment lets you get familiar with successive improvements with algorithm design for a concrete "real-world" problem. There are three programs required using cleaver brute-force, divide-and-conquer and dynamic programming. It is worth 5% of your total course marks. Please try and test with your own generated input cases before submitting to the automated marker.

Problem: Fibre Cabling

A big internet infrastructure company in New Zealand has been tasked with wiring up homes with fibre optic broadband cables. However, since this is a business for profit, not all homes will need to be hooked up. The company Circus has asked you to help them determine which homes they should connect cables, ready for sale. The main constraint of this problem is that if two homes on a street is cabled then all intermediate homes on the path between them must also be, irrespective if they sign-up for an internet plan or not. The company makes a certain amount of profit based on which plan customers purchase; it also losses money if a complicated connection is done without any plan purchase.

An input scenario is a sequence $a_1, a_2, \ldots, a_n$ of $n \leq 100000$ integers on one line, denoting the profit/loss for connecting each house on a street of length n. The entry at position i, $1 \leq i \leq n$, is positive for profit and negative for loss. We have many streets of data to process and a single line containing only one integer denotes the end of input (this scenario is not processed). We can assume each scenario has at least one positive house value a_i, since otherwise Circus would not even consider cabling that street.

The output of your program, one line per each scenario, is the maximum profit obtainable for that street.

Sample Input:

```
10 -4 5
3 -4 5
3 -10 4 41 -1
10 0 20 -4 -3 8 -7 4
4 4 -3 1 0 0 -4 1 3 -4 7 -2 -2 3
1
```

Sample Output

```
11
5
45
31
9
```

Submission 1: `fibreBrute.ext`

There is a natural simple brute-force algorithm that tries and sums every subsequence $a_i + a_{i+1} + a_{i+2} + \cdots + a_{j-1} + a_j$ for $1 \leq i \leq j \leq n$. Then return the maximum of all combinations of starting positions i and ending positions j This algorithm has running time $O(n^3)$. With a simple observation that, one can compute both the sums $a_i + \cdots + a_{j+1}$ and $a_{i+1} + \cdots + a_j$ from $a_i + \cdots + a_j$ with one addition/subtraction, we can get a $O(n^2)$ algorithm. An implementation of this algorithm is required for a program named `fibreBrute.ext`, where `ext` is an extension of the programming language of your choice.

Submission 2: `fibreDC.ext`

For this part of the assignment we want to develop an even higher performance algorithm based on the divide-and-conquer design approach. An $O(n \lg n)$ algorithm is possible by using an algorithm similar to the 'counting inversions' algorithm given in lectures. Consider the following sequence:

$$a_1, a_2, \ldots, a_{\frac{n}{2}-1}, a_{n/2}, a_{\frac{n}{2}+1}, \ldots, a_{n-1}, a_n$$

An optimal solution either uses the element $a_{n/2}$ or not. So we can take the maximum of three cases: recursively solve subproblem $a_1, \ldots, a_{\frac{n}{2}-1}$, recursively solve subproblem $a_{\frac{n}{2}+1}, \ldots, a_n$, or the "conquer" case that includes $a_{n/2}$. Submit a program named `fibreDC.ext`.

Submission 3: `fibreDP.ext`

Finally we want to improve to an $O(n)$ time algorithm. You should use dynamic programming to solve this case. Submit a problem named `fibreDP.ext`.

Comments

The markers will quickly look at the last of each of your submissions to make sure that you implement all three algorithms and not just submit the fastest one, say `fibreDP.ext`, to the automarker for all problems. The first implementation `fibreBrute.ext` is worth one mark and the other two improvements are worth two marks each. There will be easy and hard test cases loaded on the automarker for the two problems worth two marks. The number of submissions per problem is limited to 8 before a 20% deduction is applied (assuming you solve a particular problem eventually). Note that for the hard test cases the sums may require arbitrary precision integers (e.g. use Python 3 ints, Java BigInteger data type, or C/C++ GMP library).