CS29003 Algorithms Laboratory

LAB TEST: ODD PC

Last date of submission: 02–November–2017

Prof. Abracad has written two books: one on Apples and the other on Berries. Prof. Abracad lives in the developing country Bopropoti where the currency is BPP. But the publishers of his book are from a developed country Nodepotico where the currency is NPC. At the beginning of the story, one NPC can be exchanged by one hundred BPP. After that, the value of one NPC increases by eleven BPP per month.

Prof. Abracad receives cheques for the next n months for his books. The publisher of his Apples book has announced monthly installments (cheque amounts) $a_0, a_1, a_2, \ldots, a_{n-1}$ in currency NPC, and the publisher of the Berries book has announced monthly installments $b_0, b_1, b_2, \ldots, b_{n-1}$, again in currency NPC. The laws of Bopropoti prohibit one from depositing multiple foreign cheques in one's bank account in a month. So Prof. Abracad will deposit his $2n$ cheques in $2n$ consecutive months. A restriction from the publishers is that the cheques must be deposited in the order they are sent. This means that cheque a_i must be deposited before cheque a_{i+1}. Likewise, check b_j must be deposited before cheque b_{j+1}. Assume that there is no expiry date of cheques. If $n = 3$, Prof. Abracad may, for instance, choose to deposit the cheques in the order $b_0, a_0, b_1, a_1, a_2, b_2$ to get a total exchange of $100b_0 + 111a_0 + 122b_1 + 133a_1 + 144a_2 + 155b_2$ BPP.

Assume that the cheque amounts a_i and b_j are floating-point numbers. The goal of Prof. Abracad is to choose the sequence of depositing the $2n$ checks in $2n$ consecutive months starting from the beginning of the story, in such a way that his total exchange in currency BPP is maximized. He is very weak in algorithms, so you are required to help him solve his problem.

Part 1: Exhaustive Search

Suppose that cheques $a_0, a_1, \ldots, a_{i-1}$ and $b_0, b_1, \ldots, b_{j-1}$ are deposited, so Prof. Abracad has two options: deposit a_i or deposit b_j. At this point, the exchange rate for one NPC is $100 + 11(i + j)$ BPP. Exhaustive search explores both the possibilities recursively one by one, and returns the larger of the maximum exchanges that can be obtained by the two options. Write a function *esexchange* implementing this idea.

Part 2: Greedy Strategy

Since the exchange rate increases monotonically with time, it is a nice idea to deposit cheques of smaller amounts before cheques of larger amounts. Until cheques from one of the publishers are all deposited, Prof. Abracad has two deposit options a_i and b_j for some i and j. The greedy approach is to deposit a_i if $a_i \leqslant b_j$ or to deposit b_j otherwise. Implement this greedy strategy in a function *grexchange*.

Part 3: Dynamic Programming

Exhaustive search of Part 1 runs in time exponential in n, and is useless unless n is very small. The greedy strategy of Part 2 is superfast (linear time), but is not guaranteed to lead to the best possible solution. A dynamic-programming algorithm can be designed to compute the best solution in $O(n^2)$ time. The algorithm builds a two-dimensional table T such that $T[i][j]$ stands for the maximum exchange that can be obtained by depositing $a_i, a_{i+1}, \ldots, a_{n-1}$ and $b_j, b_{j+1}, \ldots, b_{n-1}$ from month $i + j$ to month $2n - 1$. The final result will be available as $T[0][0]$. Clearly mention in a comment how the entries of T are related, and what the initial conditions are. Following this comment, write a function *dpexchange* to implement this algorithm.

The *main*() function

- Read n, $a_0, a_1, a_2, \ldots, a_{n-1}$, and $b_0, b_1, b_2, \ldots, b_{n-1}$ from the user.
- Call *esexchange*, and print the return value.
- Call *grexchange*, and print the return value.
- Call *dpexchange*, and print the return value.

Sample output

```
n = 10

     71.98 88.31 89.93 54.80 62.62 27.27 64.92 43.58 64.09 40.63

     43.96 96.92 53.26 62.89 29.85 21.80 12.52 67.73 66.92 33.09

+++ Exhaustive Search
    Maximum exchange = 224949.38

+++ Greedy Strategy
    Maximum exchange = 213692.31

+++ Dynamic Programming
    Maximum exchange = 224949.38
```

Submit a single C/C++ source file. Do not use global/static variables.

CS29003 Algorithms Laboratory

LAB TEST: EVEN PC

Last date of submission: 02–November–2017

Ms. Cadabra wants to sell n items costing $c_0, c_1, c_2, \ldots, c_{n-1}$ (floating-point numbers) initially. The items are all perishable, so with time the cost of each item reduces. For $d = 0, 1, 2, 3, \ldots$, the cost of the i-th item on the d-th day is $0.9^d \times c_i$. Ms. Cadabra can sell only one item per day, so she has to sell all of the n items on n consecutive days.

The items are stored one after another in a rack in her warehouse. For simplicity, assume that n is even. At the center of the rack, there is a gate. Initially, exactly half (that is, $\frac{n}{2}$) items $c_0, c_1, c_2, \ldots, c_{\frac{n}{2}-1}$ are to the left of the gate, and the remaining $\frac{n}{2}$ items $c_{\frac{n}{2}}, c_{\frac{n}{2}+1}, c_{\frac{n}{2}+2}, \ldots, c_{n-1}$ are to the right of the gate. Ms. Cadabra can access only the exposed items to the left and to the right of the gate. That is, on Day 0, she can sell either $c_{\frac{n}{2}-1}$ or $c_{\frac{n}{2}}$. As an example, take $n = 6$. The items to the left of the gate are to be sold in the sequence c_2, c_1, c_0, and the items to the right of the gate are to be sold in the sequence c_3, c_4, c_5. If Ms. Cadabra chooses the sequence R, L, L, R, R, L (where L means left and R means right) on the six days, she is sequentially selling the items with initial costs $c_3, c_2, c_1, c_4, c_5, c_0$, and her total profit is $c_3 + 0.9c_2 + 0.9^2c_1 + 0.9^3c_4 + 0.9^4c_5 + 0.9^5c_0$.

Because of the depreciation of the items with time, the total profit of Ms. Cadabra depends upon the sequence of choosing the options L and R. Since Ms. Cadabra is very poor in algorithms, you are required to help her maximize her total profit.

Part 1: Exhaustive search

Suppose that i items are sold from left and j items from right. Suppose that both i and j are $< \frac{n}{2}$. At this point, the $(\frac{n}{2} - i - 1)$-th item is exposed to the left, the $(\frac{n}{2} + j)$-th item is exposed to the right, and the depreciation factor is 0.9^{i+j}. Exhaustive search explores both the possibilities recursively one by one, and returns the larger of the maximum profits that can be obtained by the two options. Write a function *esprofit* implementing this idea.

Part 2: Greedy Strategy

Since the item costs decrease monotonically with time, it is a nice idea to sell the costlier items before the cheaper items. If both the left and the right parts of the rack contain items yet to be sold, Ms. Cadabra has two selling choices $c_{\frac{n}{2}-i-1}$ or $c_{\frac{n}{2}+j}$ for some i and j. The greedy approach is to sell $c_{\frac{n}{2}-i-1}$ if $c_{\frac{n}{2}-i-1} \geqslant c_{\frac{n}{2}+j}$ or to sell $c_{\frac{n}{2}+j}$ otherwise. Implement this greedy strategy in a function *grprofit*.

Part 3: Dynamic Programming

Exhaustive search of Part 1 runs in time exponential in n, and is useless unless n is very small. The greedy strategy of Part 2 is superfast (linear time), but is not guaranteed to lead to the best possible solution. A dynamic-programming algorithm can be designed to compute the best solution in $O(n^2)$ time. The algorithm builds a two-dimensional table T. Let i items be sold from the left and j items from the right. The table entry $T[i][j]$ stands for the maximum profit that can be achieved from this point (in days $i + j$ to $n - 1$). The final result will be available as $T[0][0]$. Clearly mention in a comment how the entries of T are related, and what the initial conditions are. Following this comment, write a function *dpprofit* to implement this algorithm.

The *main*() function

- Read n, and $c_0, c_1, c_2, \ldots, c_{n-1}$ from the user.
- Call *esprofit*, and print the return value.
- Call *grprofit*, and print the return value.
- Call *dpprofit*, and print the return value.

Sample output

```
n = 20

     7.95 9.06 1.48 0.17 8.80 3.86 1.91 4.90 5.42 2.14 9.52 4.16 6.11 2.77 7.45
     9.50 1.19 7.52 8.99 8.03

+++ Exhaustive Search
     Maximum profit = 51.663545

+++ Greedy Strategy
     Maximum profit = 50.085989

+++ Dynamic Programming
     Maximum profit = 51.663545
```

Submit a single C/C++ source file. Do not use global/static variables.