Programming Design, Spring 2015
Lab Exam 1

Instructor: Ling-Chieh Kung
Department of Information Management
National Taiwan University

Submission and grading. In this exam, there are three problems. You need to write a C++ program for each problem. 100% of your grades will be based on the correctness of your outputs. The online grading system will input in total 50 sets of testing data and then check your outputs. These 50 sets count for 100 points, i.e., 2 points for each set. Before the due time of the exam, you may upload your programs multiple times. Only the last one you upload will be graded. Unlike what happens with your homework, you will not see the grading results during the exam. For each problem, you will only see the results for the online samples.

To submit your work, please upload the three .cpp files, one for each problem, to the online grading system at https://pdogs.ntu.im/judge/.

Just a reminder: Talking to any other person online, directly or indirectly, is definitely considered cheating.

Problem 1

(40 points) Given an order-n polynomial $f(x) = a_n x^n + a_{n-1}x^{n-1} \cdots + a_1 x_1 + a_0$, and a domain $[b, c]$, we want to find the integer x^* that attains the minimum of $f(x)$ among all integers in $[b, c]$. More precisely, we want to find $x^* \in \mathbb{Z} \cap [b, c]$ such that

$$f(x^*) \leq f(x) \quad \forall x \in \mathbb{Z} \cap [b, c].$$

If there are multiple integers that all attain the minimum, we call the smallest one *the optimal solution* and report it. For example, given $f(x) = x^3 + x^2 + 1$ and the domain $[-1, 1]$, the three candidate integers -1, 0, and 1 results in $f(-1) = f(0) = 1$ and $f(1) = 3$. As both -1 and 0 attain the minimum, we report -1 as our optimal solution.

Input/output formats

The input consists of 20 files. In each file, there are three lines of integers:

- In the first line, there is a single integer n as the order of a polynomial. It is given that $1 \leq n \leq 4$.
- In the second line, there are $n + 1$ integers a_n, a_{n-1}, ..., and a_0. They define the polynomial $f(x) = a_n x^n + a_{n-1}x^{n-1} \cdots + a_1 x_1 + a_0$. a_i may be positive, negative, or zero. It is given that $|a_i| \leq 100$. Each pair of two values are separated by a white space.
- In the third line, there are two integers b and c. They define the domain $[b, c]$. It is given that $-50 \leq b \leq c \leq 50$,

Your program should read each input file and then print out two integers, the optimal solution x^* and its function value $f(x^*)$. For example, suppose we have

```
3
1 1 0 1
-1 1
```

as a file of input, the output should be

```
-1 1
```

in a single file. If you plan to use `pow()`, be aware of possible floating-point outputs.

Problem 2

(40 points) You are given a fixed amount of budget B for buying an item as many as possible. What complicates the problem is that the average price of the item depends on the quantity you purchase. The supplier of this item has announced a *price schedule* consisting of n quantity-price pairs (q_1, p_1), (q_2, p_2), ..., (q_n, p_n), where $0 < q_1 < q_2 < \cdots < q_n$. For the kth unit you purchase, you pay p_i if $q_i < k \leq q_{i+1}$. q_n is the maximum number that you may purchase (even if you have infinitely much money). Given a price schedule and a fixed amount of budget, your task is to determine how many units you may purchase and how much you will spend.

As an example, consider the following price schedule

$$q = (100, 200, 300) \quad \text{and} \quad p = (20, 19, 18).$$

If you have $B = 4000$ dollars, you will be able to buy 100 units at $20 \times 100 = 2000$ dollars, buy another 100 units at $19 \times 100 = 1900$ dollars, and the last 5 units at $18 \times 5 = 90$ dollars. Though you still have 10 dollars, you cannot buy any more. Suppose that you have $B = 10000$ dollars, you will only buy 300 units (by spending 5700 dollars) because the maximum purchase quantity is 300.

Input/output formats

The input consists of 20 files. In each file, there are three lines of integers:

- In the first line, there are two integers n and B, which are the number of schedule items and budget amount. These two numbers are separated by a white space. It is given that $1 \leq n \leq 100$ and $0 < B \leq 1000000$.
- In the second line, there are n integers q_1, q_2, ..., and q_n, where $[q_{i-1}, q_i]$ is the ith quantity region (with $q_0 = 0$), $i = 1, ..., n$. It is given that $0 < q_1 < q_2 < \cdots < q_n$. Each pair of two values are separated by a white space.
- In the second line, there are n integers p_1, p_2, ..., and p_n, where p_i is the unit prices of region i, $i = 1, ..., n$. It is given that $p_i > 0$, $i = 1, ..., n$. Each pair of two values are separated by a white space.

Your program should read each input file and then print out two integers, the maximum quantity that can be purchased and the price spent for purchasing that amount. For example, suppose we have

```
3 4000
100 200 300
20 19 18
```

as a file of input, the output should be

```
205 3990
```

in a single file.

Problem 3

(20 points) Given some boxes, each can carry the same weight, and a few items with different weights, we are interested in using the minimum number of boxes to carry all items. As an example, suppose that each box can carry 10 kg, and you have six items weighing 8, 7, 6, 5, 4, and 3 kg. One trivial way is to use 6 boxes, each containing one item; a better ways is to use one box to contain the two items weighing 7 and 3, another box to contain the two items weighing 6 and 4, and two more boxes to contain the two items weighing 8 and 5. As the total weight of the six items is 33, using 4 boxes is the best we can do.

In this problem, we do not require you to find a best way to carry all items. Instead, we want you to implement the following algorithm:

> In each iteration, you start by using an empty box to contain a not yet carried item that is currently the heaviest. Then you ask whether there is any not yet carried item that can be put into the same box. If there is one, put it into the box and continue searching for a not carried item that can be put into the same box; if there are multiple, put a heaviest and continue searching; if there is none, this iteration is finished and you start the next iteration with a new empty box.

As an example, consider the above six-item instance. In iteration 1, we put the 8-kg item into box 1; in iteration 2, we put the 7-kg item and then the 3-kg one into box 2; in iteration 3, we put the 6-kg item and then the 4-kg one into box 3; in iteration 4, we put the 5-kg item into box 4. This is indeed the best way. There are of course some cases that this algorithm fails to find a best way (i.e., using the minimum number of items). For example, if the six items weigh 5, 4, 4, 3, 2, and 2 kg, the algorithm group items into three boxes ($\{5, 4\}$, $\{4, 3, 2\}$, and $\{2\}$) where the best way uses only two boxes ($\{4, 4, 2\}$ and $\{5, 3, 2\}$). Nevertheless, all we want in this problem is to implement this algorithm.

Input/output formats

The input consists of 10 files. In each file, there are two lines of integers:

- In the first line, there is one single integer B, which is the maximum weight that may be carried by one box. It is given that $0 < B \leq 1000000$.
- In the second line, there are $n + 1$ integers w_1, w_2, ..., w_n, and -1, where w_i is the weight of item i, $i = 1, ..., n$. It is given that $0 < w_i \leq B$ for all $i = 1, ..., n$ and $1 \leq n \leq 10000$. Each pair of two values are separated by a white space. These w_is may be not ordered.

Your program should read each input file, assign items to boxes by the given algorithm, and then print out two integers k and r, where k is the number of boxes used and r is the maximum unused capacity. For each box, the unused capacity is B minus the sum of weights of all items assigned to that box. The maximum amount of unused capacity is the maximum of the unused capacities of all boxes. As an example, suppose you are given

```
10
2 3 2 4 5 4 -1
```

as a file of input, the output should be

```
3 8
```

in a single file. Note that the unused capacities of the three boxes are 1, 1, and 8, and thus the maximum unused capacity is 8.