# University of Central Florida


## 2017 (Fall)

## Local Programming Contest

| Problems | | | |
|---|---|---|---|
| Problem# | Difficulty Level | Filename | Problem Name |
| 1 | Easy | energy | Electric Bill |
| 2 | Easy | typing | Simplified Keyboard |
| 3 | Easy/Medium | singing | Singin' in the Rain |
| 4 | Medium | editor | Editor Navigation |
| 5 | Medium | darts | Simple Darts |
| 6 | Medium | transport | Multimodal Transport |
| 7 | Medium/Hard | game | Videogame Probability |
| 8 | Medium/Hard | mnois | Maximum NOI Subseq |
| 9 | Medium/Hard | cards | Rotating Cards |
| 10 | Hard | multi | Multiples |
| 11 | Hard | kshop | K-Item Shopping Spree |

Call your program file: filename.c, filename.cpp, filename.java, or filename.py

For example, if you are solving Electric Bill:

Call your program file: energy.c, energy.cpp, energy.java, or energy.py

# UCF Local Contest — September 2, 2017

## Electric Bill

*filename:* `energy`
(*Difficulty Level:* `Easy`)

To encourage customers to conserve energy (and protect the environment), the electric companies typically have a lower rate for the first 1000 kilowatt-hour (KWH) of electric usage and a higher rate for the additional usage (KWH is a derived unit of energy equal to 3.6 mega-joules).

**The Problem:**

Given the rate (per KWH) for the first 1000 KWH usage, the rate (per KWH) for the additional usage, and a customer's energy consumption, you are to determine the charges (bill) for the customer.

**The Input:**

The first input line contains two integers (each between 2 and 20, inclusive), indicating the rate/KWH for the first 1000 KWH and the rate/KWH for the additional usage, respectively. The next input line contains a positive integer, *n*, indicating the number of customers to process. Each of the following *n* input lines contains an integer (between 1 and 50000, inclusive), indicating a customer's energy consumption.

**The Output:**

For each customer, print the energy consumption, followed by a space, followed by the charges.

**Sample Input:**

```
6 10
4
1000
1001
700
4800
```

**Sample Output:**

```
1000 6000
1001 6010
700 4200
4800 44000
```

# UCF Local Contest — September 2, 2017

## Simplified Keyboard

*filename:* `typing`
(*Difficulty Level:* `Easy`)

Consider a simplified keyboard consisting of the 26 lowercase letters as illustrated below:

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| a | b | c | d | e | f | g | h | i |
| j | k | l | m | n | o | p | q | r |
| s | t | u | v | w | x | y | z | |

We define the neighbors of a key (letter) as all the letters adjacent to it. For example, the neighbors of 'a' are {b, k, j}, neighbors of 'b' are {a, c, l, k, j}, neighbors of 'n' are {d, e, f, o, x, w, v, m}, and neighbors of 'z' are {p, q, r, y}.

**The Problem:**

Given two words consisting of lowercase letters only, you are to determine which of the following three cases applies to them:

1. identical: this is when the two words are of the same length and they match letter-by-letter. For example, "cool" and "cool" are identical, "cool" and "col" are not, and "cool" and "colo" are not.

2. similar: this is when the two words are of the same length, they are not identical words, and each corresponding two letters either match or are neighbors. For example, "aaaaa" and "abkja" are similar, "moon" and "done" are similar, "knq" and "bxz" are similar, but "ab" and "cb" are not (because of 'a' in the first word and the corresponding 'c' in the second word).

3. different: this is when neither of the above two cases applies to the two words, i.e., they are not identical and they are not similar. For example, "ab" and "abc" are different, "ab" and "az" are different, and "az" and "za" are different.

**The Input:**

The first input line contains a positive integer, *n*, indicating the number of test cases to process. Each of the following *n* input lines represents a test case, consisting of two words separated by one space. Each word consists of lowercase letters only and will be between 1 and 20 letters, inclusive.

**The Output:**

For each test case, output one line. That line should contain the digit (number) 1, 2, or 3, to indicate which of the above three cases applies to the two input words.

**Sample Input:**

```
7
a k
a a
a z
cool cool
aaaaa abkja
ab abc
az za
```

**Sample Output:**

```
2
1
3
1
2
3
3
```

# UCF Local Contest — September 2, 2017

## Singin' in the Rain

*filename:* `singing`
(*Difficulty Level:* `Easy/Medium`)

During the time of the 2016 UCF Local Programming Contest, Arup's younger daughter Anya (3 years old at the time), made Arup incessantly listen to Taylor Swift's song "Wildest Dreams". A full year later, we are proud to report that Anya's listening habits have matured greatly. Rather than wanting to hear the same song over and over again, Anya has embraced an embryonic notion of diversity in music. Now, she wants to hear various different tracks, in sequence, ***all from the same CD***.

This year, it turns out that Anya's favorite CD is "Singing in the Rain", which her older sister Simran obtained when she performed in the "Singing in the Rain" production at a local theater. Whenever Anya is in the car with Arup, she'll listen to a track and then call out the number of the next track that she wants to listen to. The problem is that Arup's car has a rather primitive CD player:

- If track number *k* has completed, then track *k+1* will play. If track *k* is the last track on the CD, the CD will wrap around and track 1 will play.

- Arup can also change tracks by pressing a forward button. If track number *k* has completed, if Arup presses the forward button, then track *k+2* will play. If *k* is the last track and Arup presses forward when *k* finishes, the CD will wrap around and track 2 will play next. If *k* is next to the last track and Arup presses forward when *k* finishes, track 1 will play next.

- Arup can also change tracks by pressing a backward button. If track number *k* has completed, if Arup presses the backward button, track number *k* will play again. If *k* is the first track and Arup presses the backward button twice right after track 1 completes, the CD will wrap around and track *t* will play next, where *t* is the number of tracks on the CD.

This means Arup is pressing either the forward or backward button a great deal. Help him minimize the number of times he presses the buttons.

**The Problem:**

Given the number of tracks on Anya's favorite CD and the sequence of tracks Anya wants to be played, determine the minimum number of button presses Arup must make to get the appropriate sequence of songs played. For the purposes of this problem, assume that at the very beginning the CD player is cued up to play the first song in the sequence (the first song Anya wants to be played), so Arup does not have to press any buttons for the first song in the sequence to be played, i.e., the first time any buttons might have to be pressed is in between the first and second songs in the sequence (after the first song in the sequence plays).

**The Input:**

The first input line contains a positive integer, *n*, indicating the number of test cases to process. Each test case starts with two space separated integers on a single line: $t$ ($1 \leq t \leq 10^9$), the number of tracks on Anya's favorite CD and $s$ ($1 \leq s \leq 1000$), the number of songs Anya would like to listen to from the CD. The second line of each test case contains $s$ space separated integers, representing the sequence of tracks from the CD that Anya would like to listen to. Each of these will be in between 1 and $t$, inclusive.

**The Output:**

For each test case, output a single integer on a line by itself indicating the minimum number of button presses Arup can use to play the desired sequence of songs from the CD.

**Sample Input:**

```
3
68 6
67 57 66 67 48 15
1000000000 7
1 500000002 3 500000004 5 500000006 7
3 3
3 1 1
```

**Sample Output:**

```
73
3000000000
1
```

**Explanation for Sample Output:**

In Case #1, we first push the backwards button 11 times after 67 completes to cue up 57. Then we press the forward button 8 times to get 66 cued up after 57 finishes. After 66 finishes, we press no buttons and let 67 play, followed by pressing the backward button 20 times to arrive at track 48. Finally, we can either press the backward button 34 times to get to 15 or the forward button 34 times to get to 15. Thus, the minimal number of button presses is 11 + 8 + 0 + 20 + 34 = 73.

In Case #2, we must either press the forward or backward button 500,000,000 times between each pair of consecutive songs on the list.

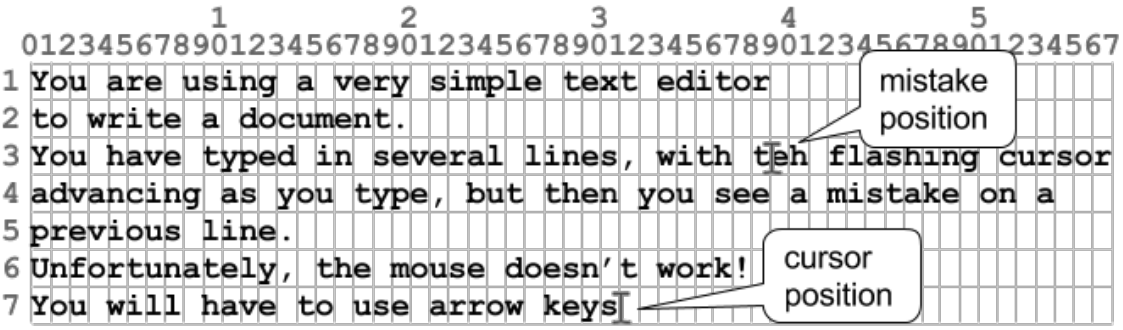
In Case #3, we must press the backward button once in between track 1 playing the first time and the second time.

# UCF Local Contest — September 2, 2017

## Editor Navigation

*filename:* `editor`
(*Difficulty Level:* `Medium`)

You are using a very simple text editor to create a document. You have typed in several lines, with the flashing cursor advancing as you type, but then you see a mistake on a previous line. Unfortunately, the mouse doesn't work! You will have to press arrow keys to move the cursor back to the position where you can fix the mistake. Of course, you want to get to this position as quickly as possible.




This simple editor uses a monospace font, so each character is exactly the same width. The cursor can be at the beginning of the line (before the first character), end of the line (after the last character), or at a horizontal position between two characters on a given line. The following keys can be pressed to move the cursor (each keypress is independent of any preceding keypress):

| | |
|---|---|
| ← (Left arrow key) | Moves the cursor one character to the left on the same line, unless the cursor is at the beginning of the line, in which case it moves to the end of the previous line. If the cursor is at the beginning of the line and there is no previous line, the cursor does not move. |
| → (Right arrow key) | Moves the cursor one character to the right on the same line, unless the cursor is at the end of the line, in which case it moves to the beginning of the next line. If the cursor is at the end of the line and there is no next line, the cursor does not move. |
| ↑ (Up arrow key) | Moves the cursor up to the previous line; keeps the same horizontal position, unless the previous line is shorter, in which |

| | |
|---|---|
| | case the cursor goes to the end of the previous line. If there is no previous line, the cursor does not move. |
| ↓ (Down arrow key) | Moves the cursor down to the next line; keeps the same horizontal position, unless the next line is shorter, in which case the cursor goes to the end of the next line. If there is no next line, the cursor does not move. |

**The Problem:**

Given the line lengths of a text file that was loaded in this simple editor, along with the current cursor position and a different, desired cursor position (e.g., to fix a mistake), you are to determine the minimum number of keypresses, using arrow keys only, required to move the cursor to the desired position.

**The Input:**

The first input line contains a positive integer, $n$, indicating the number of editor navigation scenarios to process. Each scenario will occupy exactly 4 input lines. The first line of each scenario contains an integer $f$ ($1 \leq f \leq 120$), indicating the number of lines of text in the file that is loaded in the editor. The next input line contains $f$ integers, $s_1$ to $s_f$, where each value $s_i$ ($0 \leq s_i \leq 80$) indicates the number of characters on line $i$ of the file; the values will be separated by exactly one space. A value $s_i = 0$ means that there are no characters on line $i$. The newline character (common character indicating end of a line) does not count as a character on the line.

The third input line of each scenario will contain two integers (separated by a space) providing the current cursor position in the file: $r_c$ ($1 \leq r_c \leq f$) and $c_c$ ($0 \leq c_c \leq 80$), where $r_c$ represents the line of the file, counting from 1 (as with $i$), and $c_c$ represents the horizontal position of the cursor on that line, 0 for the beginning (before the first character). It is guaranteed that the cursor position is valid, so if, for instance, $r_c = 17$ and $s_{17} = 56$, then $0 \leq c_c \leq 56$; the maximum value of $c_c$ is the end of the line. The fourth input line of each scenario is similar to the third and indicates the desired cursor position to begin fixing the mistake, i.e., this line consists of two integers separated by a space, $r_m$ ($1 \leq r_m \leq f$) and $c_m$ ($0 \leq c_m \leq 80$). The constraints on the values for $r_c$ and $c_c$ also apply to $r_m$ and $c_m$.

**The Output:**

For each scenario in the input, output a single integer on a line by itself indicating the minimum number of keypresses needed to move the cursor from its current position ($r_c$, $c_c$) to the desired position ($r_m$, $c_m$) for fixing the mistake.

**Sample Input:**

```
2
7
39 20 57 54 14 38 31
7 31
3 39
3
15 30 20
1 12
3 3
```

**Sample Output:**

```
21
8
```

**Explanation for Sample Output:**

For Case #1, one possible sequence for the minimum number of keypresses to move the cursor from its current position to the desired position is: Up, Up, Right, Up, Left, Up, Left 15 times.

# UCF Local Contest — September 2, 2017

## Simple Darts

*filename:* `darts`
(*Difficulty Level:* `Medium`)

A staple to many game rooms, darts is a fun game that doesn't require a lot of physical space. The standard dartboard (see Figure 1) consists of 6 concentric rings and 20 wedges, dividing the board into a number of regions, each with well-defined scores (note: for a higher picture resolution/clarity, fewer than 20 wedges are depicted in Figure1). The centermost ring is scored as a double bullseye, while the second ring gives the score of a single bullseye. Beyond the second ring, each wedge has a score between 1 and 20 with the spaces between certain rings having double and triple score modifiers. A new, simpler version of the game is being proposed to attract younger players to the game.

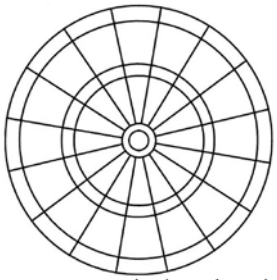


Figure 1, Standard Dartboard

**The Problem:**

The simpler version of the game is illustrated in Figure 2. More specifically, the board consists of exactly 3 (instead of 6) concentric rings and it may have fewer (instead of exactly 20) wedges. Also, in the simpler board, the wedges are of equal size. The scoring for this simple version is as follows:

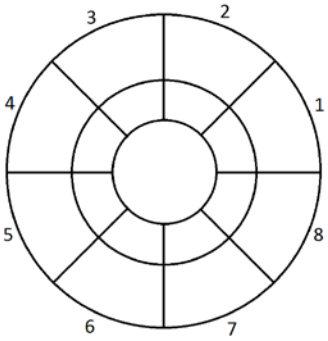



Figure 2, Example Simple Dartboard

Assume the board is centered at the origin ("0,0" in Cartesian plane). Let *w* refer to the number of wedges on the board (8 in Figure 2), and let *b, d, s* refer to (respectively) the radii of the smallest, second smallest, and the largest rings.

The wedge immediately above the positive x axis (Wedge 1 in Figure 2) is scored at 1 point and the score for each wedge is increased by 1 in a counterclockwise fashion, thus resulting in the wedge below the positive x axis (Wedge 8 in Figure 2) having a score of *w*. The area within the circle centered at the origin with radius *b* represents the bullseye and is always scored at 50 points. The area between radii *b* and *d* denotes the double ring and any dart landing within the double ring is scored at twice the value of the surrounding wedge. The area between radii *d* and *s* denotes the single ring and any dart landing within the single ring is scored at the value of the surrounding wedge. Any dart landing outside the dartboard carries a score of zero (0).

Given the description (layout) of such a board centered at the origin and a number of dart throws, you are to calculate the total score.

**The Input:**

The first input line contains a positive integer, *n*, indicating the number of test cases to process. Each test case will contain multiple input lines. The first line of each test case will contain four integers separated by a space: $w$ ($2 \leq w \leq 20$), representing the number of equal-sized wedges on the board, followed by $b, d, s$ ($0 < b < d < s < 100$), representing the radii of the bullseye, double ring and single ring regions of the board. The second input line of each test case will contain an integer, $t$ ($1 \leq t \leq 100$), indicating the number of dart throws. Each of the following $t$ input lines contains two floating point numbers (three decimal places), $x$ and $y$ ($-100 \leq x,y \leq 100$), providing the Cartesian coordinates of a dart throw. Assume that no dart will land within $10^{-5}$ of any boundary (line or arc/curve).

**The Output:**

For each test case, output a single integer on a line by itself indicating the total score for all dart throws.

**Sample Input:**

```
3
4 7 13 10
2
4.000 4.000
6.000 -4.000
10 1 6 10
1
20.000 -0.500
8 3 7 50
5
-0.750 1.207
1.180 3.132
27.111 -44.630
-43.912 -22.104
2.000 -6.000
```

**Sample Output:**

```
58
0
73
```

# UCF Local Contest — September 2, 2017

## Multimodal Transport

*filename:* `transport`
(*Difficulty Level:* `Medium`)

New methods of shipping goods have transformed the transportation industry and helped usher in an era of global commerce. Nowadays, someone can click a few buttons and have an item leave a factory on the other side of the world and show up at their doorstep the next day. To help accomplish this, there are a number of modes of transport within the shipping industry. The four most common are: air, boat, rail and truck.

Transportation companies tend to keep the mode of transport consistent throughout a packages journey (route/path). However, when customers are not very time sensitive, often times the cheapest way to move a package from one location to another is to use more than one mode of transport. One has to be careful, though, as additional costs are incurred when switching between transport modes within a city on the package path. (Switching transport mode refers to a package leaving a city in a different mode than it arrived at the city, e.g., the package arrived by air and leaves by truck.)

A new startup, KnightShip, has asked for your help in figuring out the cheapest way to ship packages when switching between transportation modes is acceptable.

**The Problem:**

Given the costs for various modes of transport between cities, and the cost of switching mode within a city, calculate the lowest cost to transport an item from an origin to a destination.

**The Input:**

The first input line contains a positive integer, *n*, indicating the number of test cases to process. Each test case will contain multiple input lines. The first line of each test case will contain an integer, $c$ ($2 \leq c \leq 400$), representing the number of cities within the transportation network. Each of the following $c$ input lines contains two values: a string (1 to 20 uppercase letters, inclusive), representing the name of a city and an integer (between 1 and 1000, inclusive), representing the cost of changing the transport mode within that city.

The city information for a test case is followed by the route information for the test case. There will be an input line containing one integer, $r$ ($1 \leq r \leq 40000$), representing the number of route

segments in the network. This will be followed by a listing of all route segments, one per line. Each route segment will contain four values: *p*, *q*, representing two cities from the previous list, *m*, representing the transport mode (one of four values AIR, BOAT, RAIL, TRUCK) for that route segment, and an integer *v* ($1 \leq v \leq 1000$), representing the cost to ship a package between the two cities (either direction). Note that there may be no route segments for a particular transport mode. There will be no duplicate city pair within a given mode of transport, but different transport modes (even all four modes) can exist between the same two cities.

The last input line for a test case contains two distinct cities *o* and *d* which represent the origin and destination cities for which we want the minimum cost to ship a package. Cities *o* and *d* are guaranteed to have a path (of finite cost) that exists between them. Any mode of transport can be used to leave city *o* and any mode can be used to reach city *d* (they don't necessarily need to match). The transport mode can change in the intermediate stages as well.

**The Output:**

For each test case, output a single integer on a line by itself indicating the minimum cost to move a package from city *o* to city *d*.

**Sample Input:**

```
2
4
ORLANDO 10
TAMPA 15
MIAMI 5
JACKSONVILLE 10
7
TAMPA JACKSONVILLE AIR 100
MIAMI TAMPA SEA 70
JACKSONVILLE MIAMI RAIL 45
ORLANDO JACKSONVILLE TRUCK 85
TAMPA ORLANDO RAIL 10
MIAMI JACKSONVILLE SEA 15
ORLANDO MIAMI TRUCK 15
JACKSONVILLE TAMPA
2
ORLANDO 15
TAMPA 10
3
ORLANDO TAMPA AIR 7
TAMPA ORLANDO TRUCK 3
ORLANDO TAMPA RAIL 19
ORLANDO TAMPA
```

**Sample Output:**

```
55
3
```

**Note:**

Due to the large input bounds on the number of cities ($2 \leq c \leq 400$) and route segments $r$ ($1 \leq r \leq 40000$), an *Order-cubed* solution will result in "time limit exceeded" with the given judge run time for this problem.

# UCF Local Contest — September 2, 2017

## Videogame Probability

*filename:* `game`
(*Difficulty Level:* `Medium/Hard`)

You have just joined the illustrious Ewokin guild (group of players) in your favorite video game. This guild is known for attempting the hardest raids the instant that they are released. For these new raids you need the best gear (items) in order to have a viable chance of success. With your skills as a programmer, you have managed to determine the current drop rates for different item types in the game (drop rate refers to the probability of catching an item when the item is dropped). Now you need to know how likely it is that you will gather your gear (items) and bring your guild one step closer to success.

**The Problem:**

Given the number of different item types in a game, how many of each item type you need, the probability of obtaining (catching) each item type when it drops, and the maximum number of possible attempts you have for catching the items, determine the probability that you will obtain the desired number of each item type.

**The Input:**

The first input line contains a positive integer, $n$, indicating the number of test cases to process. Each test case starts with an integer, $g$ ($1 \leq g \leq 50$), indicating the number of different item types in the game. The following $g$ input lines provide the information about the different item types. Each such line contains two values: an integer, $c$ ($0 \leq c \leq 50$), representing how many of this item type is needed, and a floating point number, $p$ ($0.0 \leq p \leq 1.0$), representing the probability of catching this item type when it drops. The last input line for a test case contains an integer, $a$ ($0 \leq a \leq 10000$), representing the maximum number of attempts you have for catching the items. Note that this last input represents the total number of attempts and not the attempts for each item type. An attempt can, of course, be used (applied) to obtain any item type.

**The Output:**

For each test case, print a floating point number on a line by itself, representing the probability that you will obtain the desired number of each item type. Output the results to 3 decimal places, rounded to the nearest thousandth (e.g., 0.0113 should round to 0.011, 0.0115 should round to 0.012, and 0.0117 should round to 0.012).

**Sample Input:**

```
4
2
3 0.5
3 0.3
20
4
2 0.75
1 0.01
2 1.0
3 0.8
25
2
1 0.5
1 0.3
2
2
50 .4
50 .3
250
```

**Sample Output:**

```
0.816
0.153
0.150
0.036
```

# UCF Local Contest — September 2, 2017

## Maximum Non-Overlapping Increasing Subsequences

*filename:* `mnois`
(*Difficulty Level:* `Medium/Hard`)

Given a list of integers, we can find an increasing subsequence from that list, i.e., select specific elements from the list where the selected numbers are in increasing order. There is a classic problem that asks to find the longest increasing subsequence, i.e., increasing subsequence with the maximum number of elements. In our problem, we are going to do a modified version of that problem. Instead of finding one increasing subsequence, we will find multiple increasing subsequences with the following two constraints:

1. The length of each subsequence should be *at least* of a given size *k*.
2. If a subsequence starts at index *i* and ends at index *j*, no other subsequence can start or end in the range *i* to *j* (inclusive), i.e., no other subsequence can have any elements in between index *i* and *j* (inclusive), i.e., the subsequences cannot overlap.

The objective is to find subsequences (satisfying the above two conditions) resulting in the maximum number of elements being selected, i.e., the total number of elements in all these subsequences combined is the maximum.

For example, consider the list 2 1 9 3 4 4 5 6.
If $k = 2$, we can get three non-overlapping increasing subsequences with maximum of 7 elements: [2, 9], [ 3, 4], [4, 5, 6].
If $k = 3$, we can get two non-overlapping increasing subsequences with maximum of 6 elements: [2, 3, 4], [ 4, 5, 6].

**The Problem:**

Given a list of $n$ integers, determine the maximum number of integers you can include in the non-overlapping increasing subsequences for all $k$ where $1 \leq k \leq n$.

**The Input:**

The first input line contains an integer, $t$ $(1 \leq t \leq 50)$, indicating the number of test cases to process. Each test case starts with an integer, $n$ $(1 \leq n \leq 100)$, indicating the number of integers in the list. The second line of each test case contains the list of $n$ integers (each in the range of $-10^6$ to $10^6$, inclusive).

**The Output:**

For each test case, output $n$ integers (on a single line) indicating, respectively, the maximum number of integers in the non-overlapping subsequences for all $k$ where $1 \leq k \leq n$.

**Sample Input:**

```
3
8
2 1 9 3 4 4 5 6
2
1 1
3
1 2 3
```

**Sample Output:**

```
8 7 6 5 5 0 0 0
2 0
3 3 3
```

# UCF Local Contest — September 2, 2017

## Rotating Cards

*filename:* `cards`
(*Difficulty Level:* `Medium/Hard`)

A magician has a stack of *n* cards labeled 1 through *n*, in random order. Her trick involves discarding all of the cards in numerical order (first the card labeled 1, then the card labeled 2, etc.). Unfortunately, she can only discard the card on the top of her stack and the only way she can change the card on the top of her stack is by moving the bottom card on the stack to the top, or moving the top card on the stack to the bottom. The cost of moving any card from the top to the bottom or vice versa is simply the value of the label on the card. There is no cost to discard the top card of the stack. Help the magician calculate the minimum cost for completing her trick.

**The Problem:**

Given the number of cards in the magician's stack and the order of those cards in the stack, determine the minimum cost for her to discard all of the cards.

**The Input:**

The first input line contains a positive integer, *t*, indicating the number of test cases to process. Each test case is on a separate input line by itself and starts with an integer, *c* ($1 \leq c \leq 10^5$), indicating the number of cards in the stack, followed by *c* labels for the cards in the stack (starting from the top going to the bottom). Each of these labels will be in between 1 and *c*, inclusive, and each label will be unique.

**The Output:**

For each test case, output a single integer on a line by itself indicating the minimum cost for the magician to complete her magic trick.

**Sample Input:**

```
2
5 3 5 1 4 2
3 1 2 3
```

**Sample Output:**

```
15
0
```

# UCF Local Contest — September 2, 2017

## Multiples

*filename:* `multi`
(*Difficulty Level:* `Hard`)

Sumon's 9-year-old son Samin likes programming. He becomes happy when his program compiles, runs, and gives correct output (at least for the small test cases). He does not bother much about making time-efficient solutions. It was good easy life for Samin, until the 2016 UCF Local Programming Contest.

His baby brother Saraf was born the very next day after the 2016 UCF Practice Local Contest, with a desperate interest to participate in the Real Local Contest. Unfortunately, for various reasons, we could not fulfill Saraf's wish to participate in the very special "2 0 16" (= 2 0 $2^{2*2}$ = 2 0 $(2*2)^2$) UCF Local Contest at the age of "6" (2+2+2) days. Anyway, being born in the UCF contest week, Saraf is clearly destined to grow sophisticated taste for programming, and make his older brother's programming life harder. Now, Saraf wants his brother to run hard and medium range test cases at least for 20 iterations to test his code efficiency. If Samin's code does not run fast enough, Saraf gets angry, sits on the laptop, and keeps hitting it until the result is displayed.

Samin now needs to do a project for his school. To save Samin's laptop from Saraf's rage while running that program, you are asked to help Samin by writing the best possible time efficient solution to this problem.

**The Problem:**

Given two integers $a$ and $b$, you are to find how many integers $1 \le b_i \le b$ are multiple of any integer $2 \le a_i \le a$. For example, if $a = 3$ and $b = 30$, we are interested in how many integers in the range of 1-30 are a multiple of 2 or 3; there are 20 such integers: 2, 3, 4, 6, 8, 9, 10, 12, 14, 15, 16, 18, 20, 21, 22, 24, 26, 27, 28, 30.

**The Input:**

The first input line contains a positive integer, $n$ ($1 \le n \le 100$), indicating the number of queries to process. Each query will be on a separate input line and will contain two integers separated by a space: $2 \le a \le 130$, $1 \le b \le 10^{15}$.

**The Output:**

For each query, output a single integer on a line by itself indicating the number of integers in the *b* range that are a multiple of any number in the *a* range.

**Sample Input:**

```
2
3 30
11 123
```

**Sample Output:**

```
20
97
```

# UCF Local Contest — September 2, 2017

## *K*-Item Shopping Spree

*filename:* `kshop`
(*Difficulty Level:* `Hard`)

It's back to school time and you've won a shopping spree! It's got some interesting rules. There are *n* items to choose from and you must select a sequence of *k* items. Any of the *n* items can be selected any number of times. A shopping spree is identified by the indices of the selected items (and not by the item values) and two shopping sprees are considered to be different if any of the corresponding item indices are different. For example, if there are 5 items (indexed 1 to 5) and you must select a sequence of 3 of them, any of the following would be different valid shopping sprees: (1, 2, 1), (2, 1, 1), (2, 1, 3), (5, 5, 2), (3, 5, 4). Note, again, that the shopping sprees are identified by item indices and not item values, e.g., the first two sprees listed are different because the first item chosen in the two sprees are different. Thus, the same combination of items can result in different shopping sprees if chosen in different orders.

For this problem, you are to find the number of *K*-Item Shopping Sprees that have a specified total value.

**The Problem:**

Given the number of items available in a shopping spree, their values, the value of *k*, and a price target, *X*, calculate the number of possible shopping sprees of precisely *k* items that has a total value of exactly *X*. Since the number of possible sprees can be extremely large, calculate it modulo 997.

**The Input:**

The first input line contains a positive integer, *c*, indicating the number of test cases to process. Each test case starts with an integer, $n$ ($1 \leq n \leq 10000$), representing the number of items available in the shopping spree. Each of the following $n$ input lines provides a floating point number, $p_i$ ($0.01 \leq p_i \leq 500.00$) with two digits past the decimal point, representing the value in dollars of each item available in the shopping spree.

The next input line of each test case will contain two space separated integers: $k$ ($1 \leq k \leq 10000$), representing the number of items to be selected in sequence for the shopping spree, and $q$ ($1 \leq q \leq 10000$), representing the number of queries for the test case. Each of the following $q$ lines will contain one query. Each query will be a single floating point number, $t$ ($0.01 \leq t \leq 500.00$) with two digits past the decimal point, representing the value in dollars of a target value for the shopping spree.

**The Output:**

For each query in a test case, print (on a line by itself) the number of possible shopping sprees (modulo 997) that are equal in value to the query value. Leave a blank line after the output for each test case.

**Sample Input:**

```
3
5
0.07
0.08
0.12
0.08
0.15
2 4
0.12
0.19
0.15
0.23
2
0.05
0.08
3 3
0.18
0.23
0.24
3
1.00
1.00
1.00
3 1
3.00
```

**Sample Output:**

```
0
2
4
4

3
0
1

27
```