Homework Assignment 3

Problems and point distribution:

- Stable worm holes: 100pts
- Portable shield generators: 150pts
- Faster-than-light communicators: 150pts
- Warp-drive engines: 150pts
- Photon swords: 150pts
- Carbon sequestration station: 150pts
- Plug-and-play AI: 200pts
- Hats!: 50pts (extra credit)
- Assembling the super computer: 180pts (extra credit)

Time-outs:

- Python: 5 seconds
- C++: 2 seconds (w/ g++ -O3 flag)

You may solve the programming problems in either Python or C++. To use Python, name the file containing your code `main.py`. To use C++, name the file `main.cpp`. You are encouraged to test your code locally.

At any time, you can choose to run the tests that will be used to grade your solution. To do so, navigate to the Gradescope assignment for that particular homework question and upload **only** `main.py` or `main.cpp`. Gradescope will tell you how many test cases passed, and what error occurred on the ones that failed.

When you run the tests on Gradescope, debugging output will be printed to tell you which test cases you passed, which ones you failed, and what your total score currently is for that problem. If a test case failed, the output will specify a failure mode (e.g. timeout, compiler error, wrong answer, etc.).

The first few test cases for each problem are visible test cases. This means that if you fail them, Gradescope will tell you the the expected output and the output your code actually gave. Note that these outputs may not be listed in order. The rest of the test cases for each problem are hidden test cases, which means you will not be shown the input or expected output.

All assignments MUST be manually submitted to Gradescope in order to complete submission! There is no auto-submit feature enabled for any Gradescope coursework. There will be no exemptions granted if you forget to manually submit your assignment. If time permits, you can contact the course staff and they can check on the Gradescope platform and confirm your assignment submission status.

1. Stable Worm Holes

(100 pts)

Description

Worm holes provide connections from one point in the galaxy to another. Unfortunately, they can have instabilities that will cause them to collapse when a ship tries to pass through them. Fortunately, Oganesson Dynamics Inc.'s top engineers have proposed a new way to build worm-hole stabilizers! Even better, these new stabilizers can repair multiple instabilities, if needed, though at an increasing cost for each additional one.

You have access to k stabilizers and there are n instabilities that need to be fixed. Each instability i has a base energy cost of e_i terawatts. The first instability fixed by an individual stabilizer has its cost multiplied by $\times 1$, the second by $\times 2$, and so on, so that the k^{th} instability costs $\times k$.

Given n, k, and the base cost for each instability, find and print the minimum total energy (in terawatts) for all of the instabilities to be repaired.

Note: instabilities can be repaired in ANY ORDER.

Input Format

- The first line contains two space-separated integers describing the respective values of n and k.
- The second line contains n space-separated positive integers describing the respective values of $e_0, e_1, ..., e_{n-1}$

Constraints

- $1 \le n, k \le 100$
- $1 \le e_i \le 10^6$
- answer $< 2^{31}$
- $0 \le i < n$

Output Format

Print the total energy required to repair all of the instabilities.

Example 0

Sample Input

```
3 3
2 5 6
```

Sample Output

```
13
```

Explanation

There are three instabilities and three stabilizers. Since each stabilizer can fix one instability, there is only a multiple of × 1 for each. The total cost is:

(2 × 1) + (5 × 1) + (6 × 1) = 13

Example 1

Sample Input

```
7 3
6 1 28 10 15 3 21
```

Sample Output

```
105
```

Explanation

There are seven instabilities and three stabilizers to fix them with.

The stabilizers can repair, in order:

- first stabilizer: 28, 10, and 1
- second stabilizer: 21 and 6
- third stabilizer: 15 and 3

The first stabilizer would have an energy expenditure of $28 + 10 \times 2 + 1 \times 3 = 51$

The second stabilizer would have an energy expenditure of $21 + 6 \times 2 = 33$

The third stabilizer would have an energy expenditure of $15 + 3 \times 2 = 21$

... for a grand total of $51 + 33 + 21 = 105$.

Example 2

Sample Input [1]

```
55 34
433515 22221 540709 538312 496949 902471 439983 159332
417327 399306 817804 354682 108825 970689 868286 235070
18732 848955 994152 741000 722232 564879 503093 734475
174795 129065 483929 683440 37645 670198 911023 40334
329513 669160 180034 285512 401190 629798 857703 765572
174164 849299 159367 62467 84449 523962 735995 245666
832139 879827 749840 315756 253168 659252 187178
```

Sample Output

```
30352414
```

[1] Line-wrapped lines are shown in yellow.

2. Portable Shield Generators

(150 pts)

Description

Portable shield generators can keep ships, buildings, and even people safe from harm. Oganesson Dynamics Inc. has a blueprint that will produce a device with a long series of aegis cells that can be individually turned on to produce a shield. Each cell has an individual contribution to shield strength. The total shield strength is the sum of these individual contributions.

There's a problem though! The cells get very warm when activated, and if you turn on two neighboring ones at the same time, they will overheat. What is the maximum total shield strength possible?

Given a series of N aegis cells where cell i, in order, has a shield strength of S_i, determine the maximum total shield strength that can be generated.

HINT 1: how would dynamic programming help? See handout on D2L for some ideas: Assessments > Assignments > Assignment 3 > dynamic-programming-hint.pdf

Input Format

- The first line provides N, the number of aegis cells in the row.
- The second line has N values, separated by spaces, indicating the individual contributions to total shield strength (S_0 through S_{N-1}).

Constraints

- $1 \leq N \leq 10^6$
- $1 \leq S_i \leq 10^{10}$

Output Format

A single value indicating the maximum total shield strength possible.

Example 0

Sample Input

```
5
2 10 1 3 10
```

Sample Output

```
20
```

Explanation

There are five aegis cells, with shield levels of 2, 10, 1, 3, and 10, respectively.

If we activate the second and fifth cells, we will have a total of 20 shield strength (no other legal combination can produce more than 20).

Example 1

Sample Input

```
5
20 10 1 10 20
```

Sample Output

```
41
```

Explanation

There are five aegis cells, with shield levels 20, 10, 1, 10, and 20, respectively.

If we activate the first, middle, and last cells, we will have a total of 41 shield strength (no other legal combination can produce more than 41).

Example 2

Sample Input [2]

```
200
1047129894 3797367497 394826071 9280785640 2477752358
3233669043 826145110 7250698392 5753206911 557104746
45755527 5040188463 2383658854 6431884393 2941742415
889855528 4130448825 6744534394 4998309382 5381159515
31437377 784162251 8617113464 7791869866 958451927
7766264380 216182132 9524688063 191909343 696337156
527559792 48458739 8358947530 256591505 5257977873
1890888698 1774302331 1380954890 4498127845 9477329438
964314761 895664073 3764756811 4814124537 6499114528
42713779 12677328 3724815872 579347820 7232308031
3248721374 9253461460 5040225944 217323565 904901175
469885653 1450569435 895640368 1313135786 848624542
727752853 7774904816 3699625890 9954333945 9071243361
4741424188 5981819525 219435671 392438846 632256322
416713330 2636627919 14642894 6491361194 8566527935
8940226060 1155284079 975543711 2839158855 24975203
9469705853 7697647285 154297446 3699844176 716844792
93468932 183229787 332672856 8347785286 2425677855
5320177074 984448667 7113971334 706221251 4135838375
8954934136 4846227627 1488447054 332549359 2319937211
4784329456 252575032 7847394031 529771582 38174276
839996223 217386855 164214698 4732153047 7070815361
8614836492 8954957796 855746071 6593683628 6752438134
2373249849 4222819747 6217595650 326078812 7773539660
7963143669 4549196320 393465874 76686150 6910366671
3264619443 4579972492 6511429798 863696896 63148905
412451449 6996483694 6622103819 8455893449 9752129081
83296196 996418190 684111517 504586562 521668775 5987712521
6555318727 79597565 1095763049 3838267197 483519517
4241626186 6144563855 5587546742 623957693 50325881
968794384 520388341 4696373425 126817170 3888798543
7563626787 26181986 7490363676 719771234 5528491428
3764561763 22481282 2094494459 735766962 782807288
```

[2] Line-wrapped lines are shown in yellow.

```
4913698750 7212748221 742369765 517724888 521475144
5561375368 327401185 2316447130 39686027 473623781
6157621539 1492238215 322022453 9186139121 6478555940
453749549 8176612369 287893458 1229292628 6184727073
8011562284 846544323 9445469430 4994326582 6477315439
105316655 12024433 1577177715 9918412847 239773313
3921599488 971243415 4115673545 869391813
```

Sample Output

```
444580401095
```

3. Faster-than-light Communicators

(150 pts)

Description

A set of documents describe how to build a faster-than-light communications system. The only problem is that when a message is sent, its binary representation is encoded and must be decrypted on the other side. The binary string (B) is of length N. This string is written down K times, shifted respectively by 0, 1, ..., K - 1 bits.

For example, if B = 1001010 and K = 4 it would appear as:

```
1001010
 1001010
  1001010
   1001010
```

Next, calculate XOR in every column and write it down. This number representing the encoded message is call S. For example, XOR-ing the numbers in the above example results in:

```
1110100110
```

Both the encoded message (S) and the value K are sent to the recipient.

You must implement a decoding algorithm that will take S and K and reproduce the original bitstring B (in the above case, you would output '1001010').

Input Format

- The first line contains two integers, N and K.
- The second line contains the received string S of length $N + K - 1$, consisting of ones and zeros.

Constraints

- $1 \le N,K \le 10^6$

Output Format

The decoded message of length N, consisting of ones and zeros.

Example 0

Sample Input

```
7 4
1110100110
```

Sample Output

```
1001010
```

Explanation

```
1001010
 1001010
  1001010
   1001010   XOR
----------
1110100110
```

So, how did we find this solution?

Lets replace each bit with a true/false variable; we must use the encrypted sequence we were provided to determine those original variables.

```
abcdefg
 abcdefg
  abcdefg
   abcdefg   XOR
----------
1110100110
```

The first one is easy: since the first encrypted bit is a, it means that (given the first column) $a = 1$.

So we have:

```
1bcdefg
 1bcdefg
  1bcdefg
   1bcdefg   XOR
----------
1110100110
```

Now, looking at the second column (b and 1), we see that: $b \oplus 1 = 1$

(note, the $\oplus$ is the logical symbol for XOR, "exclusive or")

To solve for b, we need to know an important transform. That is, if you have $x \oplus y = z$, this is the same as "$x \oplus z = y$" or "$y \oplus z = x$" or any other ordering.

As such, $b = 1 \oplus 1$, or $b = 0$.

Back to our example:

```
10cdefg
 10cdefg
  10cdefg
   10cdefg   XOR
----------
1110100110
```

Next up, the third column (c, 0, 1) means that $c \oplus 0 \oplus 1 = 1$, which is the same as $c = 0 \oplus 1 \oplus 1$, hence $c = 0$.

Each column will have just one new variable to decypher until the entire original code is reconstructed.

WARNING: Some of the test cases are large, and you don't want to calculate every single XOR for each column. But, if you look at the previous column, you see that it was pretty similar. Can you re-use those previous calculations?

Either way, I recommend that you get something working first for *MOST* of the test cases and then optimize further when you need to.

HINT: After you've implemented an initial solution, further speedups can be obtained by making use of properties of XOR such as those discussed above. Another nice property of XOR can be stated as follows: $a \oplus (a \oplus b \oplus c) = b \oplus c$.

Example 1

Sample Input

```
6 2
1110001
```

Sample Output

```
101111
```

Explanation

```
101111
 101111
-------
1110001
```

Example 2

Sample Input [3]

```
185 48
0101001101011000110011010101000111011011111110001001110101
1101101000100011011001010001010110110000000100110110011001
0
1110010101110101010101100110111001010001000000111111000010
1
0011010111010001000101011110100101100101001010111001001
```

Sample Output

```
0111101011110100101010111111100100110110000000100101010010
0
1001001001101101000111100010111101001101010100110100011100
0
1111111100111111010000111001100111110001110111010111011111
0
01011011
```

[3] Line-wrapped lines are shown in yellow.

4. Warp-Drive Engines

(150 pts)

Description

Of all of their discoveries, Oganesson Dynamics is most excited by their new warp engines. The problem is that they go through the quadlithium crystals that power them much too quickly. Can you help figure out how to get the crystals to last for as long as possible?

Warp engines apply one of K different frequencies to a crystal to extract power from it for an hour, after which the crystal breaks into smaller crystals (some of which may be used again!). Each frequency i is associated with a fracture value F_i indicating the size of the crystals that this one will break into when used.

Frequency i can only be used if S (the size of the crystal) is greater than F_i and can be evenly divided by F_i. The crystal then breaks into P/F_i crystals each of which is size is size F_i.

Since each application of a frequency to a crystal can power the engine for an hour, what is the LONGEST amount of time (in hours) that the ship can fly given a single starting crystal of size N?

Input Format

The first line contains an integer, T, denoting the number of tests to be performed. The subsequent T pairs of lines describe each test in the following format:

- The first line contains two space-separated integers describing the respective values of S (the size of the initial crystal) and K (the number of available frequencies).
- The second line contains K space-separated integers describing the values of F_0 through F_{K-1}.

Constraints

- $1 \le T \le 10$
- $1 \le S \le 10^{12}$
- $1 \le K \le 1000$
- $1 \le F_i \le 10^{12}$

Output Format

For each test, determine the maximum number of hours that the ship can be powered given the starting crystal; output this value on its own line.

Example 0

Sample Input

```
1
12 3
2 3 4
```

Sample Output

```
4
```

Explanation

This input has 1 test case.

That test case starts with a crystal of size 12 and three frequences that would break a crystal into sizes 2, 3, or 4 (respectively).

```
Crystals: 12
```

The longest set of crystal uses starts with the frequency that produces crystals of size 4, which shatters the initial crystal into three smaller ones.

```
Crystals: 4 4 4
```

For our second hour, use frequency "2" on one of our size-4 crystals.

```
Crystals: 4 4 2 2
```

For our third hour, we again use frequency "2" on one of our size-4 crystals.

```
Crystals: 4 2 2 2 2
```

For our fourth and final hour, we again use frequency "2" on the final crystal of size four.

```
Crystals: 2 2 2 2 2 2
```

No additional steps are possible.

Example 1

Sample Input

```
4
64 5
2 4 8 16 64
1 2
1 2
6 1
3
64 6
2 4 8 16 32 64
```

Sample Output

```
29
0
1
31
```

Example 2

Sample Input

```
5
377083280820 10
1 377083280820 2 188541640410 3 125694426940 4 94270820205 5 75416656164
377083280820 9
1 377083280820 2 188541640410 3 125694426940 94270820205 5 75416656164
798652236637 2
1 798652236637
733493187656 6
1 733493187656 2 366746593828 4 183373296914
597670549095 5
1 597670549095 3 199223516365 5
```

Sample Output

```
282812460621
188541640416
1
550119890745
199223516366
```

Example 3

Sample Input

```
3
377083280820 10
1 377083280820 2 188541640410 3 125694426940 4 94270820205 5 75416656164
377083280820 9
1 377083280820 2 188541640410 3 125694426940 94270820205 5 75416656164
99033715019 5
1 99033715019 89 1112738371 864721694069
```

Sample Output

```
282812460621
188541640416
1112738372
```

Example 4

Sample Input [4]

```
5
377083280820 10
1 377083280820 2 188541640410 3 125694426940 4 94270820205 5 75416656164
68719476736 37
1 2 4 8 16 32 64 128 256 512 1024 2048 4096 8192 16384 32768 65536 131072
262144 524288 1048576 2097152 4194304 8388608 16777216 33554432 67108864
134217728 268435456 536870912 1073741824 2147483648 4294967296 8589934592
17179869184 34359738368 68719476736
68719476736 35
1 2 4 8 16 32 64 128 256 512 1024 2048 4096 8192 16384 32768 65536 131072
262144 524288 1048576 2097152 4194304 8388608 16777216 33554432 67108864
134217728 268435456 536870912 1073741824 2147483648 4294967296 8589934592
17179869184
72421814655 355
```

Sample Output

```
282812460621
68719476735
68719476733
33803608562
43074255585
```

[4] Line-wrapped lines are shown in yellow.

Example 5

Sample Input

```
1
84 4
42 7 6 3
```

Sample Output

```
17
```

Explanation

The initial crystal size is 84.

The best path way is:

- break the size 84 crystal into 2 crystals of size 42 (1 hour)
- break both size 42 crystals into 7 crystals each of size 6 (2 hours)
- break each of the 14 size 6 crystals into size 3 crystals (14 hours)

5. Photon Swords

(150 pts)

Description

The individual engineers at Oganesson Dynamics are most excited about the photon swords that they've discovered. Now they want to see just how hot they can make those swords without melting the handles.

Each sword has N buttons, each of which increases the heat by a fixed (integer) amount, to a limit of K (also an integer), beyond which the handle will melt. Given the amount of heat each button produces (B_0 through B_{N-1}), determine the maximum heat possible for the sword.

For example, if there were two buttons that put out 9 and 12 units of heat respectively, and the sword could handle a maximum of 31 units, you would be able to produce a maximum of 30 units by pressing the first button twice and the second button once. Any more presses and the device would melt. If there were also a button for 1 unit, you would be able to maximize the heat to a perfect 31.

Input Format

- The first line contains T, the number of test cases.
- Each test comprises two lines:
  - The first line contains two integers, N and K, representing the number of buttons and maximum heat, respectively.
  - The second line consists of space separated integers, B_0 through B_{N-1}, representing the heat output of each button.

Constraints

- $1 \le T \le 10$
- $1 \le N, K \le 2000$
- $1 \le B_i \le 2000$

Output Format

Output T lines, the maximum heat that can be produced for each test case which is as near as possible including, but not exceeding, the heat limit (K).

Hint 1: Try to get an initial solution using recursion.

Hint 2: After you implement an initial recursive algorithm, you can improve efficiency by using dynamic programming.

Hint 3: Notice that the heat and heat limit values are integers. How can you exploit this observation in your dynamic programming solution?

Example 0

Sample Input

```
3
3 12
1 6 9
5 9
3 4 4 4 8
4 11
5 7 8 9
```

Sample Output

```
12
9
10
```

Explanation

- In the first test case, you can press the '6' button twice to achieve the maximum heat output of 12.
- In the second test case, you can press the '3' button three times to hit the maximum heat output of 9.
- In the third test case, there is no way to reach the limit of 11, so the closest you can come is 10, by hitting the '5' button twice.

Example 1

Sample Input

```
7
3 9
3 2 4
3 12
3 10 4
3 13
3 10 4
3 16
3 10 4
3 2000
2000 2000 2000
3 9
9 9 9
3 8
9 9 9
```

Sample Output

```
9
12
13
16
2000
9
0
```

Example 2

Sample Input

```
1
2000 1
```

Sample Output

```
0
```

Example 3

Sample Input

```
5
1 2000
1
2 10
5 9
2 2000
2 1
2 2000
1999 1999
3 3
4 4 5
```

Sample Output

```
2000
10
2000
1999
0
```

Example 4

Sample Input [5]

```
5
500 13
38 387 278 416 294 336 387 493 150 422 363 28 191 60 264 27
41 427 173 237 212 369 68 430 283 31 363 124 68 136 430 303
23 59 70 168 394 457 12 43 230 374 422 420 285 38 199 325
316 371 414 27 92 481 457 374 363 171 497 282 306 426 85
328 337 6 347 230 314 358 125 396 83 46 315 368 435 365 44
251 88 309 277 179 289 85 404 152 255 400 433 61 177 369
240 13 227 87 95 40 296 71 435 379 468 102 98 403 318 493
153 257 302 281 287 442 366 190 445 120 441 230 32 118 98
272 482 176 210 428 68 357 498 354 87 466 307 184 220 125
29 372 233 330 4 20 271 369 209 216 341 150 297 224 119 246
347 452 422 56 380 489 265 229 342 351 194 1 35 265 125 415
488 357 244 492 228 366 360 437 433 52 438 229 276 408 475
122 359 396 30 238 236 294 319 429 144 12 429 30 277 405
444 264 114 39 107 341 405 319 129 189 370 418 418 497 325
244 471 184 491 500 273 226 145 91 6 140 455 287 170 83 43
465 198 8 356 305 349 112 123 329 300 344 247 69 341 423
312 311 106 302 162 231 379 306 321 237 445 127 23 466 209
417 283 259 425 138 63 125 101 37 453 400 380 51 469 72 474
132 382 431 434 395 161 164 200 482 400 497 460 274 314 169
191 96 427 467 85 341 91 185 377 43 437 108 446 257 180 419
388 413 349 173 160 10 337 211 343 88 207 302 214 33 322
256 320 100 222 405 440 312 441 168 206 229 128 151 485 159
421 225 423 270 397 82 131 85 293 473 173 351 126 386 223
300 141 43 399 214 299 191 25 91 210 82 320 337 23 156 95 5
380 270 274 277 351 256 361 143 80 385 44 206 122 68 5 114
462 255 327 260 445 203 23 7 285 22 343 369 29 190 373 409
459 49 37 309 254 249 304 334 134 149 391 255 68 247 369 30
1 47 289 298 250 491 304 34 364 498 254 393 187 126 153 497
476 189 158 230 437 461 415 422 461 305 29 28 51 249 57 403
295 198 200 44 40 3 429 404 1 182 148 39 160 152 36 135 340
193 216 128 5 130 50 465 286 430 344 336 178 401 239 472
```

[5] Line-wrapped lines are shown in yellow.

450 290 36 489 293 296 244 145 330 391 183 341 42 70 327
233
500 13
262 3 361 118 24 262 82 310 191 426 497 368 178 235 191 127
25 58 115 169 206 359 313 387 101 37 227 45 417 53 79 30
447 291 148 471 52 81 132 4 358 128 313 387 215 356 13 91
413 480 111 40 90 25 56 14 121 434 488 389 339 67 271 285
357 48 107 261 350 238 206 60 218 19 446 284 374 459 37 138
290 484 108 479 258 315 472 230 101 460 119 439 26 389 75
234 158 182 494 359 271 200 418 340 70 364 123 295 174 348
432 463 183 391 293 292 58 116 22 158 75 492 448 452 232 22
38 241 55 31 99 326 82 7 17 3 232 140 297 405 339 81 219 22
471 363 313 380 478 186 37 405 177 484 208 260 358 245 500
412 128 451 237 61 319 106 64 50 245 212 306 435 292 376
456 115 90 269 494 419 306 383 323 483 218 31 94 75 127 94
487 254 44 75 315 214 180 378 263 276 89 420 211 233 295 18
347 236 138 192 154 444 74 329 426 292 211 19 218 337 464
56 91 359 131 405 72 162 134 186 290 74 105 352 306 251 369
4 486 7 196 140 450 121 468 227 264 178 97 482 366 61 37
456 271 19 212 343 33 197 380 322 271 485 173 428 235 41
284 73 399 331 64 348 451 31 74 215 60 23 48 425 83 436 233
205 455 444 99 487 141 279 160 263 263 184 42 349 224 325
273 123 155 336 322 458 366 248 172 277 270 219 202 204 154
434 408 460 229 307 298 221 85 309 335 199 492 377 399 216
53 172 190 60 7 11 17 225 110 40 1 379 110 54 82 115 339
490 427 68 148 224 288 232 33 123 282 376 351 180 91 255
351 132 314 358 495 182 82 104 221 434 483 182 488 416 297
326 405 223 393 52 298 33 135 182 7 416 58 209 96 500 463
298 484 277 155 478 310 88 433 383 22 267 64 361 183 212
186 87 286 431 491 84 315 477 117 321 393 26 29 340 26 491
137 361 119 144 338 429 83 122 311 456 389 226 316 71 438
354 9 223 284 351 158 98 328 127 270 72 152 150 411 29 140
399 389 111 394 78 391 477 200 53 432 88 278 100 158 67 453
18 142 236 369 299 185 196 277 306 267 429 455 29 309 94
279 198 56 173 275 446 1 326 498 284 413 128 383 422
500 13
194 335 440 335 42 160 486 458 355 262 263 473 42 217 353
351 163 483 400 218 155 174 16 7 352 365 291 264 492 173 38
38 360 329 372 281 488 357 91 342 471 353 166 12 70 18 362
84 352 113 301 7 139 168 365 490 33 155 105 376 180 142 413

39 470 137 171 457 345 261 150 315 466 315 327 387 184 40
470 36 153 122 394 291 290 110 132 174 265 236 49 296 378
314 334 199 450 356 156 294 469 157 461 434 324 287 172 359
178 141 246 182 262 491 324 51 101 455 76 365 43 125 160
420 290 345 470 239 52 477 384 20 134 344 305 457 482 476
167 12 468 413 193 230 403 369 132 3 175 208 219 217 184
378 488 473 74 458 63 126 434 298 497 419 142 154 227 475
481 394 486 449 306 31 30 60 399 161 63 425 220 281 142 403
11 481 227 84 290 141 209 224 439 58 494 432 211 221 406
191 114 392 139 271 422 168 330 172 181 244 96 400 25 89
155 387 70 233 470 211 374 31 434 164 88 280 95 150 500 352
340 465 243 331 87 16 350 416 187 382 12 135 134 388 223
288 274 144 20 243 355 245 125 140 260 64 419 354 213 270
206 405 234 300 235 320 316 436 88 354 170 451 488 303 338
63 90 111 206 461 205 412 58 330 404 317 393 322 23 106 444
80 362 29 379 448 201 46 383 288 400 52 90 387 354 427 449
295 37 7 108 93 418 165 422 321 481 167 495 355 124 438 434
485 318 313 432 18 210 166 157 109 70 246 496 423 172 296
70 60 302 177 153 72 341 426 244 173 92 90 28 67 379 313 51
197 125 334 214 187 500 371 295 69 116 142 343 140 438 264
199 91 440 203 14 132 129 258 305 72 347 184 139 226 496 41
422 473 227 135 159 226 357 453 146 473 447 340 112 384 104
162 326 43 217 340 175 345 97 331 268 295 14 258 20 361 151
293 333 377 280 491 454 136 296 99 108 242 438 71 477 41 85
302 84 301 493 110 497 441 440 264 236 305 374 107 165 24
252 350 252 31 340 205 166 487 303 126 80 92 48 408 485 132
62 420 432 54 29 428 495 320 44 82 124 269 188 140 144 439
489 395 321 181 99 487 19 253 464 99 196 11 6 180 143 67 99
426 473 479
500 13
354 467 298 249 48 273 17 87 413 160 378 401 54 198 433 4
36 452 108 499 50 155 362 407 335 4 326 285 429 298 264 134
116 61 382 15 186 398 101 98 409 330 350 314 380 135 317
415 86 276 266 487 431 127 393 117 130 70 253 410 367 16
396 334 429 277 348 114 27 301 63 287 130 264 101 9 398 417
276 483 45 41 321 327 19 66 295 500 135 47 409 354 63 304
187 343 433 35 308 311 335 370 97 316 485 197 325 383 466
452 217 362 344 38 188 362 103 482 361 89 380 121 442 294
425 129 136 209 163 443 371 497 164 467 312 1 16 488 235
333 291 451 194 134 340 381 495 294 214 355 383 445 476 324

239 252 452 374 312 466 169 182 314 84 500 478 36 15 465
270 347 108 73 392 241 412 24 8 58 337 294 440 282 121 115
372 372 419 97 35 384 265 68 198 301 68 175 336 434 491 458
133 450 30 24 191 293 148 130 350 336 423 141 469 44 256
340 267 26 437 154 261 53 221 458 205 288 484 41 222 327
498 206 128 379 229 170 171 228 299 21 64 222 161 384 117
268 224 383 293 12 36 54 64 109 363 269 396 199 161 469 377
10 174 4 388 255 174 58 482 324 430 397 45 443 281 13 210
356 248 355 367 135 260 282 243 474 402 139 172 414 459 400
423 485 256 162 91 281 72 424 104 1 321 1 443 453 13 5 308
260 359 26 395 118 159 137 91 61 127 115 474 438 366 249
422 121 410 364 401 333 287 5 334 459 357 128 411 369 132
70 129 342 447 375 459 106 11 402 166 490 368 491 427 233
239 200 206 1 63 458 333 349 462 18 308 170 498 70 39 129
140 19 471 86 393 281 43 255 34 208 244 401 51 22 486 289
221 191 141 135 148 326 484 462 195 143 131 192 64 21 321
55 39 143 493 431 423 35 186 309 95 281 209 145 303 46 285
23 88 426 158 236 103 493 49 297 487 31 341 50 52 13 457 90
155 449 373 429 483 58 89 77 338 298 221 140 195 6 15 283
283 24 370 237 16 418 385 354 300 225 255 351 237 211 293
243 159 165 24 142 222 112 70 411 261 291 403 456 148 417
90 282 440 459 18 307 376 402 12 175 479 266 378 67 477 170
162 135 334
500 13
37 128 407 500 198 317 261 340 71 68 487 487 9 268 278 467
137 436 194 38 447 220 368 213 97 435 41 118 96 175 451 132
303 209 131 352 26 243 191 448 162 177 435 170 296 64 136
432 352 181 321 298 401 189 362 497 475 402 115 70 429 417
53 83 126 183 434 3 278 124 450 291 152 236 461 447 300 448
230 3 129 50 300 29 90 14 377 64 267 491 485 195 260 37 277
385 72 210 239 201 185 40 491 336 276 451 282 427 251 363
429 379 265 80 259 354 445 135 270 212 478 106 258 237 143
35 473 66 96 211 266 133 250 108 468 377 59 102 303 161 464
83 39 228 15 297 434 459 431 55 170 408 160 428 496 302 314
468 219 261 30 336 393 132 444 213 8 354 314 163 14 129 97
52 357 111 200 142 70 482 196 91 390 355 370 237 9 183 205
227 296 234 415 188 365 358 252 225 211 65 239 224 194 335
127 402 446 326 395 367 307 90 457 48 296 327 285 304 361
341 383 156 74 149 196 291 6 447 367 68 364 105 291 409 291
269 162 236 94 56 102 252 145 411 152 292 89 436 448 449

```
276 182 457 201 330 152 343 335 450 61 402 165 165 44 73
456 312 234 43 257 141 497 360 137 259 11 429 347 298 376
295 425 57 251 126 386 254 320 72 56 381 325 220 397 368
145 204 31 230 247 287 371 95 146 359 353 9 287 199 306 14
345 231 422 448 208 159 201 379 230 108 259 54 180 8 273
176 211 303 405 309 441 127 403 87 486 107 447 124 305 252
138 2 334 411 449 393 70 1 272 11 109 382 57 288 389 329
463 452 484 19 260 424 346 15 362 183 473 308 158 130 412
95 131 245 58 79 138 127 79 261 129 187 142 185 326 383 366
140 334 349 359 445 124 56 311 486 90 284 145 247 413 56
394 43 301 451 121 290 429 51 50 57 238 43 242 415 425 459
55 110 159 265 55 282 320 217 119 261 500 264 7 264 319 400
306 471 202 278 112 130 329 13 187 418 56 280 184 480 90
238 90 248 354 496 381 25 212 500 285 212 115 144 327 285
43 133 108 97 262 219 226 442 232 264 211 287 395 395 118
336 484 59 83 338 54 316 214 118 315 499 181 281 142 7 65
36 13 24 132
```

Sample Output

```
13
13
13
13
13
```

Example 5

Sample Input

```
1
8 10
11 12 13 14 15 16 17 3
```

Sample Output

```
9
```

Explanation

The first seven buttons are useless since they immediately overheat the sword. The best we can do is hit the last button three times for a heat total of 9.

6. Carbon Sequestration Station

(150 pts)

Description

Oganesson Dynamics have blueprints for a carbon sequestration station that will pull carbon out of the atmosphere until it has been reduced to a target level.

Every day a station can pull a configurable number of units of carbon out of the atmosphere. Unfortunately, the amount removed must always be set to a power of two. So, it could pull 1, 2, 4, 8, 16, or 32 units in a day (or more!), but NOT exactly 5 or 20 units.

Given a starting number of units (N) and a goal number (G), can you figure out the fewest number of days it would take to reach the target?

Input Format

- The first line will contain a value *T* indicating the number of planets (the number of test cases).
- The next *T* lines will each contain two integers *N*, indicating a number of units of carbon in the atmosphere, and *G*, the goal number of units for when the station is finished.

Constraints

- $1 \le T \le 200{,}000$
- $1 \le G \le N \le 2^{64} - 1$

Output Format

N lines, each indicating the number of days it would take to reduce the carbon of a given planet down to the target level.

Example 0

Sample Input

```
1
12 1
```

Sample Output

```
3
```

Explanation

There is a single planet with 12 units of carbon on it, which needs to be dropped to 1. Three days are needed for the station to reduce it:

- Day one reduces the carbon by 8 units, since 8 (2^3) is the largest power of two less than 12. This leaves four units of carbon in the atmosphere.
- Four is a power of two, but one unit of carbon must remain, so the second day reduces it by 2 (2^1).
- The third and final day reduces it by 1 (2^0) to the target of 1.

Example 1

Sample Input

```
3
21 13
16 0
35 18
```

Sample Output

```
1
1
2
```

Example 2

Sample Input

```
20
2 1
2 0
7 4
8 4
12 7
11 5
10 3
14 6
20 11
21 11
18 7
12 1
35 22
31 17
37 22
18 2
47 30
38 20
21 2
59 39
```

Sample Output

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

3
3
4
1
2
2
3
2

Example 3

Sample Input

```
1
5514762744452200995 990399488
```

Sample Output

```
32
```

Example 4

Sample Input

```
5
6703734870638684097 0
7597026131563511682 2604620160
131746072624578522 373163008
6959712971461184279 0
12572864058699034273 4261406720
```

Sample Output

```
29
30
37
32
32
```

Example 5

Sample Input

```
20
1 1
2 1
3 1
4 1
5 1
6 1
7 1
8 1
9 1
10 1
11 1
12 1
13 1
14 1
15 1
16 1
17 1
18 1
19 1
20 1
```

Sample Output

```
0
1
1
2
1
2
2
3
1
2
2
3
```

2
3
3
4
1
2
2
3

7. Plug-and-Play AI

(200 pts)

Note: Only efficient solutions won't timeout.

Description

There are many types and sizes of smart bricks; all of them will automatically link together when touching. Given N types of bricks where each brick i has length L_i, how many ways are there to lay them in a line with total length L_{total}?

Input Format

- The first line of the input provides T, the total number of test cases.
- The next T pairs of lines describe a single test case:
  - The first line in a test case privides the number of brick types (N) and the total target length (L_{total}).
  - The second line has N values, each indicating the length of each available brick.

Constraints

- $1 \le T \le 10$
- $1 \le N \le 50$
- $1 \le L_{total} \le 10^6$
- $1 \le L_i \le 10^6$

Output Format

T lines, one per test case, each indicating the number of combinations possible to make a line of that total length. Some of the values are quite large, so mod your answer by 1000000009.

Example 0

Sample Input

```
2
2 5
1 2
2 7
1 3
```

Sample Output

```
8
9
```

Example 1

Sample Input

```
6
4 2
1 2 3 4
4 4
1 2 3 4
4 6
1 2 3 4
4 8
1 2 3 4
4 10
1 2 3 4
4 12
1 2 3 4
```

Sample Output

```
2
8
29
108
401
1490
```

Example 2

Sample Input

```
5
4 16
1 2 4 8
4 32
1 2 4 8
4 64
1 2 4 8
4 128
1 2 4 8
4 256
1 2 4 8
```

Sample Output

```
5271
47292950
289604230
163774346
867355878
```

Example 3

Sample Input

```
4
5 5
1 1 1 1 1
4 10
1 1 2 2
4 5
1 1 1 1
4 10
2 2 2 2
```

Sample Output

```
3125
18272
1024
1024
```

Example 4

Sample Input

```
5
10 10000
1 17 91 101 336 775 1001 2345 2468 7777
4 10000
3 8 63 3968
10 10000
2 4 8 16 32 64 128 256 512 1024
10 10000
1 3 9 27 81 243 729 2187 6561 19683
10 10000
2 4 10 28 81 243 729 2187 6562 19684
```

Sample Output

```
151999650
54679324
425079984
959307745
945717398
```

8. Hats! (Extra Credit)

(50 bonus pts)

Description

N of your best robot friends have been invited to participate in a game of 'Hats!'. Can you write the instructions (a program) for each robot to follow that guarantees victory?

In every game of 'Hats!', there are *N* players and *N* possible hat colors. At the beginning of the game, each player has a randomly colored hat (one of the known *N* possible colors) placed on their head. Players can see the hat colors of other players, but CANNOT see their own hat color. Further, no communication is allowed during a game of 'Hats!'. To win, each player must write down which color they think their hat is, and if at least one player is correct, everyone wins. If no one correctly guesses their hat color, everyone loses.

NOTE: not all possible hat colors may be assigned, and multiple players may have duplicate hat colors.

Write a program that, if executed by each of your robot friends, will guarantee they win as a group (i.e., at least one guesses their hat color).

For this problem, your program will be evaluated differently than in previous problems. For every test case, we will run your program *N* times, once for each robot. Each time, we will provide the following inputs: *N* (the number of possible colors and the number of players), *P* (the current player's ID), the set of possible colors, and the hat colors the current player sees. You will pass a test case if, for at least one of the players, your program outputs the correct hat color.

Credit for this problem is all or nothing. You must pass all tests to receive credit.

Input Format

- The first line contains two space-separated integers, *N* and *P*.
- The second line contains a space separated list of strings where each string is a possible hat color.
- The third line contains a space separated list of strings where each string represents the hat colors the current player sees.

Constraints

- $1 \leq N,P \leq 865$

Output Format

Output the (string) color written down by the current player.

Example 0

In this example, there are five players (robots) and five possible hat colors. Each robot's program (i.e., the program you submit) is run independently and given as input N (the number of hat colors and number of total players), P (the current player's ID), and the current player's view of everyone else's hats.

In this example, the true hat color assignment is as follows:

- Robot 0: orange_peel
- Robot 1: orange_peel
- Robot 2: orange_peel
- Robot 3: gainsboro
- Robot 4: yellow

Robot 0

```
5 0
gainsboro orange_peel fawn yellow white_smoke
orange_peel orange_peel gainsboro yellow
```

Robot 1

```
5 1
gainsboro orange_peel fawn yellow white_smoke
orange_peel orange_peel gainsboro yellow
```

Robot 2

```
5 2
gainsboro orange_peel fawn yellow white_smoke
orange_peel orange_peel gainsboro yellow
```

Robot 3

```
5 3
gainsboro orange_peel fawn yellow white_smoke
orange_peel orange_peel orange_peel yellow
```

Robot 4

```
5 4
gainsboro orange_peel fawn yellow white_smoke
```

```
orange_peel orange_peel orange_peel gainsboro
```

Explanation

In this example, your program would be run five times, once for each robot. Each time your program is run, output the robot's guess. In this example, your program would be correct if at least one of the following was true:

- Robot 0 output 'orange_peel'
- Robot 1 output 'orange_peel'
- Robot 2 output 'orange_peel'
- Robot 3 output 'gainsboro'
- Robot 4 output 'yellow'

9. Assembling the Super Computer (Extra Credit)

(180 bonus pts)

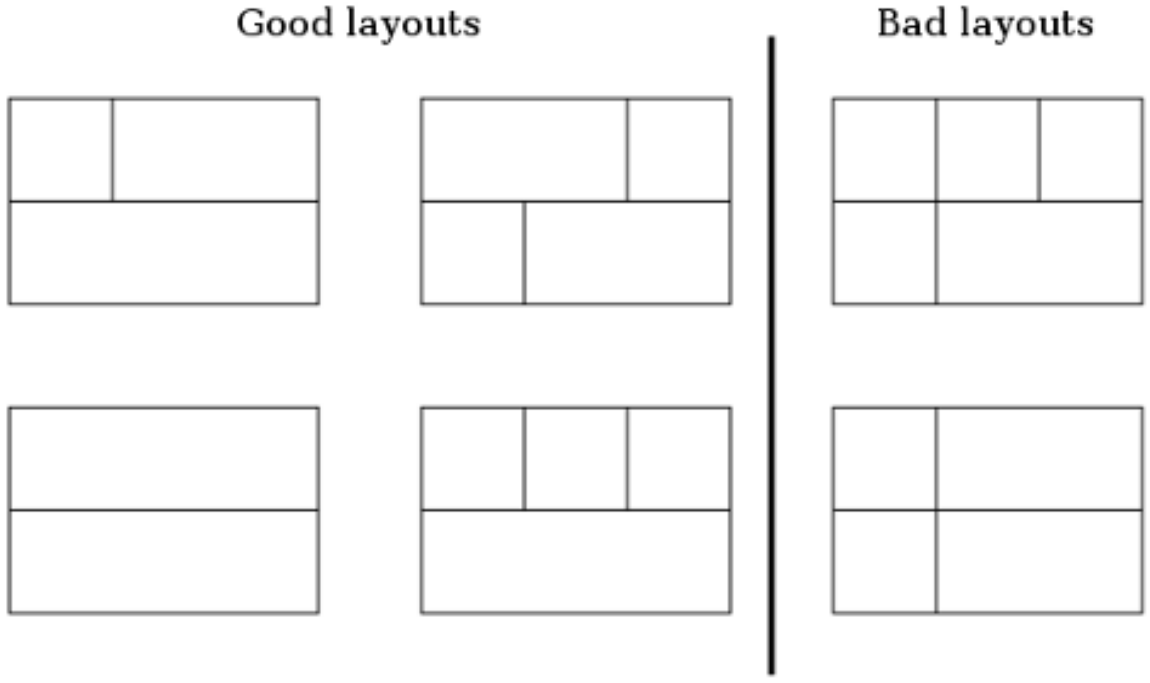
Description

Oganesson Dynamics have design a new type of modular supercomputer. Each of the computer's modules are able to stack vertically together. All of the modules are rectangular cubes of the same height and depth (1 unit long each), but their widths come in 4 lengths (1, 2, 3, and 4 units long). Each module has input ports on its bottom and output ports on its top. Thus, in order for there to be connectivity between the entire supercomputer, there must NOT be any vertical split that goes through every row in the same position (doing so would partition the computer).

Oganesson Dynamics needs to know how many possible configurations of the supercomputer are possible given a particular set of dimensions (width and height).

Important things to remember:

1. The assembled computer must be built horizontally, meaning that pieces are laid horizontally, not vertically (i.e., input ports always face up and output ports always face down).
2. The computer can't have any gaps (i.e., the entire computer must be filled with modules).
3. The computer can't be vertically partitioned. There cannot be a seam that runs from the top of the computer to the bottom: see the following diagram for some examples of an assembled computer that is two units tall and three unts wide. (note, this diagram does not show all legal configurations, just some of them).



Examples:

- If the required dimensionality for the computer is 1 unit tall and 1 unit wide, there is only one possible configuration as only 1 module can fit.
- If computer must be 2 units tall by 2 units wide, there are 3 possible configurations:
  - (1) a 2-long module on top of a 2-long module
  - (2) a 2-long module with 2 1-long modules on top
  - (3) 2 1-long modules with a 2-long module atop them
  - NOTE: 2 1-long modules atop 2 1-long modules isn't valid because there would be a vertical seam through the middle.
- If the computer must be 3 units tall by 2 units wide, there are 7 configurations: each row can be either a 2-long module or 2 1-long modules. So there are 2 x 2 x 2 possible options. However, 1 configuration (where all of the module are 1-long) results in a vertical seam that goes all the way through (like the previous example).

As the number of configurations possible becomes quite large for large computers, return your result as the number of configurations mod($10^9 + 7$).

Input Format

- The first line contains T, the number of test cases.
- Each test comprises two values separated by a space:
  - The first value, H, represents the height of the supercomputer.
  - The second value, L, represents the length of the supercomputer.

Constraints

- $1 \leq T \leq 10$
- $1 \leq H, L \leq 1000$

Output Format

Output T lines, the number of legal configurations that can be produced for each test case modulus 1000000007.

Example 0

Sample Input

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

Sample Output

```
1
3
7
49
```

Example 1

Sample Input

```
7
10 1
5 2
2 5
6 4
4 5
100 100
500 500
```

Sample Output

```
1
31
50
250047
35714
928745576
798212178
```