Data Structure and Algorithm, Spring 2023

Homework 1

Blue Correction Date: 03/19/2023 06:30

P4-P6 Release and Red Correction Date: 03/18/2023 13:00

Due: 13:00:00, Tuesday, April 11, 2023

TA E-mail: dsa_ta@csie.ntu.edu.tw

Rules and Instructions

- Any form of cheating, lying, or plagiarism will not be tolerated. Students can get zero scores and/or fail the class and/or be kicked out of school and/or receive other punishments for those kinds of misconducts.
- In Homework 1, the problem set contains a special Problem 0 (see below) and 5 other problems. The set is divided into two parts, the non-programming part (Problems 0, 1, 2, 3) and the programming part (Problems 4, 5).
- For problems in the non-programming part, you should combine your solutions in *one* PDF file. Your file should generally be legible with a white/light background—using white/light texts on a dark/black background is prohibited. Your solution must be as simple as possible. At the TAs' discretion, solutions which are too complicated can be penalized or even regarded as incorrect. If you would like to use any theorem which is not mentioned in the classes, please include its proof in your solution.
- The PDF file for the non-programming part should be submitted to Gradescope as instructed, and you should use Gradescope to tag the pages that correspond to each subproblem to facilitate the TAs' grading. Failure to tagging the correct pages of the subproblem can cost you a 20% penalty.
- For the programming part, you should have visited the *DSA Judge* (https://dsa2023.csie.org/) and familiarized yourself with how to submit your code via the judge system in Homework 0.
- For problems in the programming part, you should write your code in C programming language, and then submit the code via the judge system. In each day, you can submit up to 5 times for each problem. To encourage you to start early, we allow 10 times of submissions per day in the first week **(from 2023/03/19 to 2023/03/27)**. The judge system will compile your code with

```
gcc main.c -static -O2 -std=c11
```

- For debugging in the programming part, we strongly suggest you produce your own test-cases with a program, use tools like `gdb` or compile with flag `-fsanitize=address` can greatly help you solve issues such as illegal memory usage. For `gdb`, you are strongly encouraged to use compile flag `-g` to preserve symbols after compiling. Note: For students use IDE(VScode, DevC++...) you can modify compile command in their settings/preference section. Below are some examples:
  - `gcc main.c -O2 -std=c11 -fsanitize=address # Works on Unix-like OS`
  - `gcc main.c -O2 -std=c11 -g # use it with gdb`
- Discussions on course materials and homework solutions are encouraged. But you should write the final solutions alone and understand them fully. Books, notes, and Internet resources can be consulted, but not copied from.
- Since everyone needs to write the final solutions alone, there is **absolutely no need to lend your homework solutions and/or source codes to your classmates at any time.** In order to maximize the level of fairness in this class, lending and borrowing homework solutions are both regarded as dishonest behaviors and will be punished according to the honesty policy.
- The score of the part that is submitted after the deadline will get some penalties according to the following rule:

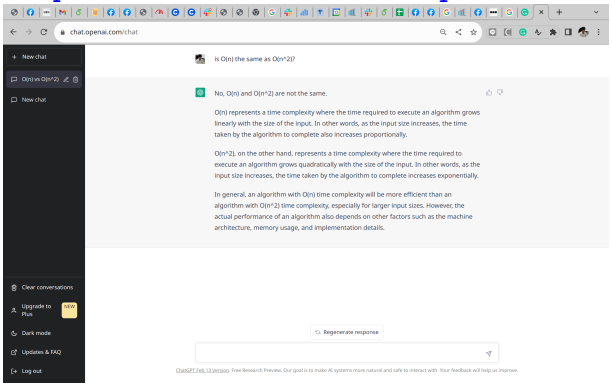
$$Late\ Score = \max\left(\left(\frac{5 \times 86400 - Delay\ Time(sec.)}{5 \times 86400}\right) \times Original\ Score, 0\right)$$

- If you have questions about HW1, please go to the discord channel and discuss (*strongly preferred*, which will provide everyone a more interactive learning experience). If you really need an email answer, please follow the rules outlined below to get a fast response:
  - The subject should contain two tags, `"[hw1]"` and `"[Px]"`, specifying the problem where you have questions. For example, `"[hw1][P5] Is k in subproblem 5 an integer"`. Adding these tags allows the TAs to track the status of each email and to provide faster responses to you.
  - If you want to provide your code segments to us as part of your question, please upload it to Gist or similar platforms and provide the link. Please remember to protect your uploaded materials with proper permission settings. Screenshots or code segments directly included in the email are discouraged and may not be reviewed.

Problem 0 - Proper References (0 pts)

For each problem below, please specify the references (the Internet URL you consulted with or the classmates/friends you discussed with) in your PDF submitted to Gradescope. If you have used chatGPT or similar tools, please provide a quick snapshot of your interaction with the tools. *You do not need to list the TAs/instructors.* If you finished any problem all by yourself (or just with the help of TAs/instructors), just say "all by myself." While we did not allocate any points on this problem, failure to complete this problem can lead to penalty (i.e. negative points). Examples are

- Problem 1: Alice (B86506002), Bob (B86506054), and https://stackoverflow.com/questions/982388/



- Problem 2: all by myself
- ...

Listing the references in this problem does *not* mean you could copy from them. We cannot stress this enough: *you should always write the final solutions alone and understand them fully.*

Problem 1 - What if you became a DSA TA? (100 pts)

Suppose that f and g are asymptotically non-negative functions defined on $\mathbb{N}$. Recall the following asymptotic notations:

- $f(n) = O(g(n))$ if and only if there exist positive numbers c and n_0 such that

$$f(n) \leq cg(n) \text{ for all } n \geq n_0.$$

- $f(n) = \Omega(g(n))$ if and only if there exist positive numbers c and n_0 such that

$$f(n) \geq cg(n) \text{ for all } n \geq n_0.$$

- $f(n) = \Theta(g(n))$ if and only if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.

Note: Any theorems that are proven in our class can be directly used. But please state the logical derivations from those theorems to what you want to prove clearly. For instance, in our class, we proved that

Theorem 1. If $\lim_{n\to\infty} \frac{f(n)}{g(n)} = c$ with **A POSITIVE** c, then $f(n) = \Theta(g(n))$.

But we did **NOT** prove that

Conjecture 1. If $\lim_{n\to\infty} \frac{f(n)}{g(n)} = c$ with **ANY** c, then $f(n) = O(g(n))$.

So if you want to use the conjecture, you need to prove it first.

When you woke up in the morning, you received an urgent message from NTU COOL. One of the DSA TAs, Casper Wang, had to leave immediately since he received a job offer from Google. The instructors thus needed someone to take over the position, and you were offered the job due to your intelligence and coding skills. Now, you are facing 414 homework submissions from all the students in the DSA class, and you need to check if their proofs in the Analysis Tools homework are correct.

1. (15 pts) In one of the homework problems, you want the students to prove that the time complexity of some algorithm is $O(n^2)$. One of the students correctly proved that the time complexity of the algorithm is $O(n^3)$. Should ze get the full mark? Why or why not? Give a formal proof to justify your decision.

   In particular, if the answer is yes, prove that if $f(n) = O(n^3)$, then $f(n) = O(n^2)$. Otherwise, prove that there is some $f(n) = O(n^3)$ that does not belong to $O(n^2)$.

2. (15 pts) A student came to your TA hour and ask you why ze got a TLE (Time Limit Exceed) when ze tries to use "binary search" to solve the problem. The student provided the following pseudo code for zir solution, and claimed to feed the algorithm with a sorted array A with $l = 1$ and $r = n$, where n is the length of the array. What is the time complexity of the algorithm, expressed with the big-O notation? Prove your answer.

```
TLE-BINARY-SEARCH(A, key, l, r)
1  while l ≤ r
2      m = l
3      if A[m] is equal to key
4          return m
5      else if A[m] > key
6          r = m − 1
7      else if A[m] < key
8          l = m + 1
9  return NIL
```

When continuing to grade the homework submissions, you noticed that some students wrote down some propositions without any explanations (which is highly discouraged!). Now you need to prove or disprove the proposition to decide whether to give the student some partial mark (if the proposition is correct) or not.

3. (15 pts) Prove or disprove that $f(n) = \Theta(n^2) \Rightarrow \lim_{n\to\infty} \frac{f(n)}{n^2} = c$, where $c > 0$ is a constant.

4. (15 pts) Prove or disprove that $\lg n = O(\sqrt{n})$

Several of the issues that you faced may require more in-depth analysis and thinking, similar to the sub-problems outlined below. We recommend that you not only reflect on them by yourself, but also discuss them with your classmates (as legally allowed by our course policy) and/or attend TA office hours to discuss with other TAs. Best of luck!

5. (20 pts) Prove or disprove that $\Sigma_{i=1}^{n} i^n = O(n^n)$

6. (20 pts) One student provided the following solution to a problem. The solution seems mistaken. Point out the mistakes and write down the correct proof.

Problem 1126. Assume that f, g are positive functions and $|\lg(f(n)) - \lg(g(n))| = O(1)$, prove that $f(n) = O(g(n))$.

B11902777's Solution.

$|\lg(f(n)) - \lg(g(n))| = O(1) \Rightarrow |\lg(f(n)) - \lg(g(n))| = c$, for all $n > n_0$, where $n_0, c > 0$ are constants.

$|\lg(f(n)) - \lg(g(n))| = c \Rightarrow \lg(\frac{f(n)}{g(n)}) = c \Rightarrow \frac{f(n)}{g(n)} = 2^c$

Take $c' = \frac{1}{2^c}$, we have $f(n) \leq c'g(n)$ for all $n > n_0$, QED.

Problem 2 - DSA Judge (100 pts)

For all sub-problems listed below, unless stated otherwise, please ensure that your answers consist of the following components:

(a) Please describe your algorithm using pseudocode and provide a **brief explanation** of why it works. Your pseudocode for each sub-problem should be concise and easy to read, **consisting of no more than 30 lines**. Please note that overly complex or lengthy pseudocode may result in point deductions.

(b) Analyze the time complexity and *extra-space* complexity of your algorithm. Please note that if your algorithm is not efficient in terms of time or space, you may not receive full credit for your solution.

DSA is one of the most popular courses at National Taiwan University. There are over 400 students in the course. To manage the large volume of programming homework submissions, the course TAs have implemented the DSA Judge, an automated grading system.

1. (20 pts) One day, there should be n submissions on the DSA Judge, each with its own unique submission ID $1 \sim n$. However, DSA TAs have just discovered that exactly 2 submissions are missing from the backend database, which stores the IDs in a sorted array of length $n-2$. Please design an algorithm to find out the IDs of the two missing submissions in $O(\log n)$-time. To protect the submission data, the array is read-only and cannot be modified.

   For this sub-problem, you do not need to analyze the *extra-space* complexity (yeah!!). Please simply analyze the time complexity.

   Hint: If you find it too challenging to identify both missing elements simultaneously, you may want to consider finding them one by one instead.

2. (30 pts) Brian, one of the DSA TAs, is validating testcases for a programming problem. A testcase for this problem consists of n positive integers in sorted order, where n is an even number. It is a valid testcase if and only if the n elements can be paired up into $n/2$ pairs, each consisting of one element a and its double $2a$.

   For example, $(1, 2, 2, 3, 4, 5, 6, 10)$ is a valid testcase since you can pair up all the elements by

   $$[(1, 2), (2, 4), (3, 6), (5, 10)],$$

   but $(1, 2, 2, 4, 7, 8, 10, 14)$ is not a valid testcase.

   Please design an algorithm to assist Brian in automatically verifying the validity of the test cases before he uploads them to DSA Judge. You can assume that the given testcases are already sorted and are of even length. Your algorithm should have a time complexity of $O(n)$ and an extra-space complexity of $O(1)$. Since Brian has already backed up all the test cases, feel free to utilize the space of the original array.

3. (20 pts) The DSA TAs have recently updated the judge system's database. As a result, the DSA Judge now uses linked lists to store submission data, making it easier to conduct insertion. In the database, there are two *singly* linked lists, one for accepted codes and the other for unaccepted codes, both sorted by submission ID, and all of the submission IDs are positive integers. The evil Hsuan-Tien asks the TAs to show a dashboard because he enjoys watching how hard the students are working, and the dashboard needs to list all (accepted and unaccepted) submissions, reversely sorted by submission ID. Please design an algorithm to merge the two lists into a single list, reversely sorted by submission ID, with a time complexity of $O(n)$ and an extra-space complexity of $O(1)$. Please return the head of the merged linked list.

   Note: with the requirement on $O(1)$ extra-space complexity, you cannot afford to create new nodes to solve this sub-problem, and need to re-use the nodes that have been allocated for the original linked lists.

4. (30 pts) The database with one *singly* linked list, sorted by reverse submission ID, has been running smoothly for a while now. However, one day the database encountered errors due to an electric outage in the CSIE building. As a result, some of the submission IDs a_i were changed to their negation, i.e., $-a_i$, after the incident.

   For example,

   before: $18 \to 15 \to 11 \to 8 \to 7 \to 4 \to 2 \to 1$

   after: $18 \to -15 \to 11 \to 8 \to 7 \to -4 \to 2 \to 1$

   That is, the linked list is not reversely sorted, causing many of the algorithms that were designed for the reversely sorted linked list to fail. Please design an algorithm to fix this issue. That is, produce

   $18 \to 11 \to 8 \to 7 \to 2 \to 1 \to -4 \to -15$

   from the example above. Your algorithm should take a *singly* linked list that is reversely sorted by the absolute value of the data, and return the head of the reversely sorted linked list by the actual value. The algorithm should run in $O(n)$-time and $O(1)$-extra space.

Problem 3 - Stack / Queue (100 pts)

Ground Work

In this subsection, the answers are directly related to what we have taught in class.

1. (15 pts) Convert the infix expression below into a postfix expression, with the usual precedence of () over $\times/$ over $+-$ and left the association. Briefly explain any *human algorithm* (which is effective for human brains but does not necessarily need to be effective for computers) that you use for conversion. No, we will not accept running the stack-based conversion algorithm in your brain. :-)

$$5 + 3 \times 4 - (8 \times (2 + 3) - (1 + 9 \times 6)/5)$$

2. (15 pts) Evaluate the outcome of the following postfix expression, where each number token is a single-digit positive integer.

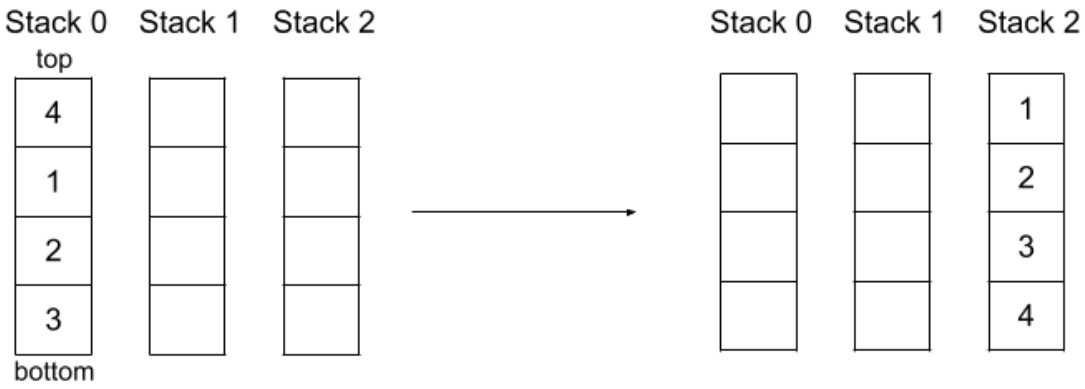
$$527 \times +634 - \times 486 \times +4/ - -$$

Please treat / as the usual division in math. Briefly explain any *human algorithm* that you use for evaluation. No, we will not accept converting this postfix into infix in your brain. :-)

Stack

There are three stacks identified by their indices, namely 0, 1, and 2. At the beginning, stacks 1 and 2 are empty, while stack 0 contains some integers as its elements. Your objective is to move all the elements from stack 0 to stack 2 and sort them in ascending order. The only permitted operation is to move (pop) the topmost element from one stack to another stack with a greater index, and you cannot use any space other than the three stacks. For instance, you can pop the element from stack 0 and push it to stack 2. But you cannot pop an element from stack 1 and push it to stack 0. You can access the element $s[i]$ at index i of the stack s in $O(1)$, for $i = 0$ (bottom), $1, \ldots, n-1$ (top).

3. (10 pts) Please check the figure below. Suppose stack 0 contains $[3, 2, 1, 4]$ with 4 being the topmost element. Can you move all elements to stack 2 and sort them to the ascending order $[4, 3, 2, 1]$, viewed from top to bottom with the operation above? If so, briefly explain the steps; if not, briefly explain the reasons.



4. (25 pts) Design an algorithm to determine whether it is possible to move all the elements of a size-n stack 0 to stack 2 and sort it in ascending order from the operation above, with $O(1)$ additional spaces. The algorithm should have a time complexity of $O(n^5)$. You can choose to either use pseudocode with no more than 30 lines, or use English/Chinese to explain the core design of your algorithm. You do *not* need to prove the correctness nor the time complexity (yeah!!).

Queue

As in Problem 1, some of the next sub-problems may require more in-depth analysis and thinking. We recommend that you not only reflect on them by yourself, but also discuss them with your classmates (as legally allowed by our course policy) and/or attend TA office hours to discuss with other TAs. Best of luck!

A total of n bikes have been parked in front of the Dertian Hall, forming a row where **each bike is closer to the one on its right than to the one on its left**, except for the first and the last bikes. The first bike has no bike on its left, while the last bike has no bike on its right. For simplicity, you can treat each bike to be of width 0.

A group of m students wants to park their bikes in the row, one at a time, between two neighboring bikes. To avoid collisions, each student aims to park their bike as far as possible from the existing bikes by parking it in the middle of (i.e. not closer to the left nor the right of) the largest distance between two neighboring bikes.

For example, suppose the initial positions of existing bikes are: $[2, 8, 10]$, the student will park zir bike at position 5, which is of the maximum distance of 3 to the nearest bikes.

5. (10 pts) Suppose the initial positions of existing bikes are: $[0, 10, 15.5, 17]$. Three students come and park their bikes one by one. What will be the distance between the third student's bike to the nearest bike? Briefly explain your computation steps.

6. (25 pts) Design an algorithm to compute the distance between the bike parked by the m-th student and the nearest bike with time and space complexity both $O(n + m)$. Use clear pseudocode with no more than 30 lines to describe your algorithm. Then, prove both the time and space complexity of your algorithm.

Programming Part

Problem 4 - King of DSA [Basic Version] (100 pts)

Time Limit: 1 s
Memory Limit: 2048 KB

Problem Description

Problem 4 described the basic (i.e. simplified) version of the game. Compared with the bonus version in Problem 5, this version is simpler by not having a "scoring system" in the game.

Little Cucumber is currently developing "Digital Savage Arena" (DSA), a video game inspired by KoH (King of the Hill), with plans to release the alpha version on April 11th to collect feedback from players. However, he has been facing difficulties with the arena system for weeks due to his lack of study in the data structures and algorithms class. Since you have successfully completed the notorious Homework 0 in the class, can you assist him with solving the final piece of the puzzle?

In DSA, N players compete in an arena to earn prestigious honor. The game consists of N rounds. The i-th player, which carries an attack power of a_i, enters the arena at the beginning of the i-th round. As soon as the player enters the arena, ze kills all other players that hold a lower attack power.

At the end of each round, the player with the highest attack power is awarded the "King of DSA." In the event of a tie, the player who entered the arena earlier will be awarded the title. However, due to memory limitations, the size of the arena must be limited. If the number of surviving players in the arena exceeds M, a revolution will occur, resulting in the King of DSA being overthrown and killed by the other M players.

In short, round i of the game executes the following three steps orderly:

1. Player i enters the arena.
2. Player i kills all other players that hold a lower attack power.
3. Check if a revolution occurs. If so, the King of DSA is killed.

Given the attack power of the N players, can you help Little Cucumber calculate the players that are killed in each round?

Input

The first line contains two integers N and M—the number of rounds (players) and the arena's maximum capacity. The second line contains N integers $a_1, a_2, \ldots, a_N$, which is the attack power of each player, separated by space.

Output

The output should consist of $N+1$ lines. The first N lines should begin with the string "`Round` i`:`", where i is the round number. Each of the N lines should include the player(s) that are killed in that round. The last line should begin with the string "`Final:`", followed by the players who are still alive at the end of the game. When outputting the players (killed ones in the first lines and the surviving ones in the last line), print out the players' **indices** (the round that each of them entered the arena), sorted in **descending** order and separated by space.

Constraints

- $1 \leq N \leq 10^6$
- $1 \leq M \leq 10^3$
- $1 \leq a_i \leq 10^9$

Subtask 1 (10 pts)

- $1 \leq N \leq 10^3$

Subtask 2 (30 pts)

- It is guaranteed that a revolution will never occur.

Subtask 3 (60 pts)

- No other constraints.

Hints

- Please note the memory limit, which requires you to use $O(M)$ memory.
- If you receive a Runtime Error (RE) on the DSA judge, it is possible that you have exceeded the memory limit.
- Little Cucumber suggests you to carefully visualize how the game is played instead of staring at the numbers.

Sample Testcases

Sample Input 1

```
10 4
8 5 2 4 7 6 5 3 4 6
```

Sample Output 1

```
Round 1:
Round 2:
Round 3:
Round 4: 3
Round 5: 4 2
Round 6:
Round 7:
Round 8: 1
Round 9: 8
Round 10: 9 7
Final: 10 6 5
```

Sample Input 2

```
8 3
5 5 5 5 6 4 4 5
```

Sample Output 2

```
Round 1:
Round 2:
Round 3:
Round 4: 1
Round 5: 4 3 2
Round 6:
Round 7:
Round 8: 7 6
Final: 8 5
```

Problem 5 - King of DSA [Bonus Version] (20 bonus pts)

Time Limit: 1 s

Memory Limit: 2048 KB

Problem Description

Problem 5 described the bonus (i.e. hard) version of the game. Compared with the basic version in Problem 4, this version is harder by having a "scoring system" in the game. We allocate **only 20** bonus (extra) points for this problem. That is, your score for HW1 can be as high as 520. Somehow calling the problem "bonus" means we do not expect everyone to try to solve the problem—you are recommended to solve other problems first before visiting this problem.

Our story is exactly the same as Problem 4, except that Little Cucumber decides to add a scoring system to the game. After confirming the players that are alive in each round, the player with the highest attack power (i.e. King of DSA) earns M points. The player with the second-highest attack power earns $M - 1$ points, and so on. Only the surviving players in the arena are given points. Ties are broken by the order that the players entered the arena—i.e. the player that enters earlier earns more points.

In short, round i of the game executes the following three steps orderly:

1. Player i enters the arena.
2. Player i kills all other players that hold a lower attack power.
3. Check if a revolution occurs. If so, the King of DSA is killed.
4. **Assign points to the surviving players.**

Given the attack power of the N players, can you help Little Cucumber calculate not only the players that are killed in each round, but also their scores (accumulated points)?

Input

The first line contains two integers N and M—the number of rounds (players) and the arena's maximum capacity. M also means the points given to King of DSA in each round. The second line contains N integers $a_1, a_2, \ldots, a_N$, which is the attack power of each player, separated by space.

Output

The output should consist of $N+1$ lines. The first N lines should begin with the string "`Round` i`:`", where i is the round number. Each of the N lines should include the player(s) that are

killed in that round. The last line should begin with the string "`Final:`", followed by the players who are still alive at the end of the game.

When outputting the players (killed ones in the first lines and the surviving ones in the last line), print out the players, separated by space, in the **descending** order of their **indices** (the round that each of them entered the arena). Each player should be outputted as a string that consists of zir index and total score, separated by a comma, like "`index,score`".

Constraints

- $1 \leq N \leq 10^6$
- $1 \leq M \leq 10^3$
- $1 \leq a_i \leq 10^9$

Subtask 1 (1 bonus pt)

- $1 \leq N \leq 10^3$

Subtask 2 (9 bonus pts)

- It is guaranteed that revolution will never occur.

Subtask 3 (10 bonus pts)

- No other constraints.

Sample Testcases

Sample Input 1

```
10 4
8 5 2 4 7 6 5 3 4 6
```

Sample Output 1

```
Round 1:
Round 2:
Round 3:
Round 4: 3,2
Round 5: 4,2 2,9
Round 6:
Round 7:
Round 8: 1,28
Round 9: 8,1
Round 10: 9,1 7,5
Final: 10,2 6,13 5,21
```

Sample Input 2

```
8 3
5 5 5 5 6 4 4 5
```

Sample Output 2

```
Round 1:
Round 2:
Round 3:
Round 4: 1,9
Round 5: 4,1 3,3 2,7
Round 6:
Round 7:
Round 8: 7,1 6,4
Final: 8,2 5,12
```

Problem 6 - Cycling Teams and Bakeries (100 pts)

Problem Description

Cycling teams resemble swarms of locusts,
swiftly moving through and consuming everything in their path.

Seven city consists of a street grid with n horizontal roads and m vertical roads. The horizontal roads are numbered by their position relative to the north, indexed by i, while the vertical roads are numbered by their position relative to the west, indexed by j. The distance between any two consecutive horizontal roads (i and $i+1$) is equal to the distance between any two consecutive vertical roads (j and $j+1$), for all $1 \leq i \leq n-1$ and $1 \leq j \leq m-1$. At every intersection of a horizontal road i and a vertical road j, there is an open bakery denoted by b_{ij}. Based on its Google Map reviews and ratings, each bakery is assigned a *unique* rank u_{ij} between 1 and mn. Furthermore, each bakery is initially stocked with a certain number of toasts, denoted by t_{ij}.

There are two training periods for cycling teams, and one racing period in between. The first training period is of T_1 days, the second training period is of T_2 days, and the racing period is for R days. On each day k within the training periods ($1 \leq k \leq T_1$ or $T_1+R+1 \leq k \leq T_1+R+T_2$), a cycling team with s_k members would be at an open bakery of rank r_k as the initial gathering bakery, and plans a tour of length at most l_k to other bakeries based on the following steps.

1. Select all open bakeries that are located on the same horizontal or vertical road as the team's current position but have not been visited on day k, excluding the gathering bakery at the current position. If no such open bakeries exist, the team ends its training and returns home.

2. Among the selected bakeries, choose the ones with the minimum distance to the gathering bakery.

3. Among the chosen bakeries, visit the one with the smallest rank.

4. Let each member of the team eat a piece of toast to restore energy at the bakery being visited. That is, decrease the number of toasts t_{ij} by s_k for the bakery. If the new $t_{ij} \leq 0$, the bakery being visited would then need to be closed.

5. If the number of bakeries that the team has visited does not exceed l_k, set the bakery being visited as the new gathering bakery, and go to step 1. Otherwise, the team ends its training and returns home.

On each day k within the racing period ($T_1 + 1 \leq k \leq T_1 + R$), the 80kings would choose an open bakery with rank r_k as the center of the racing track. The track is a square with a side length of $l_k - 1$, with an odd number l_k of intersections on each side. The many cycling teams would race through the streets at breakneck speeds, their powerful legs propelling them forward with such force that a mighty tornado formed in their wake. As they hurtled around the square track, the tornado wreaked havoc on the surrounding area, lifting and rotating bakeries within the square, including those on the racing track, by 180 degrees with tremendous force.

Given the plan on each day of the training and racing periods, output the final number of toasts of the bakeries.

Input Format

There are 2 integers n, m separated by a space in the first line.

Each of the next n lines contains m integers separated by spaces. The j-th integer of the i-th line is u_{ij} (the rank of b_{ij}).

Each of the next n lines contains m integers separated by spaces. The j-th integer of the i-th line is t_{ij} (the initial number of toasts of b_{ij}).

There are 3 integers T_1, R, T_2 separated by spaces in the next line, which denote the number of days for the first training period, the racing period, and the second training period, respectively.

Each of the next T_1 lines corresponds to each day in the first training period, and contains 3 integers r_k (the rank of the initial gathering bakery), l_k (the maximum riding length), and s_k (the size of the cycling team), separated by spaces.

Each of the next R lines corresponds to each day in the racing period, and contains 2 integers r_k (the rank of the center bakery) and l_k (the *odd* side length of the square track), separated by spaces.

Each of the next T_2 lines corresponds to each day in the second training period, and contains 3 integer r_k (the rank of the initial gathering bakery), l_k (the maximum riding length), and s_k (the size of the cycling team), separated by spaces.

Output Format

Output n lines, each line contains m integers separated by spaces. The j-th integer of the i-th line should be t_{ij}, which is the final number of toasts of b_{ij}, for any open bakery. If the bakery is closed (i.e. $t_{ij} \leq 0$), simply output 0.

Constraints

- $1 \leq n, m \leq 1000$
- $1 \leq t_{ij}, s_k \leq 10^9, \forall i, j, k$
- $1 \leq u_{ij}, r_k \leq nm, \forall i, j, k$. All u_{ij} are distinct.
- $l_k \geq 0$
- $T_1 + R + T_2 \leq 10^6$
- $\sum_k l_k \leq 10^6$

Subtasks

Subtask 1 (10 pts)

- $\sum l_k \leq 10^3$

Subtask 2 (15 pts)

- $R = 0$

Subtask 3 (15 pts)

- $T_1 = T_2 = 0$

Subtask 4 (60 pts)

- no other constraints

Hints

- Rotating is quite time-consuming. How can you reduce its time complexity?
- More specifically, can you reverse a list in $O(1)$ time?

Sample Cases

Sample Input 1

```
5 5
8 9 10 11 17
7 15 16 12 18
6 14 4 13 19
5 3 2 1 20
25 24 23 22 21
49 1 7 14 7
2 14 49 7 7
7 49 14 2 7
7 7 1 49 14
14 7 7 7 49
1 1 1
1 11 4
4 3
1 10 9
```

Sample Output 1

```
45 0 0 1 7
0 49 0 0 7
0 0 5 40 7
0 0 40 5 14
14 7 7 7 49
```