Data Structure and Algorithm, Spring 2024

Homework 4

Due: 13:00:00, Tuesday, June 11, 2024

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.
- 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 (including chatGPT) 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.
- In Homework 4, the problem set contains a special Problem 0 (see below) and 4 other problems. The set is divided into two parts, the non-programming part (Problems 0, 1, 2) and the programming part (Problems 3, 4).
- 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, including Problem 0 (Proper references), 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 in Problem 1 and Problem 2 can cost you a 20% penalty. And failure to tagging the correct pages of Problem 0 can cost you 5 points for each problem without reference.

- For problems in the programming part, you should write your code in C programming language, and then submit the code via the *DSA Judge*:

  https://dsa2024.csie.org/.

  Each day, you can submit up to 5 times for each problem. 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 programs and/or use tools like `gdb` or compile with flag `-fsanitize=address` to help you solve issues like illegal memory usage. For `gdb`, you are strongly encouraged to use compile flag `-g` to preserve symbols after compiling.

  Note: For students who use IDE (VScode, DevC++...) you can modify the compile command in the settings/preference section. Below are some examples:

  - `gcc main.c -O2 -std=c11 -fsanitize=address`
    `# Works on Unix-like OS, but possibly not M1/M2`
  - `gcc main.c -O2 -std=c11 -g # use it with gdb`

- The score of the part that is submitted after the deadline will get some penalties according to the following rule:

$$\textit{Late Score} = \max\left(\left(\frac{1 \times 86400 - \textit{Delay Time(sec.)}}{1 \times 86400}\right) \times \textit{Original Score}, 0\right)$$

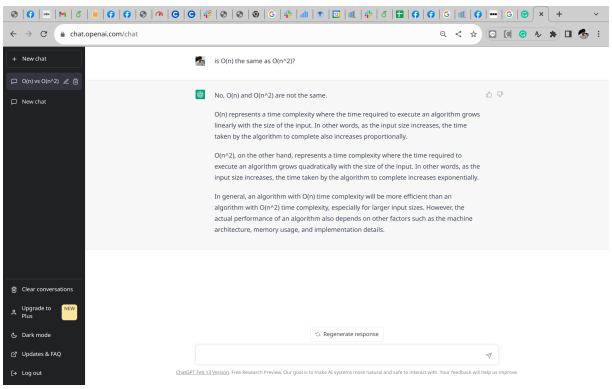
- If you have questions about HW3, please go to the Discord channel and discuss (*strongly preferred*, which will provide everyone with a more interactive learning experience). If you need an email answer, please follow the rules outlined below to get a fast response:

  - Please include the following tag in the subject of your email: `"[HW3.Px]"`, specifying the problem where you have questions. For example, `"[HW3.P2] Is k in subproblem 5 an integer?"` Adding the tag 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 **non-programming and programming 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. For P0, mark all the pages where you have listed references, whether it is on the same page or on multiple pages. 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 to this problem, failure to complete this problem can lead to penalty (i.e. negative points). Examples are

- Problem 1: Alice (B86506002), Bob (B86506054)
- Problem 2: all by myself
- Problem 3:

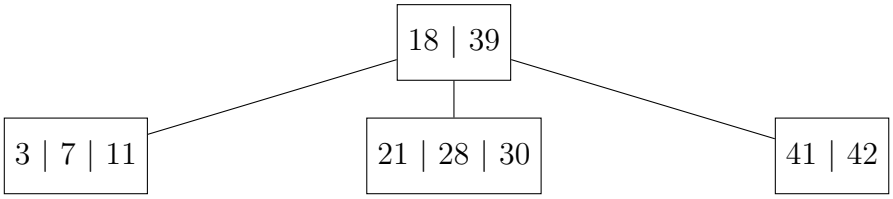


- Problem 4: https://stackoverflow.com/questions/982388/

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 - B-Tree (50 pts)

1. (12.5 pts) Given the following B tree:



Perform the following operations on the B-tree with minimum degree $t = 2$ in the figure in the given order:

(a) insert 19;

(b) insert 25;

(c) insert 23;

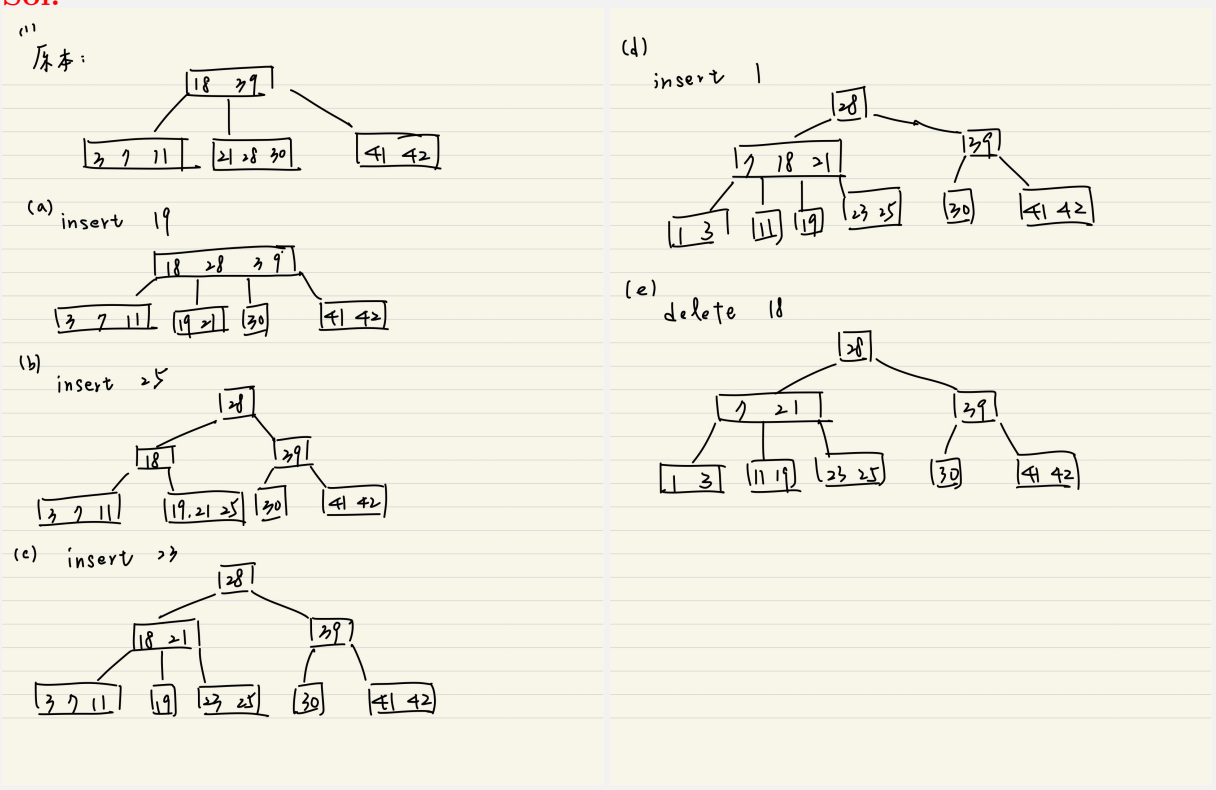
(d) insert 1;

(e) delete 18.

Note that the implementation of the operations should follow exactly what is introduced in the textbook and in the lecture[1]. Show the content of the B-tree **after each operation**. No additional explanation is needed.

[1]There exist many different versions of B-tree implementations. Obviously different implementations could produce different results given the same operations. Here we will only allow the result produced by the implementation given in the textbook.

Sol:



2. (7.5 pts) Please explain the purpose of the pseudo code in the specified lines, and discuss the impact of removing these lines. You may use some examples to help with explanation. See below for the pseudo code.

   (a) (5 points) B-Tree-Insert-Nonfull() lines 13-16

   (b) (2.5 points) B-Tree-Split-Child() lines 11-12

```
B-Tree-Insert-Nonfull(x, k)
 1  i = x.n
 2  if x.leaf
 3       while i ≥ 1 and k < x.key_i
 4            x.key_{i+1} = x.key_i
 5            i = i − 1
 6       x.key_{i+1} = k
 7       x.n = x.n + 1
 8       Disk-Write(x)
 9  else while i ≥ 1 and k < x.key_i
10            i = i − 1
11       i = i + 1
12       Disk-Read(x.c_i)
13       if x.c_i.n == 2t − 1
14            B-Tree-Split-Child(x, i)
15            if k > x.key_i
16                 i = i + 1
17       B-Tree-Insert-Nonfull(x.c_i, k)
```

B-TREE-SPLIT-CHILD(x, i)

```
1  y = x.c_i
2  z =ALLOCATE-NODE()
3  z.leaf = y.leaf
4  z.n = t − 1
5  for j = 1 to t − 1
6        z.key_j = y.key_{j+t}
7  if not y.leaf
8        for j = 1 to t
9              z.c_j = y.c_{j+t}
10 y.n = t − 1
11 for j = x.n + 1 downto i + 1
12       x.c_{j+1} = x.c_j
13 x.c_{i+1} = z
14 for j = x.n downto i
15       x.key_{j+1} = x.key_j
16 x.key_i = y.key_t
17 x.n = x.n + 1
18 DISK-WRITE(y)
19 DISK-WRITE(z)
20 DISK-WRITE(x)
```

Sol:

(a) Purpose: Ensure that inserting a key-value into the tree does not violate the degree rule.

Explanation/example: Given the insertion of any value into a node that is a leaf already having $2t - 1$ keys, lacking this line of code will result in an error.

(b) Purpose: Since the keys will be shifted to the right next, all children must first be shifted to the right here to ensure that there is space for the keys split from the child, and to ensure there is extra space for an additional subtree to place the child.

Explanation/example: For any operation that requires a B-tree split-child, lacking this line of code will result in the split tree not performing as expected.

3. (12.5 pts) In a typical computer system, accessing the *memory* (both read and write) is much faster than accessing the *hard disk*, even when the latter is a solid-state drive.

However, the capacity of the memory is usually significantly smaller than that of the hard disk. Therefore, the data is typically stored on the *hard disk*, and only moved to *memory* when frequent processing is expected.

Assuming all B-tree data is initially stored on the hard disk. And assume that disk hardware allows you to choose the size of a disk block arbitrarily, but the time it takes to read a disk block is $a + b \cdot t$, where a and b are given constants and t is the minimum degree of a B-tree using blocks of the selected size.

(a) (7.5 points) Determine a t value such that the **Disk-Read time** during the search process in the B-tree is minimized, given that $a = 1$ milliseconds , $b = 64$ microseconds and the number of keys in the tree $n = 1024$. Here, as an approximation, you can assume that all nodes have the same maximum number of keys for a chosen t value, although in reality some nodes might have less keys due to the fixed value of $n = 1024$.

In addition, describe how you determine this t value.

(Hint: What is the tree height for a given t value?)

(b) (5 points) The default method to determine whether a key exists within a B-tree node is sequential search. Assume that it takes k nanoseconds to perform such a sequential search when there are k keys in the node. Since the keys stored within a B-tree node are in non-decreasing order, it seems that we can use binary search to speed up this operation.

Can we significantly reduce the running time of the search operation on a B-tree, if we implement the above binary search method? Please briefly explain your answer. Assume that the Disk-Read time still follows what is specified in the previous question (a= 1 millisecond, b=64 microseconds). Please briefly explain your answer.

For students who want to know more: In the system, the methods for managing files and virtual memory mostly adopt a B-tree-like structure. Therefore, designing an optimal block size to accelerate data reading speed is very important. However, due to hardware limitations, we cannot allow users to decide the block size on their own, as suggested by the problem statement. However, there are still many methods available to reduce disk I/O time. You can learn more in the upcoming SP and OS courses.

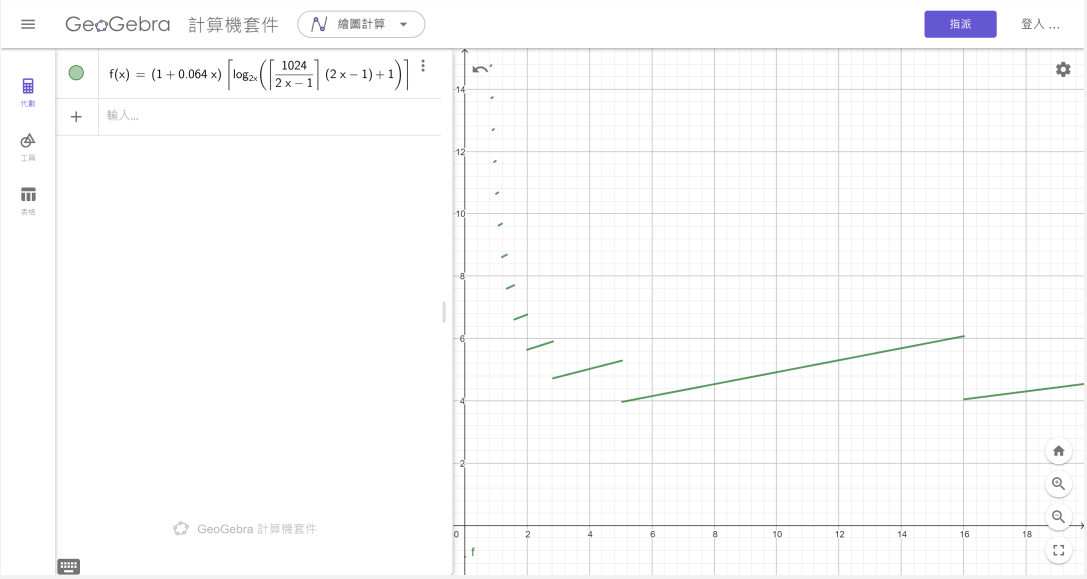
Sol:

(a) Based on the problem statement, for a tree with minimum degree t and a total of n keys, the height h of the tree is given by $h = \lceil log_{2t}((\lceil \frac{n}{2t-1} \rceil)(2t-1)+1) \rceil$.

(You can derive h by solving $1 + (2t)^1 + (2t)^2 + ... + (2t)^{h-2} < \lceil \frac{n}{2t-1} \rceil \leq 1 + (2t)^1 + (2t)^2 + ... + (2t)^{h-1}$)

The worst-case search time requires h disk reads.

Substituting the values, we can determine that the required function is $f(t) = h(a + bt) = \lceil log_{2t}((\lceil \frac{n}{2t-1} \rceil)(2t-1)+1) \rceil (1 + 0.064t)$, and our goal is to find the positive integer t that minimizes $f(t)$.



According to calculations assisted by Geogebra, we can determine that the best search time is approximately achieved when t is 6.

(b) Continuing from 3-(a), we know that the maximum time spent on search time is $(\lceil log_{2t}((\lceil \frac{n}{2t-1} \rceil)(2t-1)+1) \rceil)(2t-1)$ nanoseconds. However, through binary search, we can reduce this time to $(\lceil log_{2t}((\lceil \frac{n}{2t-1} \rceil)(2t-1)+1) \rceil) log_2(2t-1)$ nanoseconds. But we find that just the time spent on disk read/write operations requires at least 4 milliseconds, so implementing binary search to improve performance by a few nanoseconds is not significant in terms of overall performance.

4. (17.5 pts) The B-tree implementation introduced in the lecture uses *preemptive split / merge*. In this case, a check is performed *before* inserting or deleting a key to determine whether the maximum or minimum degree rule could be violated after the insertion or deletion. If so, a split or a merge is performed *before* the key insertion or deletion, respectively.

In contrast, *non-preemptive split / merge* performs the same check *after* a key has been inserted or deleted. Only when the check discovers that the the maximum or minimum degree rule is violated, a split or a merge is performed.

(a) (7.5 points) Put together the pseudo code of B-Tree-Insert-2(T, k), which inserts key k into an existing B-tree T but performs *non-preemptive split*.

You can freely use the two functions Split-Child(x, i) and Split-Root(T) without providing any explanation in your solution. These perform what is illustrated in Figure 1.1 and with the details as follows:

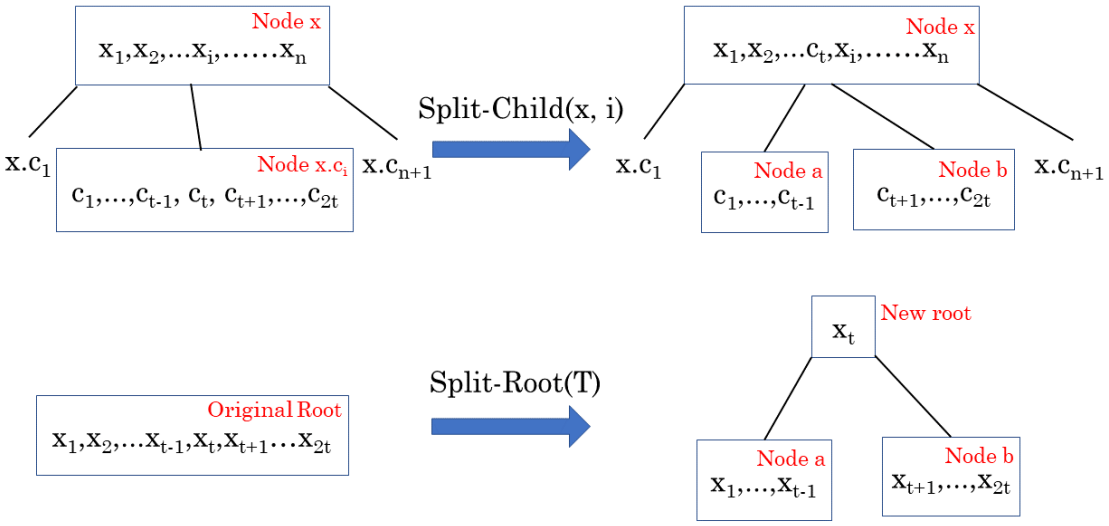


Figure 1.1: Illustration of the adjustments of the B-tree performed by Split-Child(x, i) and Split-Root(T)

- Split-Child(x, i): Split x's child node $x.c_i$ (which have exactly $2t$ keys) into two neighboring child nodes a (which have $t-1$ keys) and b (which have t keys), and insert the t-th key in original $x.c_i$ node into the i-th position of the list of the keys of x. Note that after inserting the t-th key, node x itself might have $2t$ keys after the operation.
- Split-Root(T): Split root node $T.root$ (which have exactly $2t$ keys) into two nodes a (which have $t-1$ keys) and b (which have t keys). Then, create a new node with only one key x_t, the t-th key in the original root node. Make this new node as the root node, and have a and b as its first and second child nodes, respectively.

It is not necessary to have line-by-line explanation of your pseudo code in the solution. However, feel free to provide brief explanation such that the grading TA can have better understanding of your pseudo code.

(b) (10 points) Start with an empty B-tree with minimum degree $t = 2$ and find a sequence of **insertions** (less than ten insertions) that results in a different B-tree structure when using preemptive split / merge and and when using non-preemptive split / merge. Using less than one page, please provide an analysis of why the final B-trees are different.

Sol:

(a)

B-TREE-INSERT-2(T, k)

```
 1  x = T.root
 2  //Let S be a new stack
 3  S.PUSH((x,1))
 4  while x.leaf == false
 5        i = x.n
 6        while i ≥ 1 and k < x.key_i
 7              i = i − 1
 8        i = i + 1
 9        DISK-READ(x.c_i)
10        S.PUSH(x,i)
11        x = x.c_i
12  i = x.n
13  while i ≥ 1 and k < x.key_i
14        x.key_{i+1} = x.key_i
15        i = i − 1
16  x.key_{i+1} = k
17  x.n = x.n + 1
18  DISK-WRITE(x)
19  while x.n == 2t
20        if S is empty
21              SPLIT-ROOT(T)
22              return
23        else
24              (x, i) = S.POP()
25              DISK-READ(x)
26              SPLIT-CHILD(x, i)
```

(b) You can verify the correctness of the students' answers through this website via maximum degree = 4:
https://www.cs.usfca.edu/ galles/visualization/BTree.html
Sample Answer: [2, 4, 6, 1, 3, 5, 7]
Sample Explanation: In preemptive insert method, it isn't guarantee that we will choose t-th key to be a new root, but non-preemptive insertion will.

Problem 2 - Sorting in Linear Time (50 pts)

1. (7.5 points) Algorithmic complexity attack!

   Assume that an online system supports sorting a sequence of n numbers ranging in $(0, 1]$ (excluding 0 and including 1). After some investigation, you found that it uses the following *bucket sort* algorithm BUCKET-SORT, modified from the one in the textbook. The function INSERTION-SORT is utilized in BUCKET-SORT. It takes an input singly linked list L, and sorts it with *insertion sort* to produce the list of node values in ascending order.

```
INSERTION-SORT(L)
 1  insertNode = L.head.next
 2  L.head.next = NIL
 3  while insertNode ≠ NIL
 4      nextNode = insertNode.next
 5      if insertNode.value < L.head.value
 6          insertNode.next = L.head
 7          L.head = insertNode
 8      else
 9          curNode = L.head
10          while curNode.next ≠ NIL and insertNode.value > curNode.next.value
11              curNode = curNode.next
12          insertNode.next = curNode.next
13          curNode.next = insertNode
14      insertNode = nextNode
15  return L.head
```

```
BUCKET-SORT(A)
1  n = A.length
2  Let B[1..n] be a new array
3  for i = 1 to n
4      Create an empty list and store it in B[i]
5  for i = 1 to n
6      Insert A[i] into the head of list B[⌈n · A[i]⌉]
7  for i = 1 to n
8      Sort list B[i] with INSERTION-SORT(B[i])
9  Concatenate the lists B[1], B[2], ..., B[n] together in order
```

Please construct an array A of n numbers ranging in $(0, 1]$ such that worst-case $O(n^2)$ time complexity can be achieved. Briefly explain your idea of construction.

In the real world, this is called the algorithmic complexity attack. If an input to deterministically induce worst-case running time can be constructed, then it can be used to launch an attack to consume the system resources, effectively an denial-of-service attack.

Solution.
The idea is to make all numbers fall in the same bucket. Therefore, one may choose $a_1 = \frac{n}{n^2}$, $a_2 = \frac{n-1}{n^2}, \ldots, a_n = \frac{1}{n^2}$. In this case, all numbers go to the bucket $B[1]$ and the insertion sort for a originally sorted list would take $O(n^2)$ time.

2. (17.5 points) Radix sort revisited

Given two strings of lowercase English letters, S and T. We define the $\leq$ relation *considering the alphabetical order* as follows. $S \leq T$ holds, if either one of the following two conditions are satisfied:

- $|S| \leq |T|$ and $S = T[1 : |S|]$.
- There exists $1 \leq q \leq \min(|S|, |T|)$ such that $S[1 : q-1] = T[1 : q-1]$ and $S[q] < T[q]$.

We can also define the $=$ relation *considering the alphabetical order* as follows. $S = T$ holds if both $S \leq T$ and $T \leq S$ hold.

With the $\leq$ and $=$ relations defined above, we can now sort strings *in alphabetical order*.

Now, consider sorting a set of N non-empty strings $\{S_1, S_2, \ldots, S_N\}$ formed by lowercase English letters in alphabetically ascending order. The total lengths of the N strings, L, is given by

$$L = \sum_{i=1}^{N} |S_i| \quad .$$

A student tried to use the following algorithm to sort the strings in $O(L)$-time and $O(L)$-space. You, as an acting TA, need to help him to check the correctness during the TA hour.

RADIX-SORT(S)

```
1  Let A[1..L] be a new array
2  for i = 1 to L
3      Create an empty list and store it in A[i]
4  for i = 1 to N
5      Insert S_i to A[|S_i|]
6  for i = 1 to L
7      if A[i].size ≤ 1 //  A[i].size refers to the number of elements in list A[i]
8          continue
9      for j = 1 to i
10         Perform stable counting sort on A[i]
           using the (i − j + 1)-th letter of each string as key
11 Concatenate the lists A[1], ..., A[L] in order to produce the final sorted result
```

(a) (5 points) Before checking the correctness, please first check whether the given RADIX-SORT function above satisfies the $O(L)$ time complexity.

(b) (2.5 points) Sadly, you found that the student's algorithm is actually incorrect. Please give a counter-example to show this.

(c) (10 points) Design an algorithm that runs in $O(L)$-time and $O(L)$-space to solve this problem. Write down the pseudo code of the algorithm and analyze the time and space complexity of your algorithm.

Solution.

(a) The first two loops have at most L iterations, with each running in constant time. The for loop starting from line 8 performs radix sort on each $A[i]$ and the time complexity for sorting $A[i]$ is $O(i \cdot A[i].size)$ if $A[i].size > 1$. Therefore, the total running time of the for loop is also $O(L)$ by summing up the length of each strings.

(b) Let $N = 2$ with $S_1 = b$, $S_2 = aa$. The correct output should be $[aa, b]$ but the algorithm outputs $[b, aa]$.

(c) Denote $S_{i,j}$ as the j-th character of S_i.

```
RADIX-SORT(S)
 1  let A[1..N] and C[1..L] be new arrays
 2  for i = 1 to L
 3      C[i] = 0
 4  for i = 1 to N
 5      C[|S_i|] = C[|S_i|] + 1
 6  for i = 2 to L
 7      C[i] = C[i − 1] + C[i]
 8
 9  for i = N to 1
10      A[C[|S_i|]] = i
11      C[|S_i|] = C[|S_i|] − 1
12
13  for ℓ = L to 1
14      perform regular counting sort on {A[i] | (C[ℓ] + 1) ≤ i ≤ N}
                                          using S_{A[i],ℓ} as key
15  for i = 1 to N
16      S'[i] = S[A[i]]
17  return S'
```

3. (10 points) Matrix addition

Let $A_1, A_2, \ldots, A_k$ be non-zero n-by-n square matrices. That is, each of these matrices have at least one non-zero element. The matrices are represented in the sparse format, where all non-zero elements in a matrix are represented by a linked list of (*value*, *index*)

pairs. The index of an element in the r-th row and the c-th column (both are 0-based) is $r \cdot n + c$.

For example, suppose A is a 3-by-3 matrix given by

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 3 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

which can be represented by the linked list

$$(1, 1) \to (1, 8) \to (3, 3).$$

Note that the indices of the pairs are not necessarily sorted. Assume $M = \sum_{i=1}^{k} \mathbf{nnz}(A_i)$, where $\mathbf{nnz}(A)$ denotes the number of non-zero elements in matrix A. Please design an algorithm that computes the sparse representation of the sum of the given k matrices

$$A_1 + A_2 + \cdots + A_k$$

using $O(\max(n, M))$-time and $O(n)$-extra-space. You can directly modify the lists corresponding to A_i's to output the results to avoid using extra space. Write down the pseudo code of the algorithm and analyze the time and space complexity for your algorithm.

Solution. Idea: perform radix sort using base-n representation of the indices. Then merge the nodes with the same indices. When the sum of $\mathbf{nnz}(A_i)$'s is smaller than n, the time complexity is $O(n)$. In other case, the time complexity would be $O(\sum_{i=1}^{k} \mathbf{nnz}(A_i))$.

4. (15 points) Radix sort with different bases.

In this problem, a student generates n integer numbers between $[0, 2^{31} - 1]$ and would like to compare the running time of sorting these numbers when using radix sort with different bases.

(a) (4 points) What is the time complexity for performing radix sort on n d-bits integers using base-b representation (i.e., each digit of the representation ranges from 0 to $b - 1$)?

(b) (6 points) Please help the student to finish the code snippet that computes the l-th digit (the least significant digit is regarded as the first digit) of the integer num using base-b representation.

GET-DIGIT(num, l, b)

```
1  if b == 2
2        return [A]
3  elseif b == 8
4        return [B]
5  else b == 10
6        return [C]
```

Please complete the three missing lines. For the base-2 and base-8 cases (line 2 and line 4), you have to use shift operations. Also assume that you have the value of 10^i by accessing a pre-computed table `power_of_10[i]`.

(c) (5 points) The following table shows the execution time (in seconds) to perform radix sort on the n integers in the files when representing the integers using base-2, base-8 and base-10.

	$n = 262144$	$n = 524288$	$n = 1048576$
base-2	0.0596	0.1181	0.2321
base-8	0.0223	0.0450	0.0885
base-10	0.0240	0.0490	0.0990

i. Compare the execution time for base-2 and base-8 implementation. What might be the reason for the outcome?

ii. Compare the execution time for base-8 and base-10 implementation. What might be the reason for the outcome?

(a) $O\left(\frac{d}{\log_2 b}(n+b)\right)$

(b) A. `(num & (1 << l)) >> l`

B. `(num & (((long long)1 << (3*(l+1))) - 1)) >> (3*l)`

C. `(num / power_of_10[l]) % 10`

(c) By the complexity derived in (a), we know that the needed time is inversely proportional to $\log_2 b$; therefore, the running time for using base-2 representation is roughly three times the running time for using base-8 representation.

On the other hand, since $\log_2 10$ is larger than $\log_2 8$, it seems that the running time for base-10 representation would be shorter. However, the implementation for base-8 representation uses a faster shift operation to extract each digit so it is more efficient.

Problem 3 - Dimensional Space Atlas (100 pts)

Problem Description

Story

Cross-State Interurban Express (CSIE) stands as the premier railway company within the North Taurus Union (NTU). Its flagship, the Dimensional Space Atlas (DSA), a colossal train composed of numerous carriages, ferries substantial cargo across space every day.

In the interview for the position of train guard, you are tasked to demonstrate your ability to handle goods within the carriages.

In particular, before departure, the train consists of N carriages and the total values of the goods in the carriages are a_1, a_2, $\cdots$, a_N, respectively.

After departure, T incidents happen sequentially. The incident is one of the following:

1. The k-th carriage is unloaded, i.e., the carriage is *removed* from the train.
2. A new carriage of total value a is attached *after* the k-th carriage.
3. You are tasked with calculating the total value of the goods within the carriages ranging between the l-th and the r-th carriage.

Can you answer the questions so that you are hired and get rich?

You are initially given a sequence of N integers, $a_1, a_2, \cdots, a_N$. Then, perform T operations on it, each of which belongs to one of the following types. For the description below, consider that the sequence right before performing the operation is $a_1, a_2, \cdots, a_M$.

1. **Deletion**: remove the k-th element from the sequence, $1 \leq k \leq M$.
2. **Insertion**: insert an integer a between the k-th and the $(k+1)$-th elements, $1 \leq k \leq M-1$. If $k = 0$, insert a at the beginning of the sequence. If $k = M$, insert a at the end of the sequence.
3. **Query**: calculate the sum of the elements between the l-th and the r-th elements in the sequence, including the l-th and the r-th elements. $1 \leq l \leq r \leq M$.

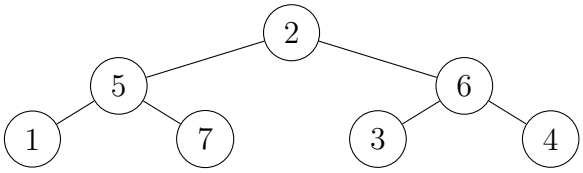
Hints for Solving the Problem

A self-balanced search tree, such as *red-black tree* or *treap* (to be introduced in the reference reading section below), can be utilized to solve this problem. Note that we *strongly recommend that you use treap* to solve this problem as we expect its implementation would be less complex and easier.

Below we will guide you through how to apply a self-balanced binary search tree to solve this problem.

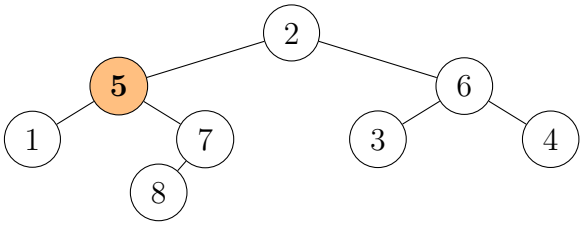
We can treat each element in the sequence as a node in a binary search tree, and its position in the sequence as the key of the node. For instance, the 3^{rd} in the sequence can be conceptualized as "node with the third smallest key" in the binary search tree.

Consider a sequence: $\{1, 5, 7, 2, 3, 6, 4\}$, a possible corresponding binary search tree would be:

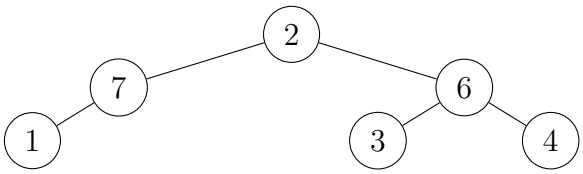


Different from the usual visualization scheme, here the number in the circle represents the original number in the sequence, not the key. You can perform an in-order traversal of the tree to verify that the keys of the nodes indeed follow the positions of the numbers in the sequence.

To add a new element between the k-th and the $(k+1)$-th elements in the sequence, find the corresponding node of the k-th element and then insert a new node that becomes its successor, i.e., the new node is "just greater" than this node. For example, if we insert 8 between the 2^{nd} position (5) and the 3^{rd} position (7) of the sequence $\{1, 5, 7, 2, 3, 6, 4\}$, the resulting tree may be:



Similarly, to delete the k-th element, simply find the corresponding node of the k-th element and delete it. For example, if we want to delete the 2^{nd} element (5) from the sequence $\{1, 5, 7, 2, 3, 6, 4\}$, the resulting tree may be:

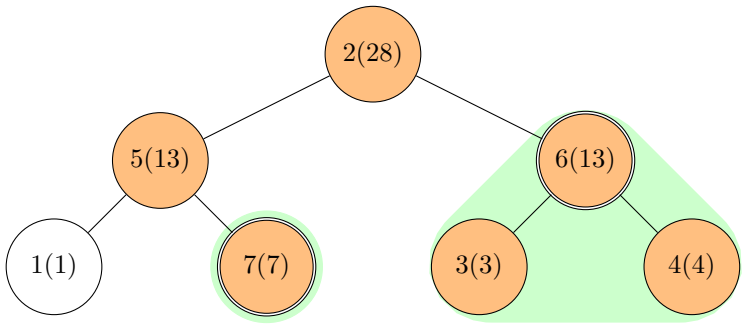


To calculate the sum of the elements from the l-th to the r-th position in the sequence, we will first need to determine the sum of the values in the nodes of a sub-tree and store this sum value in the root node of the sub-tree. This can be performed as we construct the tree. Note that the key here is the position index of the number in the original sequence. All nodes that we visit during insertion would have the sum value *increased by the value of the inserting node* (note: not the key). With this approach, the sum value in each node always represents the sum of the values in all nodes within the sub-tree rooted by that node.

Then, we can calculate the sum of the elements from the l-th to the r-th position in the sequence, by implementing the following pseudo code.

```
CALCULATE-SUM(root, l, r)
 1  if RANGE-OF-SUB-TREE(root) == [l, r]
 2      return root.sum
 3  sum = 0
 4  k = POSITION(root)
 5  // POSITION(x) returns the position of x in the sequence
 6  if k ∈ [l, r]
 7      sum = sum + root.value
 8  if l < k
 9      sum = sum + CALCULATE-SUM(root.left, l, min(k − 1, r))
10  if r > k
11      sum = sum + CALCULATE-SUM(root.right, max(l, k + 1), r)
12  return sum
```

For example, below you can find an illustration of how to calculate the sum of the elements from the 2^{nd} element (5) to the 7^{th} element (4) in the sequence $\{1, 5, 7, 2, 3, 6, 4\}$. You can verify the correctness of the above pseudo code with this given example.



The sum is $2 + 5 + 7 + 13 = 27$.

Reference Reading

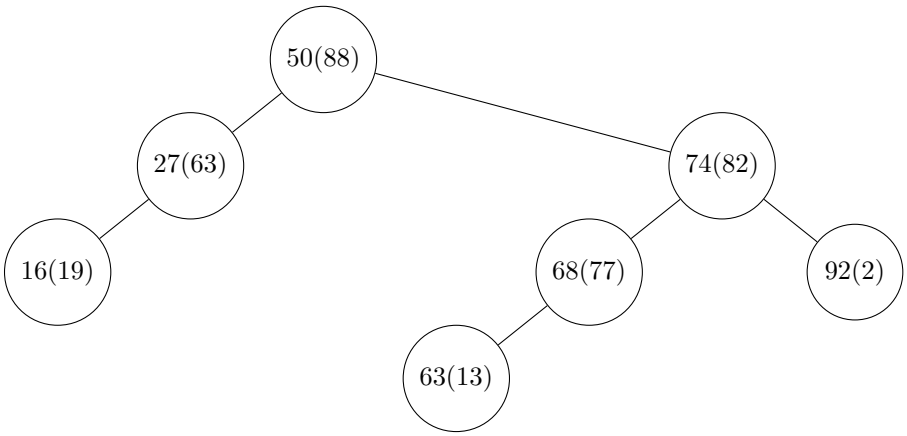
The following provides information about *Treap* that may be helpful to solve this problem.

If we insert a set of n items into a binary search tree, the resulting tree may be horribly unbalanced, leading to long search times. However, randomly built binary search trees tend to be balanced.

Property. A treap is a binary search tree with a modified way of ordering the nodes. As usual, each node x in the tree has a key value $x.key$. In addition, we assign $x.priority$, which is a random number chosen independently for each node. The nodes of the treap are ordered so that

1. The keys obey the binary-search-tree property, which implies:
   - If v is in the left sub-tree of u, then $v.key < u.key$.
   - If v is in the right sub-tree of u, then $v.key > u.key$.
2. The priorities obey the max-heap order property, which implies:
   - If v is a child of u, then $v.priority \leq u.priority$.

The data structure possesses the properties of both a binary search tree and a heap gives, and hence the name *"treap"*. Below is an example of how a treap looks like. The numbers in the circle represent the *key* (*priority*) of the node:



Suppose that we insert the nodes $x_1, x_2, \cdots, x_n$, with corresponding keys, into a treap. Then, the resulting treap is the tree that would have been formed if the nodes had been inserted into a normal binary search tree in the order given by their (randomly chosen) priorities, i.e., $x_i.priority > x_j.priority$ implies that we would insert x_i before x_j. Since the priority is randomly assigned, a treap simulates the process of building a binary search tree with all its elements inserted in a random order, resulting in an expected height of $\Theta(\log_2 n)$.

There are two primary implementations of treaps: rotation-based and merge-split. Here, we focus on the merge-split treap implementation, as it is easier to implement.

The main operations of merge-split treap are *merging* and *splitting*. To insert a new node with key k, we conceptualize it as a single-node treap. We first split the existing treap into two treaps such that one has keys smaller than k and the other has keys larger than k. Then these three treaps are sequentially merged. To delete a node, we split the treap twice to isolate the node to be deleted, and then merge the remaining two treaps.

Splitting. TREAP-SPLIT splits the original treap into two treaps, denoted as *left* and *right*, according to a given key k. Like the name suggests, it splits the tree such that all nodes in *left* have keys less than or equal to k and all nodes in *right* have keys greater than k.

We can implement this function recursively. Let $n.left$ denote the left child of a node n and $n.right$ denote its right child. If $n.key \leq k$, then both the n and its left sub-tree $n.left$ belong to *left*. On the other hand, the right sub-tree $n.right$ needs to be further split into two sub-trees, where one have keys smaller than or equal to k and the other have keys larger than k. Denote the former as *left*′ and the latter as *right*′. This can be achieved by a recursive call to split $n.right$ with k. Then, *left* would have *left*′ in addition to $n.left$ and n, and *right* would have *right*′. If $n.key > k$, similar actions are performed. See the pseudo code below for details.

TREAP-SPLIT($root, k$)

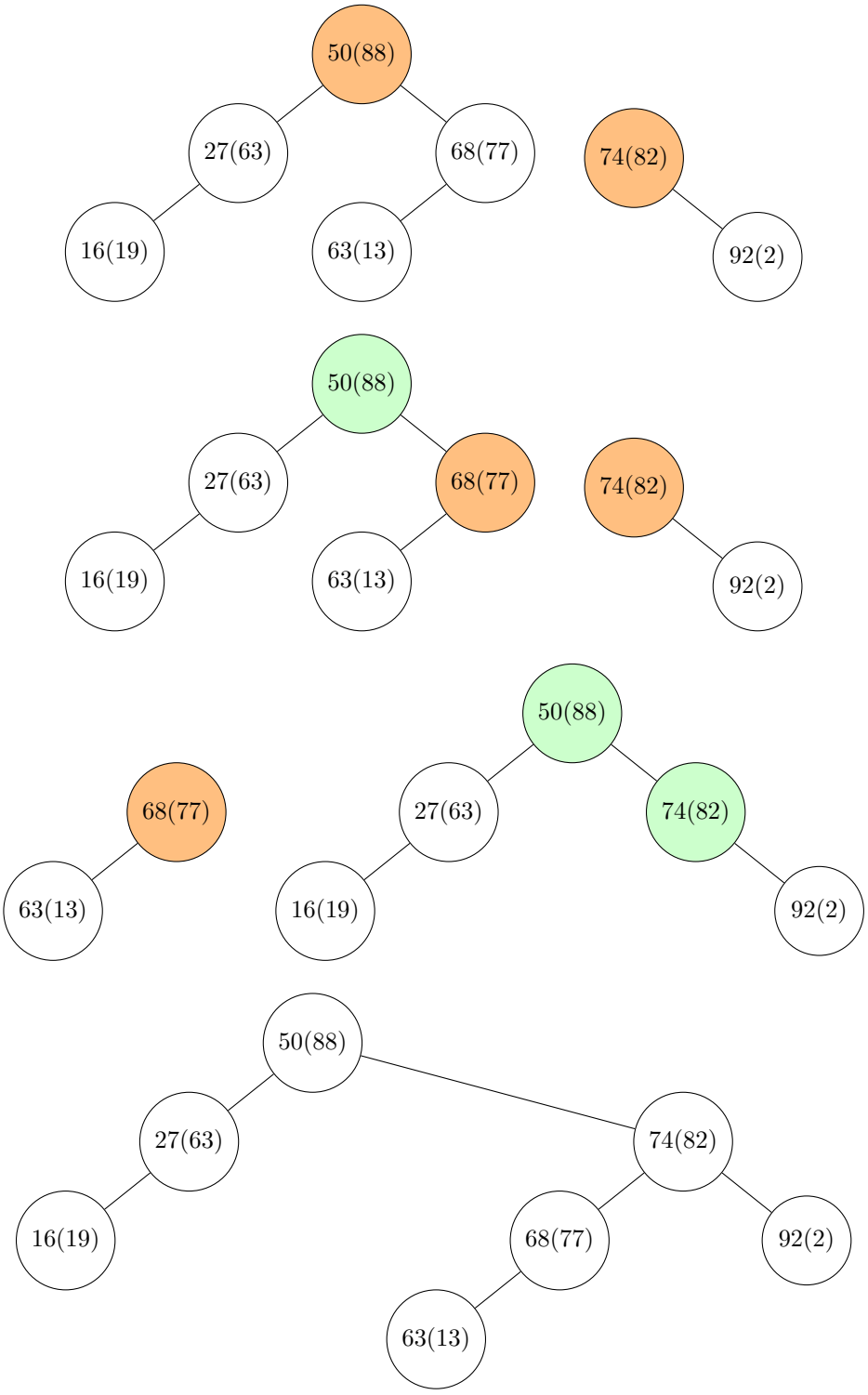
```
1  if root == NULL
2      return (NULL, NULL)
3  if root.key ≤ k
4      left = root
5      (root.right, right) = TREAP-SPLIT(root.right, k)
6  if root.key > k
7      right = root
8      (left, root.left) = TREAP-SPLIT(root.left, k)
9  root.sum = root.value + root.left.sum + root.right.sum
   (if root.left and root.right ≠ NULL)
10 return (left, right).
```

Merging. TREAP-MERGE merges two treaps *left* and *right* and returns a single treap that has the nodes of both treaps. *The prerequisite of this operation is that* $x.key \leq y.key, \forall\, x \in left, y \in right$. We compare the priorities of the root nodes of the two treaps, and select the one with higher priority as the root of the merged treap. Then, we recursively call the merge function to merge the treap with lower-priority root and one of the sub-trees of the treap with higher-priority root. See below for the pseudo code of the merge function for details.

TREAP-MERGE($left, right$)

```
1  if left == NULL
2      return right
3  if right == NULL
4      return left
5  if left.priority > right.priority
6      left.right = TREAP-MERGE(left.right, right)
7      left.sum = left.value + left.left.sum + left.right.sum
       (if left.left and left.right ≠ NULL)
8      return left
9  else
10     right.left = TREAP-MERGE(left, right.left)
11     right.sum = right.value + right.left.sum + right.right.sum
       (if right.left and right.right ≠ NULL)
12     return right
```

The graphs below demonstrate how to merge two treaps:



References:

1. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. Introduction to Algorithms, Third Edition (3rd. ed.). The MIT Press.
2. Treaps. USACO Guide. https://usaco.guide/adv/treaps?lang=cpp

Input

The first line contains two space-separated integers N and T. The second line contains N integers $a_1, a_2, \cdots, a_N$ representing the original sequence. Each of the following T lines contains two or three integers, depending on the operation:

- 1 k, indicating operation 1 is performed. Delete the k-th element from the sequence.
- 2 k a, indicating operation 2 is performed. Insert integer a between the k-th and the $(k+1)$-th elements.
- 3 l r, indicating operation 3 is performed. The sum of the elements between the l-th and the r-th elements in the sequence should be calculated.

Output

For each operation 3 specified in the input, output 1 integer in a line. The integer represents the sum.

Then output two lines after performing all T operations as follows.

1. Output one integer representing the number of elements in the array after T operations.
2. Output space-separated numbers representing final sequence after T operations.

Constraints

- $0 \leq N \leq 5 \cdot 10^5$
- $0 \leq T \leq 5 \cdot 10^5$
- $0 \leq a, a_1, a_2, \cdots, a_N \leq 10^9$
- $l \leq r$
- $N, T, a, l, r, a_1, a_2, \cdots, a_n \in \mathbb{Z}$
- All operation are performed on existing elements.

Subtasks

Subtask 1 (10 pts)

- $0 \leq N \leq 10^3$
- $0 \leq T \leq 10^3$

Subtask 2 (30 pts)

- Only operation 1 and 2 are performed.

Subtask 3 (60 pts)

- No other constraints.

Sample Test Cases

Sample Input 1

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

Sample Output 1

```
12
17
4
1 9 2 6
```

Explanation for Sample 1

The initial sequence is: 8 5 1 2 6. After each operation, the sequence becomes:

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

Sample Input 2

```
7 10
2 8 5 9 1 1 8
2 4 7
1 2
2 5 6
2 2 1
3 1 7
3 2 7
3 5 5
1 5
3 7 7
2 6 1
```

Sample Output 2

```
31
29
7
1
9
2 5 1 9 1 6 1 1 8
```

Problem 4 - Dissolve of the Short-lived Agency (100 pts)

Problem Description

Story

After a number of attempts of gaining profit, Nextgen Technology Universal Company of Software (NTUCS) finally fails and dissolves into N independent companies.

There are N former employees of NTUCS, denoted as $E_1, E_2, \ldots, E_N$. There are now N companies, denoted as $C_1, C_2, \ldots, C_N$. Currently, employee E_1 is with company C_1, employee E_2 is with company C_2, ..., and employee E_N is with company C_N. Each of them is the only employee with the corresponding company. Let $S(E_i)$ denote the salary of employee E_i, and $B(E_i)$ denote the account balance of employee $E_i, 1 \leq i \leq N$. Initially, $B(E_i) = 0$ for all $1 \leq i \leq N$, as NTUCS just fails and thus all employees have zero account balance.

There are four types of events that can happen:

- **Merge** the company that has employee E_x, denoted as C_x, and the company that has employee E_y, denoted as C_y, into one company. If C_x and C_y are the same company, then no action is performed. All employees of C_x and C_y are now with the new merged company. The salaries and the account balances are *not* altered.
- **Raise** the salaries of all employees within the company that has employee E_k by an amount r. Let C denote that company. That is, for any employee E_i in company C, his or her salary is increased by r, i.e., $S(E_i) = S(E_i) + r$.
- The company that has employee E_k **transfers** the salaries to the accounts of all its employees *once*. Let C denote that company. That is, for any employee E_i in company C, the account balance $B(E_i)$ is increased by the salary of the corresponding employee $S(E_i)$, i.e., $B(E_i) = B(E_i) + S(E_i)$.
- Employee E_k **quits** from his or her current company C_q and starts a new company C_n, with new salary c, i.e., $S(E_k) = c$. The account balance of E_k, $B(E_k)$ is *not* changed.

A total of Q events will take place. Process the given Q events and print the final account balances of all employees.

Input

The first line of input consists of two space-separated integers N, Q. The second line consists of N space-separated integers, corresponding to $S(E_1), S(E_2), \ldots, S(E_N)$, the initial salaries of the employees. Each of the final Q lines has one of the following four formats:

1. 1 x y, indicating a **merge** of companies with employee E_x and E_y.
2. 2 k r, indicating a **raise** of salary of r for company with E_k.
3. 3 k, indicating a **transfer** of salaries to the accounts for company with E_k.
4. 4 k c, indicating E_k **quits** from the original company and starts a new company with new salary c.

Note that x, y, or k are integer numbers representing the index of the employee in the original set of employees.

Output

Print a line of N integers which are the final account balances, $B(E_1), B(E_2), \ldots, B(E_N)$. Consecutive numbers in the line should be separated by a space character.

Constraints

- $1 \leq N \leq 1 \times 10^6$
- $1 \leq Q \leq 2 \times 10^6$
- $\forall i, 1 \leq S(E) \leq 10^{12}$
- $1 \leq x, y \leq N, x \neq y$
- $1 \leq k \leq N$
- $1 \leq r \leq 10^6$
- $1 \leq c \leq 10^{12}$
- Time Limit: 6 s
- Memory Limit: 524288 KB

Subtasks

Subtask 1 (10 pts)

- $N \leq 1000$
- $Q \leq 1000$

Subtask 2 (20 pts)

- Only merge and transfer events occur.

Subtask 3 (25 pts)

- Only merge, raise, and transfer events occur.

Subtask 4 (20 pts)

- Only merge, transfer, and quit events occur.

Subtask 5 (25 pts)

- No other constraints.

Sample Test Cases

Sample Input 1

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

Sample Output 1

```
5 13 0
```

Explanation for Sample 1

1. Initial:
   $\{S(E_1), S(E_2), S(E_3)\} = \{1, 2, 3\}$,
   $\{B(E_1), B(E_2), B(E_3)\} = \{0, 0, 0\}$,
   Set of Companies $= \{\{1\}, \{2\}, \{3\}\}$
2. Raise the salary of the company with E_2 by 3:
   $\{S(E_1), S(E_2), S(E_3)\} = \{1, 5, 3\}$,
   $\{B(E_1), B(E_2), B(E_3)\} = \{0, 0, 0\}$,
   Set of Companies $= \{\{1\}, \{2\}, \{3\}\}$
3. Merge companies with E_1 and E_2:
   $\{S(E_1), S(E_2), S(E_3)\} = \{1, 5, 3\}$,
   $\{B(E_1), B(E_2), B(E_3)\} = \{0, 0, 0\}$,
   Set of Companies $= \{\{1, 2\}, \{3\}\}$
4. Transfer salary to account for company with E_1:
   $\{S(E_1), S(E_2), S(E_3)\} = \{1, 5, 3\}$,

$\{B(E_1), B(E_2), B(E_3)\} = \{1, 5, 0\}$,
Set of Companies $= \{\{1, 2\}, \{3\}\}$

5. Raise the salary of the company with E_2 by 3:
   $\{S(E_1), S(E_2), S(E_3)\} = \{4, 8, 3\}$,
   $\{B(E_1), B(E_2), B(E_3)\} = \{1, 5, 0\}$,
   Set of Companies $= \{\{1, 2\}, \{3\}\}$
6. Transfer salary to account for company with E_1:
   $\{S(E_1), S(E_2), S(E_3)\} = \{4, 8, 3\}$,
   $\{B(E_1), B(E_2), B(E_3)\} = \{5, 13, 0\}$,
   Set of Companies $= \{\{1, 2\}, \{3\}\}$

Sample Input 2

```
3 6
1 1 1
1 1 2
3 1
4 2 10
3 1
1 2 3
3 2
```

Sample Output 2

```
2 11 1
```

Explanation for Sample 2

1. Initial:
   $\{S(E_1), S(E_2), S(E_3)\} = \{1, 1, 1\}$,
   $\{B(E_1), B(E_2), B(E_3)\} = \{0, 0, 0\}$,
   Set of Companies $= \{\{1\}, \{2\}, \{3\}\}$
2. Merge companies with E_1 and E_2:
   $\{S(E_1), S(E_2), S(E_3)\} = \{1, 1, 1\}$,
   $\{B(E_1), B(E_2), B(E_3)\} = \{0, 0, 0\}$,
   Set of Companies $= \{\{1, 2\}, \{3\}\}$
3. Transfer salary to account for company with E_1:
   $\{S(E_1), S(E_2), S(E_3)\} = \{1, 1, 1\}$,
   $\{B(E_1), B(E_2), B(E_3)\} = \{1, 1, 0\}$,
   Set of Companies $= \{\{1, 2\}, \{3\}\}$
4. E_2 quits and starts a company with salary 10
   $\{S(E_1), S(E_2), S(E_3)\} = \{1, 10, 1\}$,

$\{B(E_1), B(E_2), B(E_3)\} = \{1, 1, 0\}$,
Set of Companies $= \{\{1\}, \{2\}, \{3\}\}$

5. Transfer salary to account for company with E_1:
   $\{S(E_1), S(E_2), S(E_3)\} = \{1, 10, 1\}$,
   $\{B(E_1), B(E_2), B(E_3)\} = \{2, 1, 0\}$,
   Set of Companies $= \{\{1\}, \{2\}, \{3\}\}$
6. Merge companies with E_2 and E_3:
   $\{S(E_1), S(E_2), S(E_3)\} = \{1, 10, 1\}$,
   $\{B(E_1), B(E_2), B(E_3)\} = \{2, 1, 0\}$,
   Set of Companies $= \{\{1\}, \{2, 3\}\}$
7. Transfer salary to account for company with E_2:
   $\{S(E_1), S(E_2), S(E_3)\} = \{1, 10, 1\}$,
   $\{B(E_1), B(E_2), B(E_3)\} = \{2, 11, 1\}$,
   Set of Companies $= \{\{1\}, \{2, 3\}\}$

Sample Input 3

```
3 9
4 2 2
2 3 3
1 2 3
3 3
4 3 4
2 2 4
1 3 1
1 1 3
2 1 2
3 1
```

Sample Output 3

```
6 2 11
```

Explanation for Sample 3

1. Initial:
   $\{S(E_1), S(E_2), S(E_3)\} = \{4, 2, 2\}$,
   $\{B(E_1), B(E_2), B(E_3)\} = \{0, 0, 0\}$,
   Set of Companies $= \{\{1\}, \{2\}, \{3\}\}$
2. Raise the salary of the company with E_3 by 3:
   $\{S(E_1), S(E_2), S(E_3)\} = \{4, 2, 5\}$,
   $\{B(E_1), B(E_2), B(E_3)\} = \{0, 0, 0\}$,

Set of Companies $= \{\{1\}, \{2\}, \{3\}\}$

3. Merge companies with E_2 and E_3:
   $\{S(E_1), S(E_2), S(E_3)\} = \{4, 2, 5\}$,
   $\{B(E_1), B(E_2), B(E_3)\} = \{0, 0, 0\}$,
   Set of Companies $= \{\{1\}, \{2, 3\}\}$
4. Transfer salary to account for company with E_3:
   $\{S(E_1), S(E_2), S(E_3)\} = \{4, 2, 5\}$,
   $\{B(E_1), B(E_2), B(E_3)\} = \{0, 2, 5\}$,
   Set of Companies $= \{\{1\}, \{2, 3\}\}$
5. E_3 quits and starts a company with salary 4:
   $\{S(E_1), S(E_2), S(E_3)\} = \{4, 2, 4\}$,
   $\{B(E_1), B(E_2), B(E_3)\} = \{0, 2, 5\}$,
   Set of Companies $= \{\{1\}, \{2\}, \{3\}\}$
6. Raise the salary of the company with E_2 by 4:
   $\{S(E_1), S(E_2), S(E_3)\} = \{4, 6, 4\}$,
   $\{B(E_1), B(E_2), B(E_3)\} = \{0, 2, 5\}$,
   Set of Companies $= \{\{1\}, \{2\}, \{3\}\}$
7. Merge companies with E_3 and E_1:
   $\{S(E_1), S(E_2), S(E_3)\} = \{4, 6, 4\}$,
   $\{B(E_1), B(E_2), B(E_3)\} = \{0, 2, 5\}$,
   Set of Companies $= \{\{1, 3\}, \{2\}\}$
8. Merge companies with E_1 and E_3:
   (E_1 and E_3 already in the same company; no action is taken.)
   $\{S(E_1), S(E_2), S(E_3)\} = \{4, 6, 4\}$,
   $\{B(E_1), B(E_2), B(E_3)\} = \{0, 2, 5\}$,
   Set of Companies $= \{\{1, 3\}, \{2\}\}$
9. Raise the salary of the company with E_1 by 2:
   $\{S(E_1), S(E_2), S(E_3)\} = \{6, 6, 6\}$,
   $\{B(E_1), B(E_2), B(E_3)\} = \{0, 2, 5\}$,
   Set of Companies $= \{\{1, 3\}, \{2\}\}$
10. Transfer salary to account for company with E_1:
    $\{S(E_1), S(E_2), S(E_3)\} = \{6, 6, 6\}$,
    $\{B(E_1), B(E_2), B(E_3)\} = \{6, 2, 11\}$,
    Set of Companies $= \{\{1, 3\}, \{2\}\}$

New: Feedback Part

Problem 5 - Feedback (100 pts)

In previous years, we have the tradition of asking for feedback in the final exam. Some students said they could not write down extensive feedback during the exam due to the time constraint. This time, let's try having the feedback program here in HW4 instead.

As we strive to continuously improve our course over the years, we highly value your feedback. We would appreciate if you could take a few moments to share your thoughts on the following aspects of our course:

1. (25 points) **Homework Assignments**: Please comment on your experience with both the handwritten and programming parts of the assignments (including the mini-HW). What aspects do you find beneficial, and what improvements would you suggest?

2. (25 points) **Class Activities**: How do you like the experience of our three class activites: software development team game, earth game, and Kahoot game? Are there specific activities you find most engaging or beneficial? What improvements would you suggest?

3. (25 points) **Teaching Assistants (TAs)**: We would love to hear about your interactions with the TAs, both during TA hours and online interaction (email and slido). How helpful have you found their support, and what could be improved in your interactions with them?

4. (25 points) **Lectures:** Given that we have both in-person and video lectures (from the second half of the semester), and two instructors, your insights on each teaching format and instructor's approach would be invaluable. What teaching methods, elements, and subjects do you appreciate the most, and what suggestions do you have for enhancement?

Your candid feedback is crucial for us to refine our approach and make this course as effective and enjoyable as possible. Thank you for taking the time to help us enhance your learning experience!