Problem B. Let's Write Divide and Conquer Happily!

Problem Description

In a digital realm where some of the finest minds in the programming world gather, a group of esteemed individuals has gathered to tackle a complex problem. Among them are **9s6e0i3c5**, **detaomega**, **mastermindccr**, **qwe854896**, **rayray9999**, **roylin506**, **huci0062, mmi366127**, and **SorahISA**. Their mission? To master the art of Divide and Conquer in solving a range of queries concerning an array of integers.

9s6e0i3c5, our resident thinker and problem solver, takes the lead in the discussion. With a sharp mind for algorithms, **9s6e0i3c5** outlines the problem at hand:

"For this challenge, we're dealing with an array x of n integers and a series of q queries. These queries come in four flavors:

1. Update the value at position k to u.
2. Find the sum of values in the range $[ql, qr]$.
3. Discover the maximum prefix sum in the range $[ql, qr]$,
   i.e., $\max(0, \max_{i \in [ql, qr]} \sum_{k=ql}^{i} x[k])$, we denote this value as $\text{pmax}([ql, qr])$.
4. Unearth the maximum suffix sum in the same range $[ql, qr]$,
   i.e., $\max(0, \max_{j \in [ql, qr]} \sum_{k=j}^{qr} x[k])$, we denote this value as $\text{smax}([ql, qr])$.

Now, let's delve into how we can efficiently handle these queries."

detaomega, the pragmatic coder, injects a dose of realism into the conversation:

"Let's begin by considering the second operation to get started - finding the sum of values in the interval $[ql, qr]$, represented as $\text{sum}([ql, qr])$. If we compute the sum for each query naively, we're looking at a time complexity of $\mathcal{O}(n)$. With q queries, the total time complexity becomes $\mathcal{O}(qn)$, which is far from ideal. So, let's explore how we can optimize this."

mastermindccr, known for insightful observations, proposes an elegant solution:

"To enhance the efficiency of range sum queries, perhaps we can employ a Divide and Conquer strategy. For any interval $[l, r]$ and $l \neq r$, we can use $m = \left\lfloor \dfrac{l+r}{2} \right\rfloor$ to divide the interval into smaller subintervals $[l, m]$ and $[m+1, r]$, calculate the sum of each one, and then sum them to get the sum of $[l, r]$."

SorahISA, the *MASTER TA*, offers his insights:

"Now, let's analyze the time complexity. Assume $T(n)$ is the time complexity of solving one range sum query with a length of n using Divide and Conquer, we can express it as follows:

$$T(n) = T\left(\left\lfloor\frac{n}{2}\right\rfloor\right) + T\left(\left\lceil\frac{n}{2}\right\rceil\right) + \mathcal{O}(1)$$

By applying the Master Theorem (~~qwe854896 Theorem~~), we conclude that $T(n) = \mathcal{O}(n)$, it won't be any better!" Since SorahISA is an algorithm master, you can always believe what he said.

mmi366127 adds his expertise to the discussion:

"Maybe..., we can store the results for these subintervals, allowing us to query them again without recomputing them? ~~Memory is always free.~~"

rayray9999, the memory wizard, raises a valid concern:

"However, there are $\mathcal{O}(n^2)$ possible subintervals, and memorizing them might lead to a **Memory Limit Exceeded**. We need to be more clever about this."

roylin506, the guy who is always curious about everything, offers a thoughtful suggestion:

"Is there a compromise solution that allows us to avoid memorizing too many answers while still providing enough information to quickly assemble an answer?"

SorahISA delves into a creative idea (since he is always a master):

"We don't necessarily have to divide an interval right at the midpoint. We can choose other points to divide and still obtain the correct answer in the end. For instance:

- $\forall m \in [l, r-1],\ \mathrm{sum}([l, r]) = \mathrm{sum}([l, m]) + \mathrm{sum}([m+1, r])$.

And it's not limited to just one cut, for example:

- $\mathrm{sum}([l_1, r_1]) + \mathrm{sum}([l_2, r_2]) + \ldots + \mathrm{sum}([l_k, r_k])$ such that $l_i = r_{i-1} + 1$.

Perhaps we only need to memorize the answers encountered in one specific query, and then for subsequent queries, we can partition them into intervals that match the recorded answers perfectly!

If we're only recording the answer from one specific query, it would make sense to remember all the intervals encountered during the calculation of $\mathrm{sum}([1, n])$ because all queried intervals are encompassed within $[1, n]$!"

huci0062 inquires:

"Nice idea, but how can we find the specific intervals to combine them to the answer of any $[ql, qr]$?"

qwe854896 provides further insight:

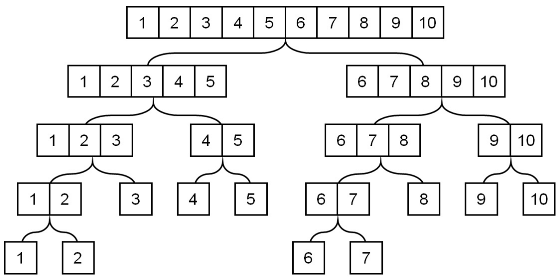
"Instead of thinking about how to divide $[ql, qr]$ into intervals we've memorized, let's consider how the memorized answers can assist us in calculating the answer for $[ql, qr]$.

Specifically, you can initially use Divide and Conquer to calculate the answer for $[1, n]$ and record the answers encountered during the recursion."

SorahISA elaborates on the process:

"Then, to calculate the answer for $[ql, qr]$, you can start from $[1, n]$ and employ Divide and Conquer once more. During the recursion, for the current interval $[l, r]$ and the queried interval $[ql, qr]$, there are three situations:

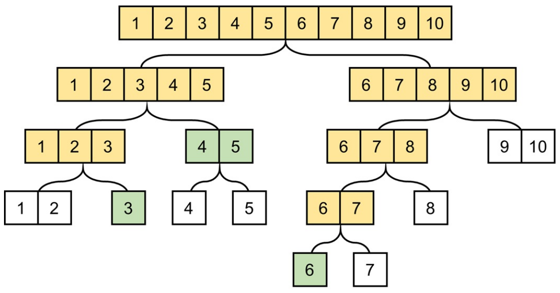
1. $[l, r]$ is entirely inside $[ql, qr]$. In this case, we can directly add $\text{sum}([l, r])$ to our answer.
2. $[l, r]$ is entirely outside $[ql, qr]$. In this scenario, this interval doesn't affect the answer.
3. Otherwise, we can divide it into two subproblems:
    1. How the answer of the interval $[l, m]$ contributes to $\text{sum}([ql, qr])$.
    2. How the answer of the interval $[m + 1, r]$ contributes to $\text{sum}([ql, qr])$."



mastermindccr seeks clarification with an example:

"Could you provide some examples? Suppose we want to find the answer for $[3, 6]$ within $[1, 10]$. How can we use these intervals to achieve that?"

detaomega responds with a visual aid:



"Similar to the image above, you can utilize these green intervals (situation 1): $[3, 3], [4, 5], [6, 6]$ to determine $\text{sum}([3, 6])$. As for the yellow intervals (situation 3), you can't predict their contribution, so you divide them. For the uncolored interval (situation 2), it won't contribute in any way."

roylin506 inquires again:

"So what is the time complexity of implementing this?"

qwe854896 adds valuable information:

"We can observe that situation 3, where $[l, r]$ is neither entirely inside nor outside of $[ql, qr]$, will always reduce to either situation 1 or situation 2 in the next recursion layer.

I've also noticed an interesting property: for intervals at the same recursion depth, situation 3 occurs at most twice!"

rayray9999 calculates the implications:

"This insight implies that the number of cases we need to consider is bounded by a constant multiplied by the depth of recursion. There, the time complexity is determined by the depth of recursion, which is $\mathcal{O}(\log n)$!"

huci0062 continues with a follow-up question:

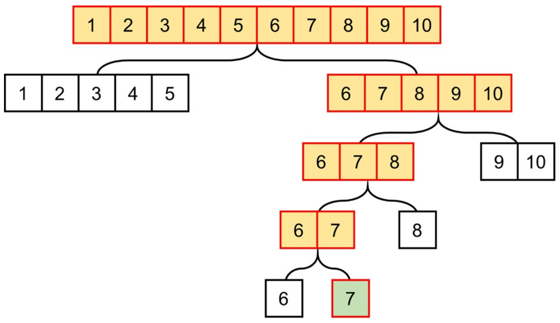
"Now that we can efficiently solve the range summation operation, how can we update the value at position k to u?"

mmi366127 expands on the idea:

"We can treat updates as a range query of $[k, k]$. The path of the range query traverses through intervals whose values will be affected if the k^{th} number changes to u, and we can handle this efficiently."

huci0062 responds with a detailed explanation and an illustrative image:

"So when we want to update 7^{th} number to 63, we first locate the interval $[7, 7]$ by following the path $[1, 10] \to [6, 10] \to [6, 8] \to [6, 7] \to [7, 7]$, and update all the answers along this path as shown in the image below:"



"Once $[7, 7]$ is updated to 63, we move back to $[6, 7]$ using the answer for $[6, 6]$ and the updated answer for $[7, 7]$ to compute the new answer for $[6, 7]$. We then proceed to $[6, 8]$ using the updated answer for $[6, 7]$ and the answer for $[8, 8]$ to calculate the new answer for $[6, 8]$, and so on, working our way back until we've updated $[1, 10]$."

roylin506 confirms:

"That's correct! Now we can also update answers in $\mathcal{O}(\log n)$!"

SorahISA seeks further understanding:

"So the last question is how to calculate the maximum prefix sum? It seems like a challenging aspect."

9s6e0i3c5 provides an answer:

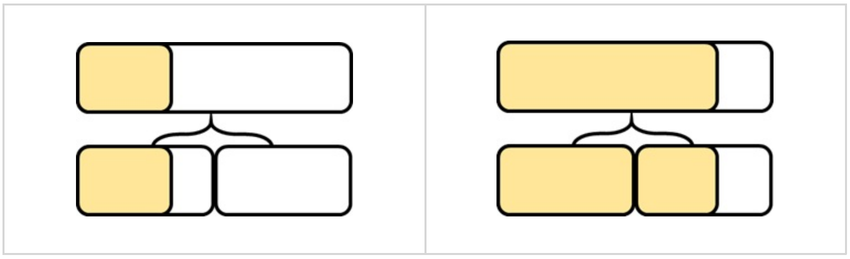
"Certainly, to calculate the maximum prefix sum, denoted as $\mathrm{pmax}([ql, qr])$, we can apply a similar Divide and Conquer approach. The key is to modify the way you merge the answer from the left side and the right side.

For each interval with a length greater than or equal to 2, the maximum prefix sum can only happen in the following two cases:

In the first case, we can obtain the candidate answer by considering $\mathrm{pmax}(\text{left interval})$.

On the other hand, we can get the other candidate answer by adding $\mathrm{sum}(\text{left interval})$ and $\mathrm{pmax}(\text{right interval})$.

By considering the maximum over these two possible cases, we can merge and determine the answer!"



And so, these brilliant minds continue their journey to Divide and Conquer this intricate problem, learning and growing together in their quest to master Divide and Conquer, all with smiles on their faces. As for you, the task of finding solutions for maximum prefix sum and maximum suffix sum and implementing all of this is assigned to you.

Input Format

The first input line has two integers n and q: the number of values and queries.

The second line has n integers $x_1, x_2, \ldots, x_n$: the array values.

Finally, there are q lines describing the queries. Each line start with one integer op.

If $op = 1$, then the last two numbers are k and u; otherwise, the last two numbers are ql and qr.

Output Format

Print the result of each query of type 2, 3, and 4.

Constraints

- $1 \le n, q \le 200\,000$.
- $-10^9 \le x_i \le 10^9$.
- $op \in \{1, 2, 3, 4\}$.
- $1 \le k \le n$.
- $-10^9 \le u \le 10^9$.
- $1 \le ql \le qr \le n$.

Subtasks

1. (5 points) $n \le 5000$; $q \le 5000$.
2. (20 points) $op = 2$ for all queries.
3. (20 points) $op \in \{1, 2\}$ for all queries.
4. (25 points) $op \in \{2, 3, 4\}$ for all queries.
5. (5 points) $ql = 1$; $qr = n$.
6. (25 points) No additional constraints.

No.	Testdata Range	Time Limit (ms)	Memory Limit (KiB)
Samples	1-3	1000	262144
1	1-6	1000	262144
2	7-10	1000	262144
3	7-14	1000	262144
4	7-10,15-18	1000	262144
5	19-22	1000	262144
6	1-26	1000	262144

Samples

Sample Input 1

```
10 4
4 -8 7 -6 3 -4 8 -7 6 -3
2 3 6
2 4 8
1 7 63
2 4 8
```

This sample input satisfies the constraints of Subtasks 1, 3, 6.

Sample Output 1

```
0
-6
49
```

Sample Input 2

```
10 7
4 -8 7 -6 3 -4 8 -7 6 -3
2 1 10
3 1 10
4 1 10
1 7 63
2 1 10
3 1 10
4 1 10
```

This sample input satisfies the constraints of Subtasks 1, 5, 6.

Sample Output 2

```
0
4
4
55
59
59
```

Sample Input 3

```
10 5
4 -8 7 -6 3 -4 8 -7 6 -3
2 3 6
3 4 8
1 7 63
3 4 8
4 2 6
```

This sample input satisfies the constraints of Subtasks 1, 6.

Sample Output 3

```
0
1
56
0
```