

# Prefix Sum Basic

누적합 기초

競技プログラミングの鉄則

KPSC Algorithm Study 24/11/14 Thu.

by Haru\_101

# Prefix Sum on 1-Dimension

- 누적합(Prefix Sum)은 배열의 특정한 범위의 합을 효율적으로 구하는 방법
- 쿼리가 다음과 같이 주어졌다고 생각해봅시다.
  - $l$ 번째 값부터  $r$ 번째 값까지의  $A_l + A_{l+1} + \dots + A_r$ 을 출력하라.
- 일단 반복문을 사용해서 풀어볼까요?

# Prefix Sum on 1-Dimension

- $l = 2, r = 6$

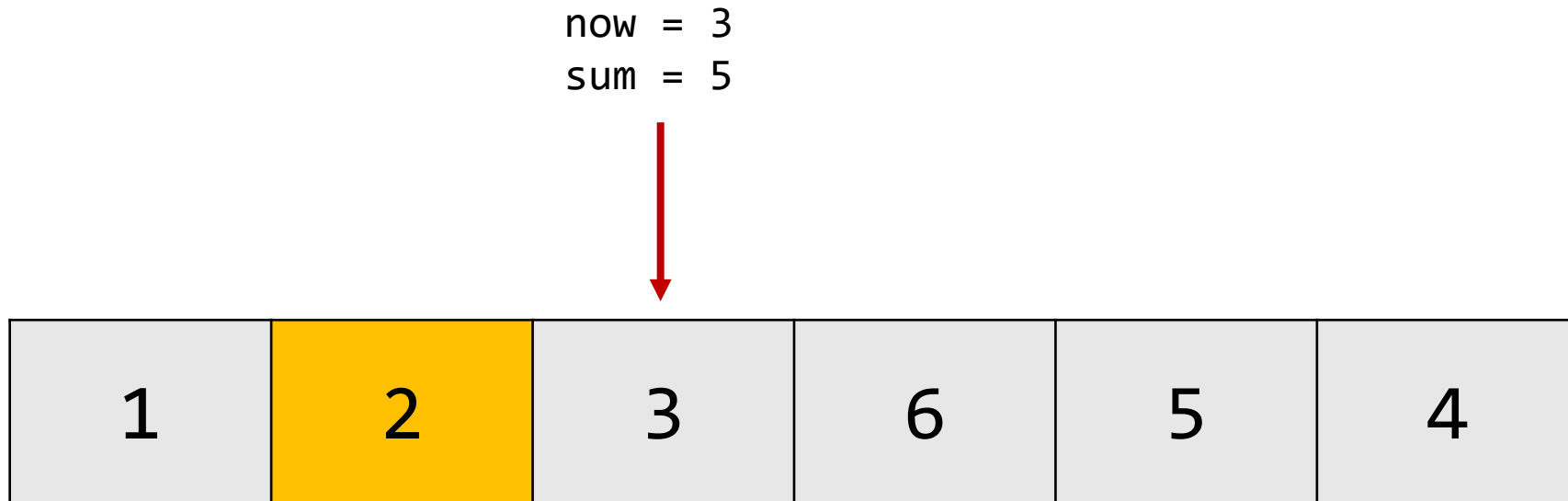
now = 2  
sum = 2



1	2	3	6	5	4
---	---	---	---	---	---

# Prefix Sum on 1-Dimension

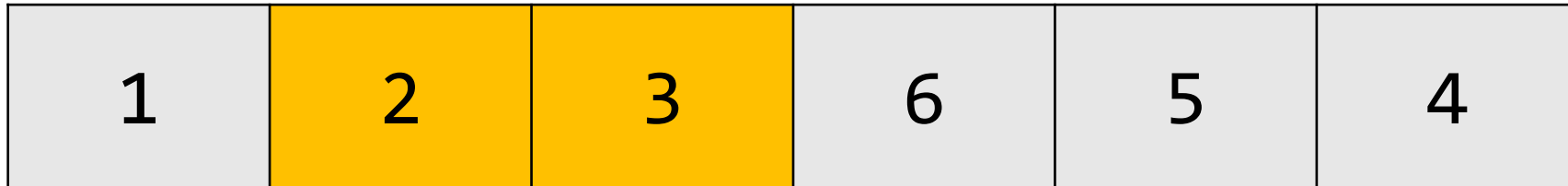
- $l = 2, r = 6$



# Prefix Sum on 1-Dimension

- $l = 2, r = 6$

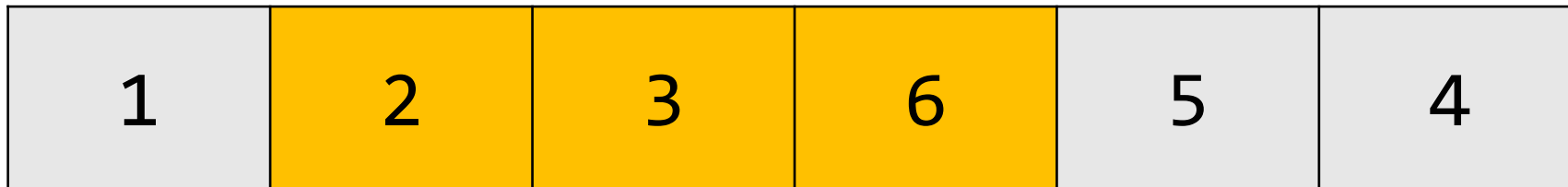
now = 4  
sum = 11



# Prefix Sum on 1-Dimension

- $l = 2, r = 6$

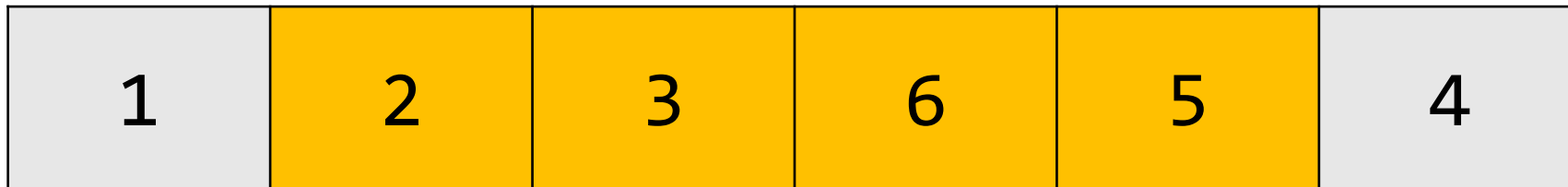
now = 5  
sum = 16



# Prefix Sum on 1-Dimension

- $l = 2, r = 6$

now = 6  
sum = 20



# Prefix Sum on 1-Dimension

- $l = 2, r = 6 \rightarrow 20$

1	2	3	6	5	4
---	---	---	---	---	---



# Prefix Sum on 1-Dimension

- 반복문을 사용하면,  $l$ 부터  $r$ 까지의 모든 원소를 반복해서 더해나가는 과정이 필요합니다.
- $l = 1, r = n$ 인 쿼리가  $Q$ 번 주어졌을 때 시간복잡도를 구해보면,
  - $O(Qn)$
  - $n = 100,000, Q = 100,000$ 이면 주어진 시간내에 해결하지 못합니다.
- 따라서 이를 효율적으로 계산하는 방법이 필요합니다.

# Prefix Sum on 1-Dimension

- 아까 봤던 배열을 변형해서 봐봅시다.

1	2	3	6	5	4
---	---	---	---	---	---

- 노랗게 칠한 부분을 어떻게 빠르게 구할 수 있을까요?

# Prefix Sum on 1-Dimension

1	2	3	6	5	4
---	---	---	---	---	---

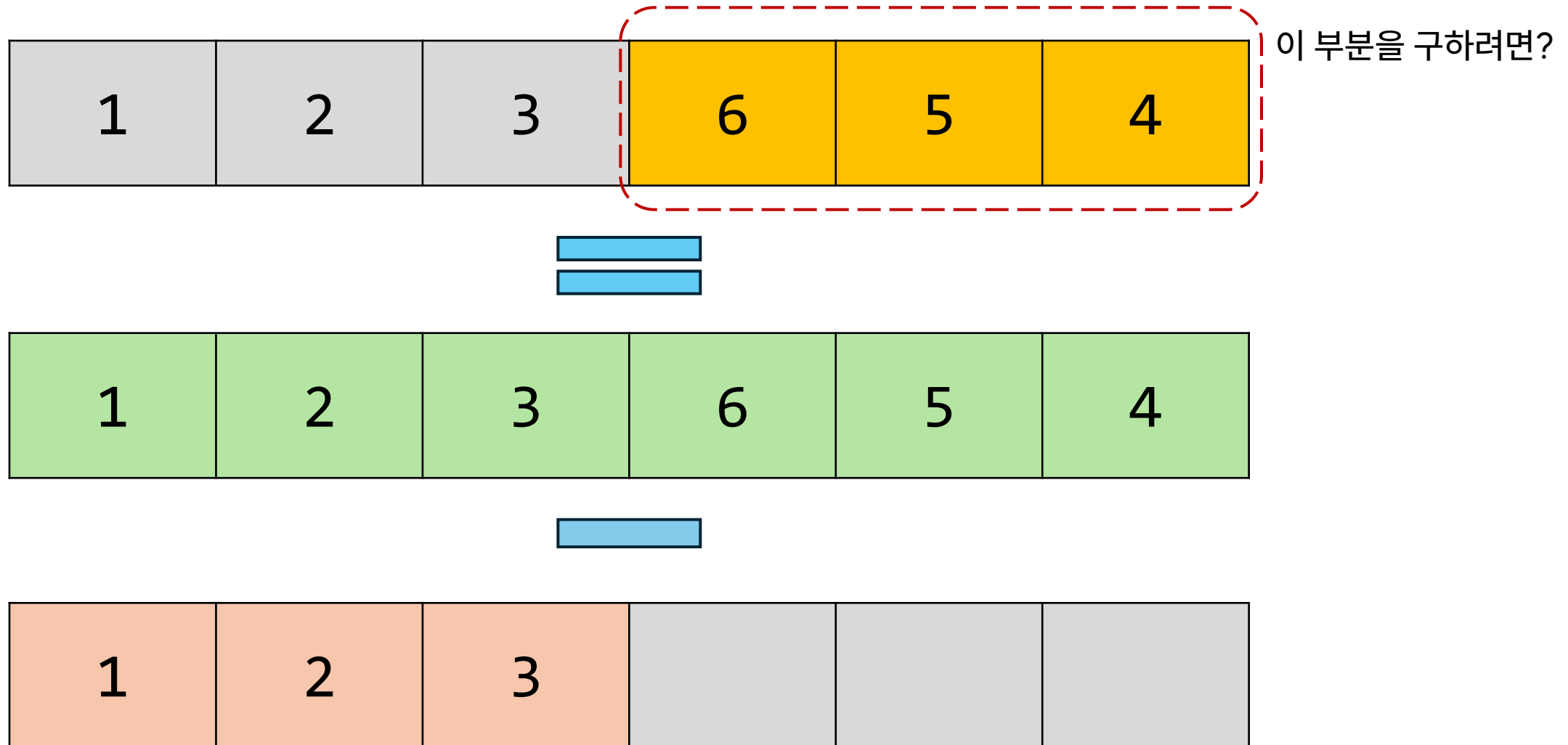
=

1	2	3	6	5	4
---	---	---	---	---	---

-

1	2	3			
---	---	---	--	--	--

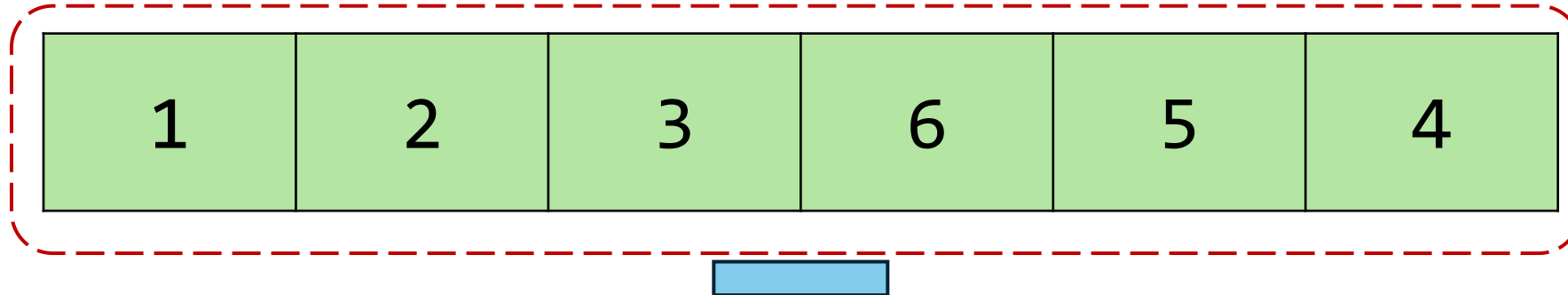
# Prefix Sum on 1-Dimension



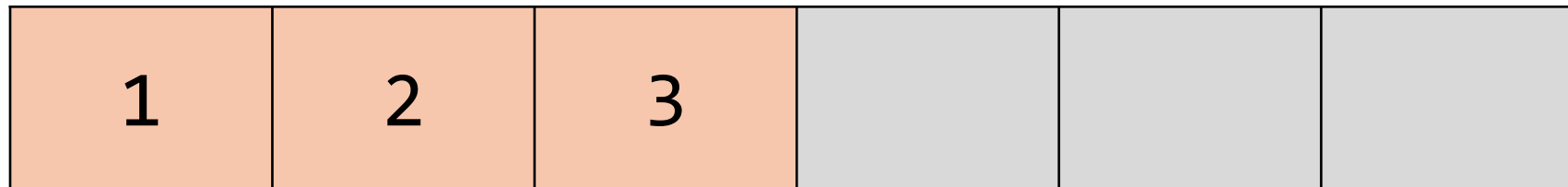
# Prefix Sum on 1-Dimension



=



전체 합을 구하고



# Prefix Sum on 1-Dimension

1	2	3	6	5	4
---	---	---	---	---	---

=

1	2	3	6	5	4
---	---	---	---	---	---

—

1	2	3			
---	---	---	--	--	--

이 부분의 합을 빼면 된다!

# Prefix Sum on 1-Dimension

- 누적합 알고리즘은 특정 범위에 대한 값을 빠르게 구할 수 있는 알고리즘입니다.
- 'l번째 값부터 r번째 값까지의  $A_l + A_{l+1} + \dots + A_r$ 을 출력하라.' 쿼리에 대해 다시 생각해보죠.
  - $l = 4, r = 6$ 일때,
  - 인덱스 1~6까지의 합에서 인덱스 1~3까지의 합을 빼면 인덱스 4~6의 합을 구할 수 있습니다!

1	2	3	6	5	4
---	---	---	---	---	---

# Prefix Sum on 1-Dimension

- 반복문으로 구하는 것과 무슨 차이가 있냐 하면,
  - 일반적인 반복문으로 구한다 -> 각 쿼리에 최대  $O(n)$ 만큼 소요
  - 누적합 알고리즘을 사용한다 -> 초기에 합을 누적해서 더하고 기록하는데에  $O(n)$ 소요, 각 쿼리에  $O(1)$ 소요

원본 배열	1	2	3	6	5	4
누적합 배열	$S_1 = 1$	$S_2$	$S_3$	$S_4$	$S_5$	$S_6$
누적합 배열	1	3	6	12	17	21



# Prefix Sum on 1-Dimension

- $l = 4, r = 6$ 일때,  $S_6 - S_3 = 21 - 6 = 6 + 5 + 4 = 15$

원본 배열	1	2	3	6	5	4
누적합 배열	$S_1 = 1$	$S_2$	$S_3$	$S_4$	$S_5$	$S_6$
누적합 배열	1	3	6	12	17	21

# Prefix Sum on 1-Dimension

- 문제를 풀어봅시다. [번역은 다음페이지]
- [https://atcoder.jp/contests/tessoku-book/tasks/math\\_and\\_algorithm\\_ai](https://atcoder.jp/contests/tessoku-book/tasks/math_and_algorithm_ai)
- 遊園地「ALGO-RESORT」では $N$ 日間にわたるイベントが開催され、 $i$ 日目( $1 \leq i \leq N$ )には $A_i$ 人が来場しました。  
以下の合計 $Q$ 個の質問に答えるプログラムを作成してください。
  - 1個目の質問： $L_1$ 日目から $R_1$ 日目までの合計来場者数は？
  - 2個目の質問： $L_2$ 日目から $R_2$ 日目までの合計来場者数は？
  - ...
  - $Q$ 個目の質問： $L_Q$ 日目から $R_Q$ 日目までの合計来場者数は？

# Prefix Sum on 1-Dimension

- 문제를 풀어봅시다.
- [https://atcoder.jp/contests/tessoku-book/tasks/math\\_and\\_algorithm\\_ai](https://atcoder.jp/contests/tessoku-book/tasks/math_and_algorithm_ai)
- 유원지 'ALGO-RESORT'에는  $N$ 일간에 걸쳐 이벤트가 개최되어,  $i$ 일째 ( $1 \leq i \leq N$ )에는  $A_i$ 명의 사람이 왔습니다. 다음과 같이  $Q$ 개의 쿼리에 대해 답하는 프로그램을 작성하시오.
  - 1번째 쿼리 :  $L_1$ 일부터  $R_1$ 일까지 온 사람의 총합은?
  - 2번째 쿼리 :  $L_1$ 일부터  $R_1$ 일까지 온 사람의 총합은?
  - ...
  - $Q$ 번째 쿼리 :  $L_Q$ 일부터  $R_Q$ 일까지 온 사람의 총합은?
- 출력 :  $Q$ 행에 걸쳐 각 쿼리의 결과를 출력하라.

# Prefix Sum on 2-Dimension

- 이번엔 2차원 배열 상에서의 누적합 알고리즘을 사용하는 법을 알아보시다.

1	3	5	7
2	4	6	8
1	2	3	4
5	6	7	8

# Prefix Sum on 2-Dimension

- 화살표 방향으로 보면, 각 행에 대해 누적합 배열을 작성할 수 있습니다.

1	3	5	7
2	4	6	8
1	2	3	4
5	6	7	8

# Prefix Sum on 2-Dimension

원본 배열

1	3	5	7
2	4	6	8
1	2	3	4
5	6	7	8

누적합 배열(행단위)

1	4	9	16
2	6	12	20
1	3	6	10
5	11	18	26

# Prefix Sum on 2-Dimension

- 하지만, 이걸로 끝나면 모든 문제가 쉽게 풀리겠지만... 현실은 그렇지 않습니다.
- 쿼리가 다음과 같다고 합시다.
  - $x_1, y_1, x_2, y_2$   
배열의  $(x_1, y_1)$ 부터  $(x_2, y_2)$ 까지의 합을 구하라.
- 이 쿼리를 풀기 위해선  $y_1$ 부터  $y_2$ 까지 각 행마다  $S_{x_2} - S_{x_1-1}$ 을 구하면 됩니다.
  - 그러면  $y_1 = 1, y_2 = n$ 일때  $O(n)$ 의 시간이 걸리고,  
이를  $Q$ 번 처리해야하므로,  $O(Qn)$ 의 시간이 걸립니다.
- 이를 어떻게 해결하면 좋을까요?

누적합 배열(행단위)

1	4	9	16
2	6	12	20
1	3	6	10
5	11	18	26

# Prefix Sum on 2-Dimension

- 화살표 방향으로 한번 더 누적합을 수행하면 됩니다.

1	4	9	16
2	6	12	20
1	3	6	10
5	11	18	26



# Prefix Sum on 2-Dimension

원본 배열

1	3	5	7
2	4	6	8
1	2	3	4
5	6	7	8

최종 누적합 배열

1	4	9	16
3	10	21	36
4	13	27	46
9	24	45	72

# Prefix Sum on 2-Dimension

- 결국 이 최종 누적합 배열은, 각  $(x, y)$ 에 대해,  $(1, 1) \sim (x, y)$ 까지의 누적합을 저장하게 됩니다.
- 근데...  $(1, 1) \sim (x, y)$  말고  $(x_1, y_1) \sim (x_2, y_2)$ 의 합은 어떻게 구하죠?

최종 누적합 배열

1	4	9	16
3	10	21	36
4	13	27	46
9	24	45	72

# Prefix Sum on 2-Dimension

- $x_1 = 2, y_1 = 2, x_2 = 3, y_2 = 4$

원본 배열

1	3	5	7
2	4	6	8
1	2	3	4
5	6	7	8

최종 누적합 배열

1	4	9	16
3	10	21	36
4	13	27	46
9	24	45	72

# Prefix Sum on 2-Dimension

- 하늘색 부분에서 노란색 부분을 빼면 초록색이 나오지 않을까요?

1	3	5	7	1	3	5	7
2	4	6	8	2	4	6	8
1	2	3	4	1	2	3	4
5	6	7	8	5	6	7	8

# Prefix Sum on 2-Dimension

- 이를 행, 열 단위로 보면, 하늘색에서 노란색을 뺀 것과 같겠죠?

1	3	5	7
2	4	6	8
1	2	3	4
5	6	7	8

1	3	5	7
2	4	6	8
1	2	3	4
5	6	7	8

1	3	5	7
2	4	6	8
1	2	3	4
5	6	7	8

# Prefix Sum on 2-Dimension

- 근데, 가장 왼쪽 윗부분이 2번 빼집니다.

1	3	5	7
2	4	6	8
1	2	3	4
5	6	7	8

1	3	5	7
2	4	6	8
1	2	3	4
5	6	7	8

1	3	5	7
2	4	6	8
1	2	3	4
5	6	7	8

# Prefix Sum on 2-Dimension

- 그러면, 보라색 부분을 2번 빼고 1번 더하면, 1번 뺀 것과 같습니다.

1	3	5	7
2	4	6	8
1	2	3	4
5	6	7	8

1	3	5	7
2	4	6	8
1	2	3	4
5	6	7	8

1	3	5	7
2	4	6	8
1	2	3	4
5	6	7	8

# Prefix Sum on 2-Dimension

- 이를 누적합 관점에서 봅시다.
- 하늘색의 합은 누적합 배열에서 노란색 부분으로 나타낼 수 있습니다.

1	3	5	7
2	4	6	8
1	2	3	4
5	6	7	8

1	4	9	16
3	10	21	36
4	13	27	46
9	24	45	72



# Prefix Sum on 2-Dimension

- 이를 누적합 관점에서 봅시다.
- 주황색은 합은 누적합 배열에서 보라색 부분으로 나타낼 수 있습니다. (가로방향)

1	3	5	7
2	4	6	8
1	2	3	4
5	6	7	8

1	3	5	7
2	4	6	8
1	2	3	4
5	6	7	8

1	4	9	16
3	10	21	36
4	13	27	46
9	24	45	72

# Prefix Sum on 2-Dimension

- 이를 누적합 관점에서 봅시다.
- 주황색은 합은 누적합 배열에서 보라색 부분으로 나타낼 수 있습니다. (세로방향)

1	3	5	7
2	4	6	8
1	2	3	4
5	6	7	8

1	3	5	7
2	4	6	8
1	2	3	4
5	6	7	8

1	4	9	16
3	10	21	36
4	13	27	46
9	24	45	72

# Prefix Sum on 2-Dimension

- 이를 누적합 관점에서 봅시다.
- 이제 2번 겹치는 부분을 보라색 부분으로 나타내봅시다.

1	3	5	7
2	4	6	8
1	2	3	4
5	6	7	8

1	3	5	7
2	4	6	8
1	2	3	4
5	6	7	8

1	4	9	16
3	10	21	36
4	13	27	46
9	24	45	72

# Prefix Sum on 2-Dimension

- 이를 누적합 관점에서 봅시다.
- 결론적으로, 쿼리에 대한 출력을 구하기 위해서는 노란색 - (보라색 2개) + 주황색을 하면 됩니다.

1	3	5	7
2	4	6	8
1	2	3	4
5	6	7	8

1	3	5	7
2	4	6	8
1	2	3	4
5	6	7	8

1	4	9	16
3	10	21	36
4	13	27	46
9	24	45	72

# Prefix Sum on 2-Dimension

- 문제를 풀어봅시다. [번역은 다음페이지]
- [https://atcoder.jp/contests/tessoku-book/tasks/tessoku\\_book\\_h](https://atcoder.jp/contests/tessoku-book/tasks/tessoku_book_h)
- $H \times W$ 의マス目があります. 上から $i$ 行目, 左から $j$ 列目にあるマス $(i, j)$ には, 整数  $x_{i,j}$ が書かれています. これについて, 以下の $Q$ 個の質問に答えるプログラムを作成してください.
  - $i$ 個目の質問: 左上 $(A_i, B_i)$  右下 $(C_i, D_i)$ の長方形領域に書かれた整数の総和は?

# Prefix Sum on 2-Dimension

- 문제를 풀어봅시다.
- [https://atcoder.jp/contests/tessoku-book/tasks/tessoku\\_book\\_h](https://atcoder.jp/contests/tessoku-book/tasks/tessoku_book_h)
- 크기가  $H \times W$ 인 격자가 있습니다. 위에서  $i$ 번째 행, 왼쪽에서  $j$ 번째 열에는 정수  $X_{i,j}$ 가 쓰여져 있습니다. 이 격자에 대해  $Q$ 개의 쿼리를 수행하는 프로그램을 작성하시오.
  - $i$ 번째 쿼리 : 왼쪽 위  $(A_i, B_i)$ 에서부터 오른쪽 아래  $(C_i, D_i)$ 까지의 써져있는 수들의 합은?
- 출력 :  $Q$ 행에 걸쳐 각 쿼리의 결과를 출력하라.

# Prefix Sum on 2-Dimension

- 문제를 풀어봅시다.
- [https://atcoder.jp/contests/tessoku-book/tasks/tessoku\\_book\\_i](https://atcoder.jp/contests/tessoku-book/tasks/tessoku_book_i)

ALGO 王国は  $H \times W$  のマス目で表されます. 最初は, どのマスにも雪が積もっていませんが, これから  $N$  日間にわたって雪が降り続けます.

上から  $i$  行目, 左から  $j$  列目のマスを  $(i, j)$  とするとき,  $t$  日目には「マス  $(A_t, B_t)$  を左上とし, マス  $(C_t, D_t)$  を右下とする長方形領域」の積雪が  $1\text{cm}$  だけ増加することが予想されています. 最終的な各マスの積雪を出力するプログラムを作成してください.

- ALGO 왕국은  $H \times W$  크기의 격자 형태로 나타낼 수 있습니다. 처음에는 격자의 모든 칸에 눈이 쌓여있진 않지만, 지금부터  $N$  일동안 눈이 내릴 예정입니다.
- 위에서  $i$  번째 행, 왼쪽에서  $j$  번째 열에 있는 칸을  $(i, j)$  라고 할 때,  $t$  일에는 왼쪽 위  $(A_t, B_t)$  에서부터 오른쪽 아래  $(C_t, D_t)$  까지 눈이  $1\text{cm}$  쌓입니다.
- 최종적으로 각 칸에 쌓인 눈의 양을 출력하는 프로그램을 작성하시오.

# Prefix Sum on 2-Dimension

- 이 문제를 단순히 2중 반복문으로 해결할 수 있지만, 최악의 경우  $O(HW)$ 의 계산을  $Q$ 번 수행해야 하므로,  $O(QHW)$ 의 시간이 걸립니다.
- 이를 부분합으로 어떻게 풀까요?

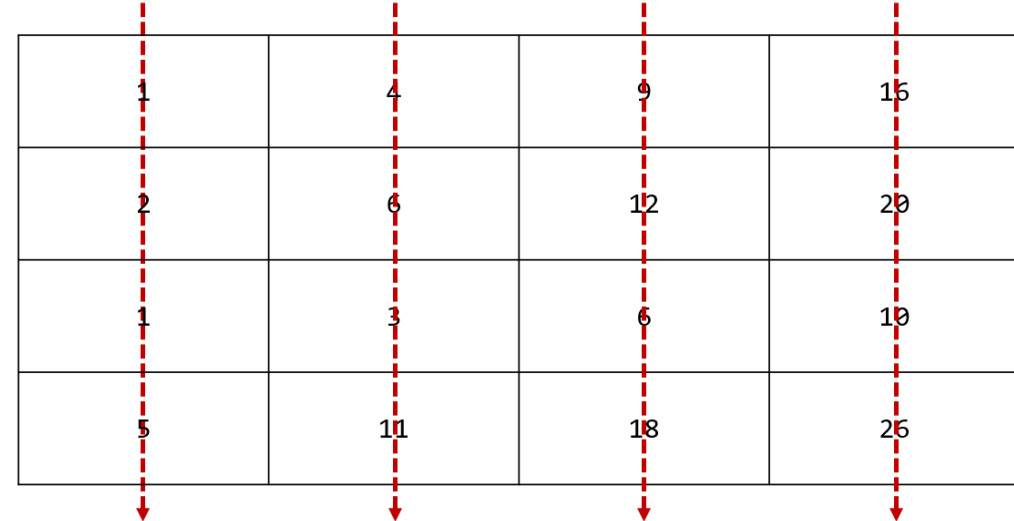


# Prefix Sum on 2-Dimension

- 아까 2차원 누적합을 계산할 때, 행 방향 -> 열 방향으로 2번 계산을 해줘야 함을 알 수 있었습니다.



1	3	5	7
2	4	6	8
1	2	3	4
5	6	7	8



1	4	9	15
2	6	12	20
1	3	6	10
5	11	18	26

# Prefix Sum on 2-Dimension

- 이 아이디어를 이용해서,  $O(Q + HW)$ 에 풀 수 있습니다.

1	3	5	7
2	4	6	8
1	2	3	4
5	6	7	8

1	4	9	15
2	6	12	20
1	3	6	10
5	11	18	26

# Prefix Sum on 2-Dimension

- 먼저  $t$ 일째에 오는 눈의 범위중 가장 왼쪽 위에 1을 더해봅시다. (예 :  $(2, 2) \sim (3, 3)$ )

현재 배열

0	0	0	0
0	+1	0	0
0	0	0	0
0	0	0	0

최종 누적합 배열

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

# Prefix Sum on 2-Dimension

- 최종 누적합 배열을 계산해봅시다.

현재 배열

0	0	0	0
0	+1	0	0
0	0	0	0
0	0	0	0

최종 누적합 배열

0	0	0	0
0	+1	+1	+1
0	0	0	0
0	0	0	0

# Prefix Sum on 2-Dimension

0	0	0	0
0	+1	+1	+1
0	0	0	0
0	0	0	0

- 최종 누적합 배열을 계산해봅시다.

현재 배열

0	0	0	0
0	+1	0	0
0	0	0	0
0	0	0	0

최종 누적합 배열

0	0	0	0
0	+1	+1	+1
0	+1	+1	+1
0	+1	+1	+1

# Prefix Sum on 2-Dimension

- 근데, 우리가 풀고자 하는 문제에는 (2, 4)와 같은 위치에 눈이 쌓이길 원하지 않습니다.

현재 배열

0	0	0	0
0	+1	0	0
0	0	0	0
0	0	0	0

최종 누적합 배열

0	0	0	0
0	+1	+1	+1
0	+1	+1	+1
0	+1	+1	+1

# Prefix Sum on 2-Dimension

- 그러면 어떻게 해야할까요?

현재 배열

0	0	0	0
0	+1	0	0
0	0	0	0
0	0	0	0

최종 누적합 배열

0	0	0	0
0	+1	+1	+1
0	+1	+1	+1
0	+1	+1	+1

# Prefix Sum on 2-Dimension

- (2, 2)~(3, 3)까지 눈이 쌓이는걸 원하니 (2, 4)에 -1을 배열에 더해봅시다.

현재 배열

0	0	0	0
0	+1	0	-1
0	0	0	0
0	0	0	0

최종 누적합 배열

0	0	0	0
0	+1	+1	+1
0	+1	+1	+1
0	+1	+1	+1



# Prefix Sum on 2-Dimension

- 그 다음 최종 누적합 배열을 다시 계산해봅시다.

현재 배열

0	0	0	0
0	+1	0	-1
0	0	0	0
0	0	0	0

최종 누적합 배열

0	0	0	0
0	+1	+1	0
0	0	0	0
0	0	0	0

# Prefix Sum on 2-Dimension

- 최종 누적합 배열을 계산해봅시다.

0	0	0	0
0	+1	+1	0
0	0	0	0
0	0	0	0

현재 배열

0	0	0	0
0	+1	0	-1
0	0	0	0
0	0	0	0

최종 누적합 배열

0	0	0	0
0	+1	+1	0
0	+1	+1	0
0	+1	+1	0

# Prefix Sum on 2-Dimension

- 근데, 아직도 (4, 2), (4, 3)에 눈이 쌓여있습니다.



현재 배열

0	0	0	0
0	+1	0	-1
0	0	0	0
0	0	0	0

최종 누적합 배열

0	0	0	0
0	+1	+1	0
0	+1	+1	0
0	+1	+1	0

# Prefix Sum on 2-Dimension

- 이제, 현재 배열의 (4, 2)에 -1을 다시 넣어서 누적합 배열을 계산해봅시다.

현재 배열

0	0	0	0
0	+1	0	-1
0	0	0	0
0	-1	0	0

최종 누적합 배열

0	0	0	0
0	+1	+1	0
0	0	0	0
0	-1	-1	-1

# Prefix Sum on 2-Dimension

0	0	0	0
0	+1	+1	0
0	0	0	0
0	-1	-1	-1

- 근데, 최종 누적합 배열의 (4, 4)에 -1이 있습니다. 눈이 -1cm 내릴 수 있진 않겠죠...

현재 배열

0	0	0	0
0	+1	0	-1
0	0	0	0
0	-1	0	0

최종 누적합 배열

0	0	0	0
0	+1	+1	0
0	+1	+1	0
0	0	0	-1

# Prefix Sum on 2-Dimension

- 그래서 현재 배열의 (4, 4)에 1을 더해서 최종 누적합 배열을 계산해봅시다.

현재 배열

0	0	0	0
0	+1	0	-1
0	0	0	0
0	-1	0	+1

최종 누적합 배열

0	0	0	0
0	+1	+1	0
0	0	0	0
0	-1	-1	0

# Prefix Sum on 2-Dimension

0	0	0	0
0	+1	+1	0
0	0	0	0
0	-1	-1	0

- 그래서 현재 배열의 (4, 4)에 1을 더해서 최종 누적합 배열을 계산해봅시다.

현재 배열

0	0	0	0
0	+1	0	-1
0	0	0	0
0	-1	0	+1

최종 누적합 배열

0	0	0	0
0	+1	+1	0
0	+1	+1	0
0	0	0	0

# Prefix Sum on 2-Dimension

- 결론적으로,  $(A_i, B_i)$ 에 1을 더하고,  $(C_i + 1, B_i)$ ,  $(A_i, B_i + 1)$ 에는 -1,  $(C_i + 1, D_i + 1)$ 에 1을 더합니다.

현재 배열

0	0	0	0
0	+1	0	-1
0	0	0	0
0	-1	0	+1

최종 누적합 배열

0	0	0	0
0	+1	+1	0
0	+1	+1	0
0	0	0	0



# Prefix Sum on 2-Dimension

- 더하기만 하는걸  $Q$ 개의 쿼리에 대해 수행하고, 이를 가장 마지막에 누적합 한번만 돌리면 해결됩니다.

# Challenge

- 문제를 풀어봅시다.
- [https://atcoder.jp/contests/tessoku-book/tasks/tessoku\\_book\\_j](https://atcoder.jp/contests/tessoku-book/tasks/tessoku_book_j)

あるリゾートホテルには, 1号室から  $N$  号室までの  $N$  個の部屋があります.  $i$  号室は  $A_i$  人部屋です. このホテルでは  $D$  日間にわたって工事が行われることになっており,  $d$  日目は  $L_d$  号室から  $R_d$  号室までの範囲を使うことができません.  $d = 1, 2, \dots, D$  について,  $d$  日目に使える中で最も大きい部屋は何人部屋であるか, 出力するプログラムを作成してください.

- 어느 한 리조트호텔에선, 1호실부터  $N$ 호실까지  $N$ 개의 방이 있습니다.  $i$  호실은  $A_i$ 인이 묵을 수 있습니다.
- 이 호텔에선  $D$ 일에 걸쳐 공사가 진행될 예정인데,  $d$ 일째에는  $L_d$ 호실부터  $R_d$ 호실까지 사용할 수 없습니다.
- $d = 1, 2, \dots, D$ 에 대해,  $d$ 일째에 사용가능한 방 중에서 한 방에 묵을 수 있는 최대 인원의 수를 출력하는 프로그램을 작성하시오.
- 출력 :  $d = 1, 2, \dots, D$ 에 대해 정답을 한 줄에 하나씩 출력하라.

# Challenge

- (힌트) 누적합을 누적최대값이라고 하면 어떻게 될까요?
  - 억지로 용어를 만들긴 했지만 한번 생각해 보세요.

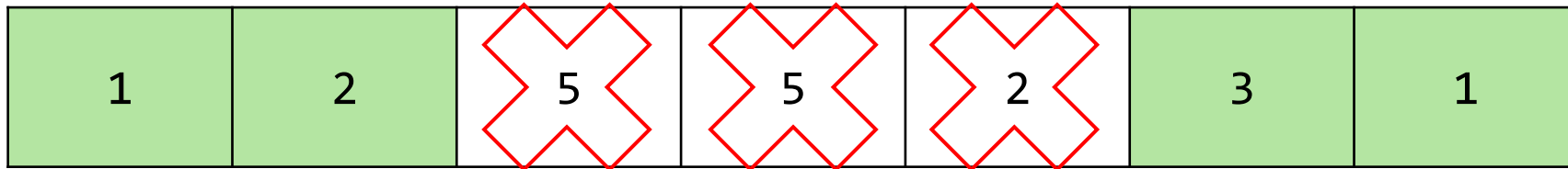
# Challenge

- $L_1 = 3, R_1 = 5$

1	2	5	5	2	3	1
---	---	---	---	---	---	---

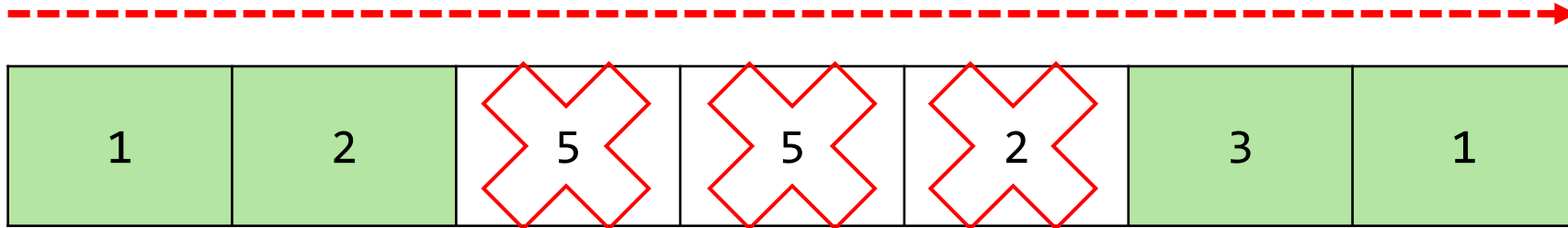
# Challenge

- $L_1 = 3, R_1 = 5$



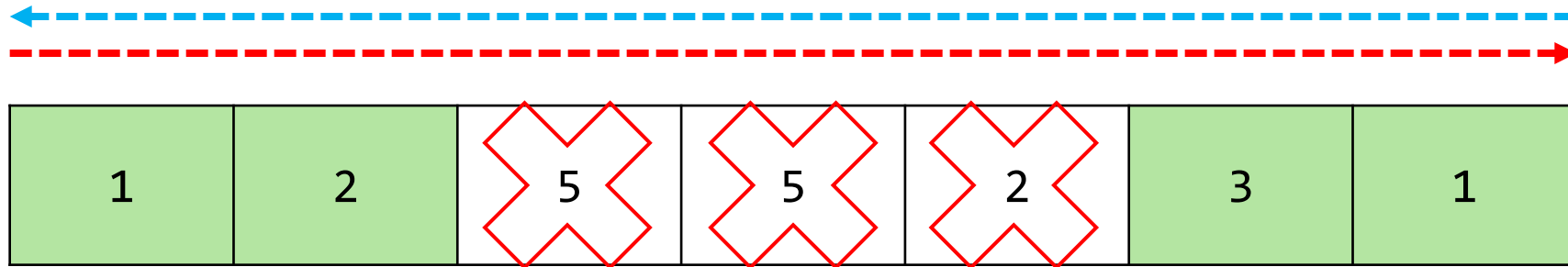
# Challenge

- $L_1 = 3, R_1 = 5$ 
  - 기존 누적합의 경우 이 방향으로만 누적합을 계산해보았습니다.
  - 오른쪽 [3, 1]에 대해선 어떻게 적용할 수 있을까요?



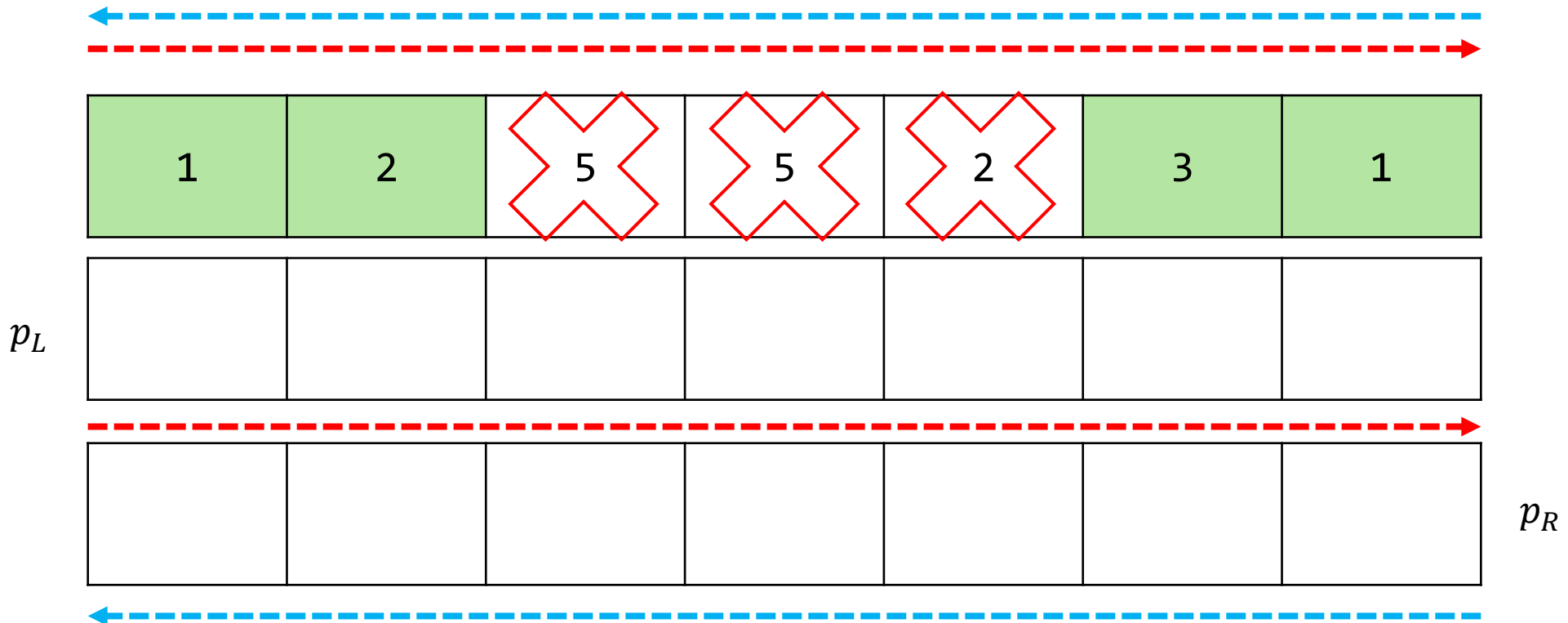
# Challenge

- $L_1 = 3, R_1 = 5$ 
  - 화살표를 뒤집어봅시다.



# Challenge

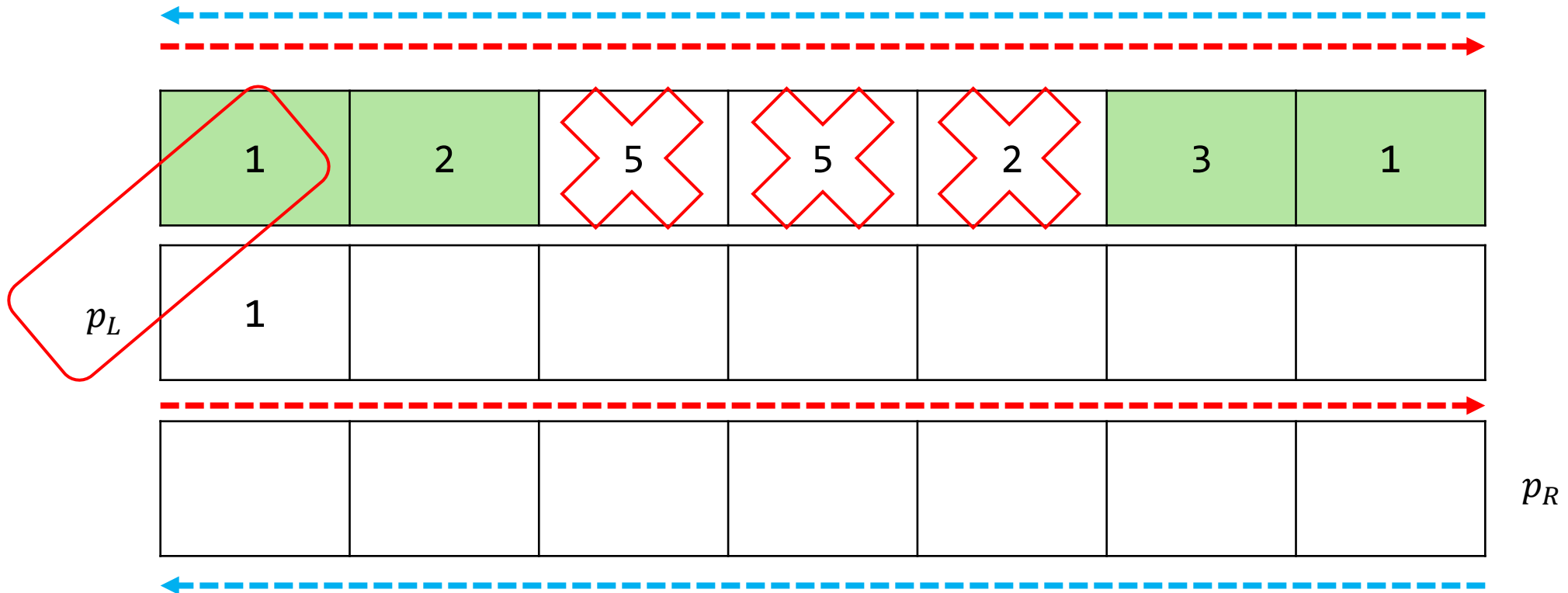
- $L_1 = 3, R_1 = 5$ 
  - '누적최대값'의 정의를  $p_L[cur] = \max(p[cur - 1], arr[cur])$ 로 정의할 수 있습니다. (빨간색 화살표)
  - 반대로는  $p_R[cur] = \max(p[cur + 1], arr[cur])$ 로 정의할 수 있습니다. (파란색 화살표)
  - $p$ 는 누적최대값 배열,  $arr$ 은 원래 값들이 있는 배열입니다.





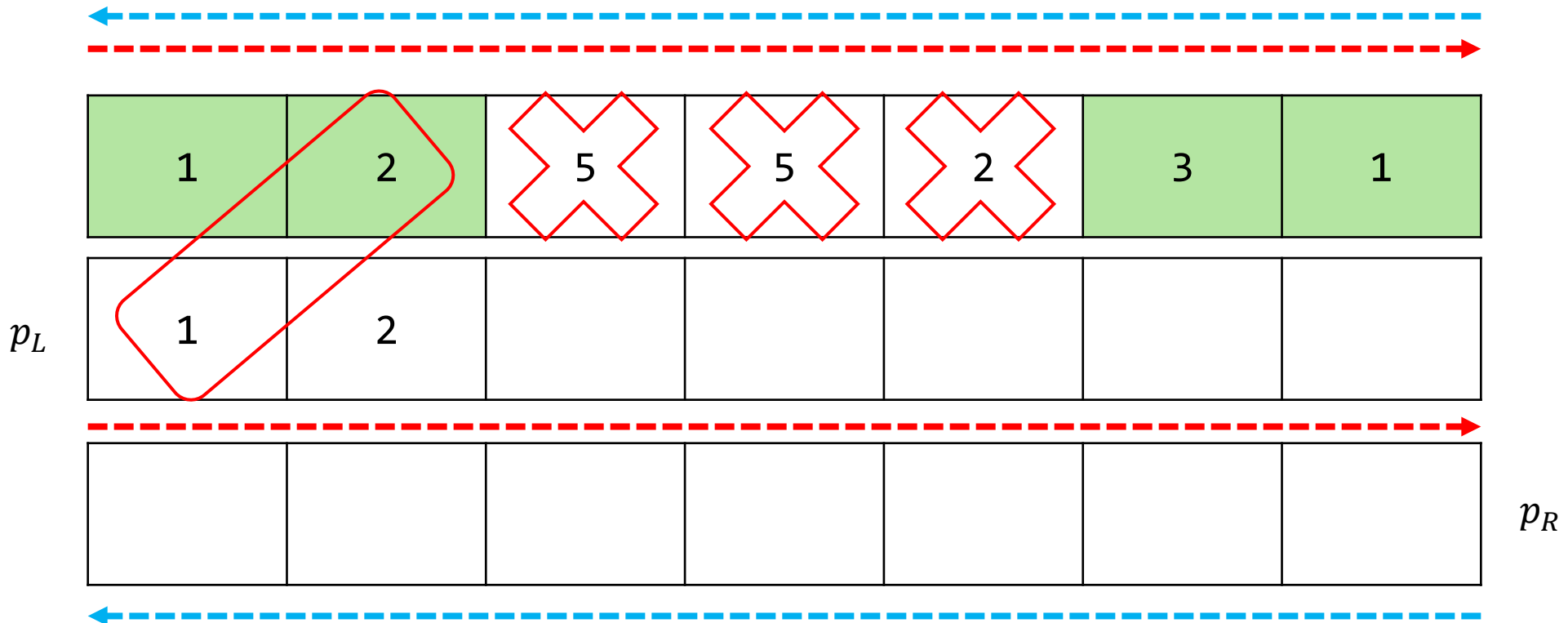
# Challenge

- $L_1 = 3, R_1 = 5$ 
  - '누적최대값'의 정의를  $p_L[cur] = \max(p[cur - 1], arr[cur])$ 로 정의할 수 있습니다. (빨간색 화살표)
  - 반대로는  $p_R[cur] = \max(p[cur + 1], arr[cur])$ 로 정의할 수 있습니다. (파란색 화살표)
  - $p$ 는 누적최대값 배열,  $arr$ 은 원래 값들이 있는 배열입니다.



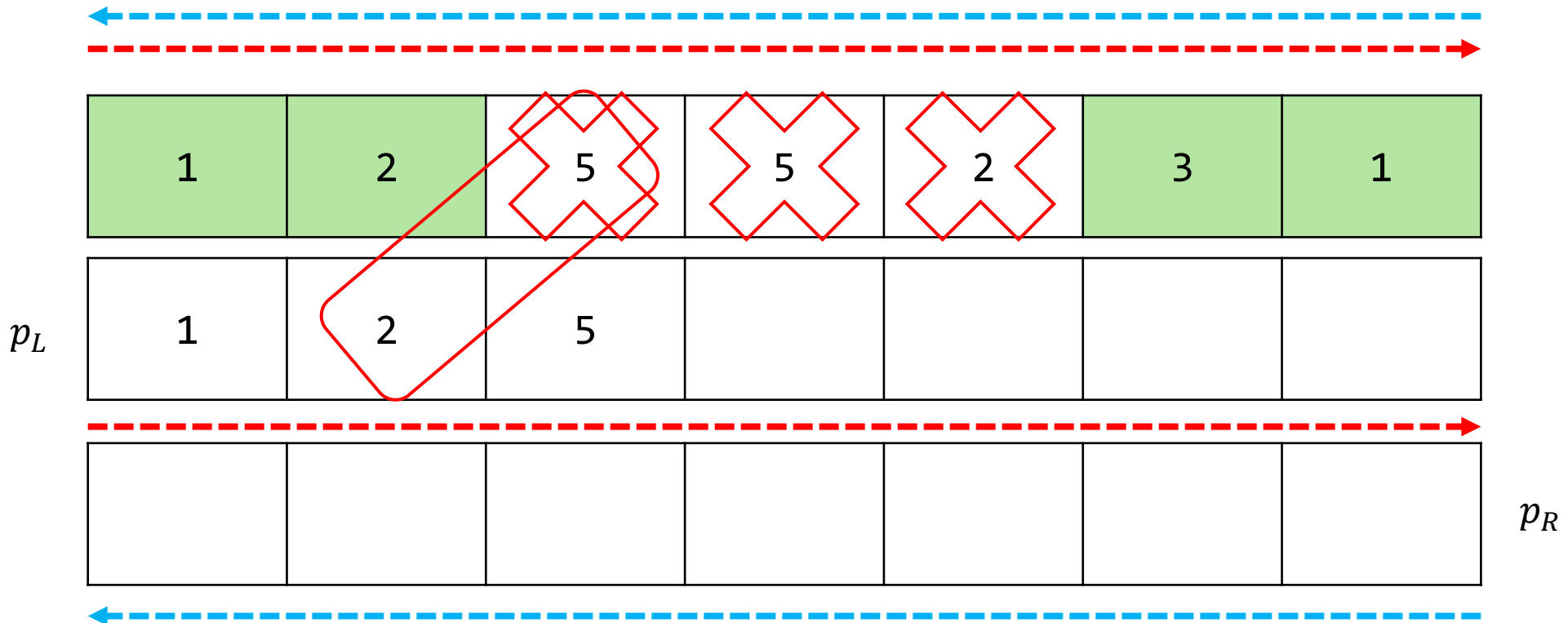
# Challenge

- $L_1 = 3, R_1 = 5$ 
  - '누적최대값'의 정의를  $p_L[cur] = \max(p[cur - 1], arr[cur])$ 로 정의할 수 있습니다. (빨간색 화살표)
  - 반대로는  $p_R[cur] = \max(p[cur + 1], arr[cur])$ 로 정의할 수 있습니다. (파란색 화살표)
  - $p$ 는 누적최대값 배열,  $arr$ 은 원래 값들이 있는 배열입니다.



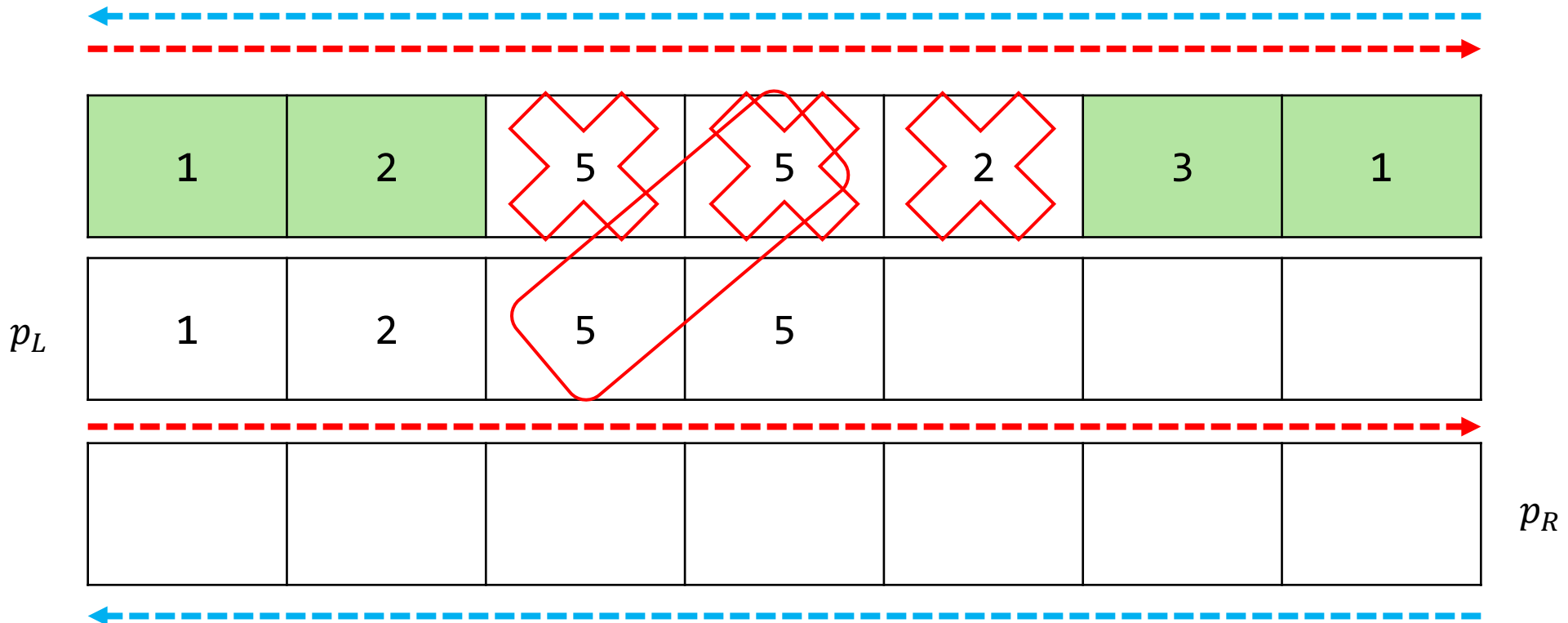
# Challenge

- $L_1 = 3, R_1 = 5$ 
  - '누적최대값'의 정의를  $p_L[cur] = \max(p[cur - 1], arr[cur])$ 로 정의할 수 있습니다. (빨간색 화살표)
  - 반대로는  $p_R[cur] = \max(p[cur + 1], arr[cur])$ 로 정의할 수 있습니다. (파란색 화살표)
  - $p$ 는 누적최대값 배열,  $arr$ 은 원래 값들이 있는 배열입니다.



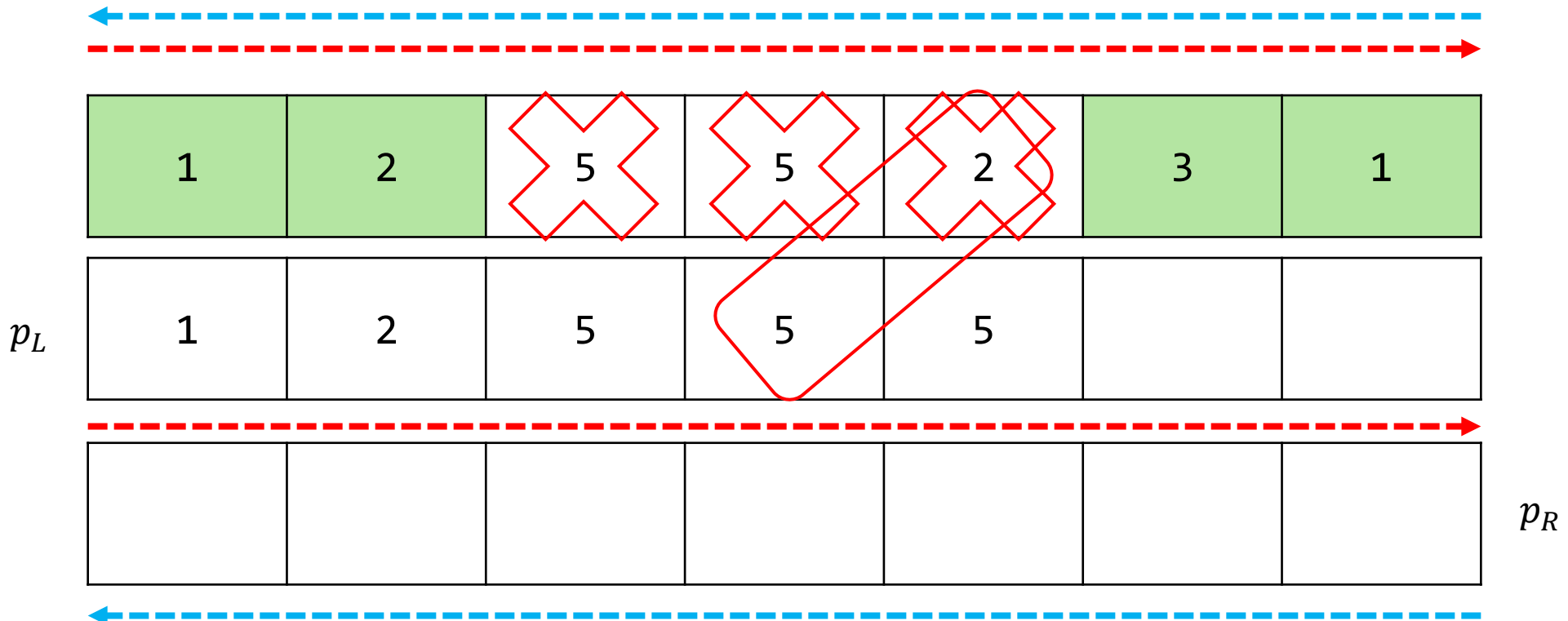
# Challenge

- $L_1 = 3, R_1 = 5$ 
  - '누적최대값'의 정의를  $p_L[cur] = \max(p[cur - 1], arr[cur])$ 로 정의할 수 있습니다. (빨간색 화살표)
  - 반대로는  $p_R[cur] = \max(p[cur + 1], arr[cur])$ 로 정의할 수 있습니다. (파란색 화살표)
  - $p$ 는 누적최대값 배열,  $arr$ 은 원래 값들이 있는 배열입니다.



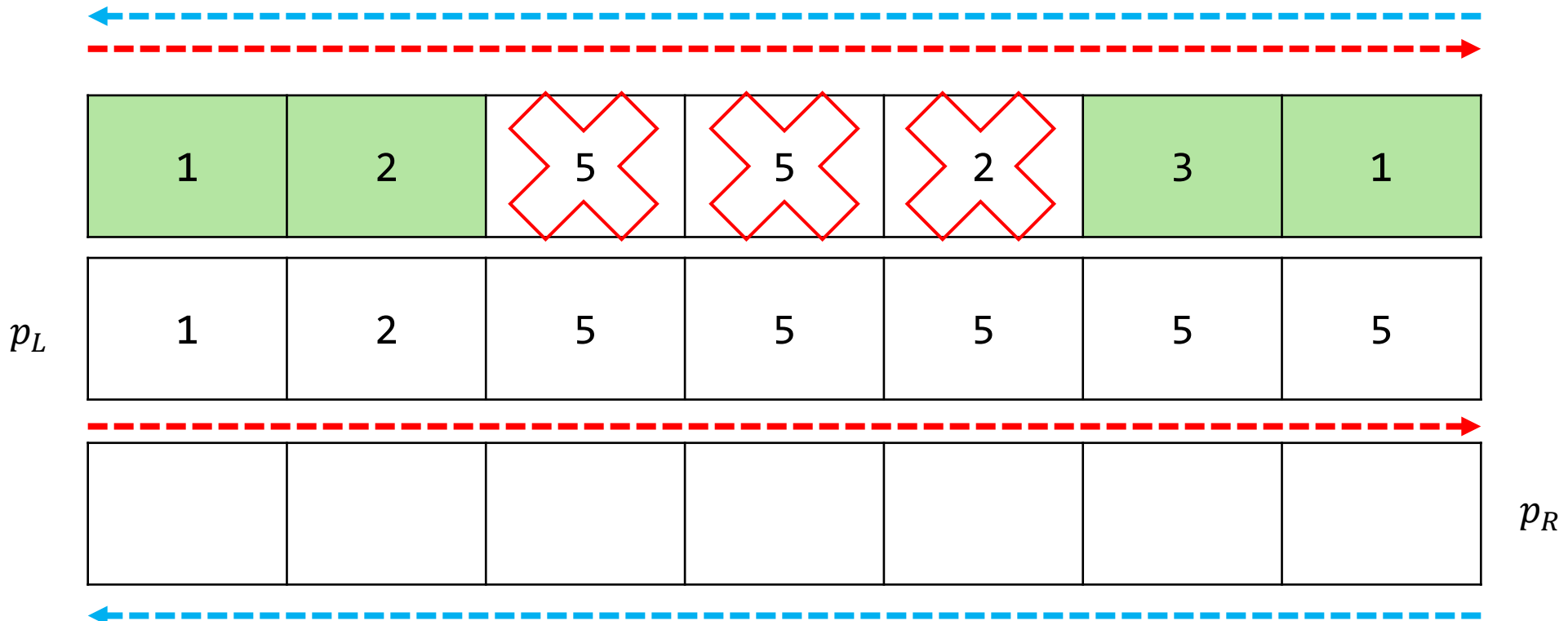
# Challenge

- $L_1 = 3, R_1 = 5$ 
  - '누적최대값'의 정의를  $p_L[cur] = \max(p[cur - 1], arr[cur])$ 로 정의할 수 있습니다. (빨간색 화살표)
  - 반대로는  $p_R[cur] = \max(p[cur + 1], arr[cur])$ 로 정의할 수 있습니다. (파란색 화살표)
  - $p$ 는 누적최대값 배열,  $arr$ 은 원래 값들이 있는 배열입니다.



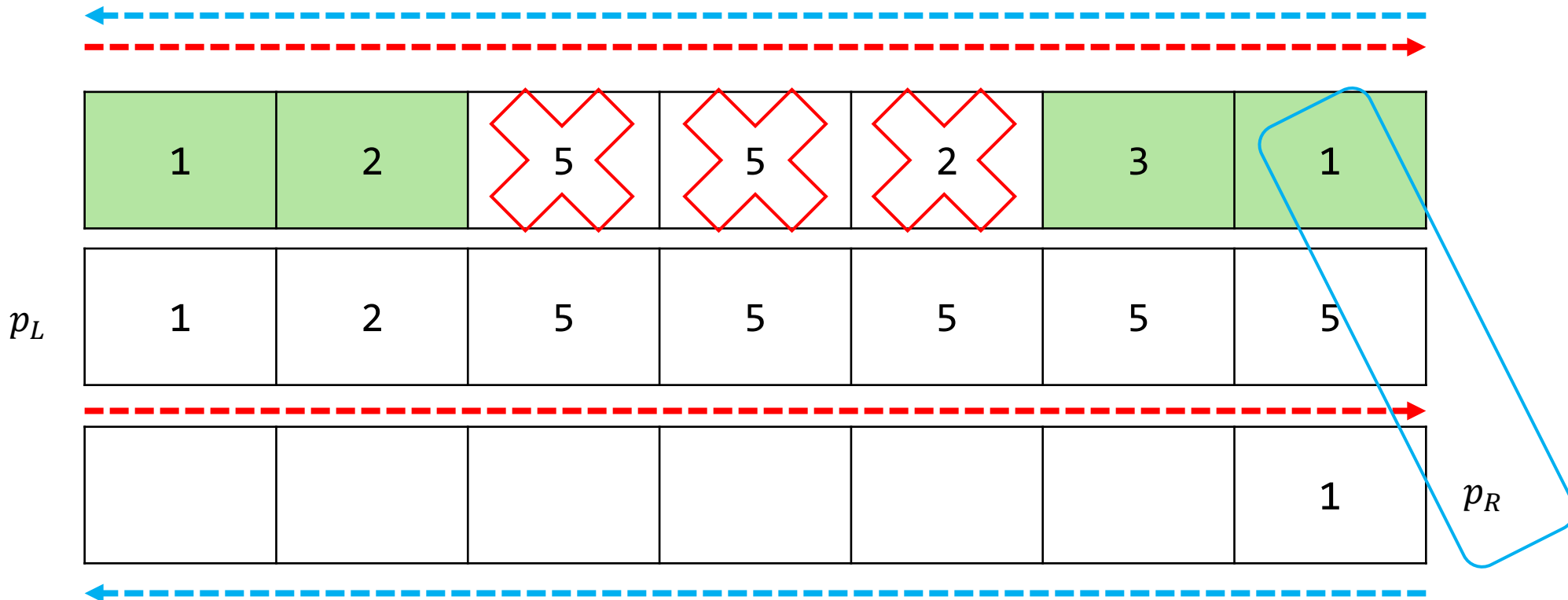
# Challenge

- $L_1 = 3, R_1 = 5$ 
  - '누적최대값'의 정의를  $p_L[cur] = \max(p[cur - 1], arr[cur])$ 로 정의할 수 있습니다. (빨간색 화살표)
  - 반대로는  $p_R[cur] = \max(p[cur + 1], arr[cur])$ 로 정의할 수 있습니다. (파란색 화살표)
  - $p$ 는 누적최대값 배열,  $arr$ 은 원래 값들이 있는 배열입니다.



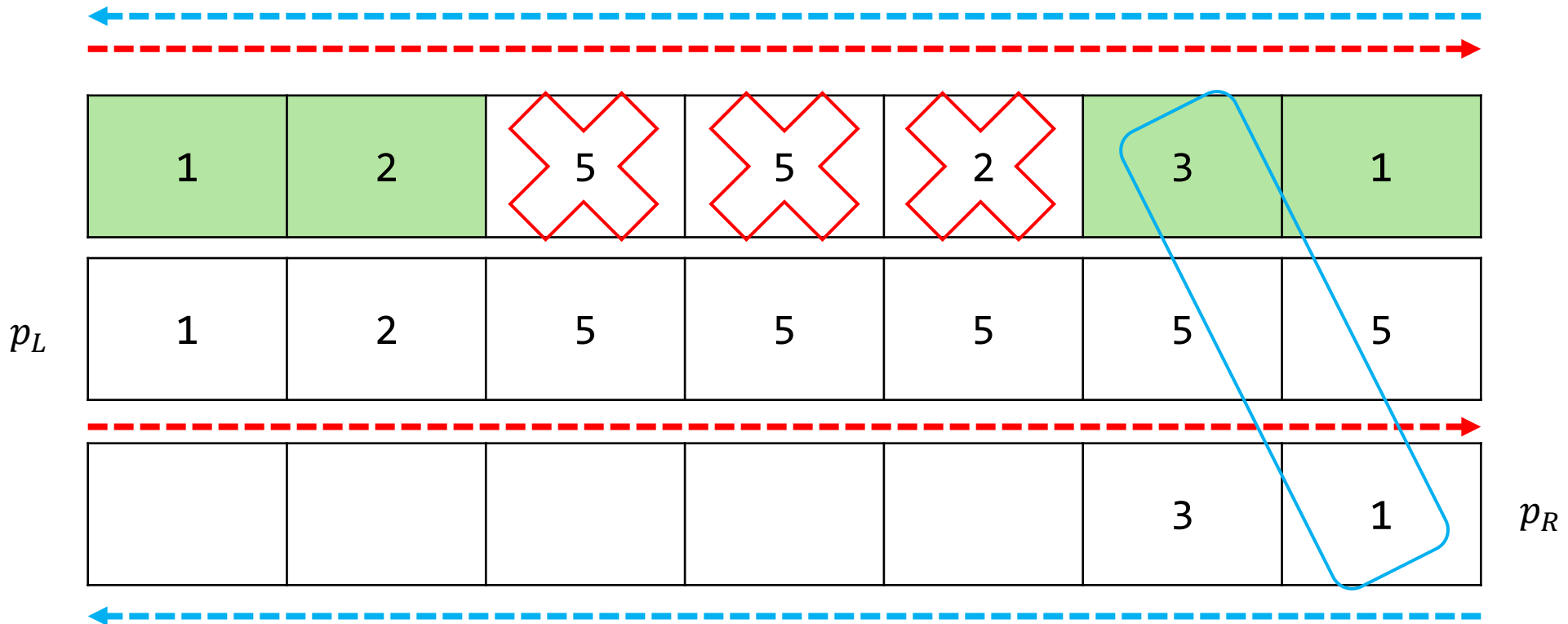
# Challenge

- $L_1 = 3, R_1 = 5$ 
  - '누적최대값'의 정의를  $p_L[cur] = \max(p[cur - 1], arr[cur])$ 로 정의할 수 있습니다. (빨간색 화살표)
  - 반대로는  $p_R[cur] = \max(p[cur + 1], arr[cur])$ 로 정의할 수 있습니다. (파란색 화살표)
  - $p$ 는 누적최대값 배열,  $arr$ 은 원래 값들이 있는 배열입니다.



# Challenge

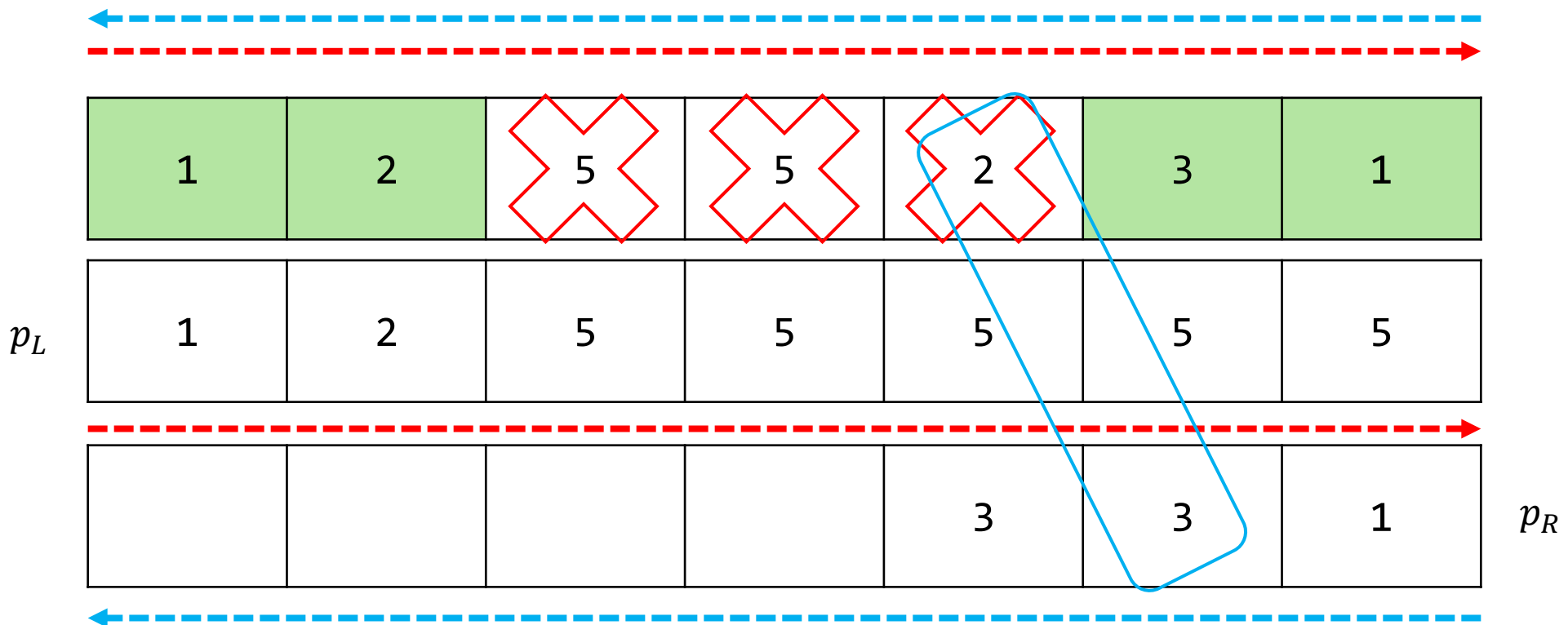
- $L_1 = 3, R_1 = 5$ 
  - '누적최대값'의 정의를  $p_L[cur] = \max(p[cur - 1], arr[cur])$ 로 정의할 수 있습니다. (빨간색 화살표)
  - 반대로는  $p_R[cur] = \max(p[cur + 1], arr[cur])$ 로 정의할 수 있습니다. (파란색 화살표)
  - $p$ 는 누적최대값 배열,  $arr$ 은 원래 값들이 있는 배열입니다.





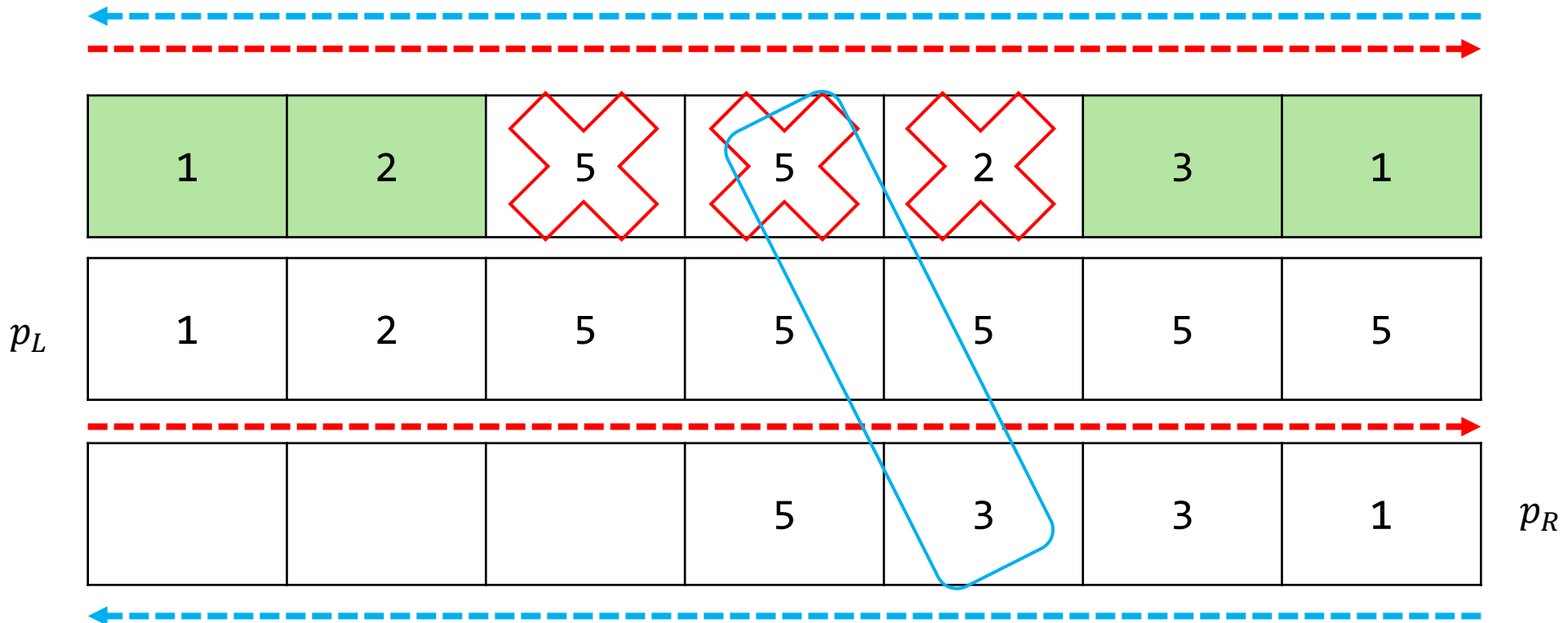
# Challenge

- $L_1 = 3, R_1 = 5$ 
  - '누적최대값'의 정의를  $p_L[cur] = \max(p[cur - 1], arr[cur])$ 로 정의할 수 있습니다. (빨간색 화살표)
  - 반대로는  $p_R[cur] = \max(p[cur + 1], arr[cur])$ 로 정의할 수 있습니다. (파란색 화살표)
  - $p$ 는 누적최대값 배열,  $arr$ 은 원래 값들이 있는 배열입니다.



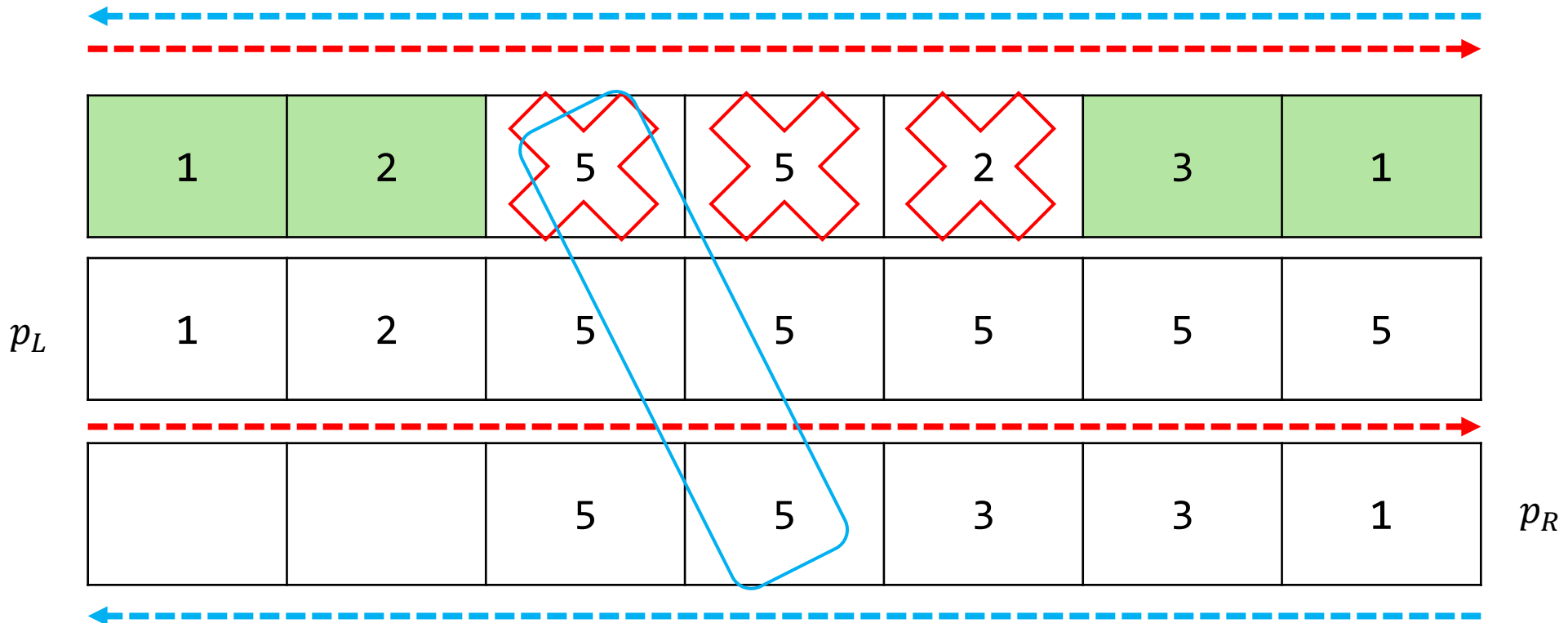
# Challenge

- $L_1 = 3, R_1 = 5$ 
  - '누적최대값'의 정의를  $p_L[cur] = \max(p[cur - 1], arr[cur])$ 로 정의할 수 있습니다. (빨간색 화살표)
  - 반대로는  $p_R[cur] = \max(p[cur + 1], arr[cur])$ 로 정의할 수 있습니다. (파란색 화살표)
  - $p$ 는 누적최대값 배열,  $arr$ 은 원래 값들이 있는 배열입니다.



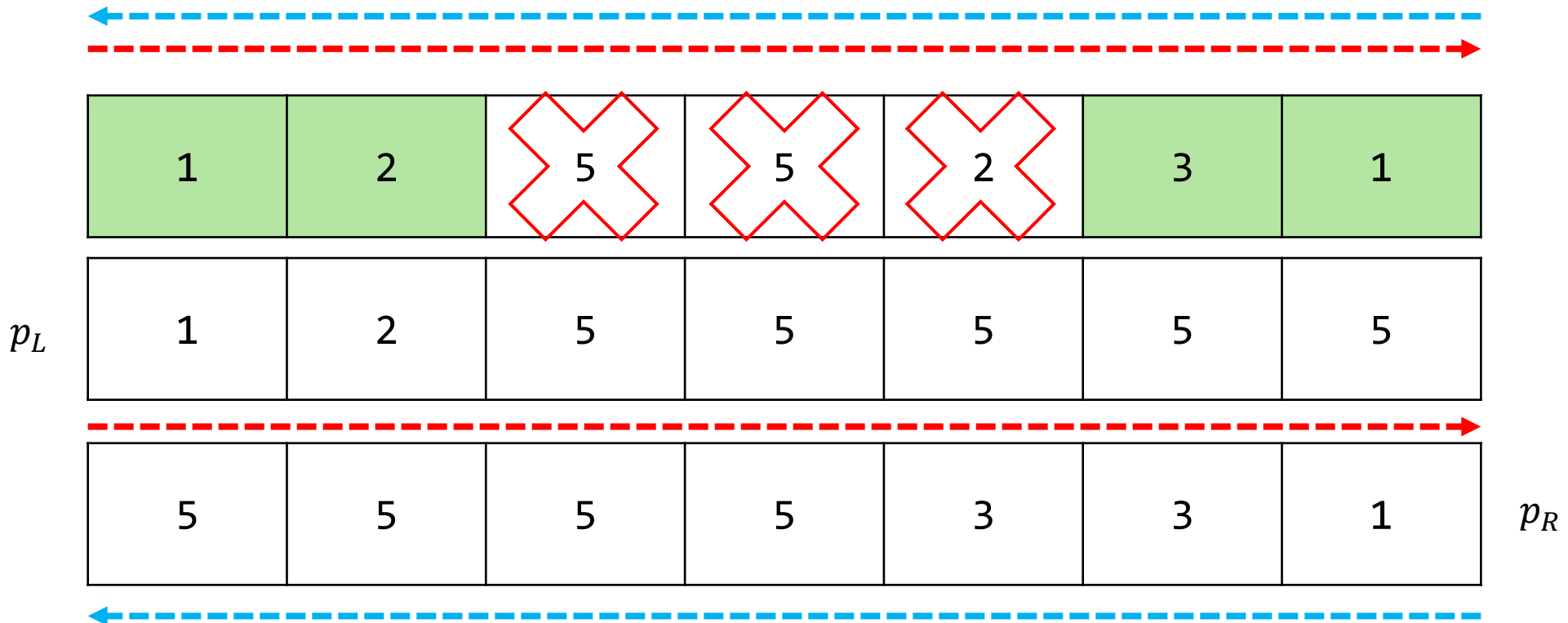
# Challenge

- $L_1 = 3, R_1 = 5$ 
  - '누적최대값'의 정의를  $p_L[cur] = \max(p[cur - 1], arr[cur])$ 로 정의할 수 있습니다. (빨간색 화살표)
  - 반대로는  $p_R[cur] = \max(p[cur + 1], arr[cur])$ 로 정의할 수 있습니다. (파란색 화살표)
  - $p$ 는 누적최대값 배열,  $arr$ 은 원래 값들이 있는 배열입니다.



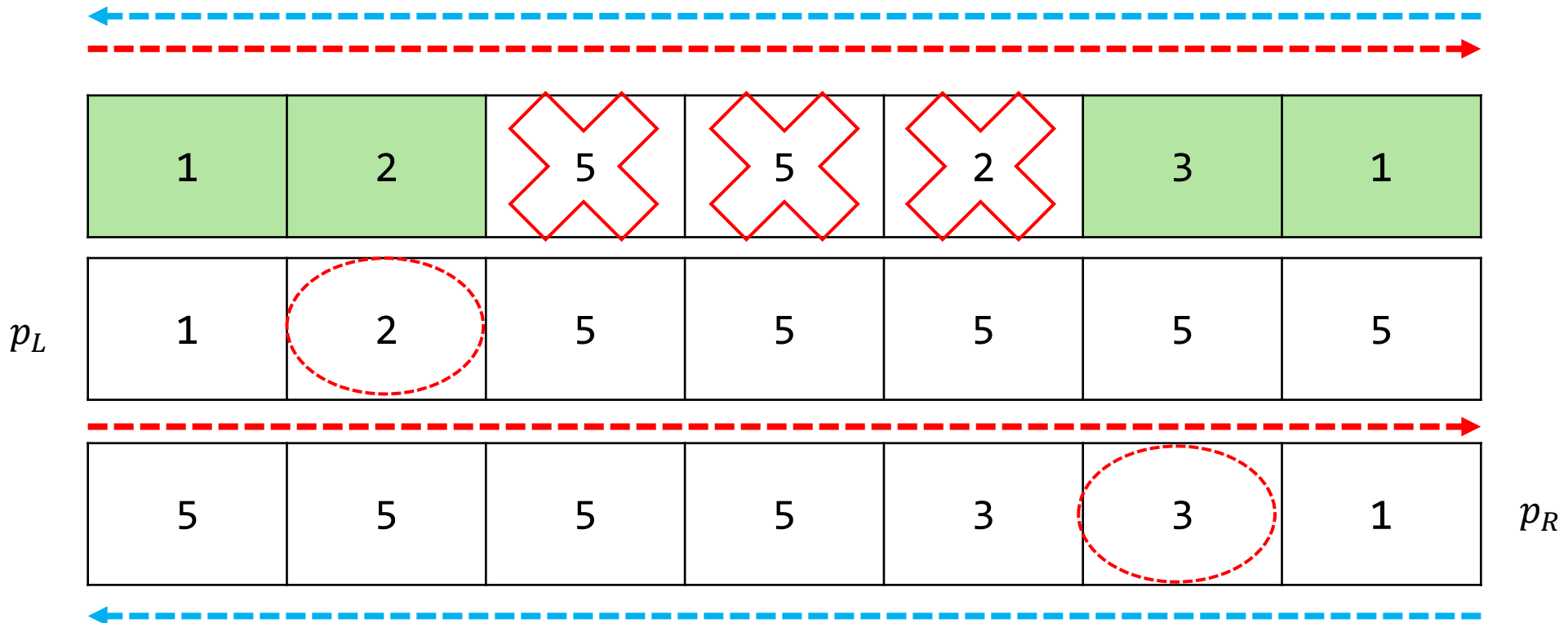
# Challenge

- $L_1 = 3, R_1 = 5$ 
  - '누적최대값'의 정의를  $p_L[cur] = \max(p[cur - 1], arr[cur])$ 로 정의할 수 있습니다. (빨간색 화살표)
  - 반대로는  $p_R[cur] = \max(p[cur + 1], arr[cur])$ 로 정의할 수 있습니다. (파란색 화살표)
  - $p$ 는 누적최대값 배열,  $arr$ 은 원래 값들이 있는 배열입니다.



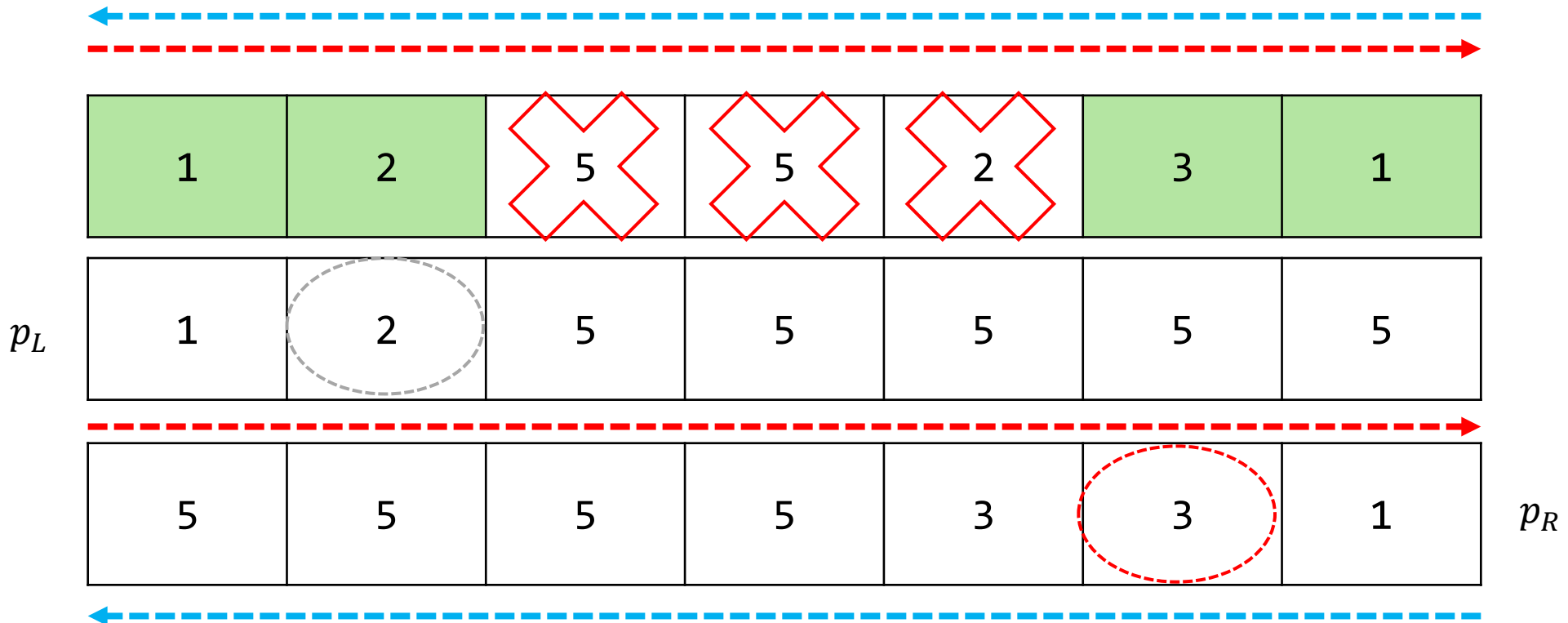
# Challenge

- $L_1 = 3, R_1 = 5$ 
  - 이제  $L$ 값에 대해선  $p_L[L - 1]$ 의 값이 가장 최대이고,  $R$ 값에 대해선  $p_R[R + 1]$ 이  $[L, R]$ 을 제외한 구간의 최댓값 후보가 됩니다.



# Challenge

- $L_1 = 3, R_1 = 5$ 
  - 따라서, 동그라미 친 두 값 중 최댓값을 출력하면 그게 쿼리의 정답이 됩니다.



끝

감사합니다.