

2025 겨울방학 알고리즘 스터디

이분 탐색

컴퓨터 공학과 20230546 서보경

목차

1. 이분 탐색이란?
2. 이분 탐색 응용 – 하한과 상한
3. 이분 탐색 응용 – 매개 변수 탐색

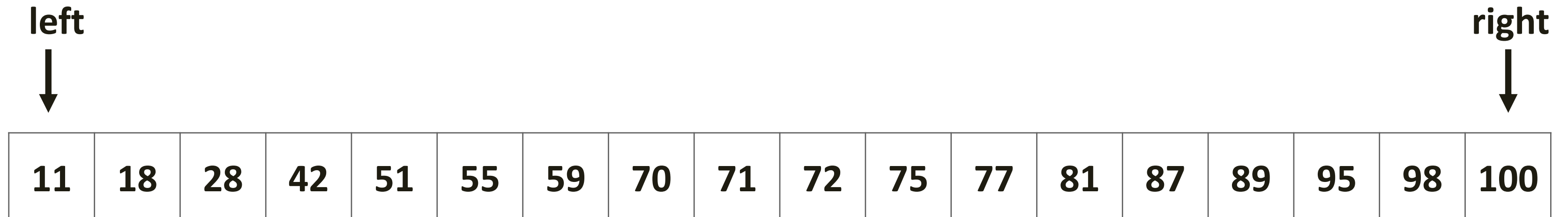
이분 탐색이란?

이분 탐색(Binary Search)은 정렬된 데이터에서 원하는 값을 빠르게 찾는 탐색 알고리즘이다. 가장 큰 특징은 탐색 범위를 절반씩 줄이며 값을 찾는다는 점이다. 항상 중앙값(mid)을 기준으로 비교하여 탐색 범위를 조정하며, 시간 복잡도는 $O(\log n)$ 으로 매우 효율적이다. 이분 탐색은 단순한 값 찾기뿐만 아니라 하한(lower Bound), 상한(upper Bound), 최적화 문제 해결 등 다양한 방식으로 활용된다.



일반적인 이분 탐색의 형태

이분 탐색



먼저 이분 탐색을 위한 선결 조건이 있다. 탐색을 하려는 대상이 단조성을 유지해야 한다는 점이다.

가령, 배열에서 이분 탐색을 하려고 하면 오름 / 내림 차순으로 정렬이 되어 있어야 한다.

배열을 정렬하고 left포인터를 왼쪽 끝, right포인터를 오른쪽 끝에 배치시킨다.

이분 탐색

left ↓																		right ↓	
11	18	28	42	51	55	59	70	71	72	75	77	81	87	89	95	98	100		
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17		

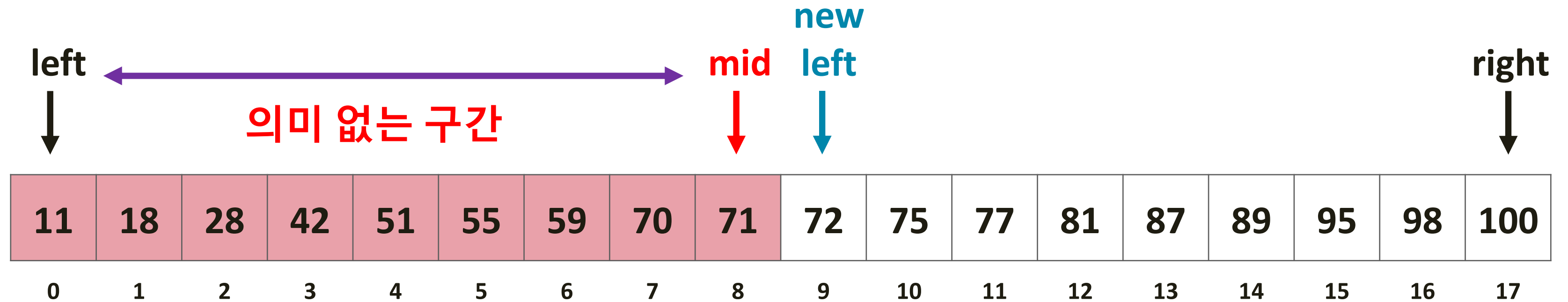
query : search 77 | left : 0, right : 17, mid : 8

이분 탐색은 left와 right의 중간 인덱스로 피벗을 잡고 그 피벗이 찾고자 하는 값보다 작으면 left 포인터를 늘리고, 크다면 right 포인터를 줄이는 방식을 사용한다.

현재 mid는 8이다. 인덱스 8의 값은 71이고, 이는 우리가 찾으려는 값인 77보다 작다.

고로, left 포인터를 늘려준다.

이분 탐색



query : search 77 | left : 0, right : 17, mid : 8

포인터를 옮기기 전에 잘 생각해 두어야 할 게 있다. mid 값이 원하는 값이 아니면 포인터를 움직여야 하는데, 어떻게 움직여야 할까?

답은 간단하다. 현재 mid 포인터가 찾고자 하는 값보다 작다면 mid 이하의 값은 전부 쓸모가 없는 값이 될 것이다. 찾고자 하는 값보다 큰 경우 또한 비슷하다.

고로, left 포인터가 이동할 곳은 $\text{mid} + 1$ 이 되고 right를 움직여야 한다면 $\text{mid} - 1$ 일 것이다.

이분 탐색

									left					mid					right
									↓					↓					↓
11	18	28	42	51	55	59	70	71	72	75	77	81	87	89	95	98	100		
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17		

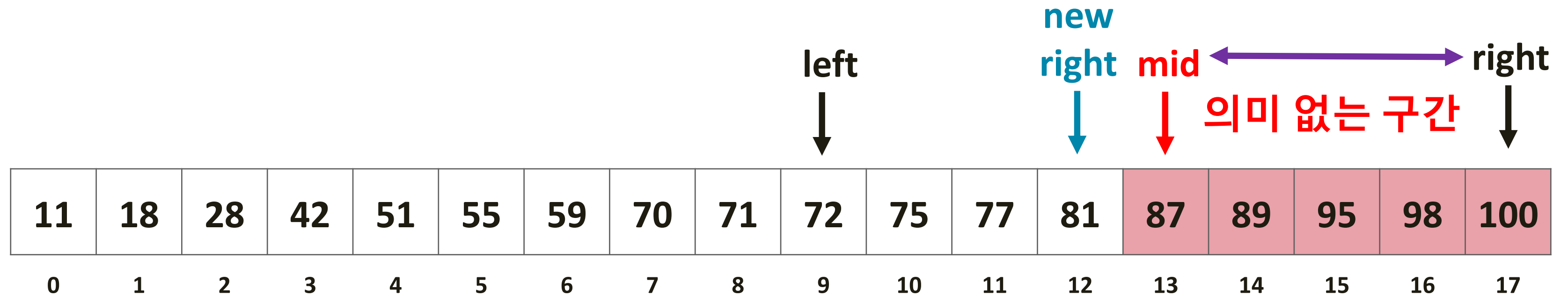
query : search 77 | left : 9, right : 17, mid : 13

현재 left 포인터는 인덱스 9의 위치에 있고 right포인터는 인덱스 17의 위치에 있다.

이때 mid는 13의 위치에 있을 것이다.

배열의 인덱스 13에는 87이라는 값이 있다. 이는 우리가 찾고자 하는 값 보다 크다.

이분 탐색



query : search 77 | left : 9, right : 17, mid : 13

목표 값 보다 mid가 크니 범위의 상한선을 줄여야 한다.

즉 mid부터 현재 right사이에 있는 구간은 의미가 없다는 뜻이다.

right를 mid -1로 바꾸고 탐색을 계속 진행한다.

이분 탐색

									left	mid	right						
									↓	↓	↓						
11	18	28	42	51	55	59	70	71	72	75	77	81	87	89	95	98	100
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

query : search 77 | left : 9, right : 12, mid : 10

현재 left 포인터는 인덱스 9의 위치에 있고 right포인터는 인덱스 12의 위치에 있다.

이때 mid는 10의 위치에 있을 것이다.

배열의 인덱스 10에는 75이라는 값이 있다. 이는 우리가 찾고자 하는 값 보다 작다.

이분 탐색

									left	mid	new left	right					
									↓	↓	↓	↓					
11	18	28	42	51	55	59	70	71	72	75	77	81	87	89	95	98	100
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

query : search 77 | left : 9, right : 12, mid : 10

목표 값 보다 mid가 작으니 범위의 하한을 늘려야 한다.

즉 left부터 현재 mid사이에 있는 구간은 의미가 없다는 뜻이다.

left를 mid + 1로 바꾸고 탐색을 계속 진행한다.

이분 탐색

mid

left right

11	18	28	42	51	55	59	70	71	72	75	77	81	87	89	95	98	100
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

query : search 77 | left : 11, right : 12, mid : 11

현재 left 포인터는 인덱스 11의 위치에 있고 right 포인터는 인덱스 12의 위치에 있다.

이때 mid는 11의 위치에 있을 것이다.

배열의 인덱스 11에는 77이라는 값이 있다. **값을 찾았다! 77은 인덱스 11의 위치에 있다.**

이분 탐색의 응용 - 하한

left

right

11	18	28	42	51	55	59	70	71	72	75	77	81	87	89	95	98	100
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

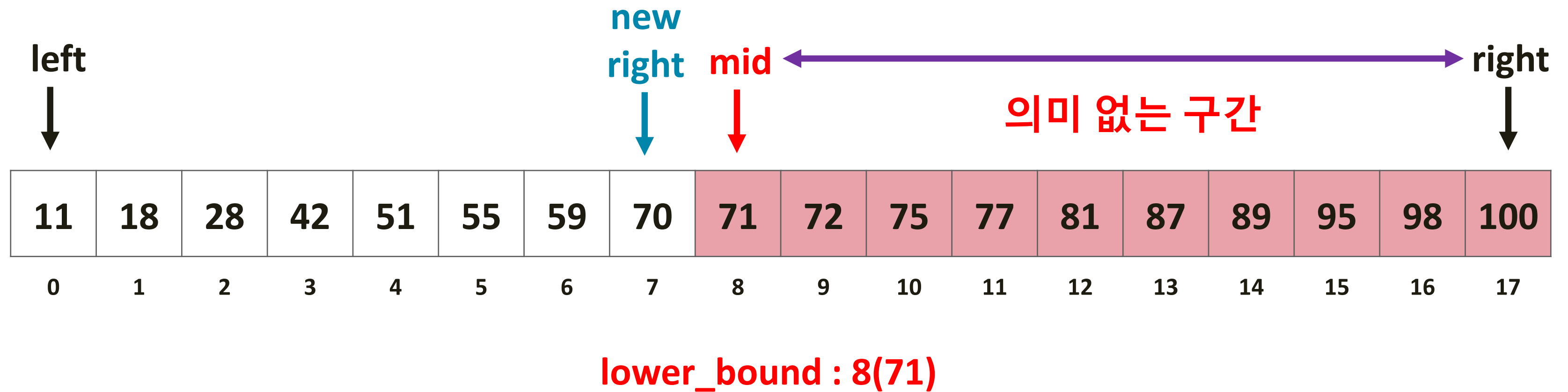
lower_bound : ?

아까와 똑같은 배열을 가지고 왔다.

실제로 많이 쓰고, 코딩 테스트에도 빈번하게 나오는 하한의 개념을 여러분에게 설명하겠다.

이분 탐색에서 하한이란, 어떤 값보다 크거나 같은 값이 처음 나오는 곳을 찾는 방법이다.

이분 탐색의 응용 - 하한



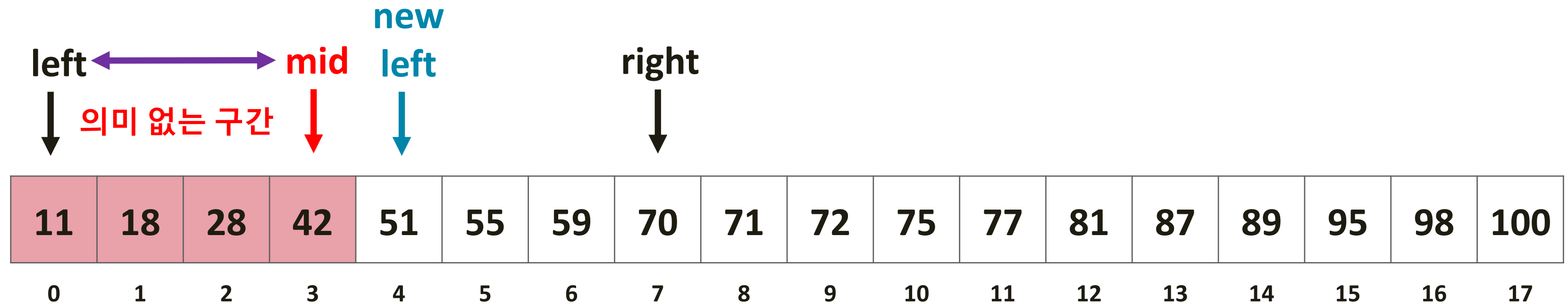
query : lower_bound 62 | left : 0, right : 17, mid : 8

하한 탐색은 이분 탐색이랑 아주 많이 유사하다. 단지 찾고자 하는 값이 조금 다를 뿐이다.

현재 left 포인터는 인덱스 0의 위치에 있고 right 포인터는 인덱스 17의 위치에 있다.
고로 현재 mid는 8의 위치에 있을 것이다. 이는 피벗 값인 62보다 크므로 lower_bound에 8을 저장한다.

더 작은 하한값을 찾기 위해 right 포인터를 mid - 1로 바꿔준다.

이분 탐색의 응용 - 하한



lower_bound : 8(71)

query : lower_bound 62 | left : 0, right : 7, mid : 3

현재 left 포인터는 인덱스 0의 위치에 있고 right포인터는 인덱스 7의 위치에 있다.

고로 현재 mid는 3의 위치에 있을 것이다. 이는 피벗 값인 62보다 작으므로 lower_bound를 업데이트 하지 않는다. (하한이란, 같거나 큰 값이 처음 시작하는 곳이기 때문에)

하한값을 찾기 위해 left포인터를 mid + 1로 바꿔준다.

이분 탐색의 응용 - 하한

				left	mid	new left	right										
				↓	↓	↓	↓										
11	18	28	42	51	55	59	70	71	72	75	77	81	87	89	95	98	100
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

lower_bound : 8(71)

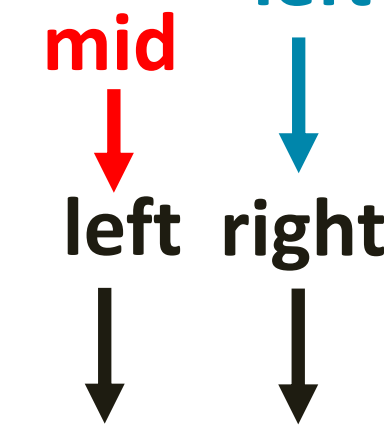
query : lower_bound 62 | left : 4, right : 7, mid : 5

현재 left 포인터는 인덱스 4의 위치에 있고 right포인터는 인덱스 7의 위치에 있다.

고로 현재 mid는 5의 위치에 있을 것이다. 이는 피벗 값인 62보다 작으므로 lower_bound를 업데이트 하지 않는다. (하한이란, 같거나 큰 값이 처음 시작하는 곳이기 때문에)

하한값을 찾기 위해 left포인터를 mid + 1로 바꿔준다.

이분 탐색의 응용 - 하한



The diagram shows the state of a binary search for the lower bound of 62. Above the array, a red arrow labeled 'mid' points to index 6, and a blue arrow labeled 'new left' points to index 7. Below these, black arrows labeled 'left' and 'right' also point to indices 6 and 7 respectively. The array itself contains 18 sorted integers, with the element at index 6 (59) highlighted in pink.

11	18	28	42	51	55	59	70	71	72	75	77	81	87	89	95	98	100
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

lower_bound : 8(71)

query : lower_bound 62 | left : 6, right : 7, mid : 6

현재 left 포인터는 인덱스 6의 위치에 있고 right포인터는 인덱스 7의 위치에 있다.

고로 현재 mid는 6의 위치에 있을 것이다. 이는 피벗 값인 62보다 작으므로 lower_bound를 업데이트 하지 않는다. (하한이란, 같거나 큰 값이 처음 시작하는 곳이기 때문에)

하한값을 찾기 위해 left포인터를 mid + 1로 바꿔준다.

이분 탐색의 응용 - 하한

mid
↓
left
↓
right
↓

11	18	28	42	51	55	59	70	71	72	75	77	81	87	89	95	98	100
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

lower_bound : 7(70)

query : lower_bound 62 | left : 7, right : 7, mid : 7

현재 left 포인터는 인덱스 7의 위치에 있고 right포인터는 인덱스 7의 위치에 있다.

고로 현재 mid는 7의 위치에 있을 것이다. 이는 피벗 값인 62보다 크므로 lower_bound를 7로 업데이트 해준다!

left와 right가 같을 때 까지 이분탐색을 진행 하므로, 이분탐색이 끝났다.

이분 탐색의 응용 - 하한

11	18	28	42	51	55	59	70	71	72	75	77	81	87	89	95	98	100
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

lower_bound : 7(70)

여기서 알 수 있는 점은, 이분 탐색의 탐색 과정으로 인해 자연스럽게 하한을 탐색 할 수 있다는 점이다.

여러분도 아시겠지만 62보다 크거나 같은 원소는 70이다.

상한에 대해서는 특별히 설명을 하지 않겠다. 하지만 상한은 특정 원소보다 무조건 큰 값이 처음 시작되는 곳을 찾게 되는데, 하한에서 보여준 것 처럼 하면 무리 없이 구해질 것이다.

매개변수 탐색

상근이는 나무 M미터가 필요하다. 근처에 나무를 구입할 곳이 모두 망해버렸기 때문에, 정부에 벌목 허가를 요청했다. 정부는 상근이네 집 근처의 나무 한 줄에 대한 벌목 허가를 내주었고, 상근이는 새로 구입한 목재절단기를 이용해서 나무를 구할것이다.

목재절단기는 다음과 같이 동작한다. 먼저, 상근이는 절단기에 높이 H를 지정해야 한다. 높이를 지정하면 톱날이 땅으로부터 H미터 위로 올라간다. 그 다음, 한 줄에 연속해있는 나무를 모두 절단해버린다. 따라서, 높이가 H보다 큰 나무는 H 위의 부분이 잘릴 것이고, 낮은 나무는 잘리지 않을 것이다. 예를 들어, 한 줄에 연속해있는 나무의 높이가 20, 15, 10, 17이라고 하자. 상근이가 높이를 15로 지정했다면, 나무를 자른 뒤의 높이는 15, 15, 10, 15가 될 것이고, 상근이는 길이가 5인 나무와 2인 나무를 들고 집에 갈 것이다. (총 7미터를 집에 들고 간다) 절단기에 설정할 수 있는 높이는 양의 정수 또는 0이다.

상근이는 환경에 매우 관심이 많기 때문에, 나무를 필요한 만큼만 집으로 가져가려고 한다. 이때, 적어도 M미터의 나무를 집에 가져가기 위해서 절단기에 설정할 수 있는 높이의 최댓값을 구하는 프로그램을 작성하시오.

입력

첫째 줄에 나무의 수 N과 상근이가 집으로 가져가려고 하는 나무의 길이 M이 주어진다. ($1 \leq N \leq 1,000,000$, $1 \leq M \leq 2,000,000,000$)

둘째 줄에는 나무의 높이가 주어진다. 나무의 높이의 합은 항상 M보다 크거나 같기 때문에, 상근이는 집에 필요한 나무를 항상 가져갈 수 있다. 높이는 1,000,000,000보다 작거나 같은 양의 정수 또는 0이다.

출력

적어도 M미터의 나무를 집에 가져가기 위해서 절단기에 설정할 수 있는 높이의 최댓값을 출력한다.

??? 이거랑 이분 탐색이 무슨 관련이 있는 거지?
어디 한번 보자.. 일단 내가 자를 높이를 정해야 하는데 높이를 정해서 브루트 포스를 돌릴까? 근데 가져가려는 나무의 길이가 20억까지 되네. $20억 * 100만 = ?????$ 이거 절대 안돌아가지 않나?

-> 이분 탐색으로 설정 높이를 구하면 되지 않을까? 만약 그 설정 높이로 원하는 만큼 잘려지면 더 높은 높이를 보면 되고, 안 잘리면 더 낮은 높이를 보면 되잖아!

매개변수 탐색

left : 0

right : 46

mid : 23

need = 20

answer = ?

높이	4	42	40	26	46
잘린 후	-	-	-	-	-

먼저 left와 right를 정하자.

설정할 수 있는 높이의 하한선은 0이다. 또한 상한선은 46억이다. (나무 길이 최대치가 46)

고로 left를 0으로, right를 46으로 맞춰준다. mid 값은 두 값을 더한 것의 반이므로 23이다.

필요한 나무의 길이는 20이므로 만약 이 조건을 충족하지 못하면 범위를 바꿔야 할 것이다.

매개변수 탐색

left : 0 -> 24

right : 46

mid : 23 -> 35

need = 20

answer = 23

높이	4	42	40	26	46
얻는 양	0	19	17	3	23

총합 : 62

mid = 23

현재 mid는 23이다. 모든 나무에 대해서 이 높이로 잘라보자.

자른 결과, 총합이 62로 필요한 양(20)보다 훨씬 크다!

당연히 높이를 낮게 둘수록 많은 총합을 얻을 수 있을 것이다. 우리가 원하는 건 높이 설정의 최댓값이기 때문에 현재 mid를 answer에 반영하고 하한(left)를 mid + 1로 늘리자.

매개변수 탐색

left : 24 -> 36

right : 46

mid : 35 -> 41

need = 20

answer = 23 -> 35

높이	4	42	40	26	46
얻는 양	0	7	5	0	11

총합 : 23

mid = 35

현재 mid는 35이다. 모든 나무에 대해서 이 높이로 잘라보자.

자른 결과, 총합이 23로 필요한 양(20)보다 크다!

우리가 원하는 건 높이 설정의 최댓값이기 때문에 현재 mid를 answer에 반영하고 하한(left)를 mid + 1로 늘리자.

매개변수 탐색

left : 24

right : 46 -> 34

mid : 35 -> 29

need = 20

answer = 35

높이	4	42	40	26	46
얻는 양	0	1	0	0	5

총합 : 6

mid = 41

현재 mid는 35이다. 모든 나무에 대해서 이 높이로 잘라보자.

자른 결과, 총합이 6로 필요한 양(20)보다 작다..

mid가 어느 지점을 넘어가니, 갑자기 총합이 필요한 양에 미치지 못하였다. 이때는, 상한을 줄여 더 낮은 구간을 탐색할 수 있게 하여야 한다. answer에 반영하지 말고 상한(right)를 mid -1로 줄이자.

매개변수 탐색

left : 24 -> 30

right : 46

mid : 29 -> 38

need = 20

answer = 35 -> 29

높이	4	42	40	26	46
얻는 양	0	13	11	0	17

총합 : 41

mid = 29

현재 mid는 29이다. 모든 나무에 대해서 이 높이로 잘라보자.

자른 결과, 총합이 41로 필요한 양(20)보다 크다!

우리가 원하는 건 높이 설정의 최댓값이기 때문에 현재 mid를 answer에 반영하고 하한(left)를 mid + 1로 늘리자. answer가 줄어들어도 상관이 없는 이유는 단조성 때문이다. 추후 설명.

매개변수 탐색

left : 30

right : 46 -> 37

mid : 38 -> 33

need = 20

answer = 29

높이	4	42	40	26	46
얻는 양	0	4	2	0	8

총합 : 14

mid = 38

현재 mid는 38이다. 모든 나무에 대해서 이 높이로 잘라보자.

자른 결과, 총합이 14로 필요한 양(20)보다 작다..

현재 mid를 answer에 반영하지 말고 상한(left)를 mid - 1로 줄이자.

매개변수 탐색

left : 30 -> 34

right : 37

mid : 33 -> 35

need = 20

answer = 29 -> 33

높이	4	42	40	26	46
얻는 양	0	9	7	0	13

총합 : 29

mid = 33

현재 mid는 33이다. 모든 나무에 대해서 이 높이로 잘라보자.

자른 결과, 총합이 29로 필요한 양(20)보다 크다!!

우리가 원하는 건 높이 설정의 최댓값이기 때문에 현재 mid를 answer에 반영하고 하한(left)를 mid + 1로 늘리자.

매개변수 탐색

left : 34 -> 36

right : 37

mid : 35 -> 36

need = 20

answer = 33 -> 35

높이	4	42	40	26	46
얻는 양	0	7	5	0	11

총합 : 23

mid = 35

현재 mid는 35이다. 모든 나무에 대해서 이 높이로 잘라보자.

자른 결과, 총합이 23로 필요한 양(20)보다 크다!!

우리가 원하는 건 높이 설정의 최댓값이기 때문에 현재 mid를 answer에 반영하고 하한(left)를 mid + 1로 늘리자.

매개변수 탐색

left : 36 -> 37

right : 37

mid : 36 -> 37

need = 20

answer = 33 -> 36

높이	4	42	40	26	46
얻는 양	0	6	4	0	10

총합 : 20

mid = 36

현재 mid는 36이다. 모든 나무에 대해서 이 높이로 잘라보자.

자른 결과, 총합이 20로 필요한 양(20)보다 같다!!

우리가 원하는 건 높이 설정의 최댓값이기 때문에 현재 mid를 answer에 반영하고 하한(left)를 mid + 1로 늘리자.

매개변수 탐색

left : 37

right : 37

mid : 37

need = 20

answer = 36

높이	4	42	40	26	46
얻는 양	0	5	3	0	9

총합 : 17

mid = 37

현재 mid는 36이다. 모든 나무에 대해서 이 높이로 잘라보자.

자른 결과, 총합이 17로 필요한 양(20)보다 작다..

(left <= right)구문에 의해 right를 줄이는 순간 탐색이 종료되게 된다.

매개변수 탐색

answer = 36

높이	4	42	40	26	46
얻는 양	0	5	3	0	9

총합 : 17

고로 저런 나무들이 주어졌을 때 설정할 수 있는 높이의 최댓값은 36이다.

매개 변수 탐색(Parametric Search)은 어떤 값을 기준으로 설정한 후, 배열을 순회하면서 이분 탐색을 활용하여 결정 문제를 해결하는 기법이다.

매개 변수 탐색 문제를 해결하기 위해선 설정 값이 (여기서는 mid가 설정 높이) 단조성을 가져야 하며, 이를 바탕으로 이분 탐색을 통해 조건을 만족하는 최적의 값을 찾는다.

즉, 설정 값(mid)가 증가할수록(혹은 감소할수록) 문제의 정답이 항상 True -> False 혹은 False -> True 형태로 변화해야 이분 탐색을 적용할 수 있다.

이러한 성질을 이용하여 최적의 값을 빠르게 찾을 수 있다.

왜 answer을 낮은 값으로 덮어씌워도 최적해를 찾을까?

매개 변수 탐색에서는 어떤 값 x 가 특정 조건을 만족하는지 여부를 확인하는 **결정 문제**를 해결한다.
일반적으로 이분 탐색을 사용하여 조건을 만족하는 **최댓값(또는 최솟값)**을 찾는다.

수식 정의

- $P(x)$: 값 x 가 조건을 만족하는지 여부를 결정하는 함수(결정 함수), $P(x)$ 는 단조성을 만족해야 한다.
- 최댓값을 찾는 경우에서, 어떤 x 에서 $P(x) = \text{True}$ 이면, $x' < x$ 인 모든 x' 에 대하여 $P(x') = \text{True}$ 가 성립한다.
- 만약, 어떤 x 에서 $P(x) = \text{False}$ 이면, $x' > x$ 인 모든 x' 에 대하여 $P(x') = \text{False}$ 가 성립한다.
- 즉, $P(x)$ 는 $\text{True} \rightarrow \text{False}$ 로 한 방향으로만 바뀌는 단조 함수여야 한다.

약식 증명

answer가 기존보다 작은 값으로 갱신될 가능성

- 이분탐색에서는 mid 가 조건을 만족하면 $\text{answer} = \text{mid}$ 로 갱신한 뒤 더 큰 값을 탐색한다. ($\text{left} = \text{mid} + 1$). 하지만 나중에 더 큰 mid' 를 선택했을 때 조건을 만족하지 않으면, $\text{right} = \text{mid}' - 1$ 이 되면서 탐색 범위가 줄어든다. 즉, 조건을 만족하는 값들 중 가장 큰 값을 찾으려 했지만, 다시 작은 값으로 돌아올 수 있다.

answer가 최적해임을 증명

- 우리가 찾으려는 최적의 x 를 x^* 라고 하자.
- 최댓값을 찾는 경우, $P(x)$ 는 $\text{True} \rightarrow \text{False}$ 로 변화하므로, 특정 x^* 에서 $P(x^*) = \text{True}$ 이며, 그보다 큰 값들은 $P(x) = \text{False}$ 를 만족해야 한다. ($P(x^*) = \text{True}$, $P(x^* + 1) = \text{False}$)
- 탐색 과정에서 answer 가 더 작은 값으로 갱신되는 경우가 있다고 하더라도, x^* 보다 작은 값에서 $P(x) = \text{True}$ 를 만족하는 가장 큰 값이 answer 로 남는다. 즉, answer 는 $P(\text{answer}) = \text{True}$ 를 만족하는 가장 큰 값이 된다.
- 이분 탐색이 종료될 때 $\text{left} > \text{right}$ 조건이 성립하므로, 탐색 가능한 x 의 최댓값이 answer 에 남는다. 이는 $P(x) = \text{True}$ 인 최대 x 이므로 우리가 찾고자 하는 값과 일치한다.

결론적으로, $\text{answer} = \max\{x \mid P(x) = \text{True}\}$ 을 보장한다.