

분할 정보

최백준 choi@startlink.io

분할 정보

분할 정복

Divide & Conquer

DP

Memorization

- 분할 정복은 문제를 2개 또는 그 이상의 작은 부분 문제로 나눈 다음 푸는 것(분할)
- 푸는 다음에는 다시 합쳐서 정답을 구할 때도 있음 (정복)
- 대표적인 분할 정복 알고리즘
 - 퀵 소트
 - 머지 소트
 - 큰 수 곱셈 (카라추바 알고리즘)
 - FFT

정복 X

분할 정복

Divide & Conquer

- 분할 정복과 다이나믹은
- 문제를 작은 부분 문제로 나눈다는 점은 동일하다
- 분할 정복: 문제가 겹치지 않음
- 다이나믹: 문제가 겹쳐서 겹치는 것을 Memoization으로 해결

이분 탐색

이분 탐색

Binary Search

6

- 정렬되어 있는 리스트에서 어떤 값을 빠르게 찾는 알고리즘
- 리스트의 크기를 N 이라고 했을 때
- $O(\lg N)$ 의 시간이 걸린다.

가장

$\lg N$

$O(N)$

N
 \downarrow
 $1/2$
 2

$2 \leftarrow \frac{2}{4}$
 8

\leftarrow
 1

이분 탐색

Binary Search

• 4를 찾아보자

• $L = 0, R = 8, M = 4$

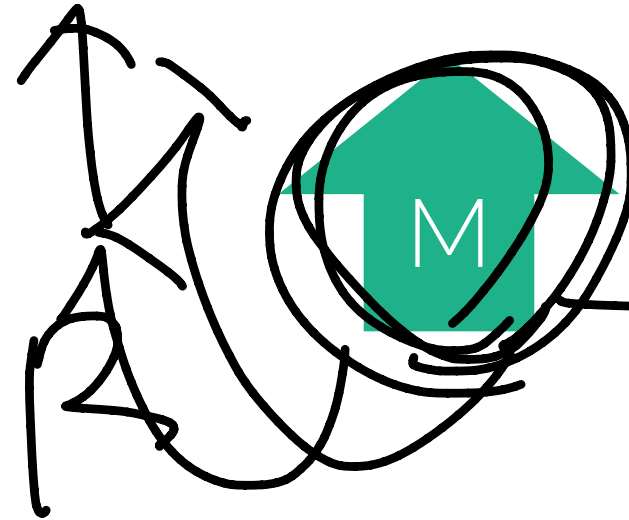
• $4 < 7$ 이기 때문에 왼쪽 ($L \sim M-1$)에 4가 있을 수 있다.

L R

가운데!

$$\frac{L+R}{2} = 4$$

0	1	2	3	4	5	6	7	8
1	3	4	5	7	8	9	20	30

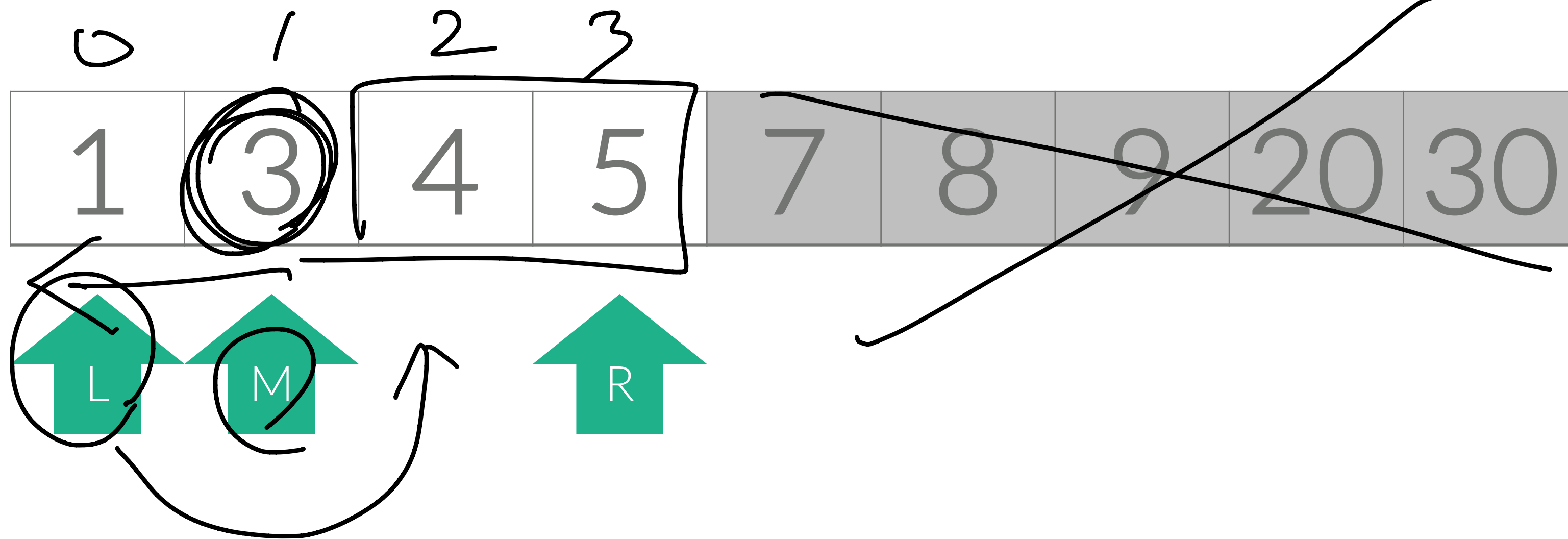


이분 탐색

Binary Search

8

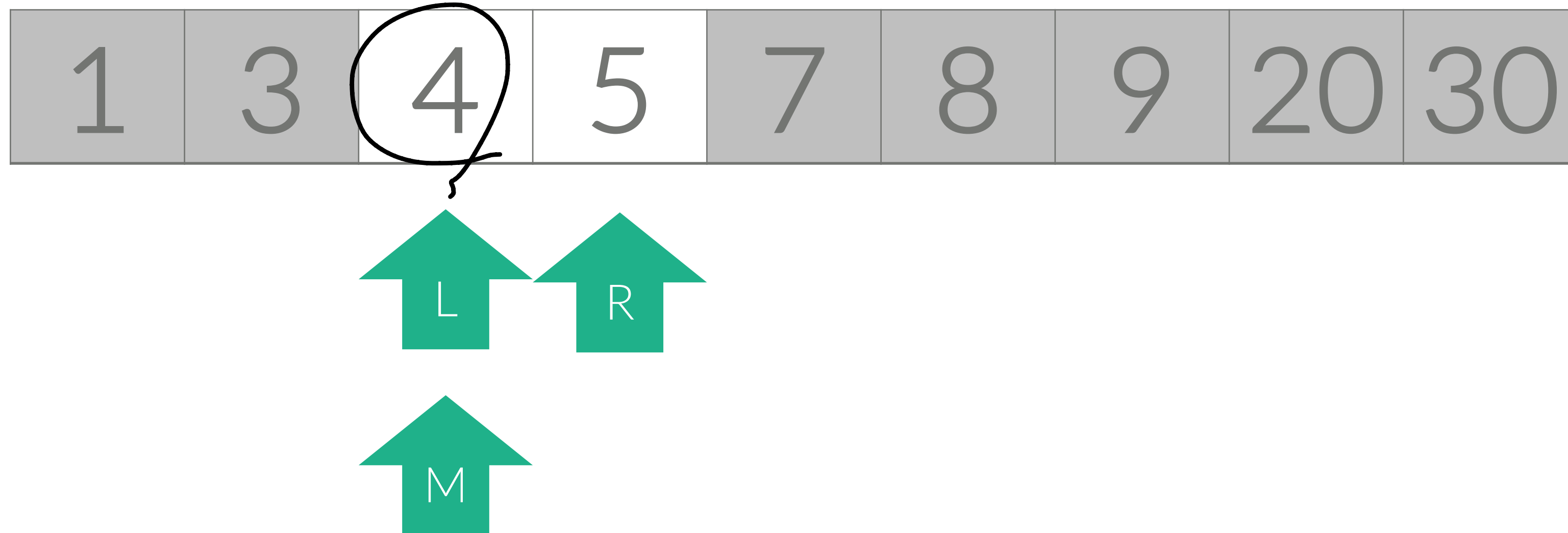
- 4를 찾아보자
- $L = 0, R = 3, M = 1$
- $4 > 3$ 이기 때문에 오른쪽 ($M+1 \sim R$)에 4가 있을 수 있다.



이분 탐색

Binary Search

- 4를 찾아보자
- $L = 2, R = 3, M = 2$
- $4 == 4$ 이다. 4를 찾았다.

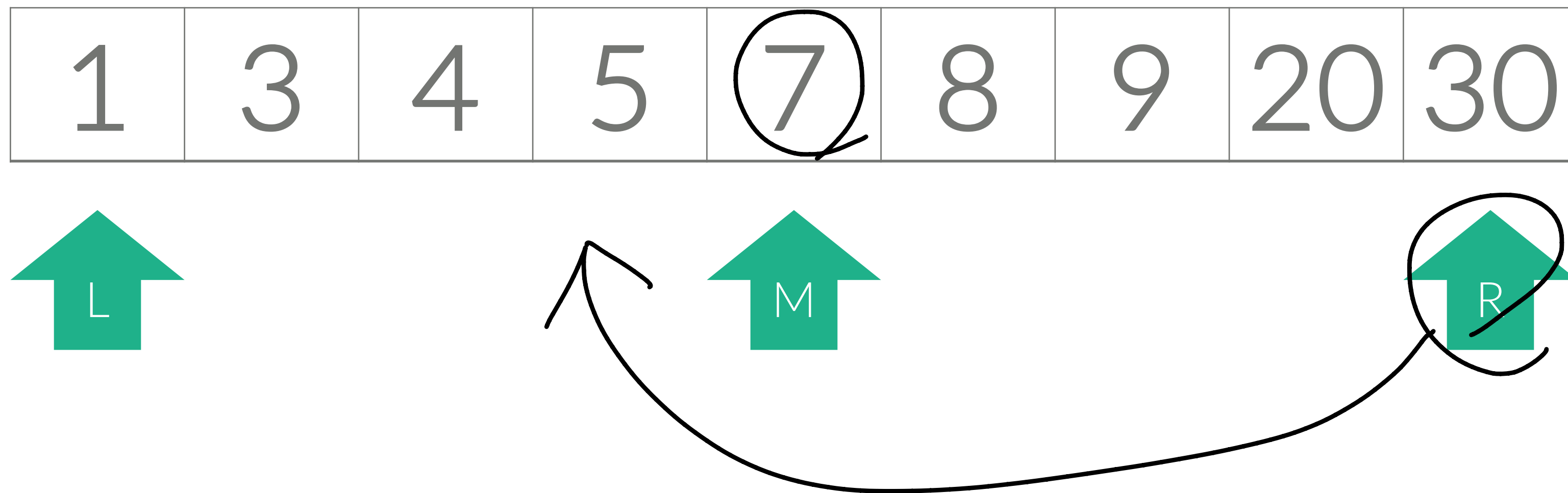


이분 탐색

10

Binary Search

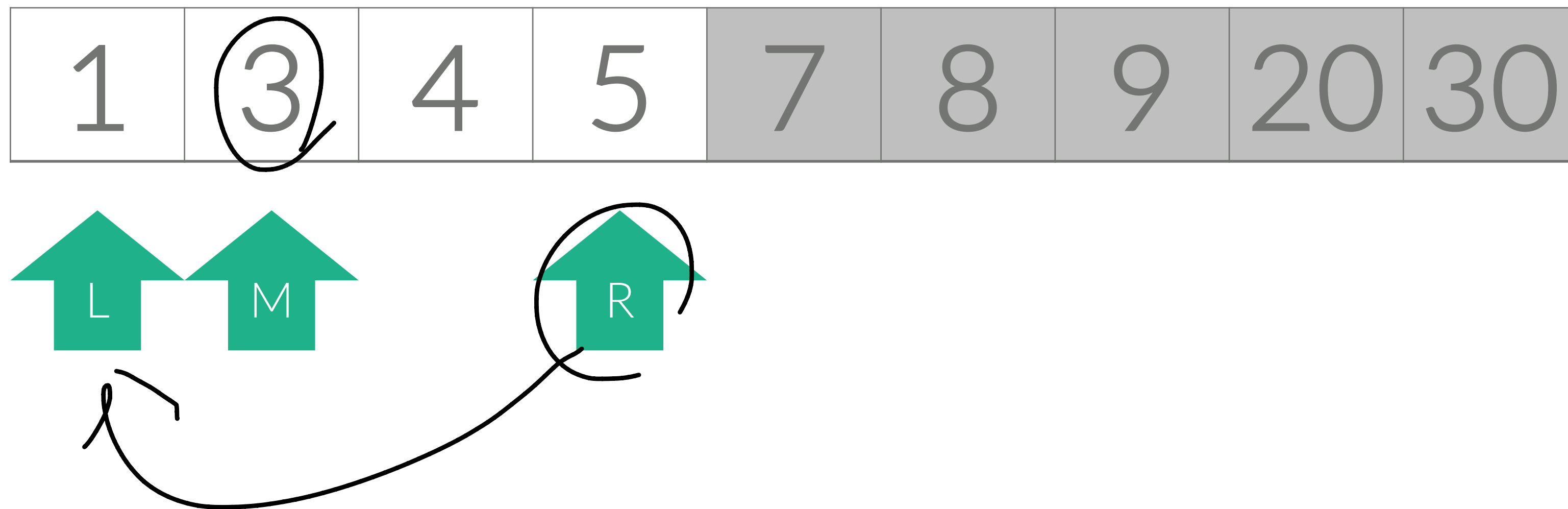
- 2를 찾아보자
- $L = 0, R = 8, M = 4$
- $2 < 7$ 이기 때문에 왼쪽 ($L \sim M-1$)에 4가 있을 수 있다.



이분 탐색

Binary Search

- 2를 찾아보자
- $L = 0, R = 3, M = 1$
- $2 < 3$ 이기 때문에 왼쪽 ($L \sim M-1$)에 2가 있을 수 있다.



이분 탐색

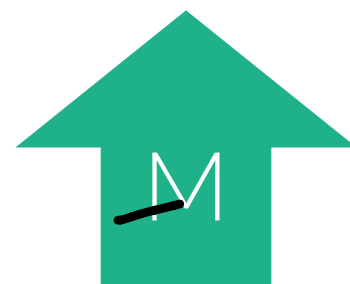
12

Binary Search

• 2를 찾아보자

• $L = 0, R = 0, M = 0$

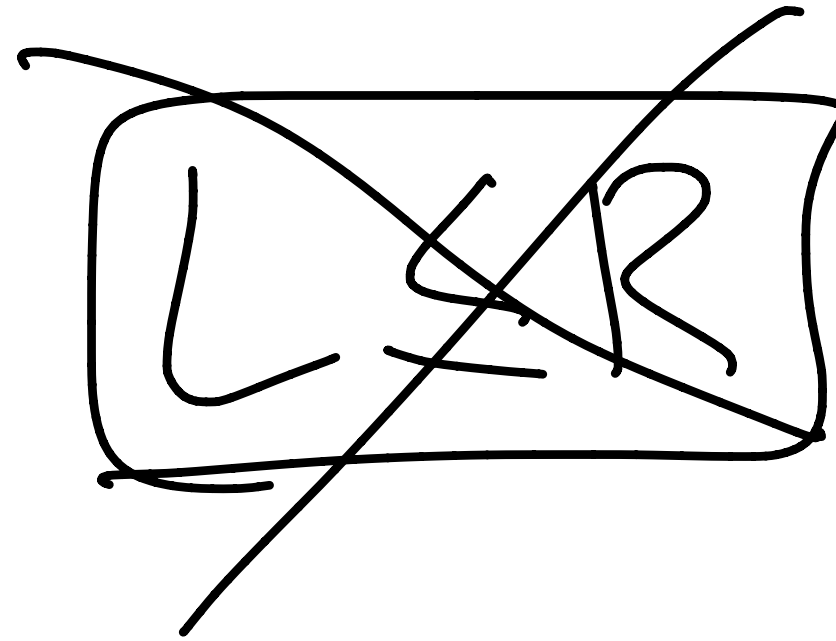
• $2 > 1$ 이기 때문에 오른쪽 ($M+1 \sim R$)에 2가 있을 수 있다.



이분 탐색

Binary Search

13



- 2를 찾아보자
- $L = 1, R = 0, M = 0$
- $L < R$ 이기 때문에, 이분 탐색을 종료한다. 2는 리스트에 없다.

1	3	4	5	7	8	9	20	30
---	---	---	---	---	---	---	----	----



이분 탐색

Binary Search

- 정렬되어 있는 리스트에서 어떤 값을 빠르게 찾는 알고리즘
- 리스트의 크기를 N 이라고 했을 때
- $\lg N$ 의 시간이 걸린다.
- 시간 복잡도가 $\lg N$ 인 이유는
- 크기가 N 인 리스트를 계속해서 절반으로 나누기 때문이다.
- $2^k = N$ 일 때, $k = \lg N$

$$O(\lg N)$$

이분 탐색

Binary Search

```
while (left <= right) {  
    int mid = (left + right) / 2;  
    if (a[mid] == x) {  
        position = mid;  
        break;  
    } else if (a[mid] > x) {  
        right = mid - 1;  
    } else {  
        left = mid + 1;  
    }  
}
```

left <= right
9/21 9/21

X는
A에

left <= right
right > left 이
오래 걸

숫자 카드

<https://www.acmicpc.net/problem/10815>

- 이분 탐색을 이용해 풀 수 있다.

숫자 카드

<https://www.acmicpc.net/problem/10815>

- 소스: <http://codeplus.codes/4e188439e8044b0db595d3f58bb7a383>

상한과 하한

Upper & Lower Bound

18

Upper bound

- 어떤 수열 A가 있을 때, K의 상한은 크거나 같은 수, 하한은 작거나 같은 수
- 보통 구현을 할 때는 아래와 같은 의미로 사용한다.

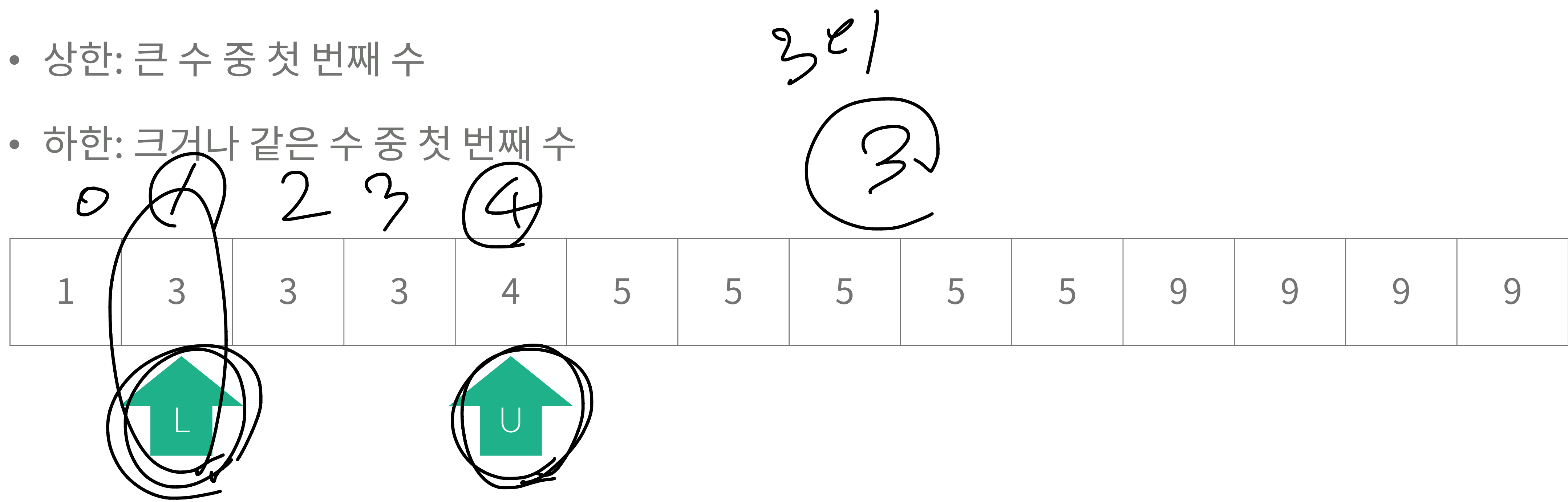
lower bound

- 상한: 큰 수 중 첫 번째 수
- 하한: 크거나 같은 수 중 첫 번째 수

상한과 하한

Upper & Lower Bound

- 어떤 수열 A가 있을 때, K의 상한은 크거나 같은 수, 하한은 작거나 같은 수
- 보통 구현을 할 때는 아래와 같은 의미로 사용한다.
- 상한: 큰 수 중 첫 번째 수
- 하한: 크거나 같은 수 중 첫 번째 수



상한과 하한

Upper & Lower Bound

- 어떤 수열 A가 있을 때, K의 상한은 크거나 같은 수, 하한은 작거나 같은 수
- 보통 구현을 할 때는 아래와 같은 의미로 사용한다.
- **상한**: 큰 수 중 첫 번째 수
- 하한: 크거나 같은 수 중 첫 번째 수



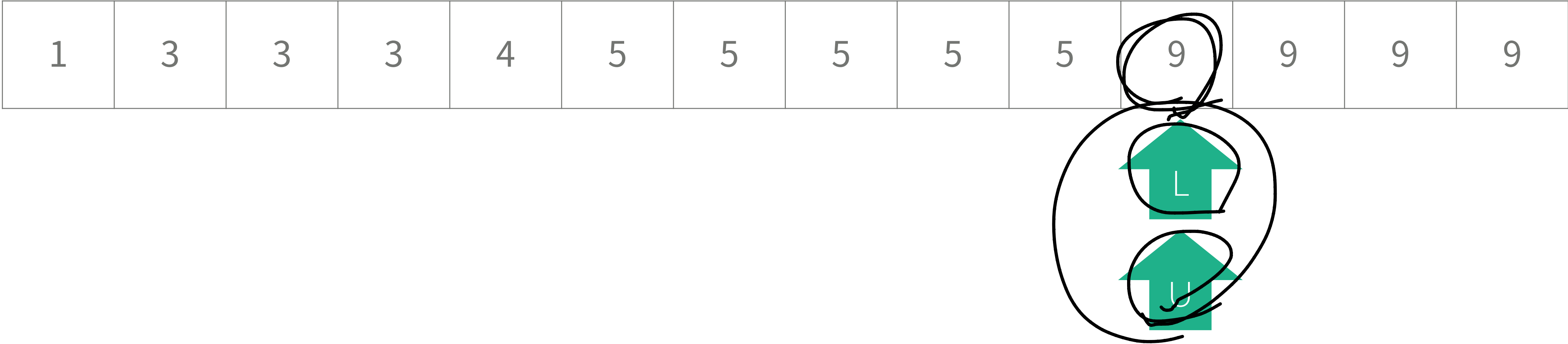
1	3	3	3	4	5	5	5	5	5	9	9	9	9
---	---	---	---	---	---	---	---	---	---	---	---	---	---

상한과 하한

Upper & Lower Bound

- 어떤 수열 A가 있을 때, K의 상한은 크거나 같은 수, 하한은 작거나 같은 수
- 보통 구현을 할 때는 아래와 같은 의미로 사용한다.
- 상한: 큰 수 중 첫 번째 수
- 하한: 크거나 같은 수 중 첫 번째 수

8의 lower bound
upp



상한과 하한

Upper & Lower Bound

- 이분 탐색을 이용해서 구현할 수 있다.

상한과 하한

Upper & Lower Bound

// lower bound

```
while (l <= r) {  
    int mid = (l+r)/2;  
    if (a[mid] == num) {  
        ans = mid;  
        r = mid-1;  
    } else if (a[mid] > num) {  
        r = mid-1;  
    } else {  
        l = mid+1;  
    }  
}
```



상한과 하한

Upper & Lower Bound

```
// upper bound
```

```
while (l <= r) {  
    int mid = (l+r)/2;  
    if (a[mid] == num) {  
        ans = mid+1;  
        l = mid+1;  
    } else if (a[mid] > num) {  
        r = mid-1;  
    } else {  
        l = mid+1;  
    }  
}
```



숫자 카드 2

25

<https://www.acmicpc.net/problem/10816>

- 이분 탐색을 이용해 풀 수 있다.

숫자 카드 2

<https://www.acmicpc.net/problem/10816>

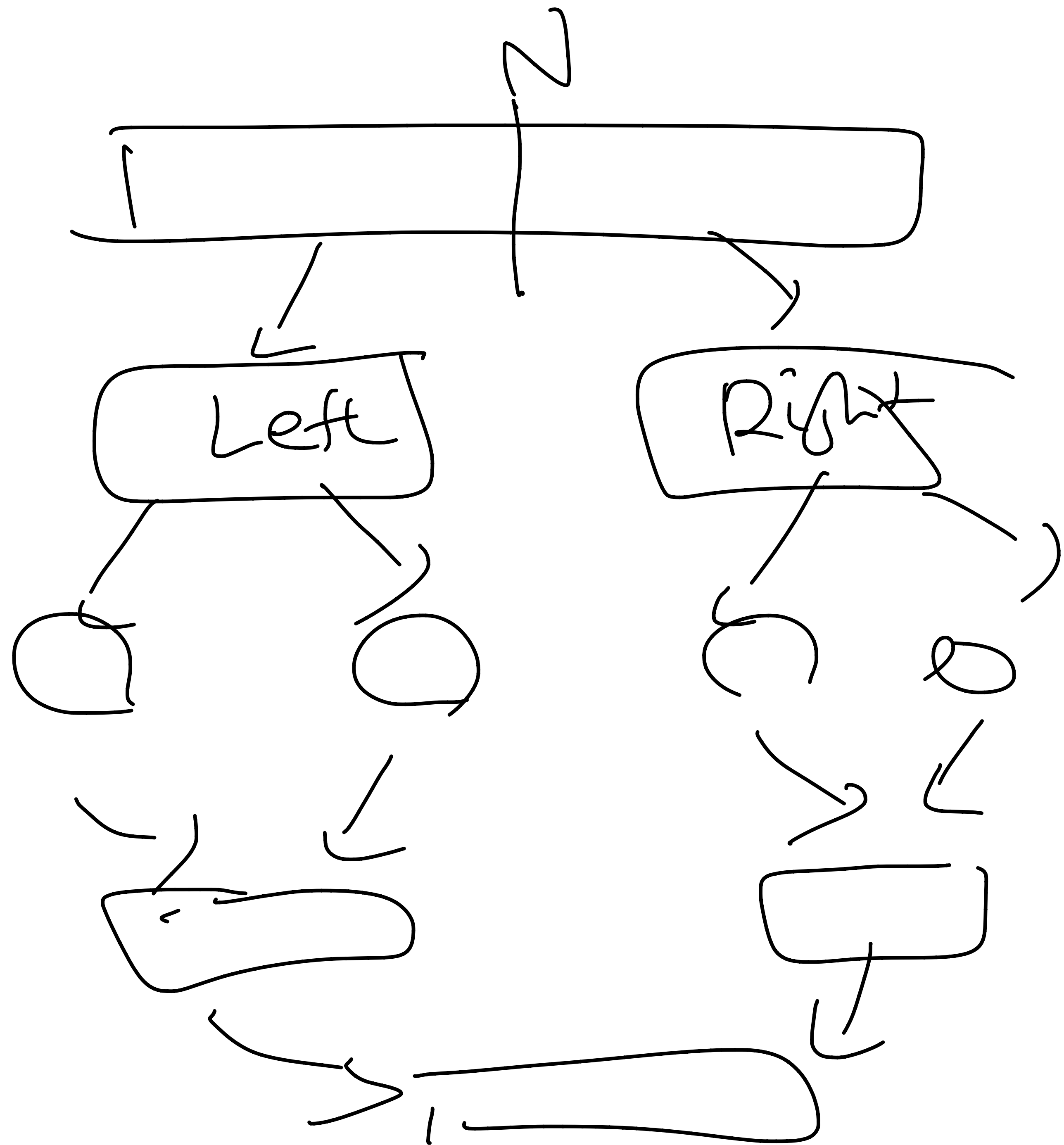
- 소스: <http://codeplus.codes/4b0580d84145483dbab368477d257505>

머지 소트

머지 소트

Merge Sort

- N개를 정렬하는 알고리즘
- N개를 $N/2$, $N/2$ 개로 나눈다.
- 왼쪽 $N/2$ 와 오른쪽 $N/2$ 를 정렬한다.
- 두 정렬한 결과를 하나로 합친다.

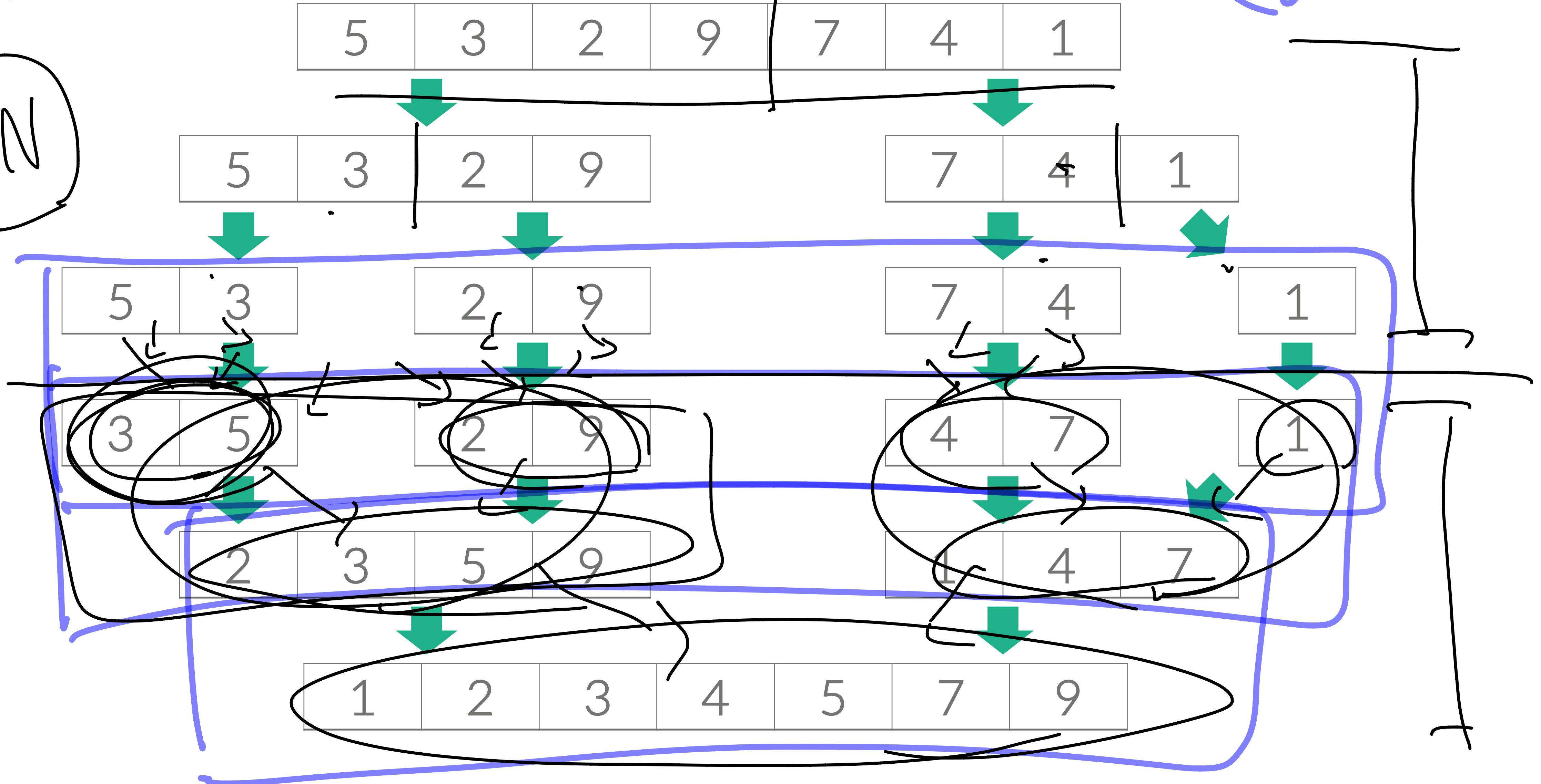


머지 소트 (N log N)

Merge Sort

$O(N)$

①



머지 소트

Merge Sort

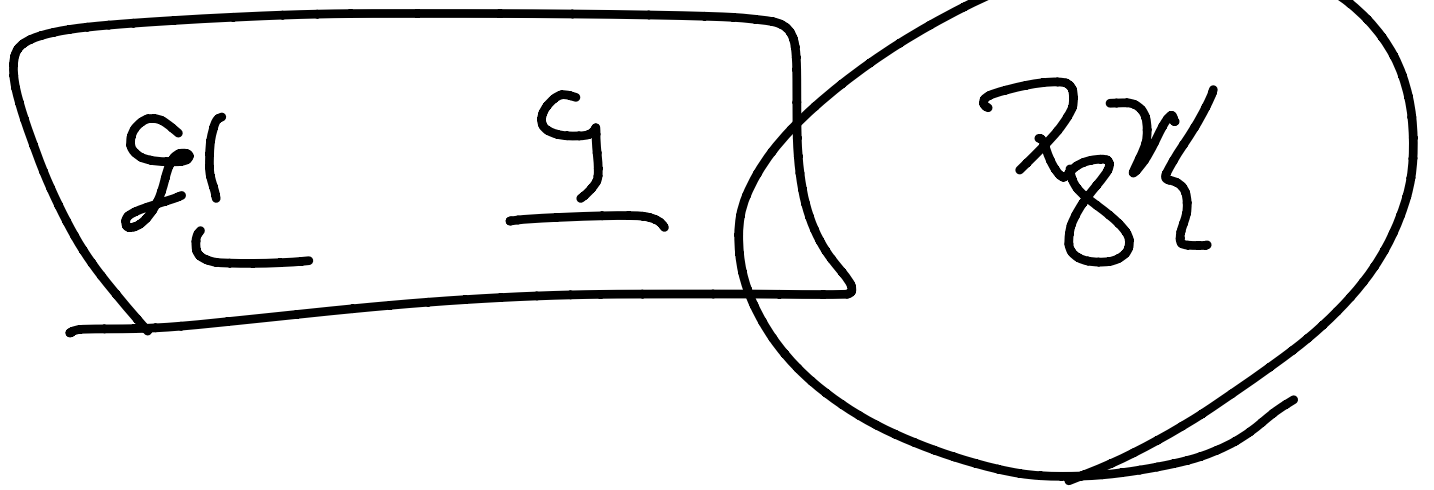
```
void sort(int start, int end) {  
    if (start == end) {  
        return;  
    }  
    int mid = (start+end)/2;  
    sort(start, mid);  
    sort(mid+1, end);  
    merge(start, end);  
}
```

base case
1) base

Start + 배열 크기
end 배열 크기
정렬하는 것

5 1 2 3
9 2 3

0 1 2 3

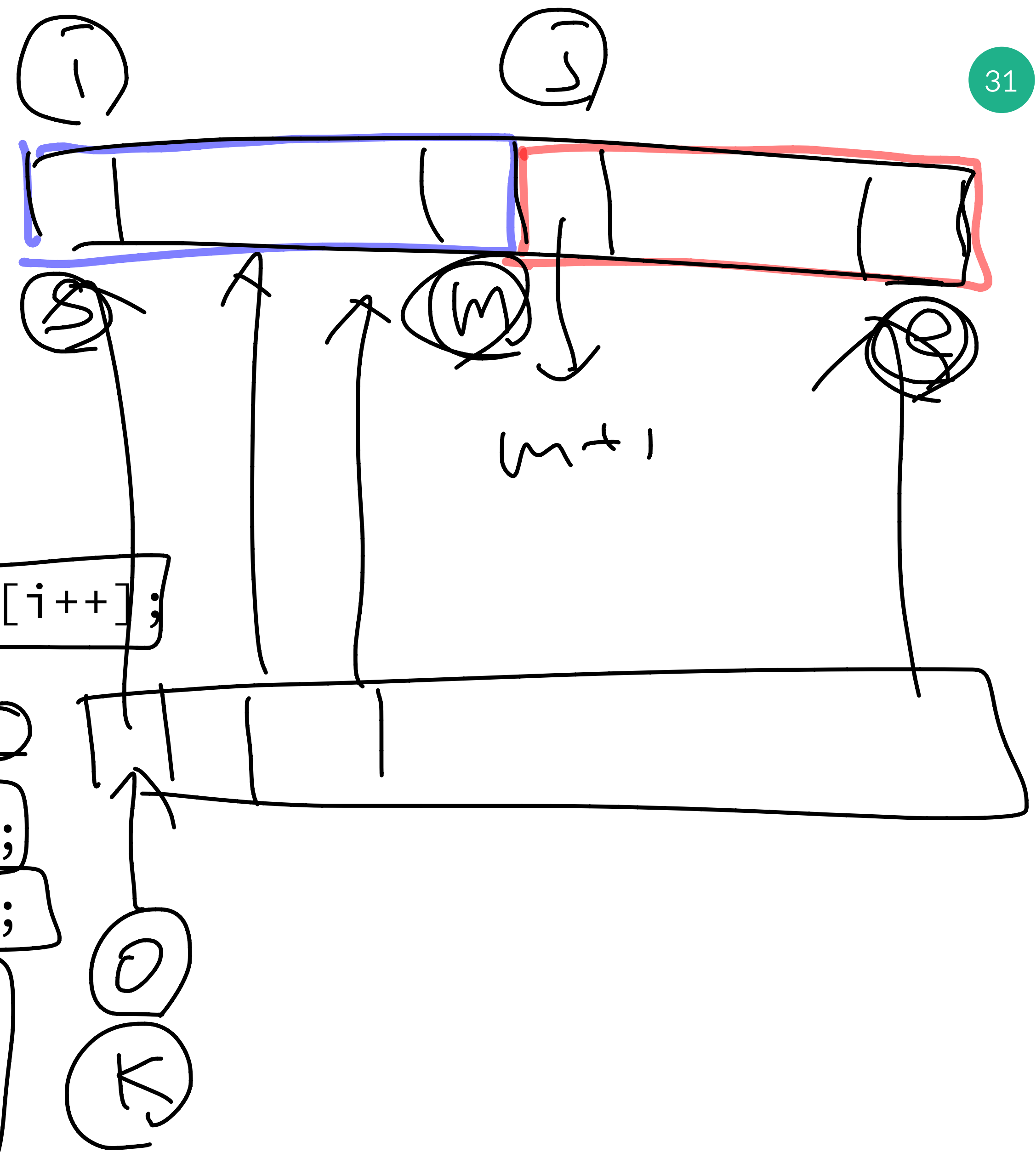


머지 소트



Merge Sort

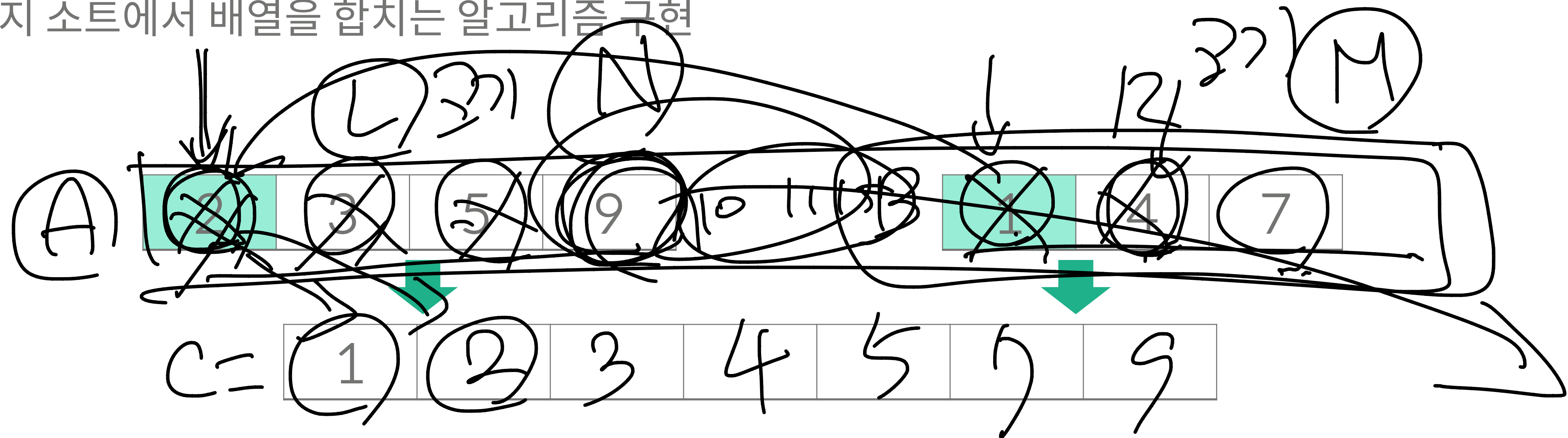
```
void merge(int start, int end) {  
    int mid = (start+end)/2;  
    int i = start, j = mid+1, k = 0;  
    while (i <= mid && j <= end) {  
        if (a[i] <= a[j]) b[k++] = a[i++];  
        else b[k++] = a[j++];  
    }  
    while (i <= mid) b[k++] = a[i++];  
    while (j <= end) b[k++] = a[j++];  
    for (int i=start; i<=end; i++) {  
        a[i] = b[i-start];  
    }  
}
```



배열 합치기

<https://www.acmicpc.net/problem/11728>

- 머지 소트에서 배열을 합치는 알고리즘 구현

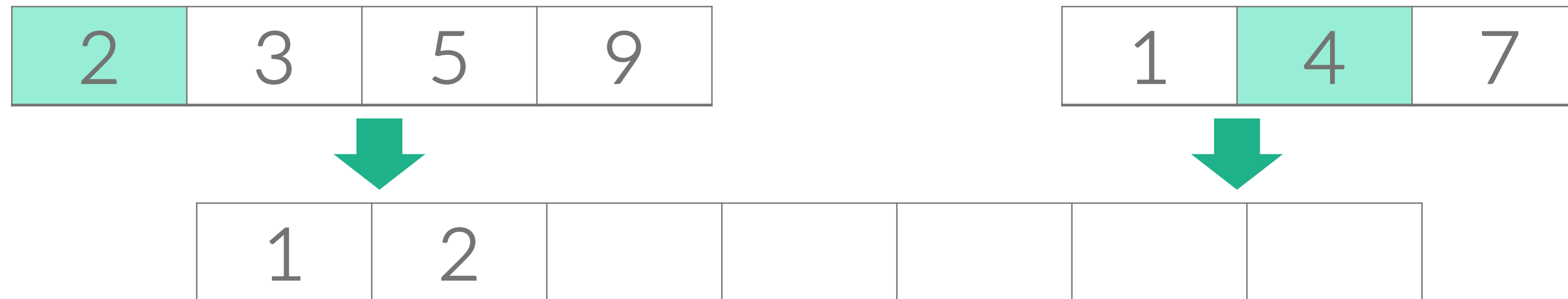


$$O(N+M)$$

배열 합치기

<https://www.acmicpc.net/problem/11728>

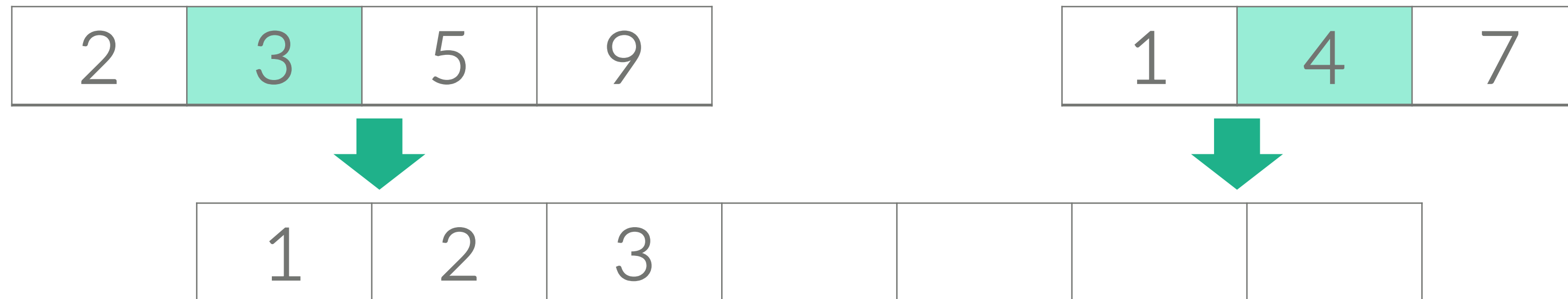
- 머지 소트에서 배열을 합치는 알고리즘 구현



배열 합치기

<https://www.acmicpc.net/problem/11728>

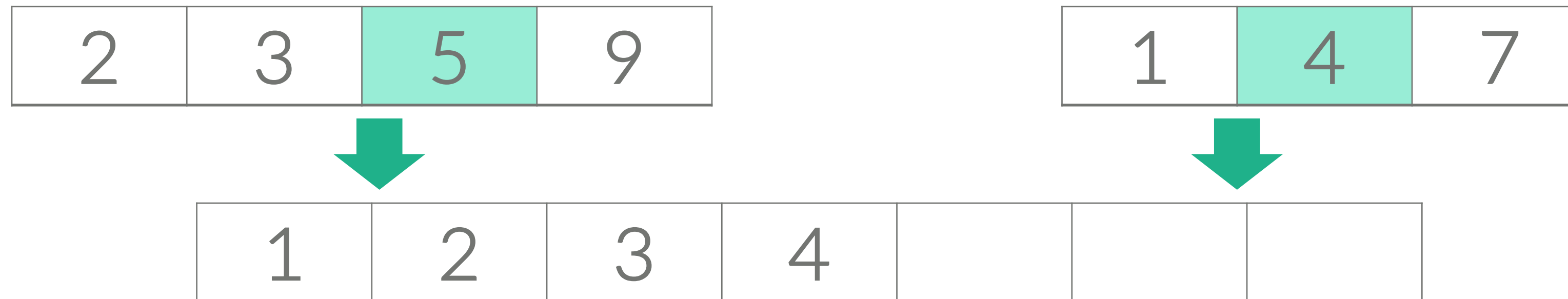
- 머지 소트에서 배열을 합치는 알고리즘 구현



배열 합치기

<https://www.acmicpc.net/problem/11728>

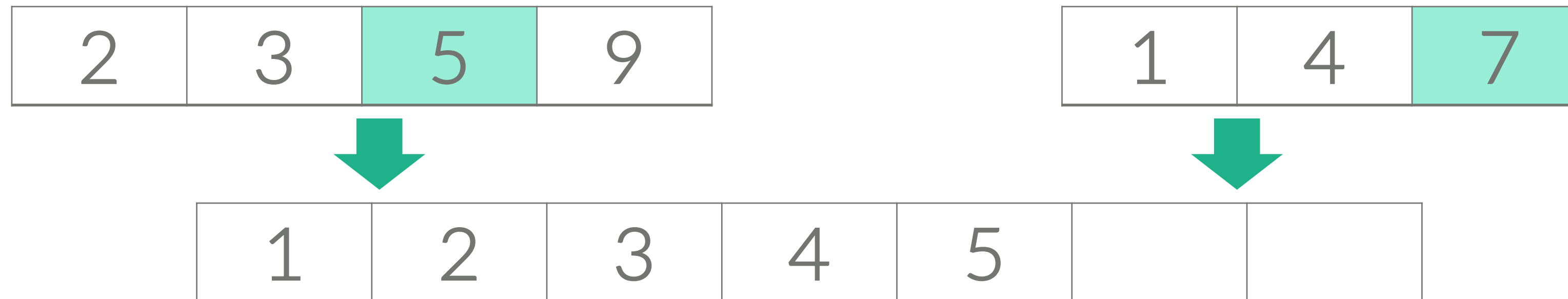
- 머지 소트에서 배열을 합치는 알고리즘 구현



배열 합치기

<https://www.acmicpc.net/problem/11728>

- 머지 소트에서 배열을 합치는 알고리즘 구현



배열 합치기

37

<https://www.acmicpc.net/problem/11728>

- 머지 소트에서 배열을 합치는 알고리즘 구현



배열 합치기

<https://www.acmicpc.net/problem/11728>

- 머지 소트에서 배열을 합치는 알고리즘 구현



배열 합치기

<https://www.acmicpc.net/problem/11728>

- 시간 복잡도: $O(|A|+|B|)$

배열 합치기

40

<https://www.acmicpc.net/problem/11728>

- 소스: <http://codeplus.codes/e6498ddc3e5546f2a1609ed175ccb360>

퀵 소트

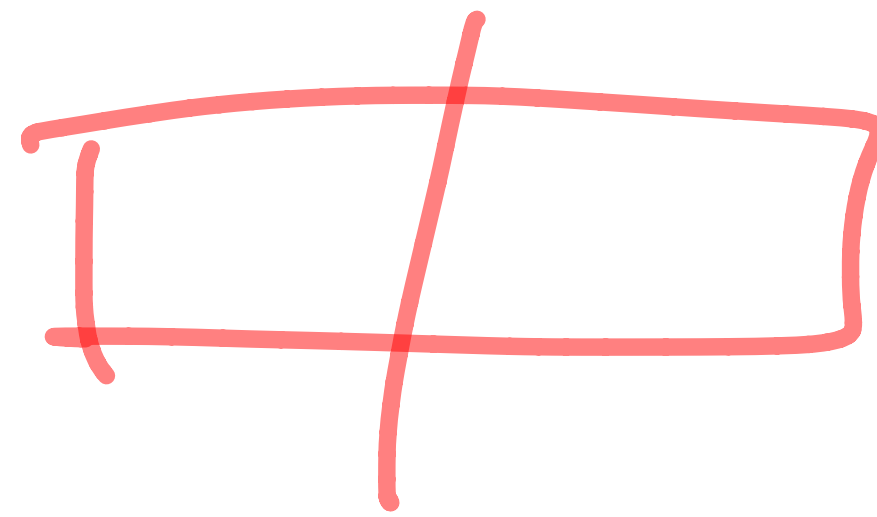
퀵 소트

Quick Sort

42

- 평균적인 상황에서 최고의 성능을 자랑하는 알고리즘
- 피벗(pivot)을 하나 고른 다음, 그것보다 작은 것을 앞으로 큰 것을 뒤로 보낸다.
- 그 다음, 피벗의 앞과 뒤에서 퀵 정렬을 수행한다.
- 최악의 경우에는 $O(N^2)$ 이 걸린다.

$N \log N$

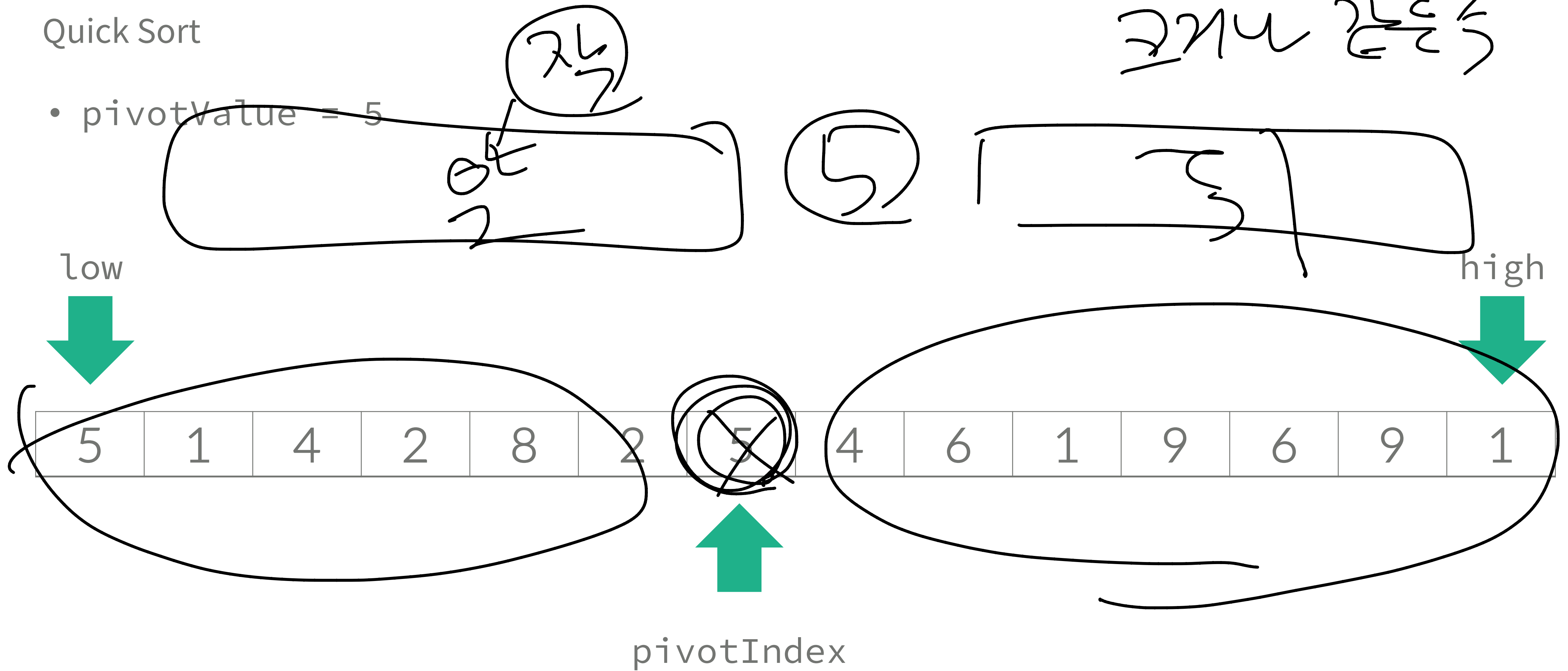


퀵 소트

43

Quick Sort

- pivotValue = 5

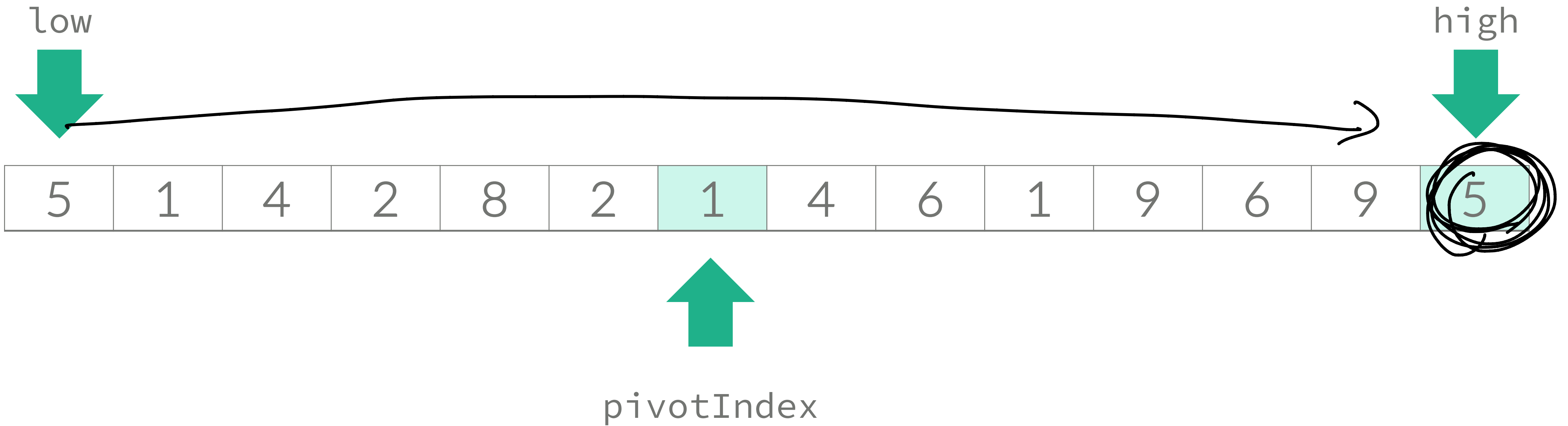


퀵 소트

Quick Sort

44

• pivotValue = 5

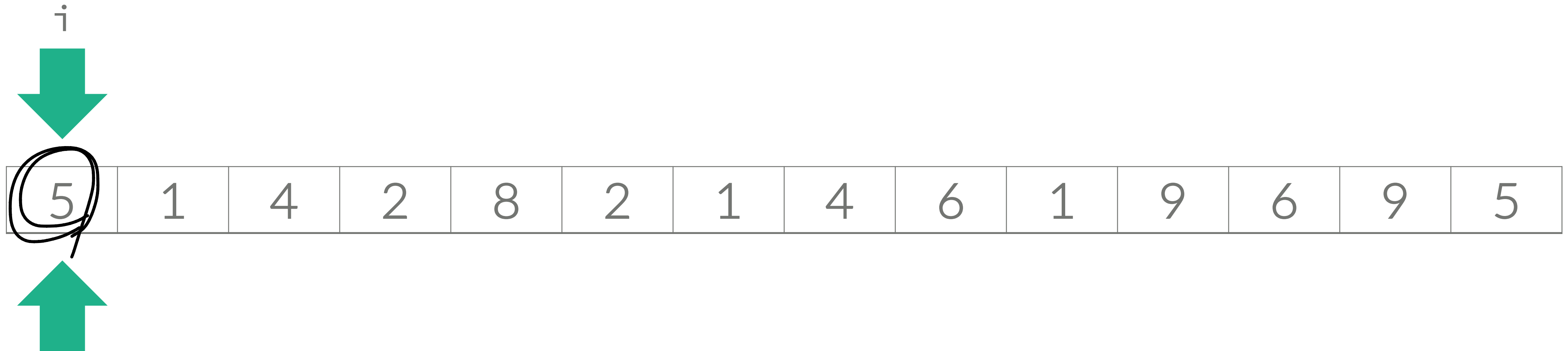


퀵 소트

45

Quick Sort

- `pivotValue = 5`
- `if (a[i] < pivotValue) swap(a[i], a[storeIndex]), storeIndex += 1`



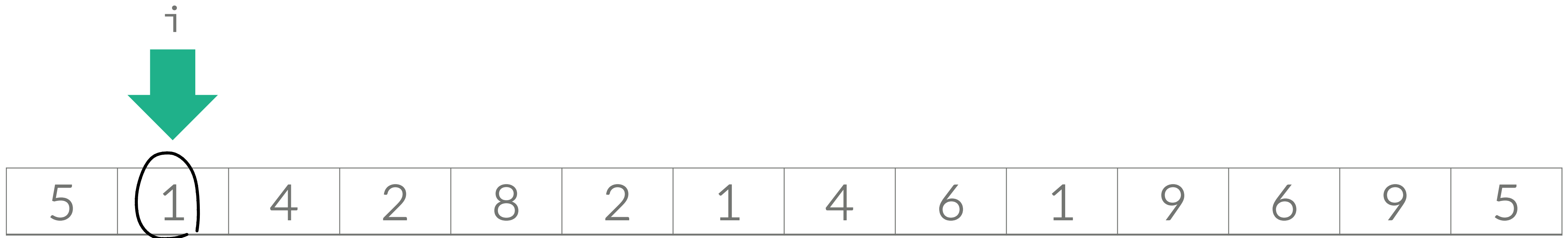
storeIndex

퀵 소트

46

Quick Sort

- pivotValue = 5
- if (a[i] < pivotValue) swap(a[i], a[storeIndex]), storeIndex += 1



storeIndex

퀵 소트

47

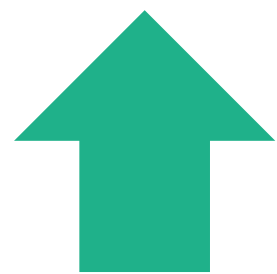
Quick Sort

- `pivotValue = 5`
- `if (a[i] < pivotValue) swap(a[i], a[storeIndex]), storeIndex += 1`

i



1	5	4	2	8	2	1	4	6	1	9	6	9	5
---	---	---	---	---	---	---	---	---	---	---	---	---	---



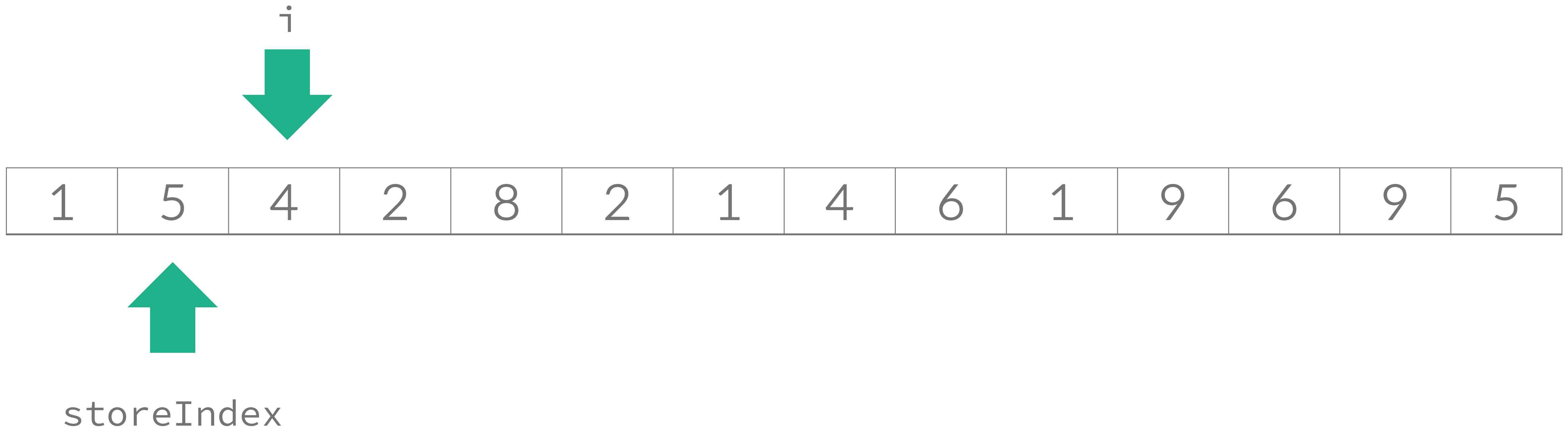
storeIndex

퀵 소트

48

Quick Sort

- `pivotValue = 5`
- `if (a[i] < pivotValue) swap(a[i], a[storeIndex]), storeIndex += 1`

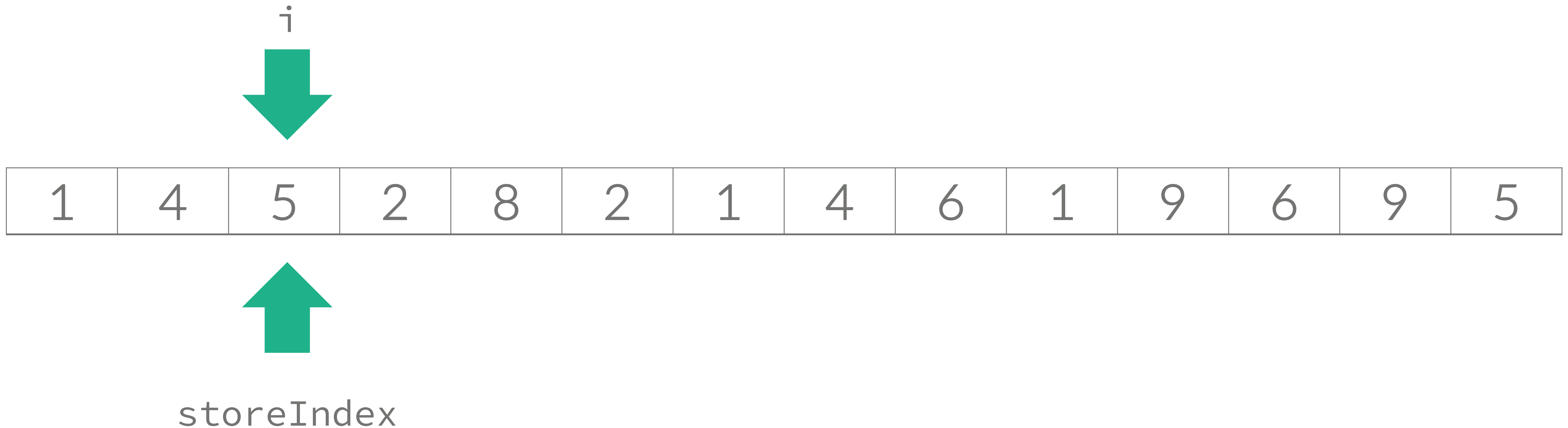


퀵 소트

49

Quick Sort

- `pivotValue = 5`
- `if (a[i] < pivotValue) swap(a[i], a[storeIndex]), storeIndex += 1`

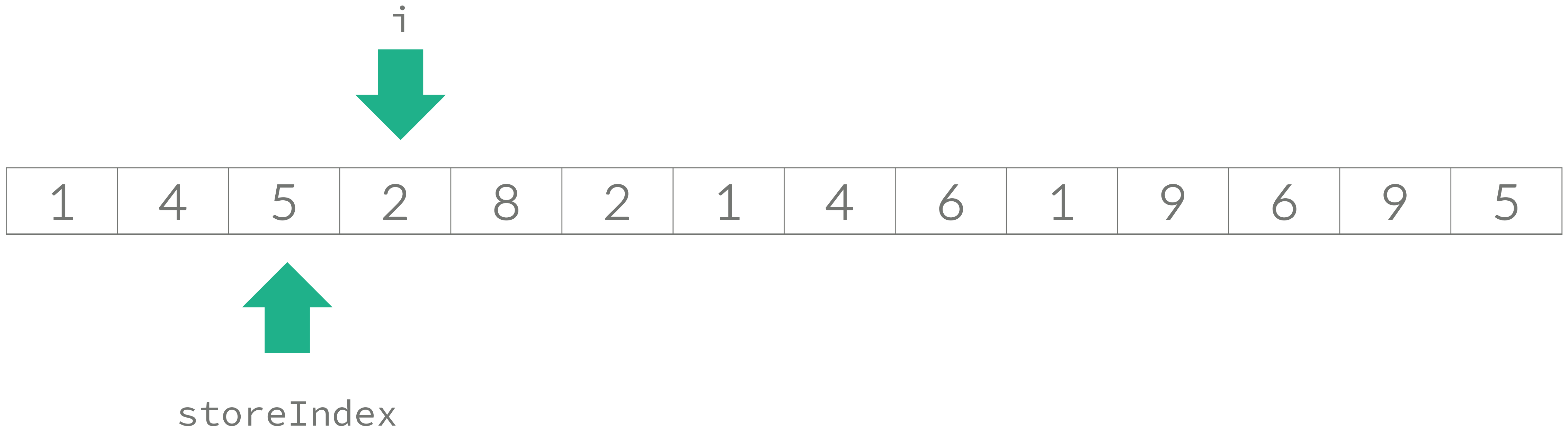


퀵 소트

50

Quick Sort

- `pivotValue = 5`
- `if (a[i] < pivotValue) swap(a[i], a[storeIndex]), storeIndex += 1`

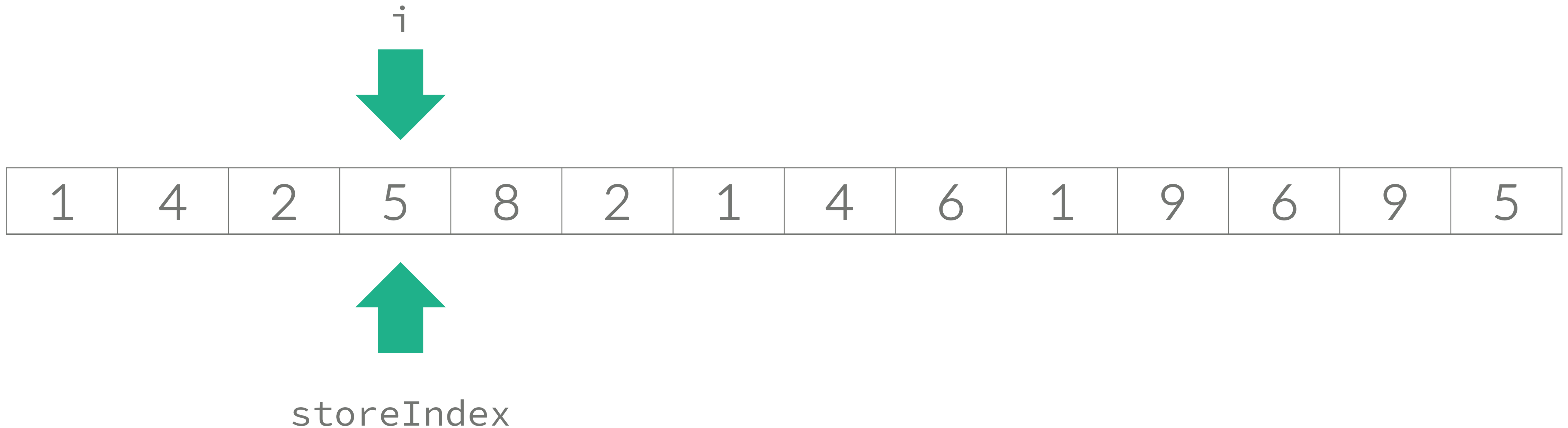


퀵 소트

51

Quick Sort

- `pivotValue = 5`
- `if (a[i] < pivotValue) swap(a[i], a[storeIndex]), storeIndex += 1`

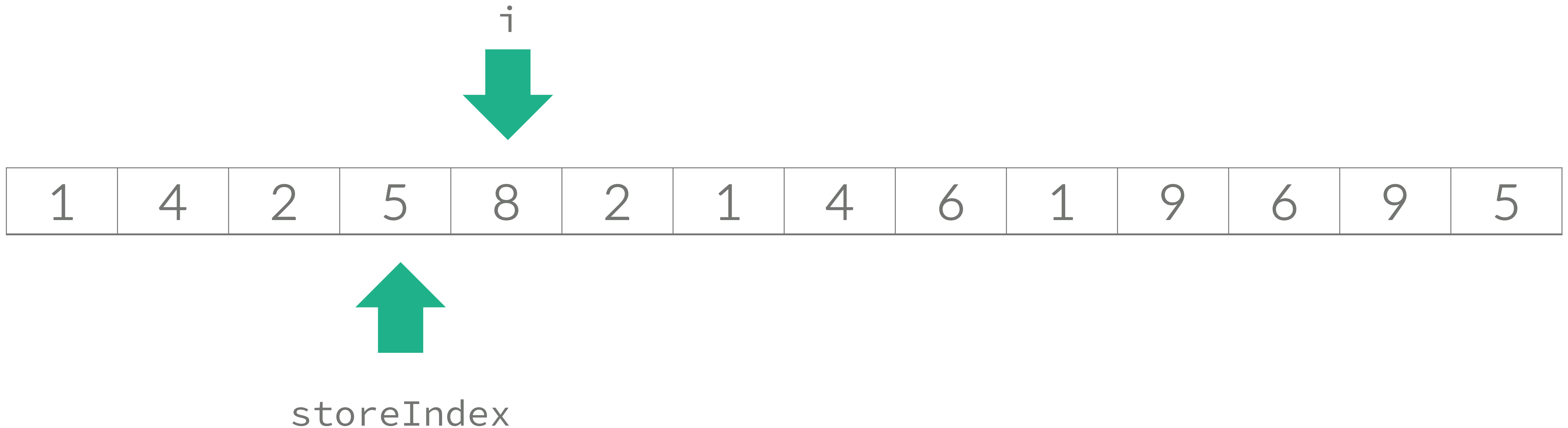


퀵 소트

52

Quick Sort

- `pivotValue = 5`
- `if (a[i] < pivotValue) swap(a[i], a[storeIndex]), storeIndex += 1`

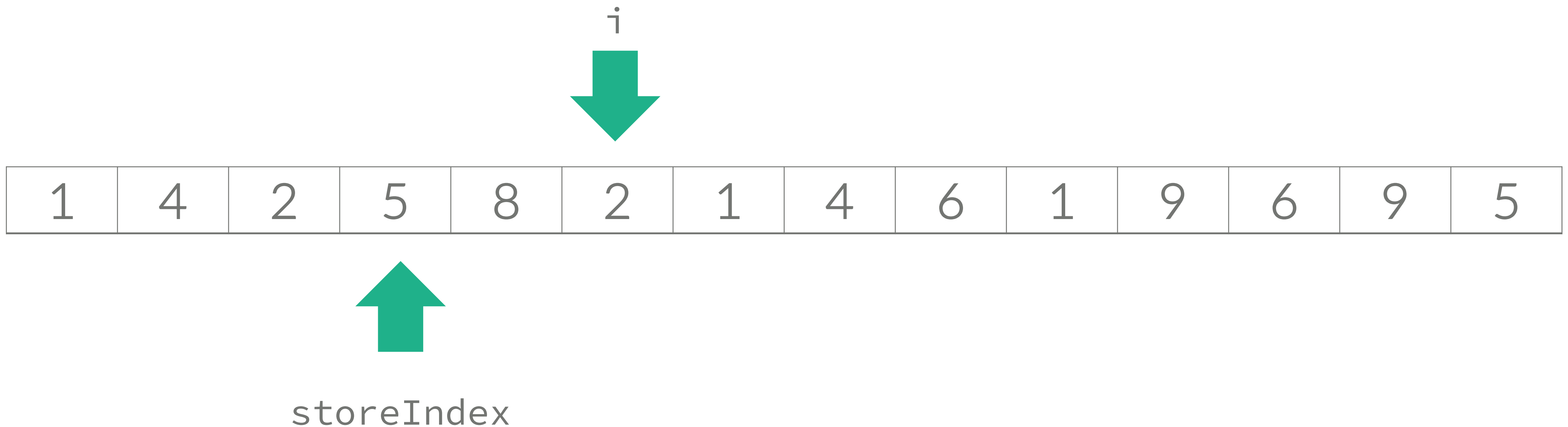


퀵 소트

53

Quick Sort

- `pivotValue = 5`
- `if (a[i] < pivotValue) swap(a[i], a[storeIndex]), storeIndex += 1`

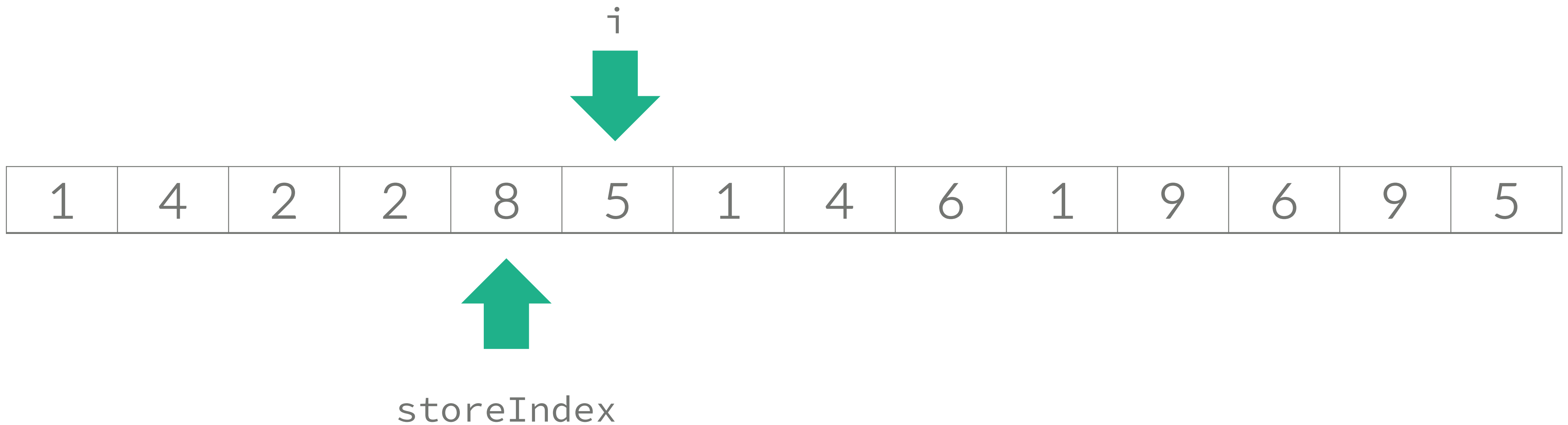


퀵 소트

54

Quick Sort

- `pivotValue = 5`
- `if (a[i] < pivotValue) swap(a[i], a[storeIndex]), storeIndex += 1`

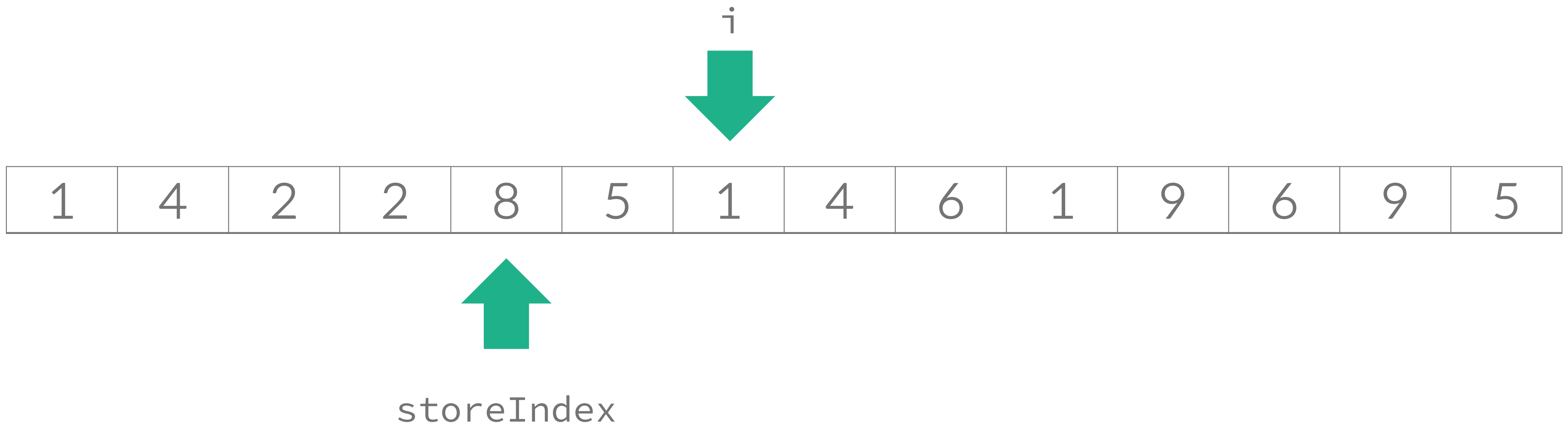


퀵 소트

55

Quick Sort

- `pivotValue = 5`
- `if (a[i] < pivotValue) swap(a[i], a[storeIndex]), storeIndex += 1`

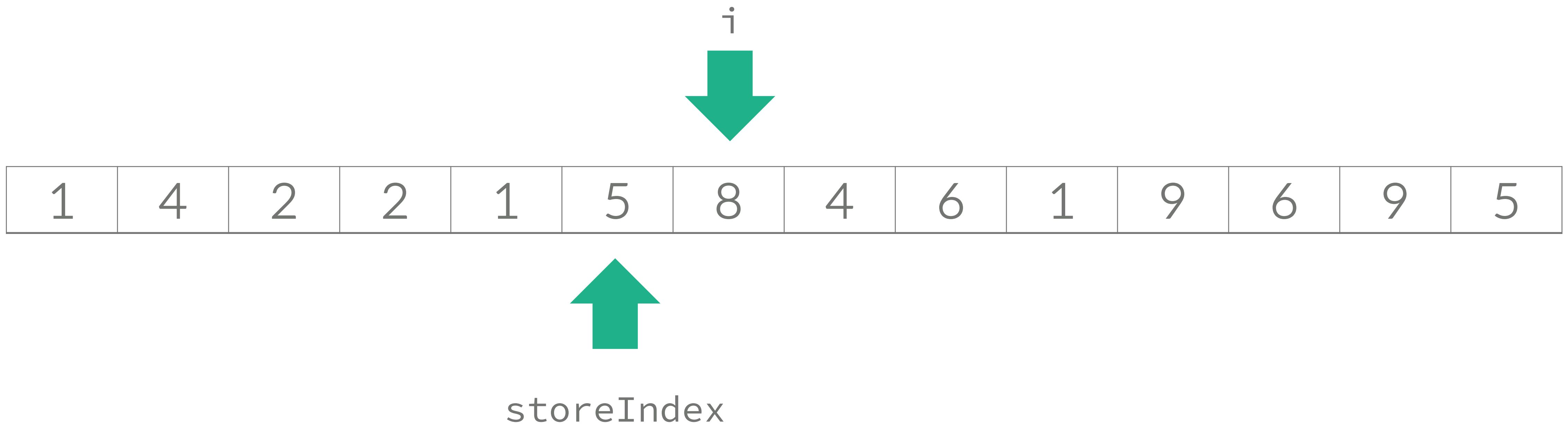


퀵 소트

56

Quick Sort

- `pivotValue = 5`
- `if (a[i] < pivotValue) swap(a[i], a[storeIndex]), storeIndex += 1`

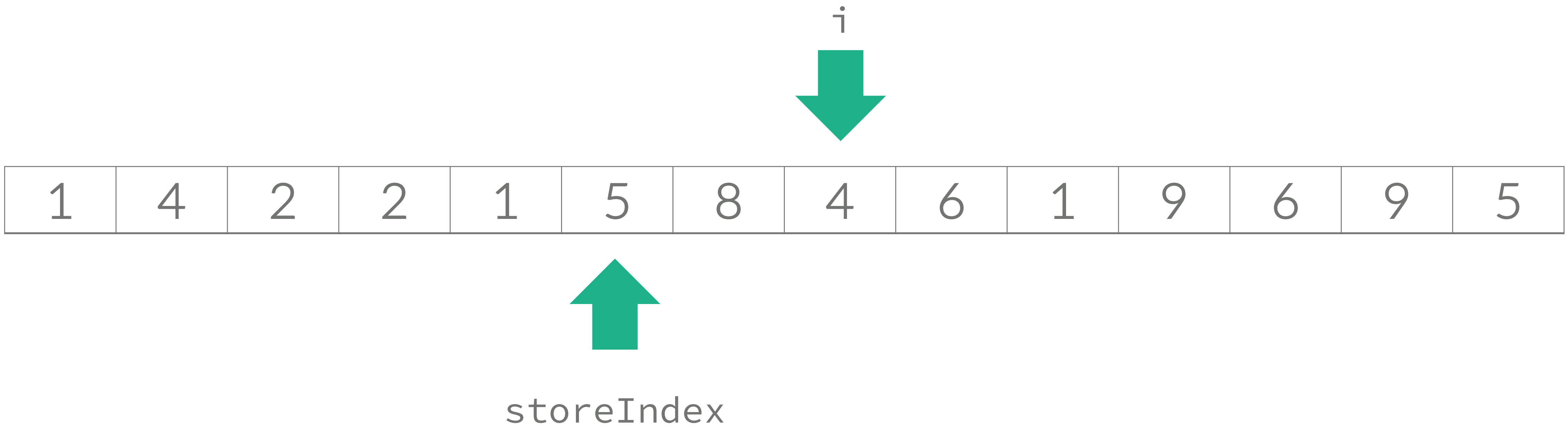


퀵 소트

57

Quick Sort

- pivotValue = 5
- if (a[i] < pivotValue) swap(a[i], a[storeIndex]), storeIndex += 1

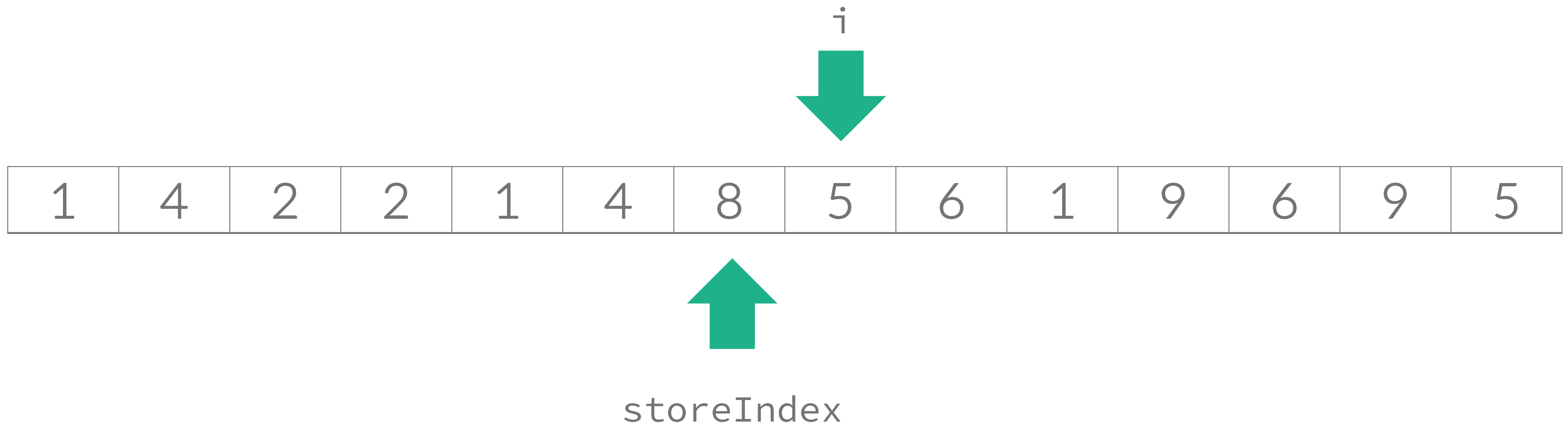


퀵 소트

58

Quick Sort

- `pivotValue = 5`
- `if (a[i] < pivotValue) swap(a[i], a[storeIndex]), storeIndex += 1`

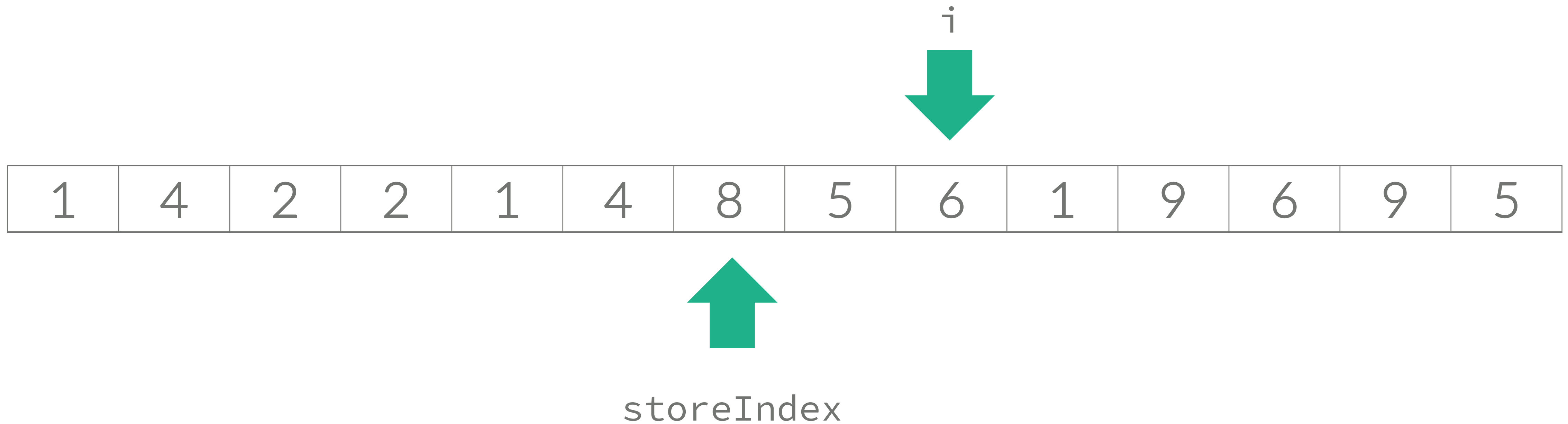


퀵 소트

59

Quick Sort

- `pivotValue = 5`
- `if (a[i] < pivotValue) swap(a[i], a[storeIndex]), storeIndex += 1`



퀵 소트

60

Quick Sort

- `pivotValue = 5`
- `if (a[i] < pivotValue) swap(a[i], a[storeIndex]), storeIndex += 1`



퀵 소트

61

Quick Sort

- `pivotValue = 5`
- `if (a[i] < pivotValue) swap(a[i], a[storeIndex]), storeIndex += 1`



퀵 소트

62

Quick Sort

- `pivotValue = 5`
- `if (a[i] < pivotValue) swap(a[i], a[storeIndex]), storeIndex += 1`



퀵 소트

63

Quick Sort

- `pivotValue = 5`
- `if (a[i] < pivotValue) swap(a[i], a[storeIndex]), storeIndex += 1`



퀵 소트

64

Quick Sort

- `pivotValue = 5`
- `if (a[i] < pivotValue) swap(a[i], a[storeIndex]), storeIndex += 1`

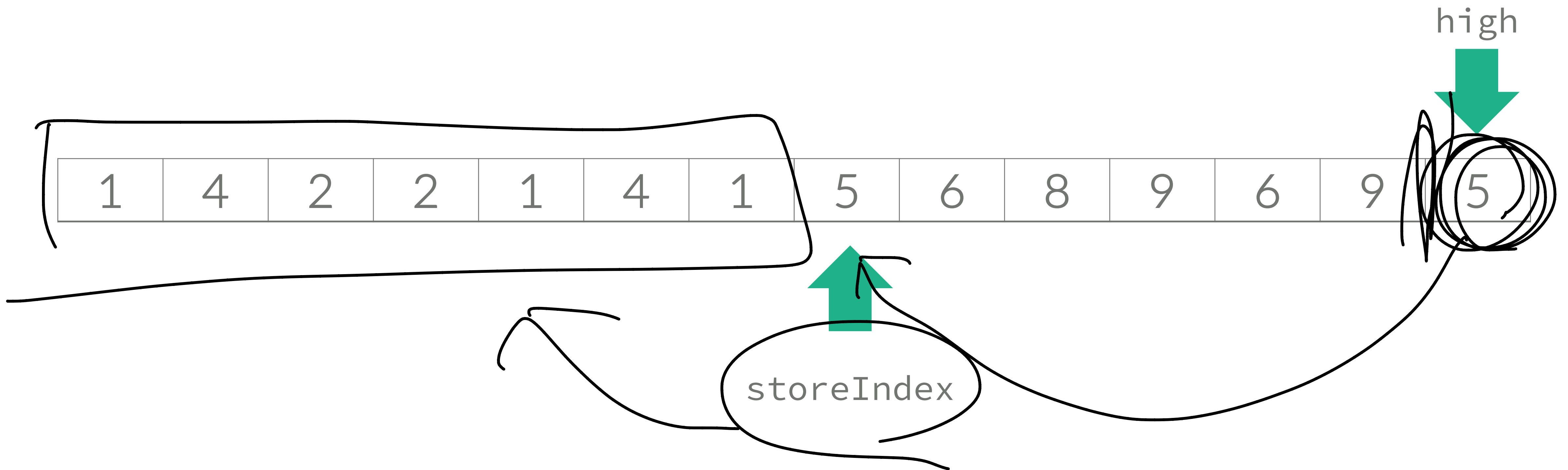


퀵 소트

65

Quick Sort

- pivotValue = 5
- if (a[i] < pivotValue) swap(a[i], a[storeIndex]), storeIndex += 1



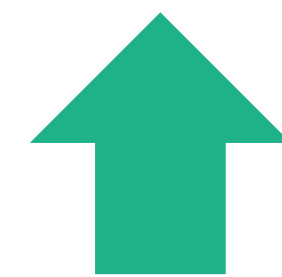
퀵 소트

66

Quick Sort

- `pivotValue = 5`
- `if (a[i] < pivotValue) swap(a[i], a[storeIndex]), storeIndex += 1`

1	4	2	2	1	4	1	5	6	8	9	6	9	5
---	---	---	---	---	---	---	---	---	---	---	---	---	---



pivot

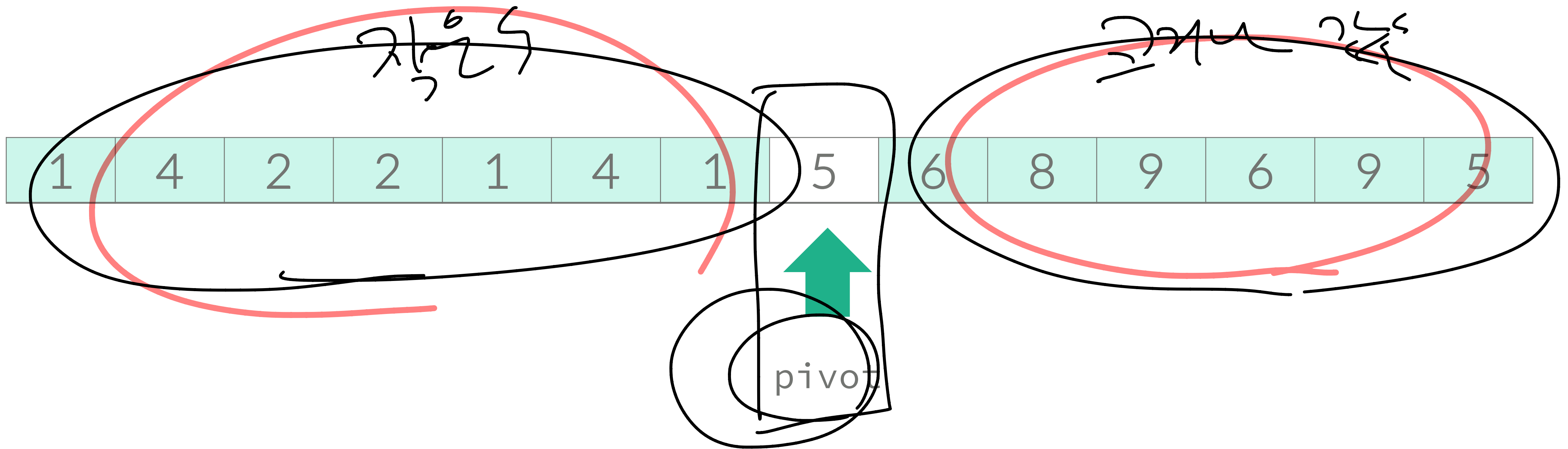
퀵 소트

Quick Sort

67

- pivotValue = 5
- if (a[i] < pivotValue) swap(a[i], a[storeIndex]), storeIndex += 1

(NEN)



퀵 소트

Quick Sort

```
int choosePivot(int low, int high) {  
    return low + (high-low)/2;  
}
```

$$l + \cancel{h/2} - l/2$$

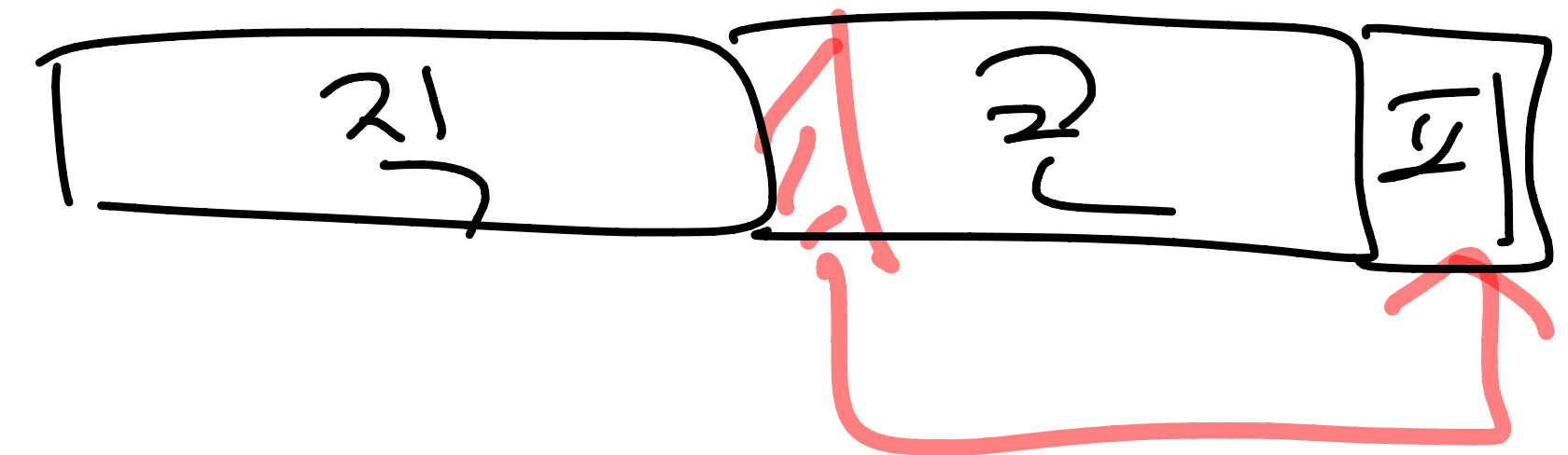
$$(l/2 + h/2)$$

$$(low + high) / 2 \leftarrow$$

퀵 소트

Quick Sort

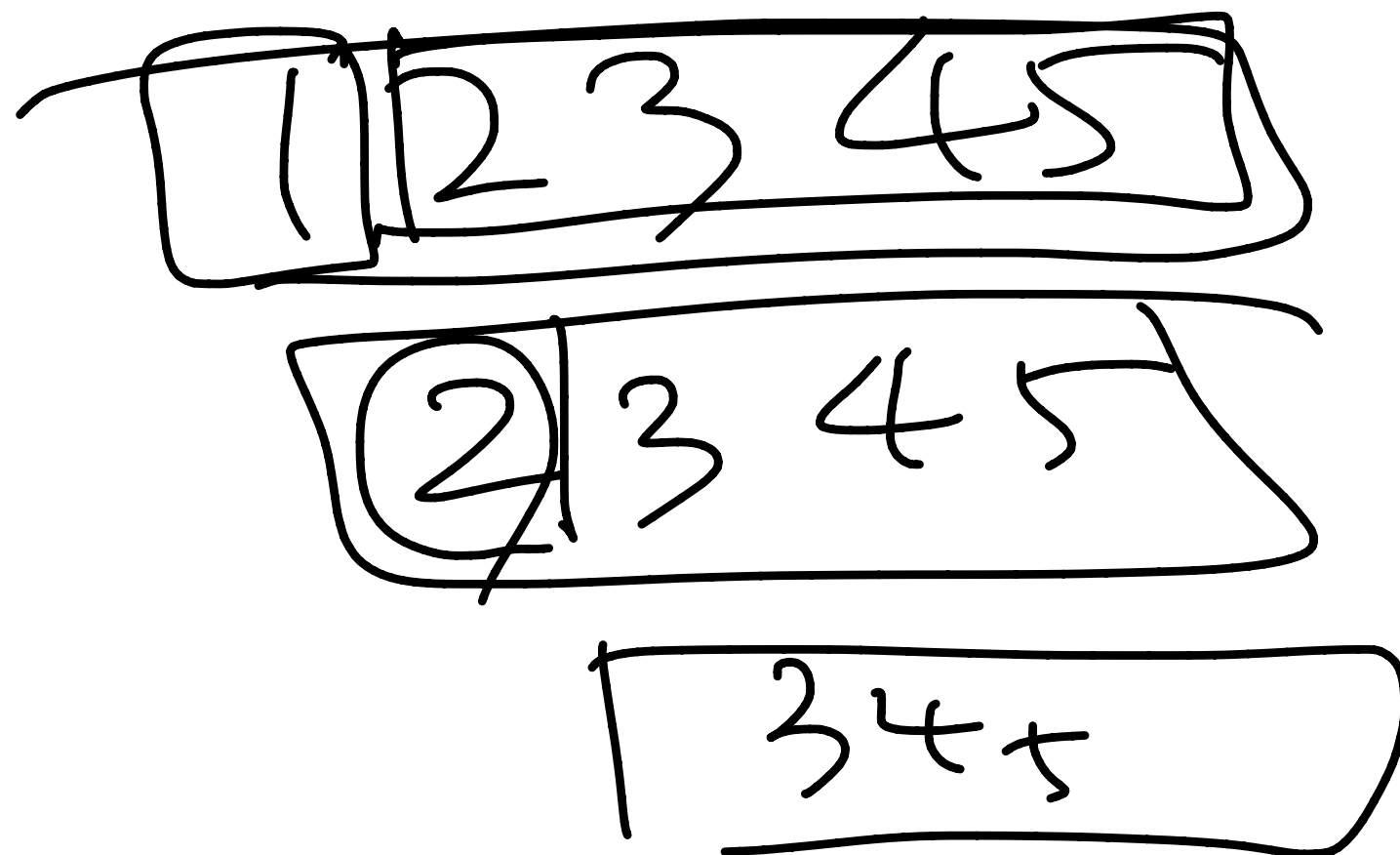
```
int partition(int low, int high) {  
    int pivotIndex = choosePivot(low, high);  
    int pivotValue = a[pivotIndex];  
    swap(a[pivotIndex], a[high]); → (2/15)  
    int storeIndex = low;  
    for (int i=low; i<high; i++) {  
        if (a[i] < pivotValue) {  
            swap(a[i], a[storeIndex]);  
            storeIndex += 1;  
        }  
    }  
    swap(a[storeIndex], a[high]);  
    return storeIndex;  
}
```



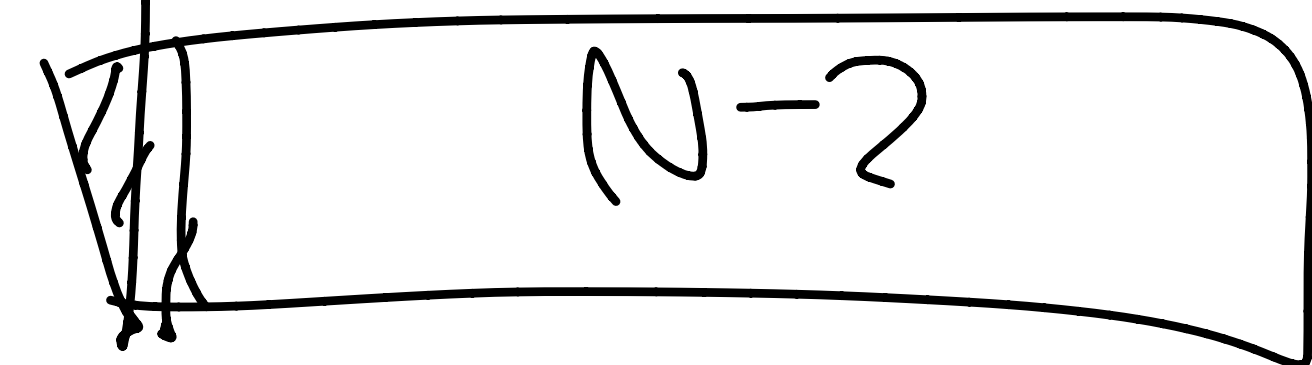
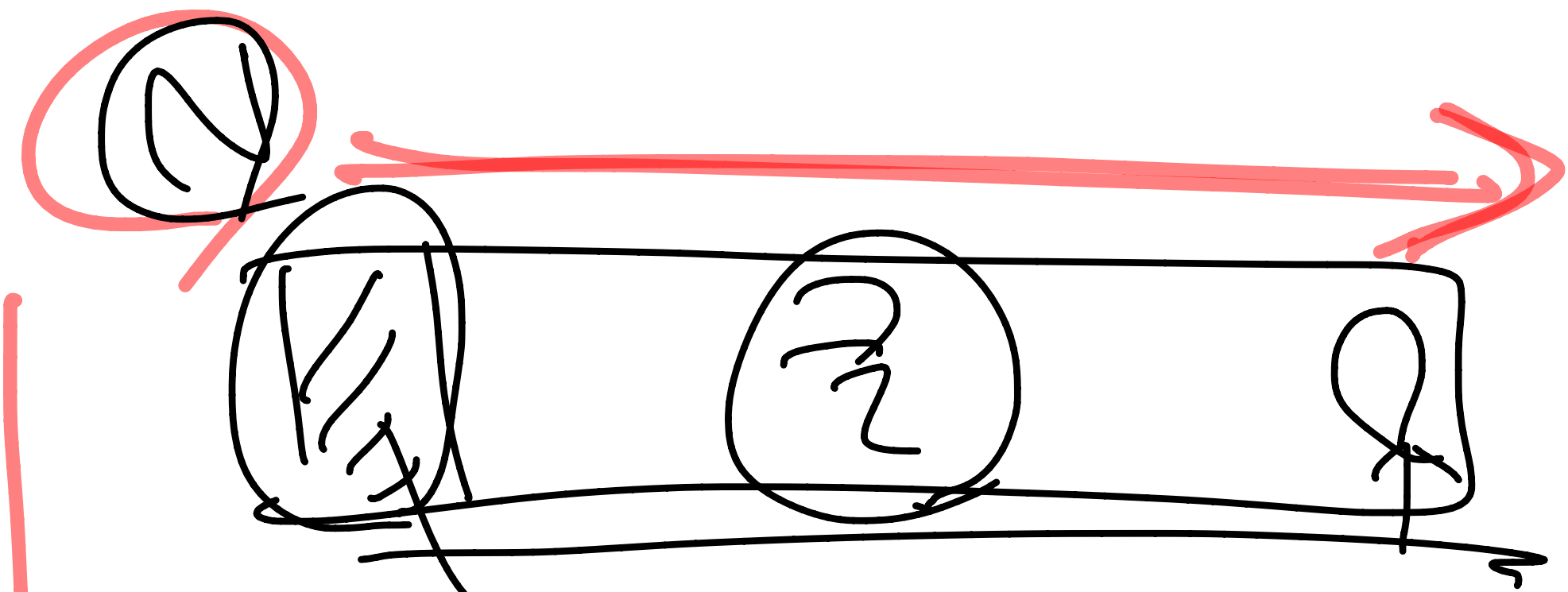
퀵 소트

Quick Sort

```
void quicksort(int low, int high) {  
    if (low < high) {  
        int pivot = partition(low, high);  
        quicksort(low, pivot-1);  
        quicksort(pivot+1, high);  
    }  
}
```



1 (N & N)



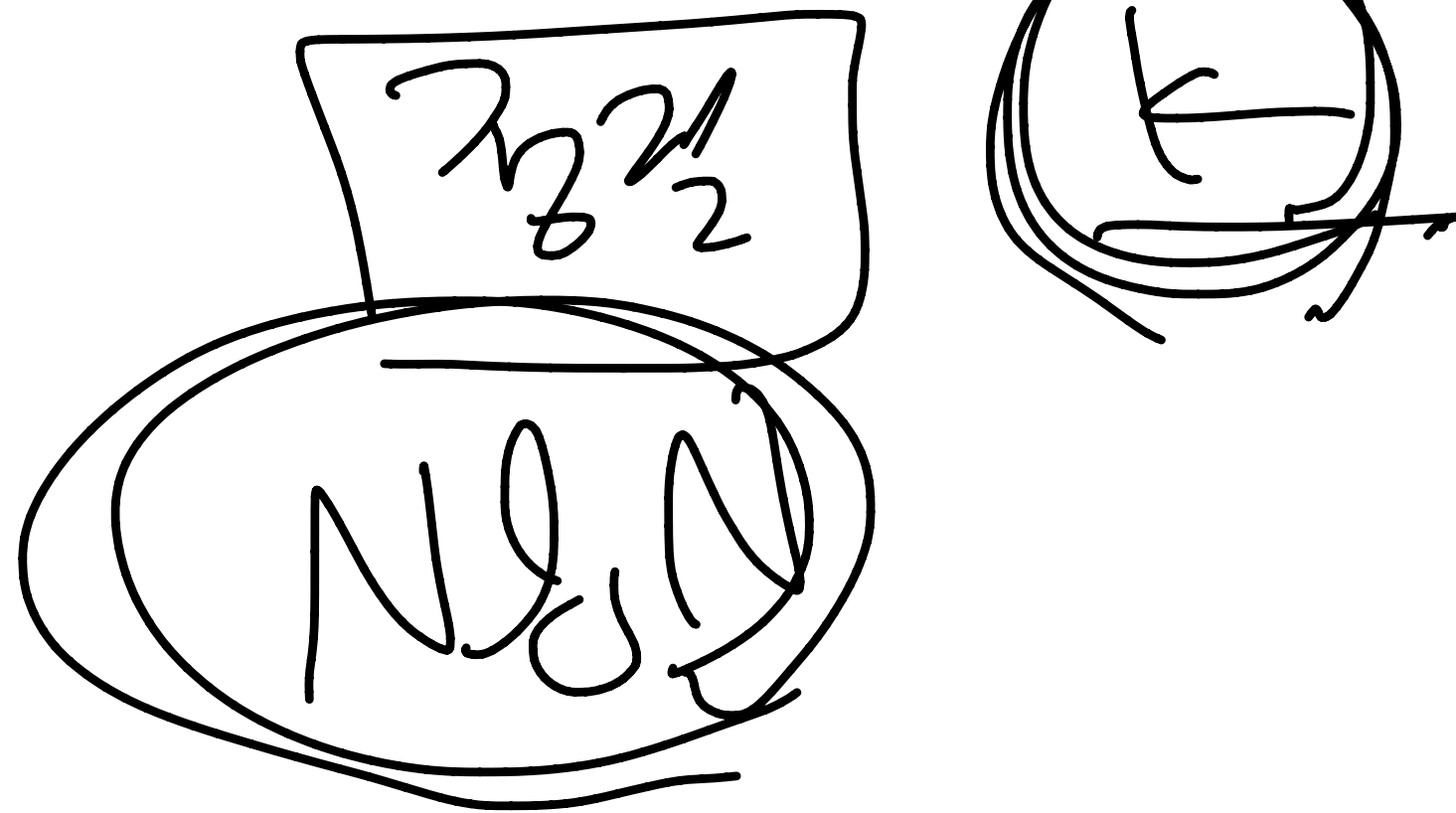
(N)

$O(N^2)$

퀵 셀렉트

Quick Sort

- 정렬되지 않은 리스트에서 k 번째 작은 수를 찾는 알고리즘
- 퀵 소트와 같지만, 한 쪽만 호출한다.
- 따라서, 시간 복잡도가 $O(N)$ 으로 줄어들지만, 최악의 경우에는 $O(N^2)$ 이다.



퀵 선택

Quick Sort

```
int quickselect(int low, int high, int k) {  
    int pivot = partition(low, high);  
    if (pivot == k) {  
        return a[k];  
    } else if (k < pivot) {  
        return quickselect(low, pivot-1, k);  
    } else {  
        return quickselect(pivot+1, high, k);  
    }  
}
```