동양미래대학교

# 고급 알고리즘

컴퓨터소프트웨어공학과 강 은 영

# 배열의 모든 왼쪽 값은 자신의 값보다 작고, 모든 오른쪽 값은 자신의값보다 큰 항목 찾기

(출처)
https://www.geeksforgeeks.org/

배열의 모든 왼쪽 값은 자신의 값 보다 작고, 모든 오른쪽 값은 자신의 값 보다 큰 항목을 찾기
**Find the element before which all the elements are smaller than it, and after which all are greater**

배열이 주어지고, 배열의 모든 왼쪽 값은 자신의 값 보다 작고, 모든 오른쪽 값은 자신의 값 보다 큰 항목을 찾는다. 그런 조건에 맞는 항목을 찾으면 그 항목의 색인 값을 반환하고, 그런 항목이 없으면 -1 값을 반환한다. 예상 시간 복잡도는 O(n)

Given an array, find an element before which all elements are smaller than it, and after which all are greater than it. Return the index of the element if there is such an element, otherwise, return –1. Expected time complexity: O(n).

| | | |
|---|---|---|
| **입력** | **Input: arr[] = {5, 1, 4, 3, 6, 8, 10, 7, 9}**<br>정수 배열 | |
| **출력** | **Output: 4**<br>**All elements on left of arr[4] are smaller than it and all elements on right are greater.** | |
| **입출력 예** | 입력<br>**Input: arr[] = {5, 1, 4, 4};** | 출력<br>**Output: -1**<br>**No such index exits.** |

# 배열의 모든 왼쪽 값은 자신의 값보다 작고, 모든 오른쪽 값은 자신의값보다 큰 항목 찾기

(출처)
https://www.geeksforgeeks.org/

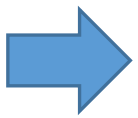
배열의 모든 왼쪽 값은 자신의 값 보다 작고, 모든 오른쪽 값은 자신의 값 보다 큰 항목을 찾기
**Find the element before which all the elements are smaller than it, and after which all are greater**

배열이 주어지고, 배열의 모든 왼쪽 값은 자신의 값 보다 작고, 모든 오른쪽 값은 자신의 값 보다 큰 항목을 찾는다. 그런 조건에 맞는 항목을 찾으면 그 항목의 색인 값을 반환하고, 그런 항목이 없으면 -1 값을 반환한다. 예상 시간 복잡도는 O(n)

Given an array, find an element before which all elements are smaller than it, and after which all are greater than it. Return the index of the element if there is such an element, otherwise, return –1. Expected time complexity: O(n).

1 문제의 이해

arr = {5, 1, 4, 3, 6, 8, 10, 7, 9}



{5} 1 {4, 3, 6, 8, 10, 7, 9}
{5, 1} 4 {3, 6, 8, 10, 7, 9}
{5, 1, 4} 3 {6, 8, 10, 7, 9}
{5, 1, 4, 3} 6 {8, 10, 7, 9}

arr[4]

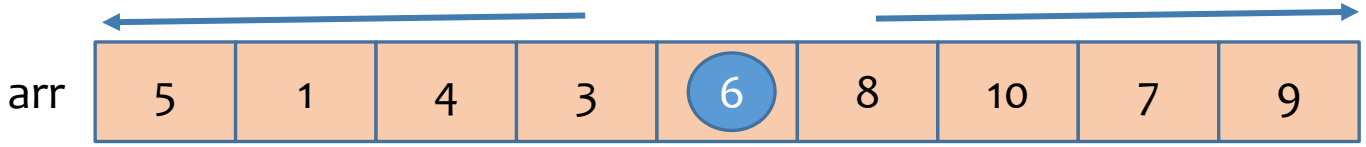
# 배열의 모든 왼쪽 값은 자신의 값보다 작고, 모든 오른쪽 값은 자신의값보다 큰 항목 찾기

(출처)
https://www.geeksforgeeks.org/

배열의 모든 왼쪽 값은 자신의 값 보다 작고, 모든 오른쪽 값은 자신의 값 보다 큰 항목을 찾기
**Find the element before which all the elements are smaller than it, and after which all are greater**

배열 각각의 항목에 대해 모든 왼쪽 값은 자신의 값 보다 작고, 모든 오른쪽 값은 자신의 값 보다 큰가를 비교한다.

Think-1 배열 각각의 항목에 대해 모든 왼쪽, 오른쪽 항목 값을 비교 → $O(n^2)$




{5} 1 {4, 3, 6, 8, 10, 7, 9}

{5, 1} 4

{5, 1, 4} 3 {6, 8, 10, 7, 9}

{5, 1, 4, 3} 6 {8, 10, 7, 9}

# 배열의 모든 왼쪽 값은 자신의 값보다 작고, 모든 오른쪽 값은 자신의값보다 큰 항목 찾기

(출처)
https://www.geeksforgeeks.org/

배열의 모든 왼쪽 값은 자신의 값 보다 작고, 모든 오른쪽 값은 자신의 값 보다 큰 항목을 찾기
**Find the element before which all the elements are smaller than it, and after which all are greater**

**Think-1** 배열 각각의 항목에 대해 왼쪽, 오른쪽 모든 항목을 비교

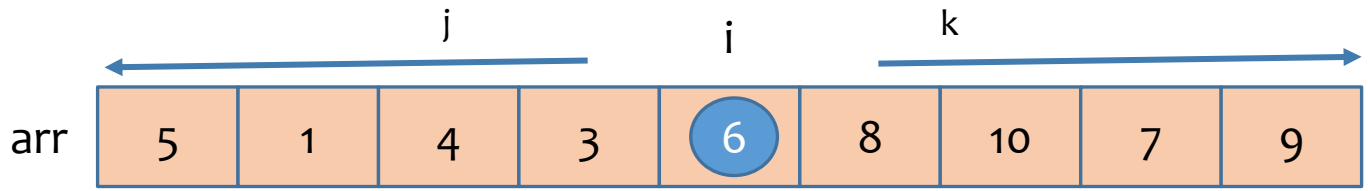
```
int findElement(int arr[], int n) {
   for (int i = 1; i < n-1; ++i) {
      int j = i-1;
      int k = i+1;
      while(arr[j] < arr[i] && arr[k] > arr[i]){
         if(j == 0 && k == n-1) {
            return i;
         }
         if(j > 0) {
            j--;
         }
         if(k < n-1){
            k++;
         }
      }
   }
   return -1;
}
```

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
```

```
int main()
{
   int arr[] = {5, 1, 4, 3, 6, 8, 10, 7, 9};
   int n = sizeof(arr) / sizeof(arr[0]);
   int idx =  findElement(arr, n);

   printf("%d", idx);
   return 0;
}
```

$O(n^2)$

4



# 배열의 모든 왼쪽 값은 자신의 값보다 작고, 모든 오른쪽 값은 자신의값보다 큰 항목 찾기

(출처)
https://www.geeksforgeeks.org/

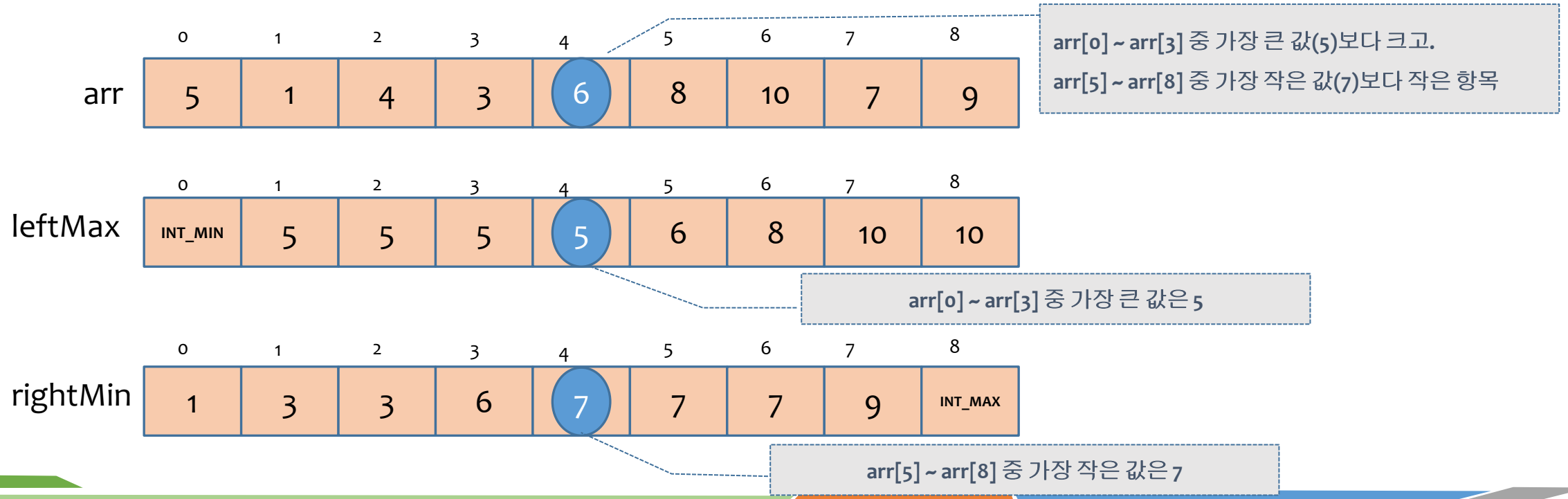
배열의 모든 왼쪽 값은 자신의 값 보다 작고, 모든 오른쪽 값은 자신의 값 보다 큰 항목을 찾기
**Find the element before which all the elements are smaller than it, and after which all are greater**

배열 각각의 항목에 대해 모든 왼쪽 값은 자신의 값 보다 작고, 모든 오른쪽 값은 자신의 값 보다 큰가를 비교한다. $O(n^2)$

→ 두 개의 추가 배열을 생성
- leftMax[] : 왼쪽 부터 시작하여 가장 큰 수를 저장
- righrMin[] : 오른쪽 부터 시작하여 가장 작은 수를 저장

O(n)

**Think-2** 두 개의 추가 배열을 생성하여 시간 복잡도를 O(n)으로 해결

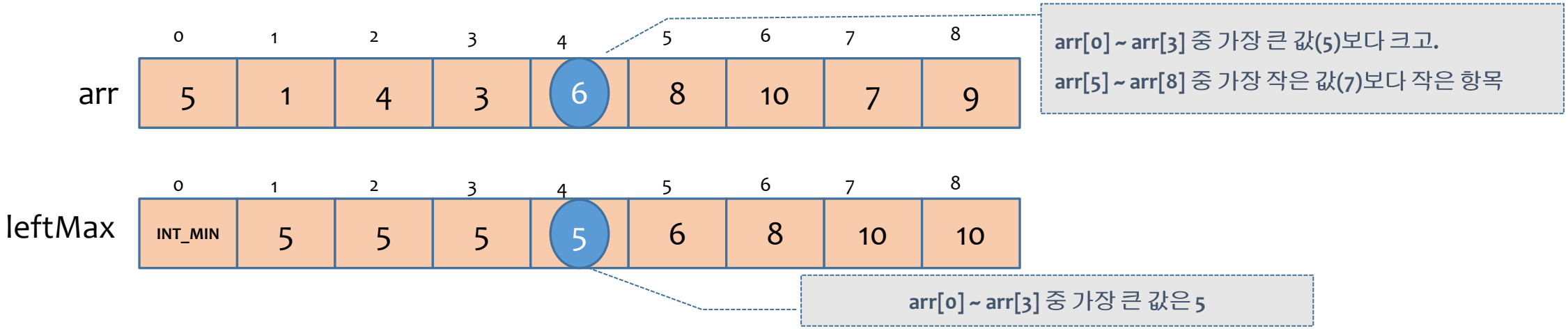


# 배열의 모든 왼쪽 값은 자신의 값보다 작고, 모든 오른쪽 값은 자신의값보다 큰 항목 찾기

(출처)
https://www.geeksforgeeks.org/

배열의 모든 왼쪽 값은 자신의 값 보다 작고, 모든 오른쪽 값은 자신의 값 보다 큰 항목을 찾기
**Find the element before which all the elements are smaller than it, and after which all are greater**

**Think-2** 두 개의 추가 배열을 생성하여 시간 복잡도를 O(n)으로 해결




```
leftMax[0] = INT_MIN;
for (int i = 1; i < n; i++)
   if(leftMax[i-1] > arr[i-1]) leftMax[i] = leftMax[i-1];
   else leftMax[i] = arr[i-1];
```

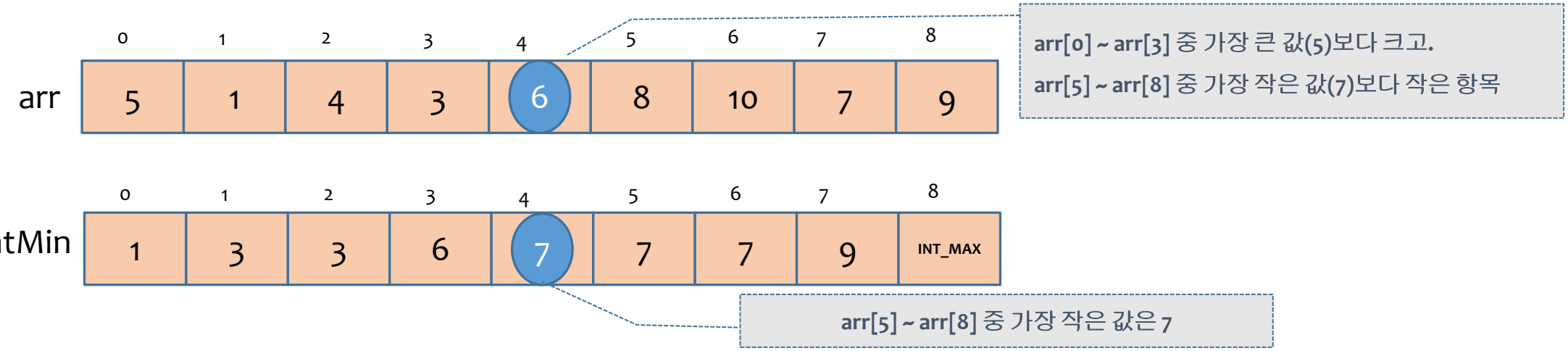
O(n)

# 배열의 모든 왼쪽 값은 자신의 값보다 작고, 모든 오른쪽 값은 자신의값보다 큰 항목 찾기

(출처)
https://www.geeksforgeeks.org/

배열의 모든 왼쪽 값은 자신의 값 보다 작고, 모든 오른쪽 값은 자신의 값 보다 큰 항목을 찾기
**Find the element before which all the elements are smaller than it, and after which all are greater**

**Think-2** 두 개의 추가 배열을 생성하여 시간 복잡도를 O(n)으로 해결




```
rightMin[n-1] = INT_MAX;
for (int i = n-2; i >= 0; i--)
   if(rightMin[i+1] > arr[i+1]) rightMin[i] = arr[i+1];
   else rightMin[i] = rightMin[i+1];
```

O(n)

# 배열의 모든 왼쪽 값은 자신의 값보다 작고, 모든 오른쪽 값은 자신의값보다 큰 항목 찾기

(출처)
https://www.geeksforgeeks.org/

배열의 모든 왼쪽 값은 자신의 값 보다 작고, 모든 오른쪽 값은 자신의 값 보다 큰 항목을 찾기
**Find the element before which all the elements are smaller than it, and after which all are greater**

배열 각각의 항목에 대해 모든 왼쪽 값은 자신의 값 보다 작고, 모든 오른쪽 값은 자신의 값 보다 큰가를 비교한다. $O(n^2)$

⇒ 두 개의 추가 배열을 생성
- leftMax[] : 왼쪽 부터 시작하여 가장 큰 수를 저장
- rightMin[] : 오른쪽 부터 시작하여 가장 작은 수를 저장

} $O(n)$

**Think-2** 두 개의 추가 배열을 생성하여 시간 복잡도를 O(n)으로 해결

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| arr | 5 | 1 | 4 | 3 | 6 | 8 | 10 | 7 | 9 |

arr[0] ~ arr[3] 중 가장 큰 값(5)보다 크고.
arr[5] ~ arr[8] 중 가장 작은 값(7)보다 작은 항목

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| leftMax | INT_MIN | 5 | 5 | 5 | 5 | 6 | 8 | 10 | 10 |

arr[0] ~ arr[3] 중 가장 큰 값은 5

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| rightMin | 1 | 3 | 3 | 6 | 7 | 7 | 7 | 9 | INT_MAX |

arr[5] ~ arr[8] 중 가장 작은 값은 7

```
for (int i=0; i < n; i++)
 {
    if (leftMax[i] < arr[i] && rightMin[i] > arr[i])
       return i;
 }
```

O(n)

# 배열의 모든 왼쪽 값은 자신의 값보다 작고, 모든 오른쪽 값은 자신의값보다 큰 항목 찾기

Think-2 두 개의 추가 배열을 사용하여 시간 복잡도 개선

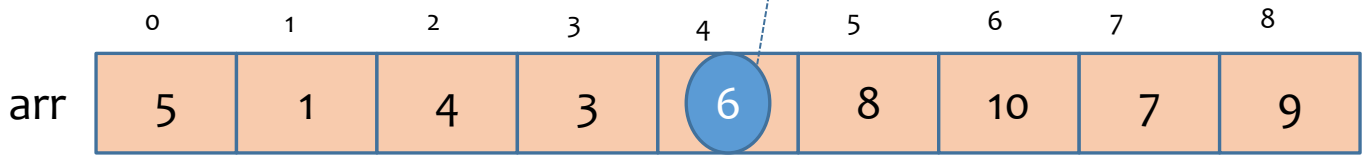
```
public static void main(String args[])    {
    int[] arr = {5, 1, 4, 3, 6, 8, 10, 7, 9};
    int n = arr.length;
    System.out.println("Index of the element is " +
    findElement(arr, n));
}
static int findElement(int[] arr, int n)    {
     int[] leftMax  = new int[n];
     int[] rightMin = new int[n];
     leftMax[0] = Integer.MIN_VALUE;
     for (int i = 1; i < n; i++)
          leftMax[i] = Math.max(leftMax[i - 1], arr[i - 1]);
     rightMin[n-1] = Integer.MAX_VALUE;
     for (int i = n-2; i >= 0; i--) {
         if(rightMin[i+1] > arr[i+1]) rightMin[i] = arr[i+1];
         else rightMin[i] = rightMin[i+1];
         //System.out.println("i=" + i +","+ rightMin[i] + ","+ arr[i] + "," + rightMin[i+1]);
     }

     for (int i = n - 1; i >= 0; i--)      {
          System.out.print("i=" + i +","+ leftMax[i] + ","+ arr[i] + ",");
          if (leftMax[i] < arr[i] && rightMin[i] > arr[i])
              return i;
          rightMin[i] = Math.min(rightMin[i], arr[i]);
          System.out.println(rightMin[i]);
     }
     return -1;
 }
```

O(n)

4

arr[0] ~ arr[3] 중 가장 큰 값(5)보다 크고.

arr[5] ~ arr[8] 중 가장 작은 값(7)보다 작은 항목



# 배열의 모든 왼쪽 값은 자신의 값보다 작고, 모든 오른쪽 값은 자신의값보다 큰 항목 찾기

**Think-2** 두 개의 추가 배열을 사용하여 시간 복잡도 개선

```
int findElement(int arr[], int n) {
   int leftMax[n], rightMin[n];

   leftMax[0] = INT_MIN;
   for (int i = 1; i < n; i++)
      if(leftMax[i-1] > arr[i-1]) leftMax[i] = leftMax[i-1];
      else leftMax[i] = arr[i-1];

   rightMin[n-1] = INT_MAX;
   for (int i = n-2; i >= 0; i--)
      if(rightMin[i+1] > arr[i+1]) rightMin[i] = arr[i+1];
      else rightMin[i] = rightMin[i+1];

   for (int i=0; i < n; i++)  {
      if (leftMax[i] < arr[i] && rightMin[i] > arr[i])
          return i;
   }
   return -1;
}
```

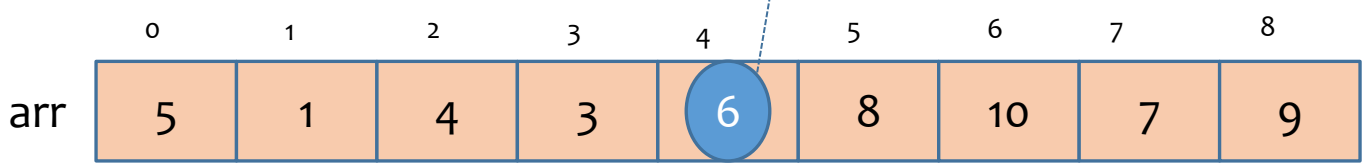
```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <limits.h>
```

```
int main()
{
   int arr[] = {5, 1, 4, 3, 6, 8, 10, 7, 9};
   int n = sizeof(arr) / sizeof(arr[0]);
   int idx =  findElement(arr, n);

   printf("%d", idx);
   return 0;
}
```

O(n)

4

arr[0] ~ arr[3] 중 가장 큰 값(5)보다 크고.
arr[5] ~ arr[8] 중 가장 작은 값(7)보다 작은 항목



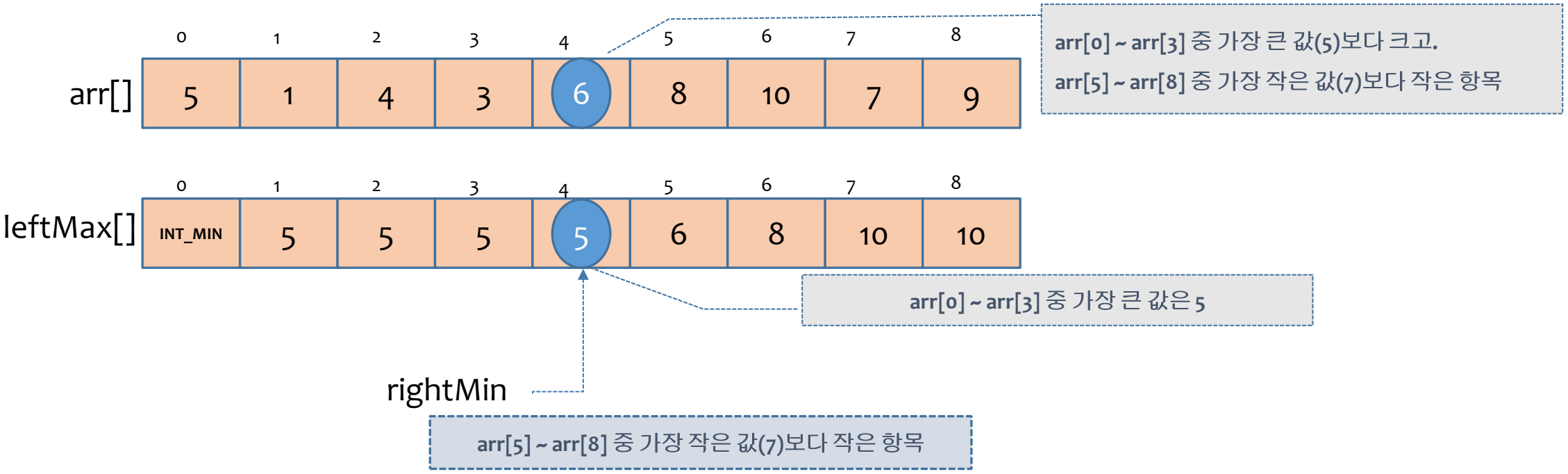
# 배열의 모든 왼쪽 값은 자신의 값보다 작고, 모든 오른쪽 값은 자신의값보다 큰 항목 찾기

두 개의 추가 배열을 생성
- leftMax[] : 왼쪽 부터 시작하여 가장 큰 수를 저장
- righrMin[] : 오른쪽 부터 시작하여 가장 작은 수를 저장

O(n)

➡ 2개의 추가 공간(leftMax[], rightMin[])을 1개(leftMax[])로 줄임

**Think-3** 2개의 추가 공간(leftMax[], rightMin[])을 1개(leftMax[])로 줄임



# 배열의 모든 왼쪽 값은 자신의 값보다 작고, 모든 오른쪽 값은 자신의값보다 큰 항목 찾기 (Java예시)

**Think-3** 2개의 추가 공간(leftMax[], rightMin[])을 1개(leftMax[])로 줄임

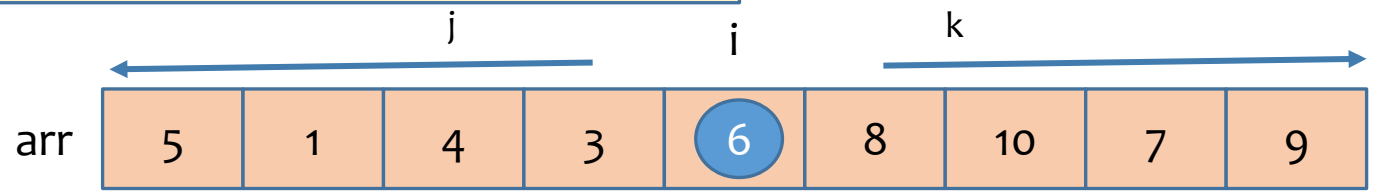
# 배열의 모든 왼쪽 값은 자신의 값보다 작고, 모든 오른쪽 값은 자신의값보다 큰 항목 찾기 (C예시)

Think-3 배열 각각의 항목에 대해 왼쪽, 오른쪽 모든 항목을 비교

```
int main()
{
   int arr[] = {5, 1, 4, 3, 6, 8, 10, 7, 9};
   int n = sizeof(arr) / sizeof(arr[0]);
   int idx =  findElement(arr, n);

   printf("%d", idx);
   return 0;
}
```

O(n)

4



# <문제2>문자열 재정렬 : 문제 조건

- 알파벳 대문자와 숫자(0 ~ 9)로만 구성된 문자열이 입력으로 주어집니다. 이때 모든 알파벳을 오름차순으로 정렬하여 이어서 출력한 뒤에, 그 뒤에 모든 숫자를 더한 값을 이어서 출력합니다.
- 예를 들어 K1KA5CB7이라는 값이 들어오면 ABCKK13을 출력합니다.

# <문제>문자열 재정렬 : 문제 조건

**난이도** ●○○ | **풀이 시간** 20분 | **시간 제한** 1초 | **메모리 제한** 128MB | **기출** Facebook 인터뷰

**입력 조건**
- 첫째 줄에 하나의 문자열 S가 주어집니다. (1 ≤ S의 길이 ≤ 10,000)

**출력 조건**
- 첫째 줄에 문제에서 요구하는 정답을 출력합니다.

**입력 예시 1**

```
K1KA5CB7
```

**출력 예시 1**

```
ABCKK13
```

**입력 예시 2**

```
AJKDLSI412K4JSJ9D
```

**출력 예시 2**

```
ADDIJJJKKLSS20
```

# <문제>문자열 재정렬 : 문제 해결 아이디어

- 요구사항대로 충실히 구현하면 되는 문제입니다.
- 문자열이 입력되었을 때 문자를 하나씩 확인합니다.
  - 숫자인 경우 따로 합계를 계산합니다.
  - 알파벳 경우 별도의 리스트에 저장합니다.
- 결과적으로 리스트에 저장된 알파벳을 정렬해 출력하고, 합계를 뒤에 붙여 출력하면 정답입니다.

# <문제>문자열 재정렬 : 답안 예시(Python)

# <문제>문자열 재정렬 : 답안 예시(Java)

# <문제>문자열 재정렬 : 답안 예시(C++)