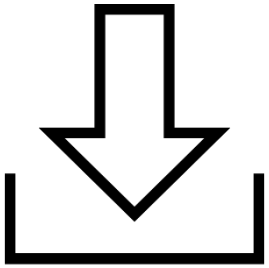
C언어
정렬 알고리즘
세미나

숭실대학교 로보틱스 25기 김동현

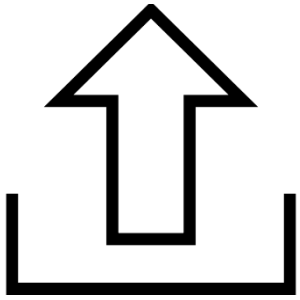
목차

1 알고리즘(Algorithm)

어떠한 문제를 해결하기 위한 여러 동작들의 모임.
입력, 출력, 명확성, 유한성, 효율성을 가지고 있어야 한다.

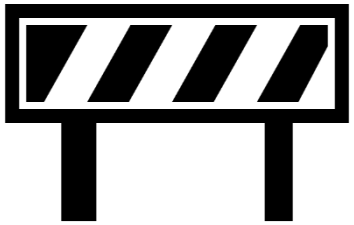


입력
외부에서 제공되는 자료가
0개 이상 존재한다.

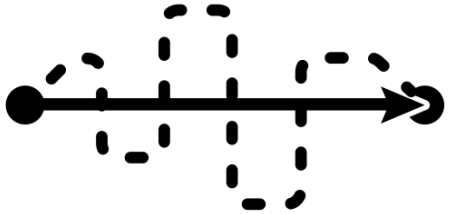


출력
알고리즘은 최소 1개 이상의
결과를 가진다.

명확성
수행 과정은 명확하고
모호하지 않은 명령어로
구성되어야 한다.



유한성
유한 번의 명령어를 수행 후
시간 내에 종료한다.



효율성
모든 과정은 명백하게
실행 가능한 것이어야 한다.

1 알고리즘(Algorithm)

같은 문자열을 10번 출력할 때, 같은 명령을 10번 내리는 것 대신
반복문을 이용하면 더 짧고 간결한 소스를 짤 수 있다.

“로보틱스” 문자열 10번 출력

```
printf(“로보틱스₩n”);
printf(“로보틱스₩n”);
printf(“로보틱스₩n”);
printf(“로보틱스₩n”);
printf(“로보틱스₩n”);
printf(“로보틱스₩n”);
printf(“로보틱스₩n”);
printf(“로보틱스₩n”);
printf(“로보틱스₩n”);
printf(“로보틱스₩n”);
```

VS

```
int i;
for(i=0;i<10;i++)
{
    printf(“로보틱스₩n”);
}
```

1 알고리즘(Algorithm)

알고리즘의 복잡도는 크게 시간복잡도와 공간복잡도로 나뉘고
복잡도를 최소화 할수록 더 좋은 알고리즘이다.

알고리즘 복잡도

시간복잡도

알고리즘 패턴을 통해 대략적인 시간을 고려.

Big O notation을 이용하여 나타냄.

빠름빠름 $O(1)<O(\log_2 n)<O(n)<O(n\log_2 n)<O(n^2)<O(n^3)<O(2^n)<O(n!)$ 극혐

1 알고리즘(Algorithm)

알고리즘의 복잡도는 크게 시간복잡도와 공간복잡도로 나뉘고
복잡도를 최소화 할수록 더 좋은 알고리즘이다.

알고리즘 복잡도

공간의 낭비를 최소화 하여 메모리 절약

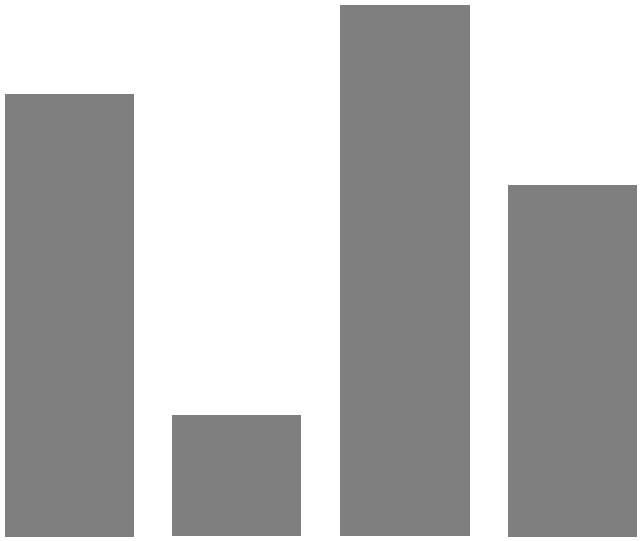
분할정복 알고리즘과 DP알고리즘에서 중요

Malloc함수를 이용하여 포인터에 동적 할당된 메모리를 배열처럼 사용가능

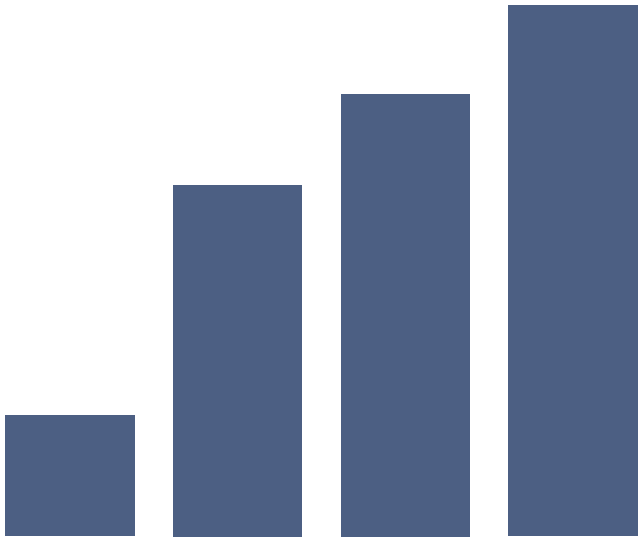
1 알고리즘(Algorithm)

본격적인 정렬알고리즘 설명
화면에 사용된 애니메이션은 버블정렬을 이용한 정렬이다.

정렬알고리즘?



Before



After

1 알고리즘(Algorithm)

정렬알고리즘의 종류와 시간복잡도
이번 세미나에서는 $O(n^2)$시간복잡도의 정렬을 설명한다.

$O(n^2)$ 정렬

버블정렬

선택정렬

삽입정렬

$O(n \log_2 n)$ 정렬

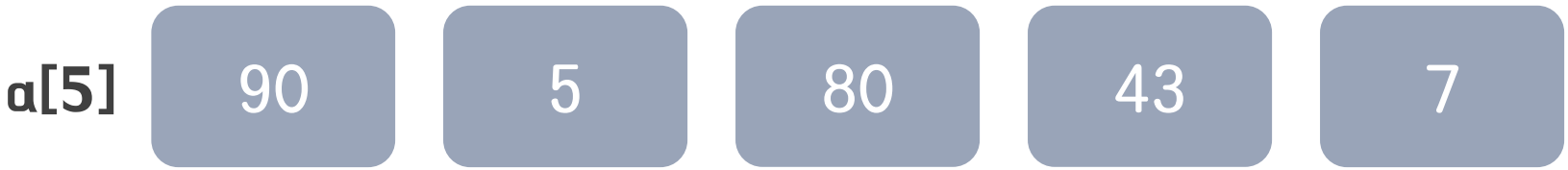
합병정렬

힙정렬

퀵정렬

2 버블(교환)정렬

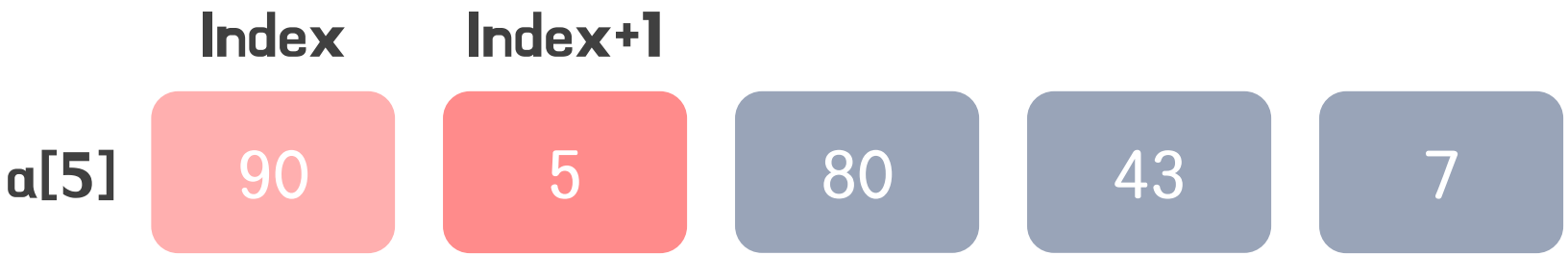
Index를 1씩 증가시키며 (index)번째와 (index+1)번째의 값을 비교하며
최대 n(n-1)/2번 정렬한다.



2 버블(교환)정렬

Index를 1씩 증가시키며 (index)번째와 (index+1)번째의 값을 비교하며 최대 n(n-1)/2번 정렬한다.

```
void swap(int *cmp1, int *cmp2)
{
    int tmp;
    tmp=*cmp1;
    *cmp1=*cmp2;
    *cmp2=tmp;
}
```

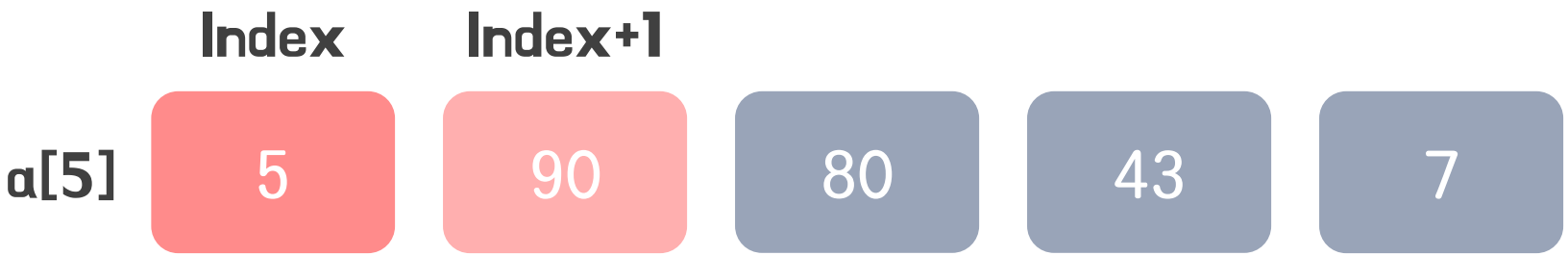


If(a[index]>a[index+1])

swap(&a[index], &a[index+1])

2 버블(교환)정렬

Index를 1씩 증가시키며 (index)번째와 (index+1)번째의 값을 비교하며
최대 n(n-1)/2번 정렬한다.

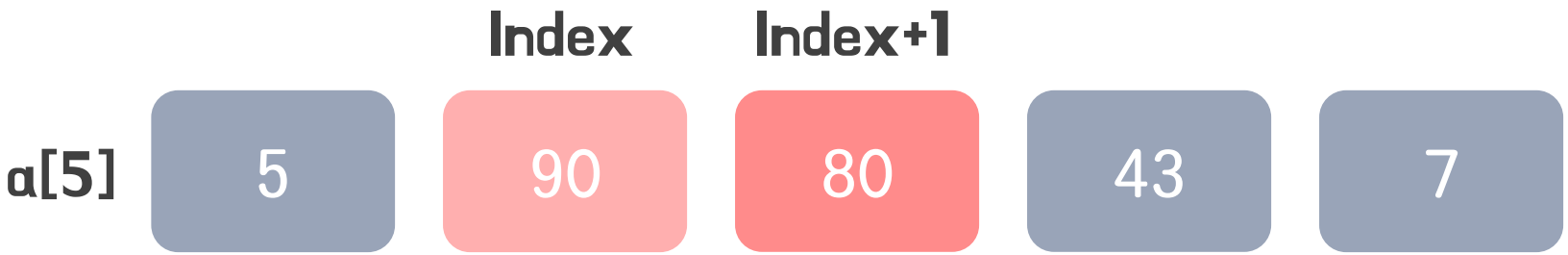


If(a[index]>a[index+1])

swap(&a[index], &a[index+1])

2 버블(교환)정렬

Index를 1씩 증가시키며 (index)번째와 (index+1)번째의 값을 비교하며
최대 n(n-1)/2번 정렬한다.

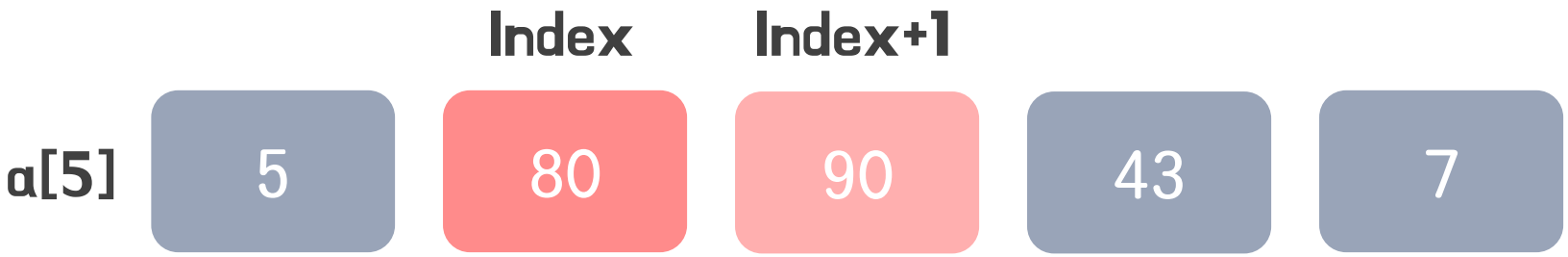


If(a[index]>a[index+1])

swap(&a[index], &a[index+1])

2 버블(교환)정렬

Index를 1씩 증가시키며 (index)번째와 (index+1)번째의 값을 비교하며
최대 n(n-1)/2번 정렬한다.

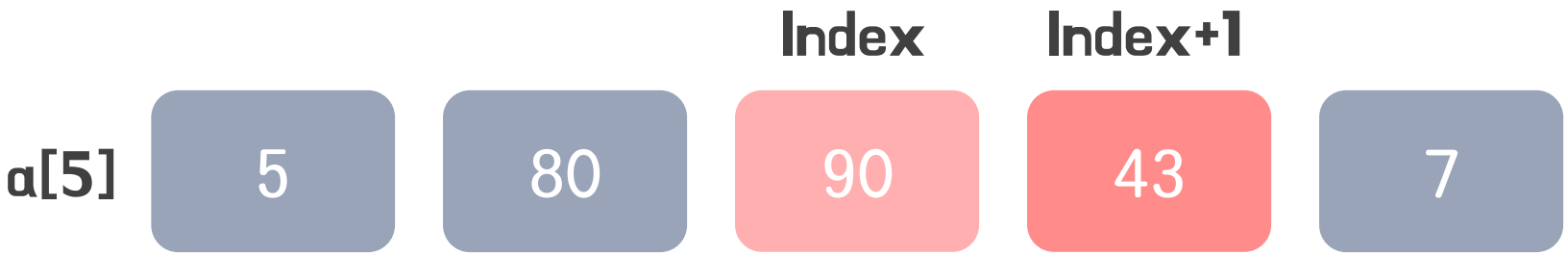


If(a[index]>a[index+1])

swap(&a[index], &a[index+1])

2 버블(교환)정렬

Index를 1씩 증가시키며 (index)번째와 (index+1)번째의 값을 비교하며
최대 n(n-1)/2번 정렬한다.

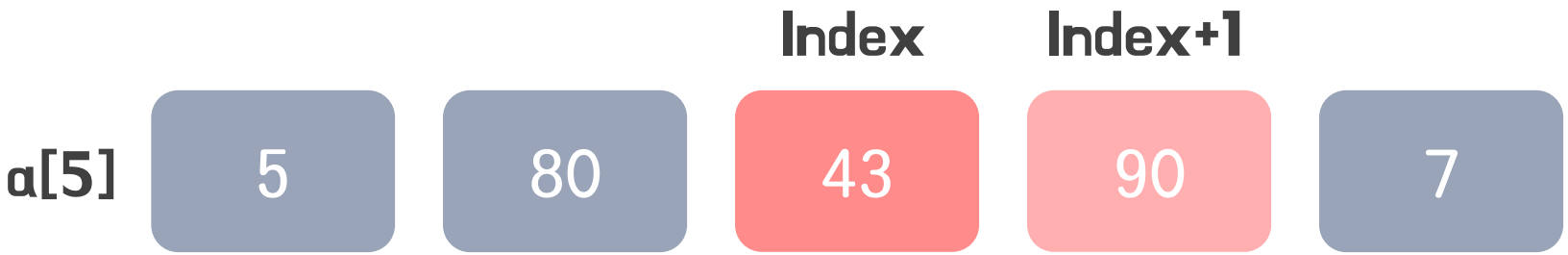


If(a[index]>a[index+1])

swap(&a[index], &a[index+1])

2 버블(교환)정렬

Index를 1씩 증가시키며 (index)번째와 (index+1)번째의 값을 비교하며
최대 n(n-1)/2번 정렬한다.

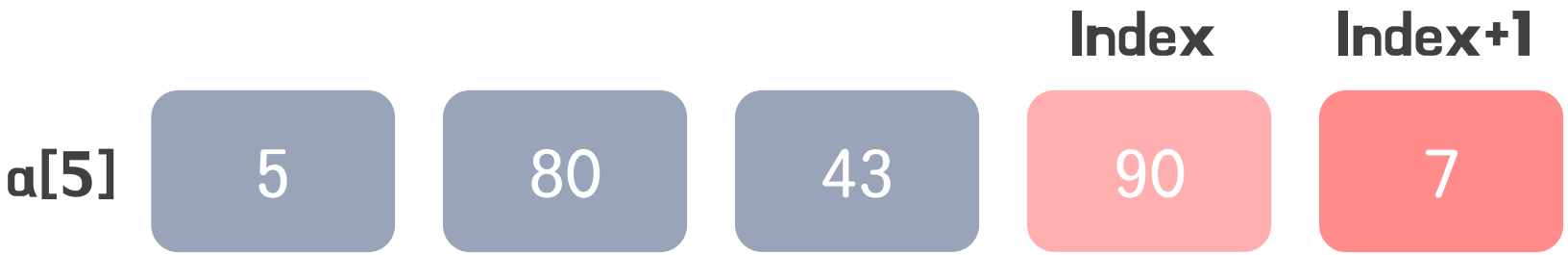


If(a[index]>a[index+1])

swap(&a[index], &a[index+1])

2 버블(교환)정렬

Index를 1씩 증가시키며 (index)번째와 (index+1)번째의 값을 비교하며
최대 n(n-1)/2번 정렬한다.

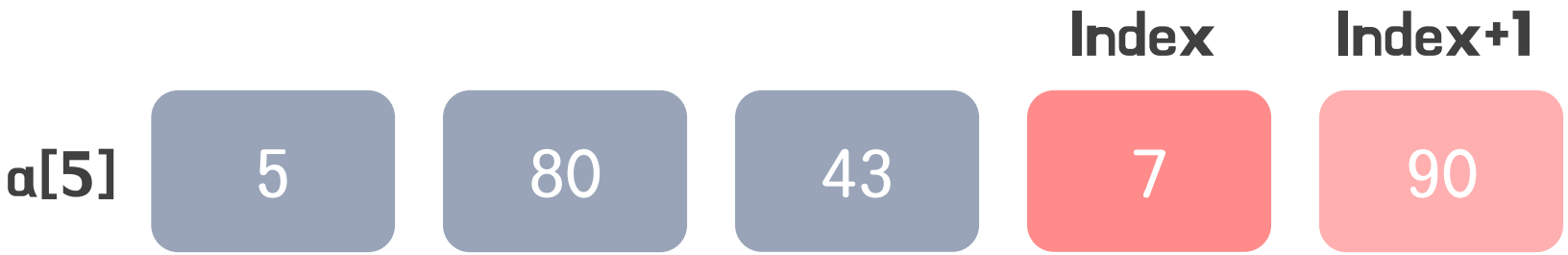


If(a[index]>a[index+1])

swap(&a[index], &a[index+1])

2 버블(교환)정렬

Index를 1씩 증가시키며 (index)번째와 (index+1)번째의 값을 비교하며
최대 n(n-1)/2번 정렬한다.

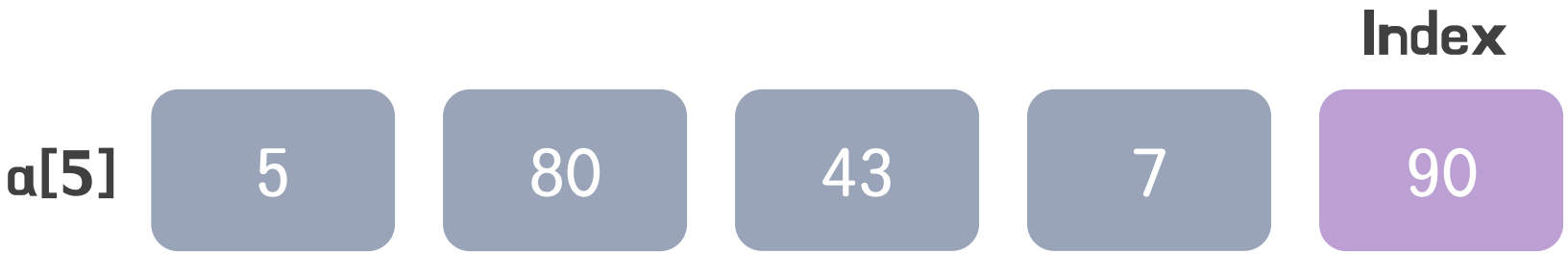


If(a[index]>a[index+1])

swap(&a[index], &a[index+1])

2 버블(교환)정렬

Index를 1씩 증가시키며 (index)번째와 (index+1)번째의 값을 비교하며
최대 n(n-1)/2번 정렬한다.

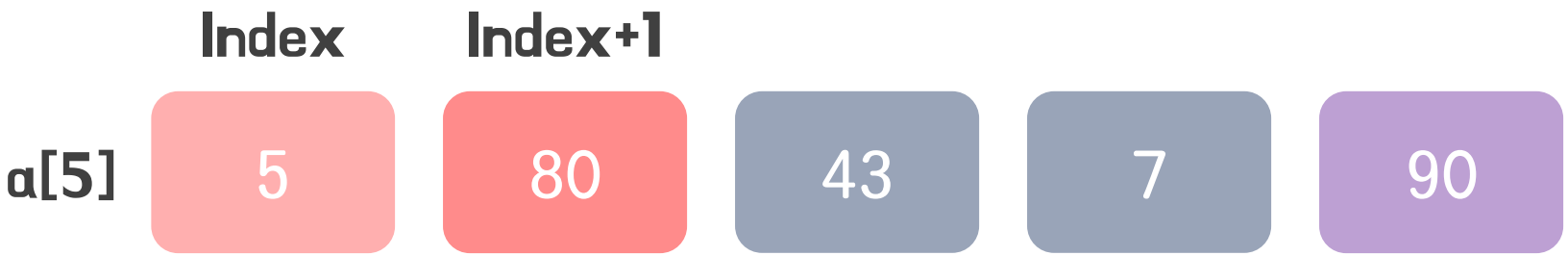



```
If(a[index]>a[index+1])

swap(&a[index], &a[index+1])
```

2 버블(교환)정렬

Index를 1씩 증가시키며 (index)번째와 (index+1)번째의 값을 비교하며
최대 n(n-1)/2번 정렬한다.

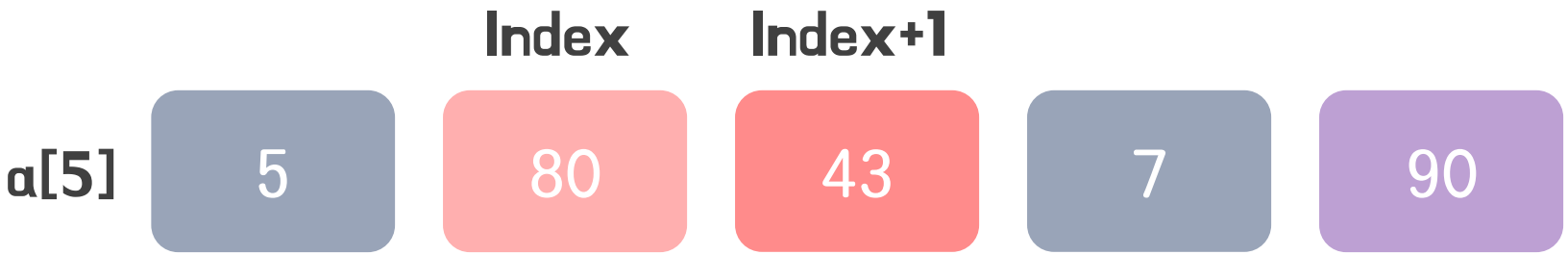


If(a[index]>a[index+1])

swap(&a[index], &a[index+1])

2 버블(교환)정렬

Index를 1씩 증가시키며 (index)번째와 (index+1)번째의 값을 비교하며
최대 n(n-1)/2번 정렬한다.

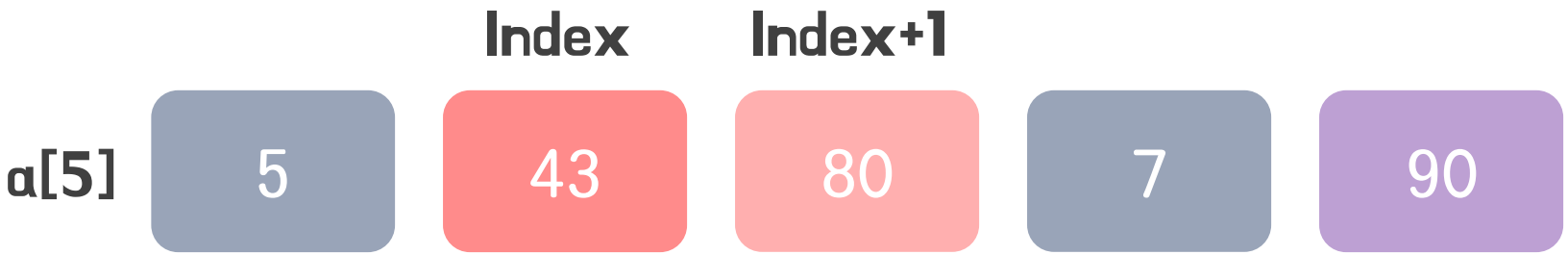


If(a[index]>a[index+1])

swap(&a[index], &a[index+1])

2 버블(교환)정렬

Index를 1씩 증가시키며 (index)번째와 (index+1)번째의 값을 비교하며
최대 n(n-1)/2번 정렬한다.

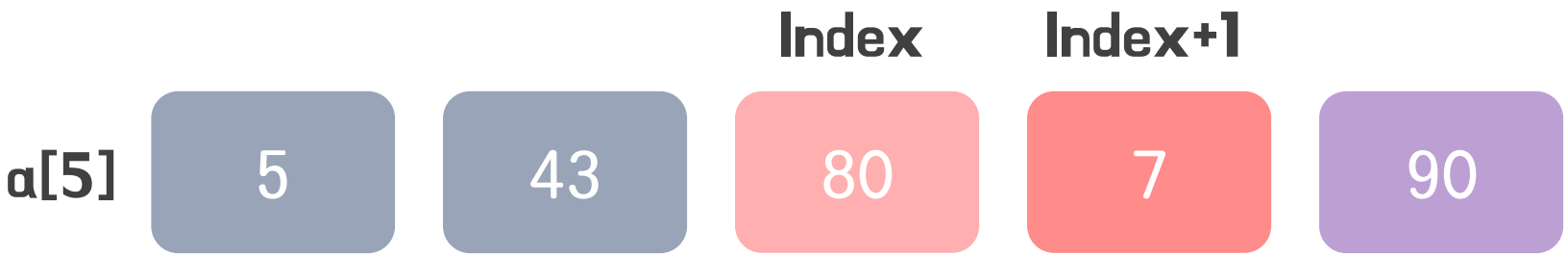


If(a[index]>a[index+1])

swap(&a[index], &a[index+1])

2 버블(교환)정렬

Index를 1씩 증가시키며 (index)번째와 (index+1)번째의 값을 비교하며
최대 n(n-1)/2번 정렬한다.

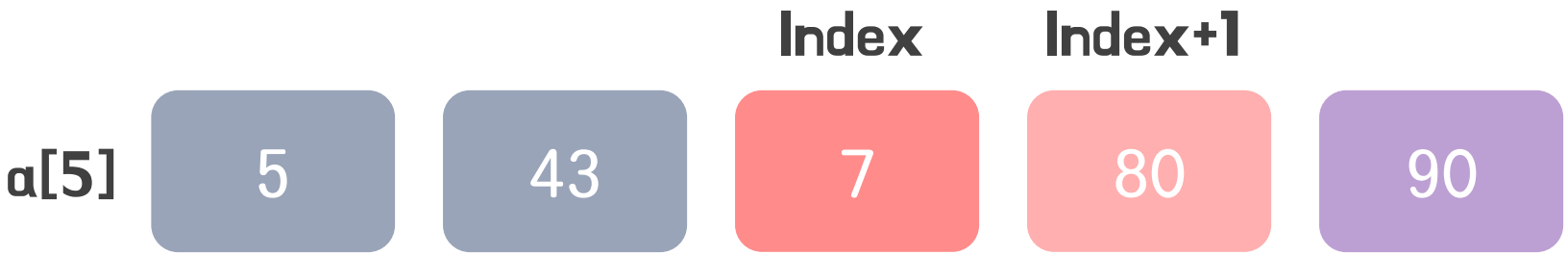


If(a[index]>a[index+1])

swap(&a[index], &a[index+1])

2 버블(교환)정렬

Index를 1씩 증가시키며 (index)번째와 (index+1)번째의 값을 비교하며
최대 n(n-1)/2번 정렬한다.



If(a[index]>a[index+1])

swap(&a[index], &a[index+1])

2 버블(교환)정렬

Index를 1씩 증가시키며 (index)번째와 (index+1)번째의 값을 비교하며
최대 n(n-1)/2번 정렬한다.


```
If(a[index]>a[index+1])

swap(&a[index], &a[index+1])
```

2 버블(교환)정렬

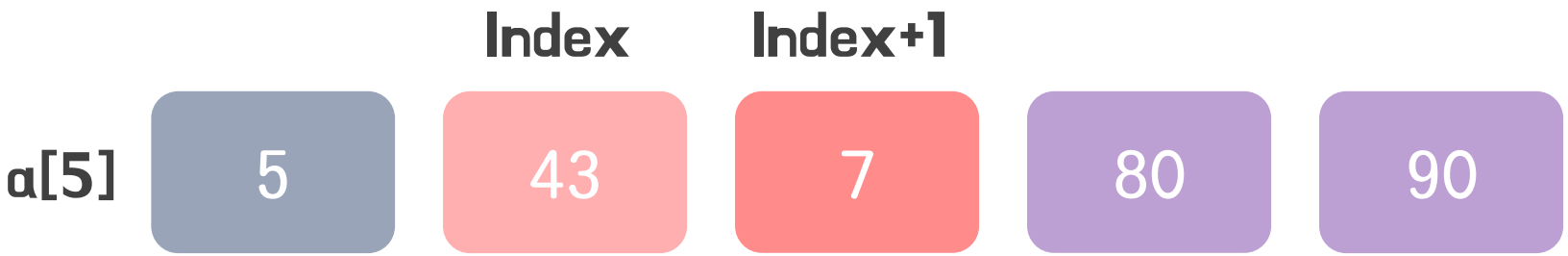
Index를 1씩 증가시키며 (index)번째와 (index+1)번째의 값을 비교하며
최대 n(n-1)/2번 정렬한다.

If(a[index]>a[index+1])

swap(&a[index], &a[index+1])

2 버블(교환)정렬

Index를 1씩 증가시키며 (index)번째와 (index+1)번째의 값을 비교하며
최대 n(n-1)/2번 정렬한다.

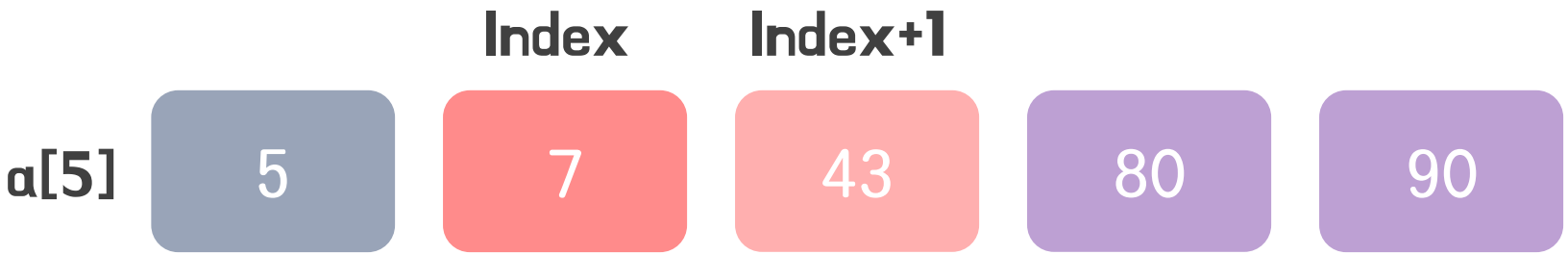


If(a[index]>a[index+1])

swap(&a[index], &a[index+1])

2 버블(교환)정렬

Index를 1씩 증가시키며 (index)번째와 (index+1)번째의 값을 비교하며
최대 n(n-1)/2번 정렬한다.

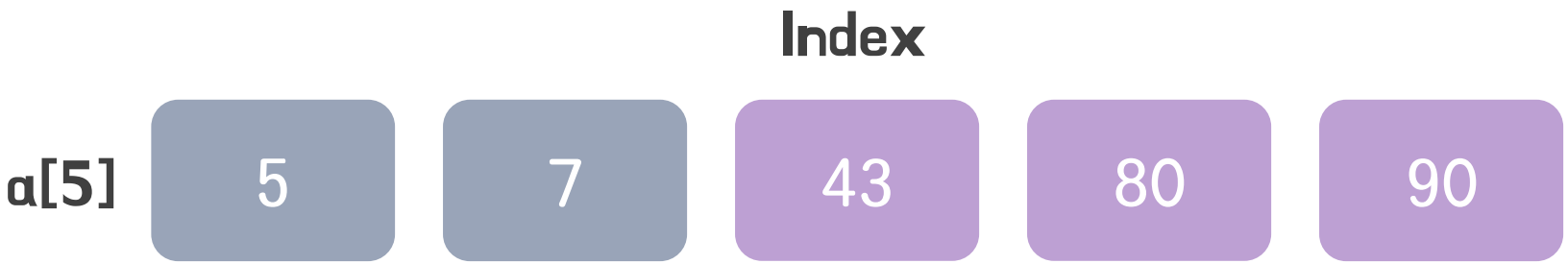


If(a[index]>a[index+1])

swap(&a[index], &a[index+1])

2 버블(교환)정렬

Index를 1씩 증가시키며 (index)번째와 (index+1)번째의 값을 비교하며
최대 n(n-1)/2번 정렬한다.

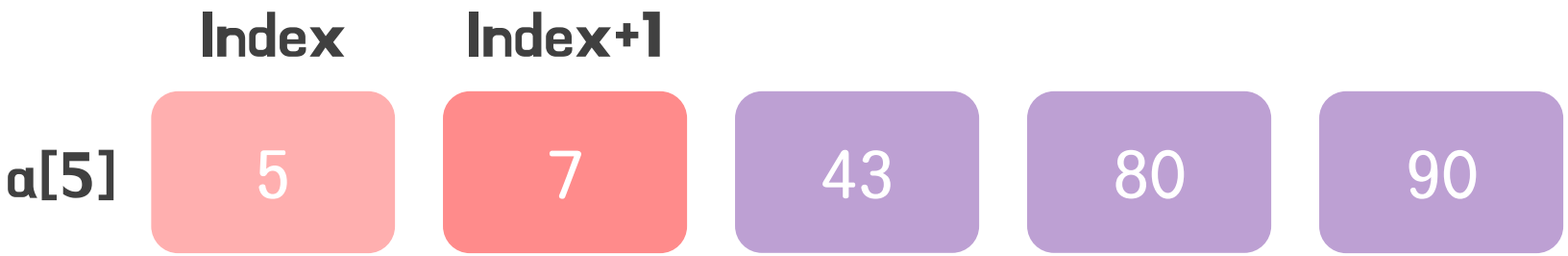



```
If(a[index]>a[index+1])

swap(&a[index], &a[index+1])
```

2 버블(교환)정렬

Index를 1씩 증가시키며 (index)번째와 (index+1)번째의 값을 비교하며
최대 n(n-1)/2번 정렬한다.

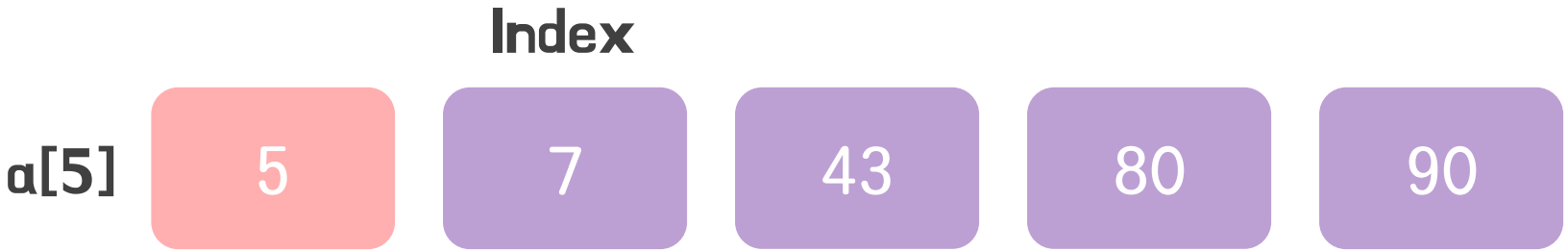


If(a[index]>a[index+1])

swap(&a[index], &a[index+1])

2 버블(교환)정렬

Index를 1씩 증가시키며 (index)번째와 (index+1)번째의 값을 비교하며
최대 n(n-1)/2번 정렬한다.

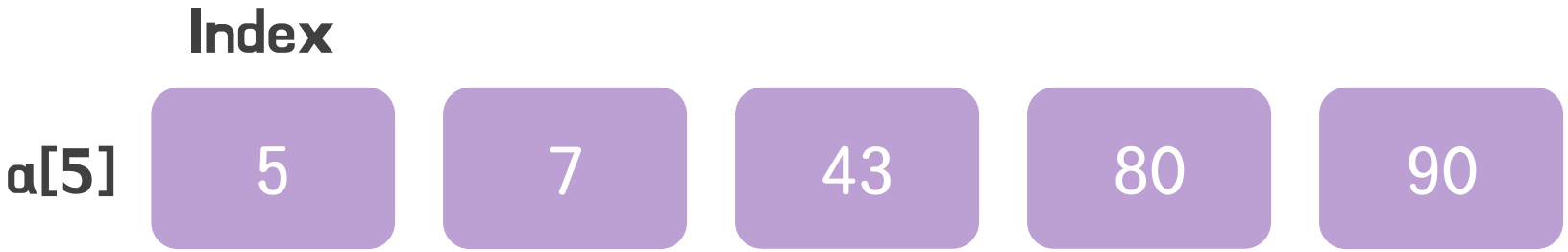



```
If(a[index]>a[index+1])

swap(&a[index], &a[index+1])
```

2 버블(교환)정렬

Index를 1씩 증가시키며 (index)번째와 (index+1)번째의 값을 비교하며
최대 n(n-1)/2번 정렬한다.




```
If(a[index]>a[index+1])

swap(&a[index], &a[index+1])
```

2 버블(교환)정렬

Index를 1씩 증가시키며 (index)번째와 (index+1)번째의 값을 비교하며
최대 n(n-1)/2번 정렬한다.

알고리즘 복잡도

시간복잡도

	최선의 경우	최악의 경우
비교횟수	N	N(N-1)/2
교환횟수	0	N(N-1)/2

평균 시간복잡도 $O(n^2)$

2 버블(교환)정렬

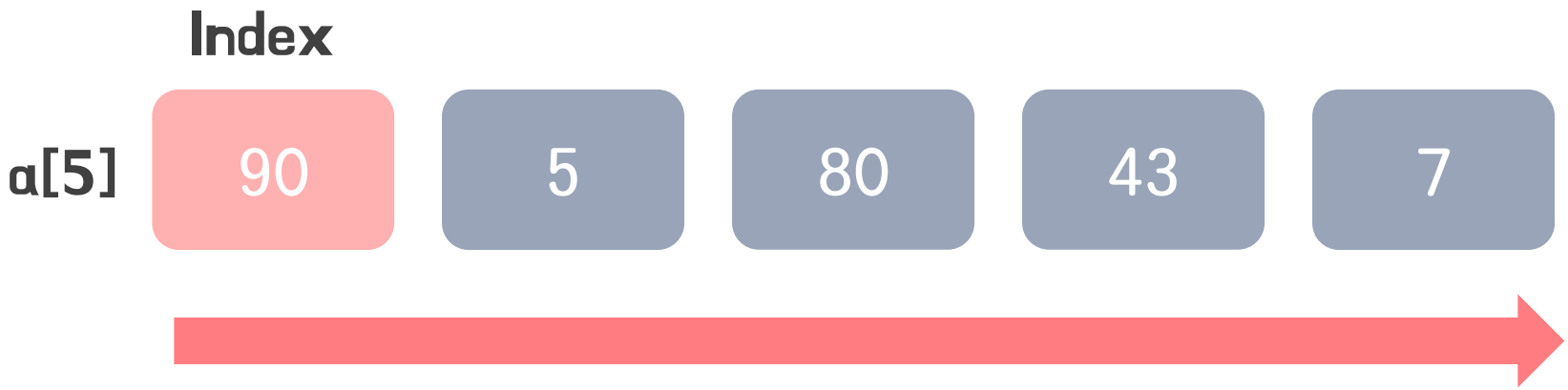
Index를 1씩 증가시키며 (index)번째와 (index+1)번째의 값을 비교하며
최대 n(n-1)/2번 정렬한다.

알고리즘 복잡도

N개의 데이터 -> N개의 공간

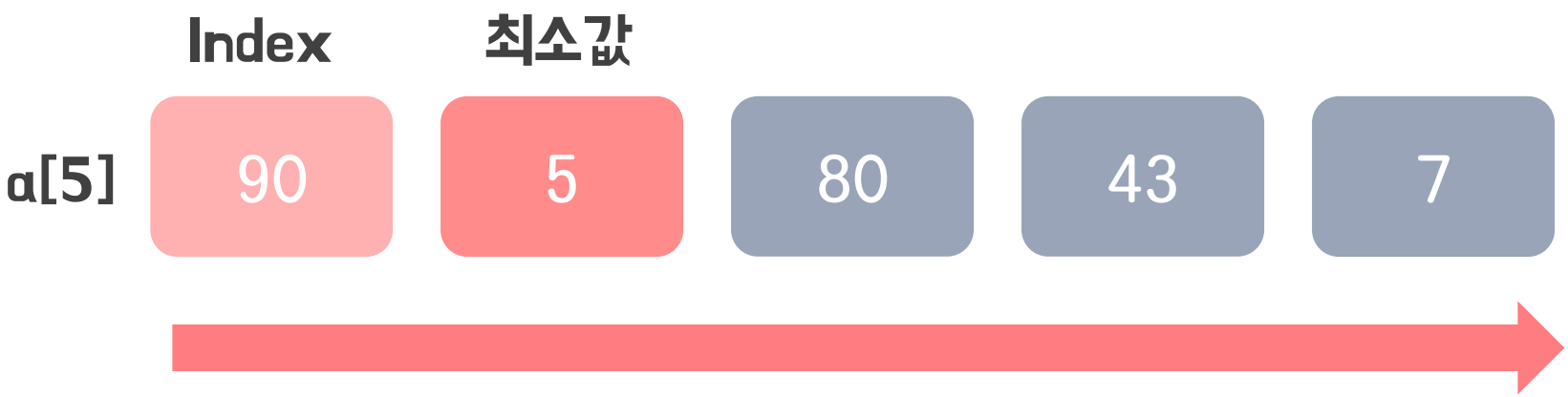
3 선택정렬

Index를 기준으로 최소(오름차순)값을 찾고
Index보다 작은 최소값이 있다면, 최소값과 index를 교환한다.



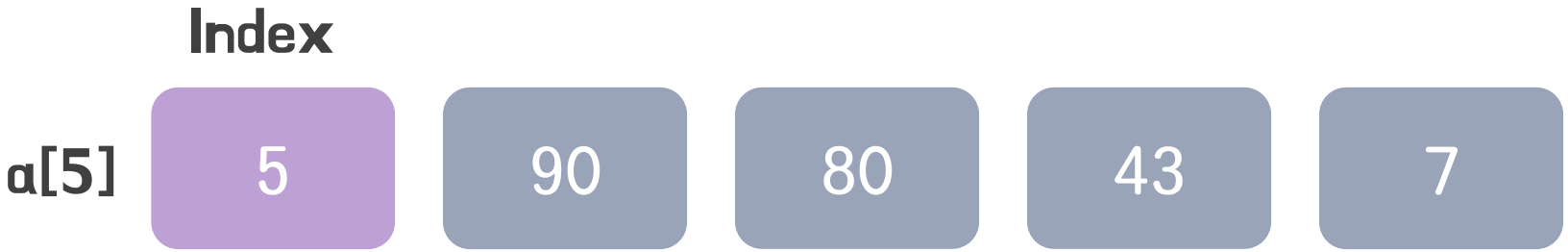
3 선택정렬

Index를 기준으로 최소(오름차순)값을 찾고
Index보다 작은 최소값이 있다면, 최소값과 index를 교환한다.



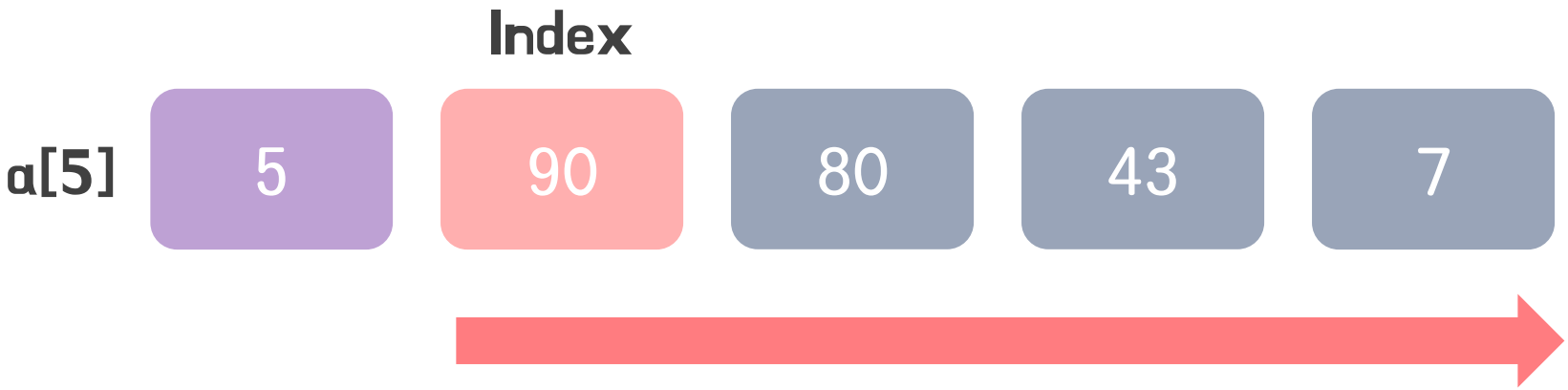
3 선택정렬

Index를 기준으로 최소(오름차순)값을 찾고
Index보다 작은 최소값이 있다면, 최소값과 index를 교환한다.



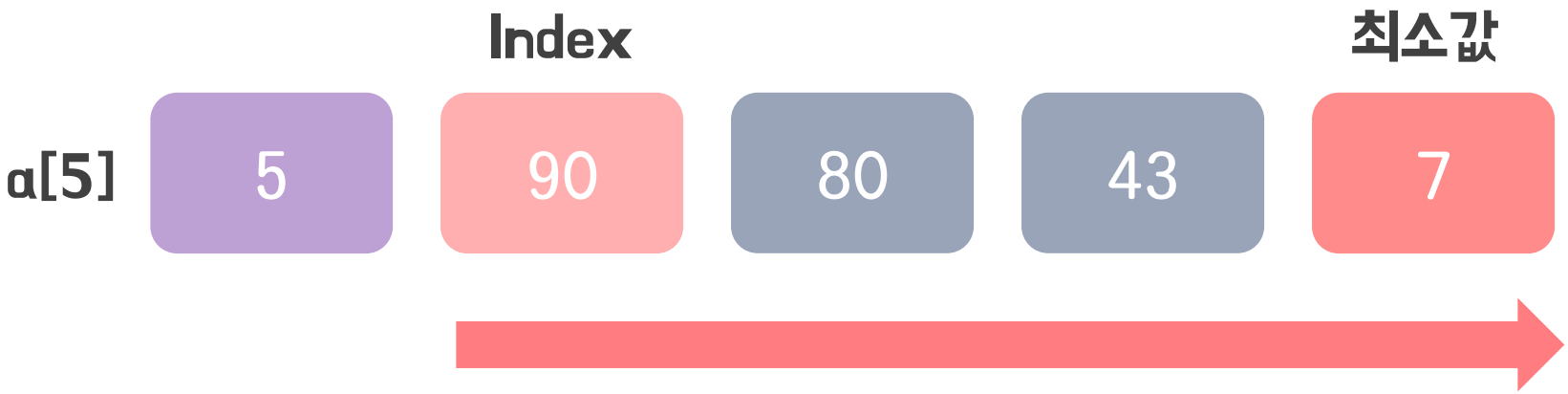
3 선택정렬

Index를 기준으로 최소(오름차순)값을 찾고
Index보다 작은 최소값이 있다면, 최소값과 index를 교환한다.



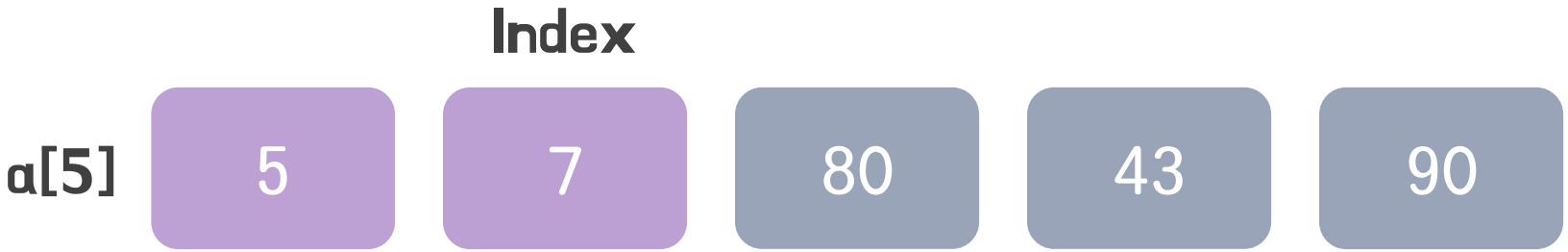
3 선택정렬

Index를 기준으로 최소(오름차순)값을 찾고
Index보다 작은 최소값이 있다면, 최소값과 index를 교환한다.



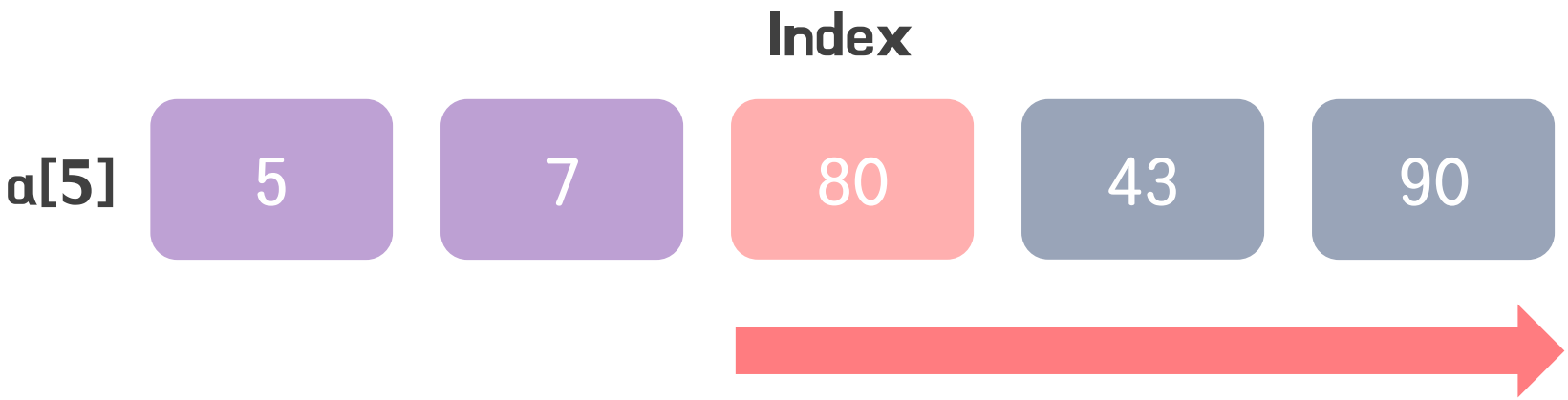
3 선택정렬

Index를 기준으로 최소(오름차순)값을 찾고
Index보다 작은 최소값이 있다면, 최소값과 index를 교환한다.



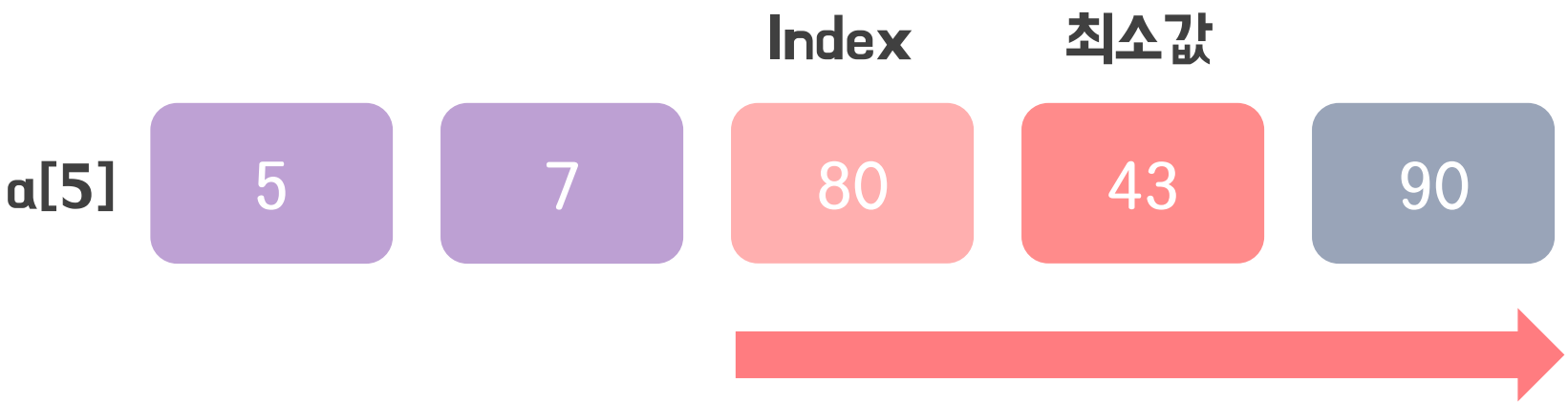
3 선택정렬

Index를 기준으로 최소(오름차순)값을 찾고
Index보다 작은 최소값이 있다면, 최소값과 index를 교환한다.



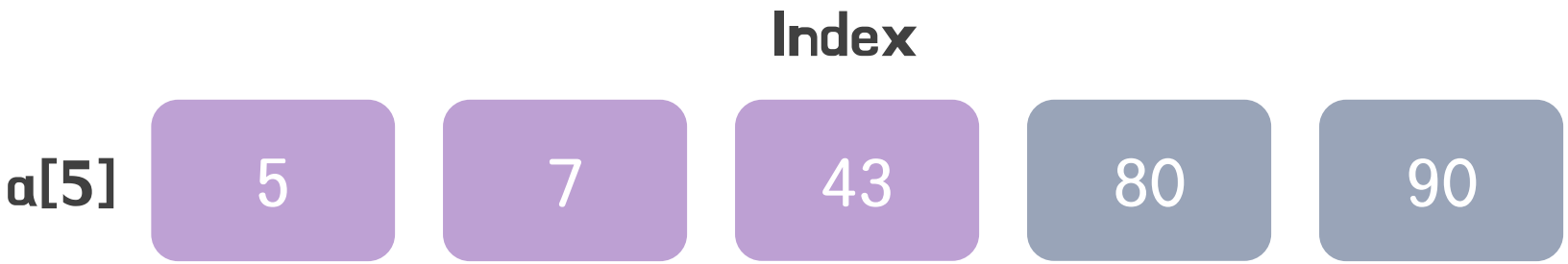
3 선택정렬

Index를 기준으로 최소(오름차순)값을 찾고
Index보다 작은 최소값이 있다면, 최소값과 index를 교환한다.



3 선택정렬

Index를 기준으로 최소(오름차순)값을 찾고
Index보다 작은 최소값이 있다면, 최소값과 index를 교환한다.

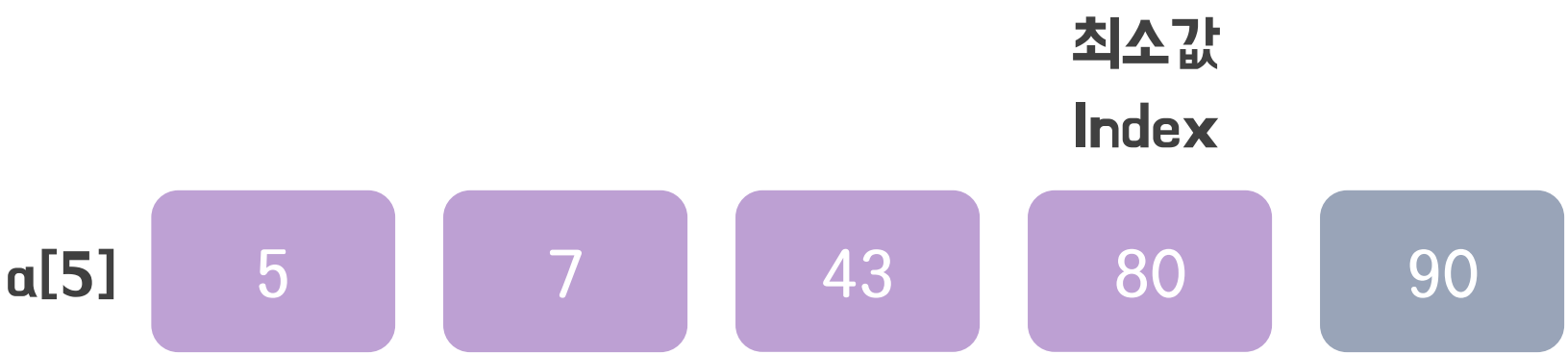


3 선택정렬

Index를 기준으로 최소(오름차순)값을 찾고
Index보다 작은 최소값이 있다면, 최소값과 index를 교환한다.

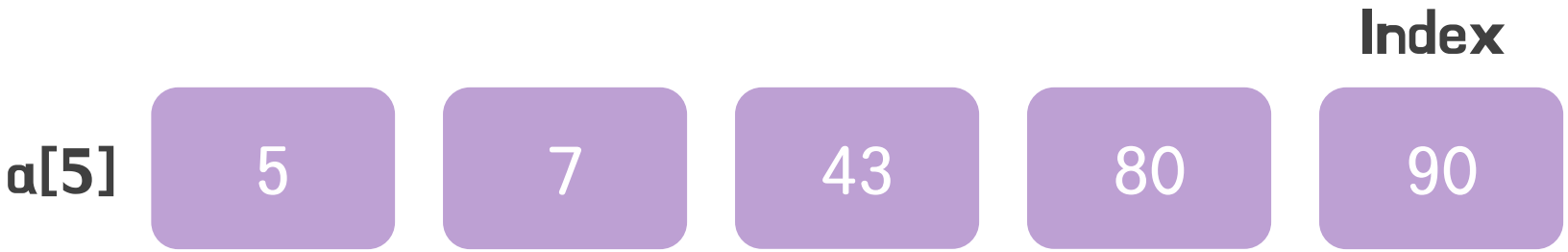
3 선택정렬

Index를 기준으로 최소(오름차순)값을 찾고
Index보다 작은 최소값이 있다면, 최소값과 index를 교환한다.



3 선택정렬

Index를 기준으로 최소(오름차순)값을 찾고
Index보다 작은 최소값이 있다면, 최소값과 index를 교환한다.



3 선택정렬

Index를 기준으로 최소(오름차순)값을 찾고
Index보다 작은 최소값이 있다면, 최소값과 index를 교환한다.

알고리즘 복잡도

시간복잡도

	최선의 경우	최악의 경우
비교횟수	N	N(N-1)/2
교환횟수	0	N(N-1)/2

평균 시간복잡도 $O(n^2)$

3 선택정렬

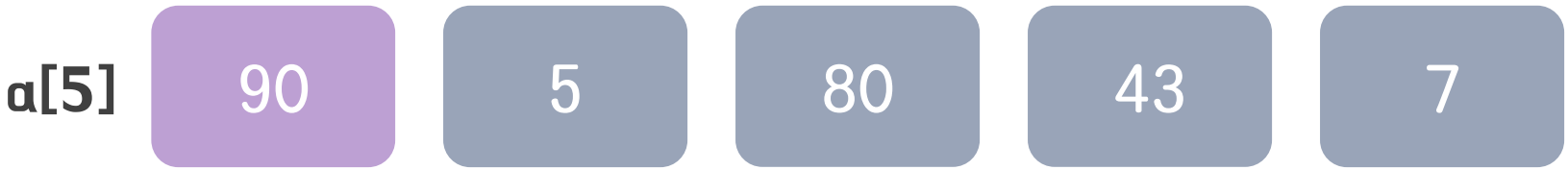
Index를 기준으로 최소(오름차순)값을 찾고
Index보다 작은 최소값이 있다면, 최소값과 index를 교환한다.

알고리즘 복잡도

N개의 데이터 -> N개의 공간

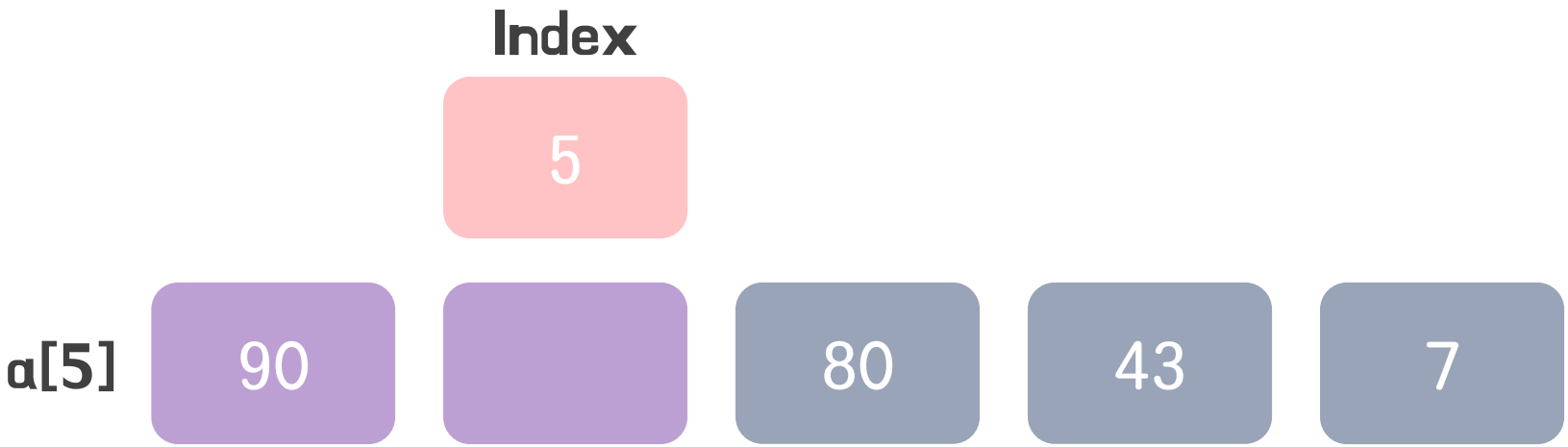
4 삽입정렬

오름차순 정렬일때, index를 왼쪽 방향으로 움직여
a [index-1] 에 a [index] 보다 작은 수가 올때까지 데이터를 옮긴다



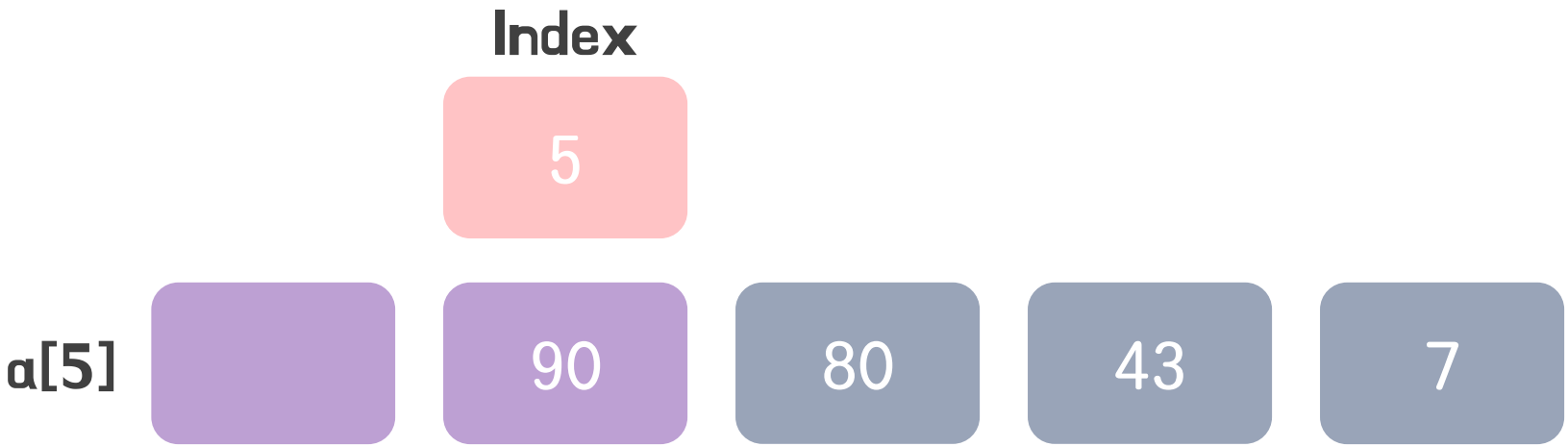
4 삽입정렬

오름차순 정렬일때, index를 왼쪽 방향으로 움직여
a [index-1] 에 a [index] 보다 작은 수가 올때까지 데이터를 옮긴다



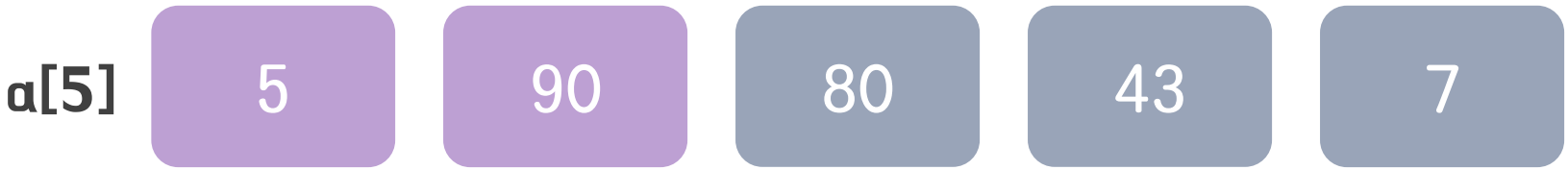
4 삽입정렬

오름차순 정렬일때, index를 왼쪽 방향으로 움직여
a [index-1] 에 a [index] 보다 작은 수가 올때까지 데이터를 옮긴다



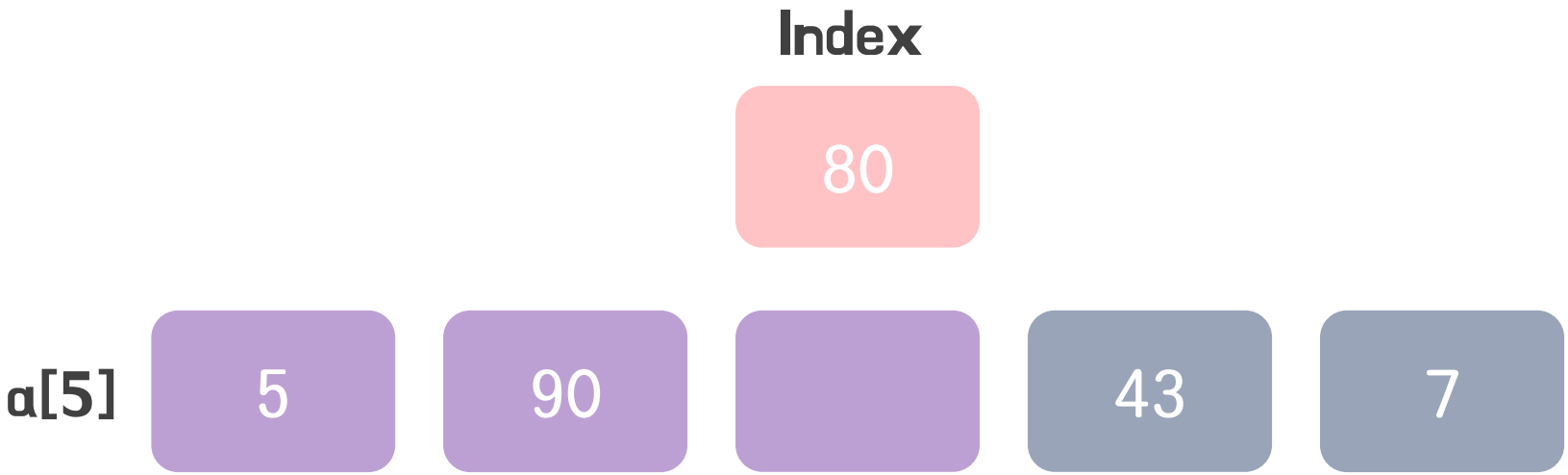
4 삽입정렬

오름차순 정렬일때, index를 왼쪽 방향으로 움직여
a [index-1] 에 a [index] 보다 작은 수가 올때까지 데이터를 옮긴다



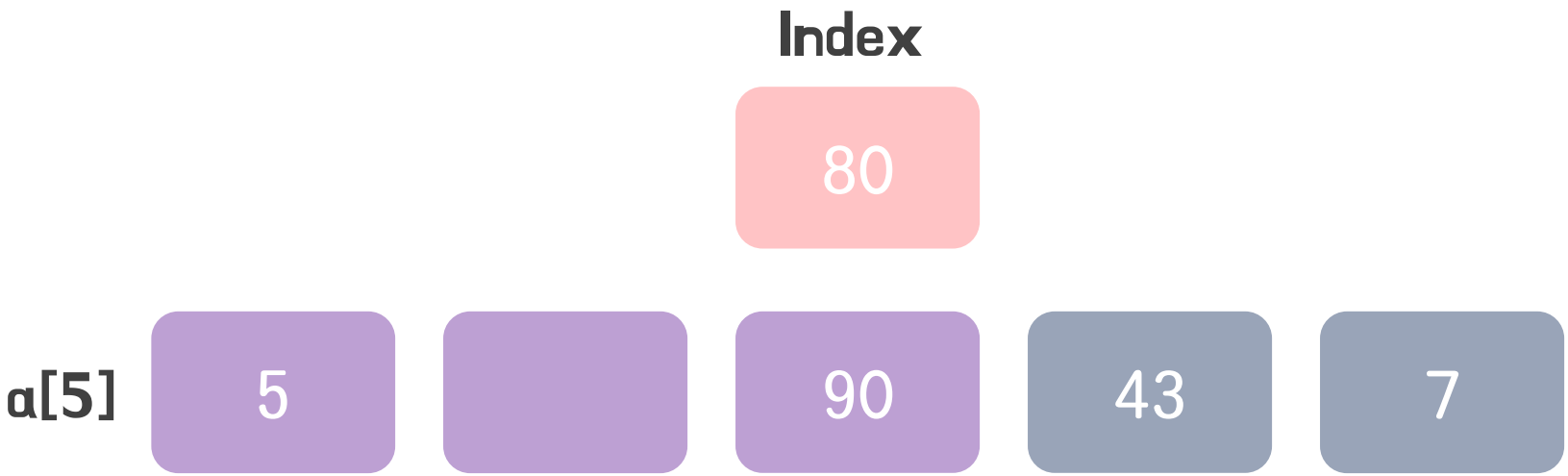
4 삽입정렬

오름차순 정렬일때, index를 왼쪽 방향으로 움직여
a [index-1] 에 a [index] 보다 작은 수가 올때까지 데이터를 옮긴다



4 삽입정렬

오름차순 정렬일때, index를 왼쪽 방향으로 움직여
a [index-1] 에 a [index] 보다 작은 수가 올때까지 데이터를 옮긴다

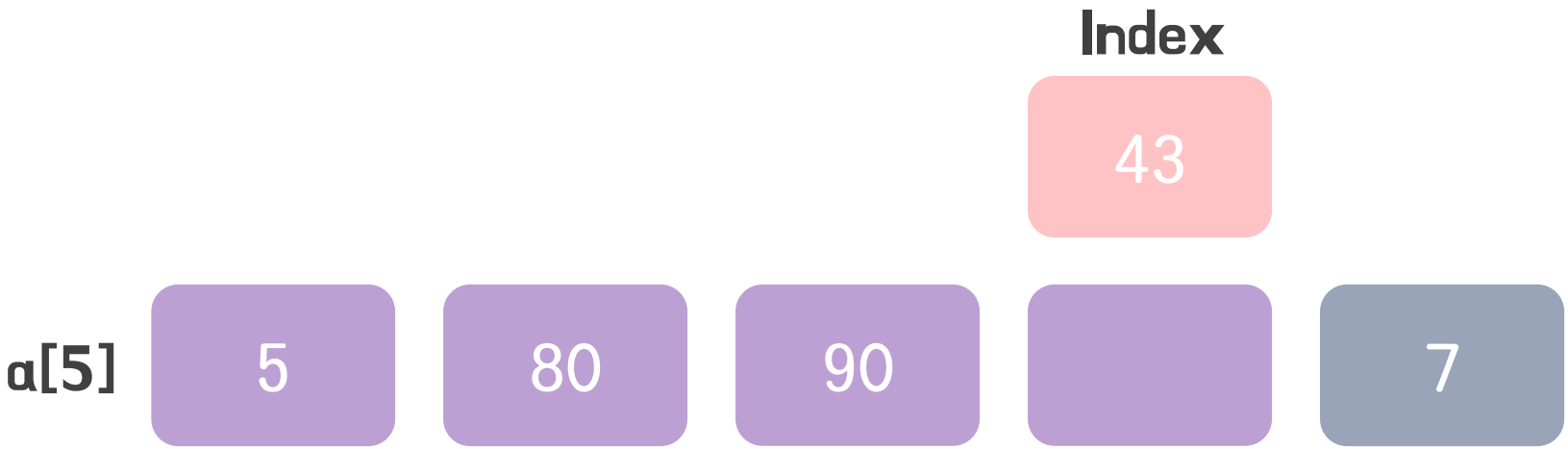


4 삽입정렬

오름차순 정렬일때, index를 왼쪽 방향으로 움직여
a [index-1] 에 a [index] 보다 작은 수가 올때까지 데이터를 옮긴다

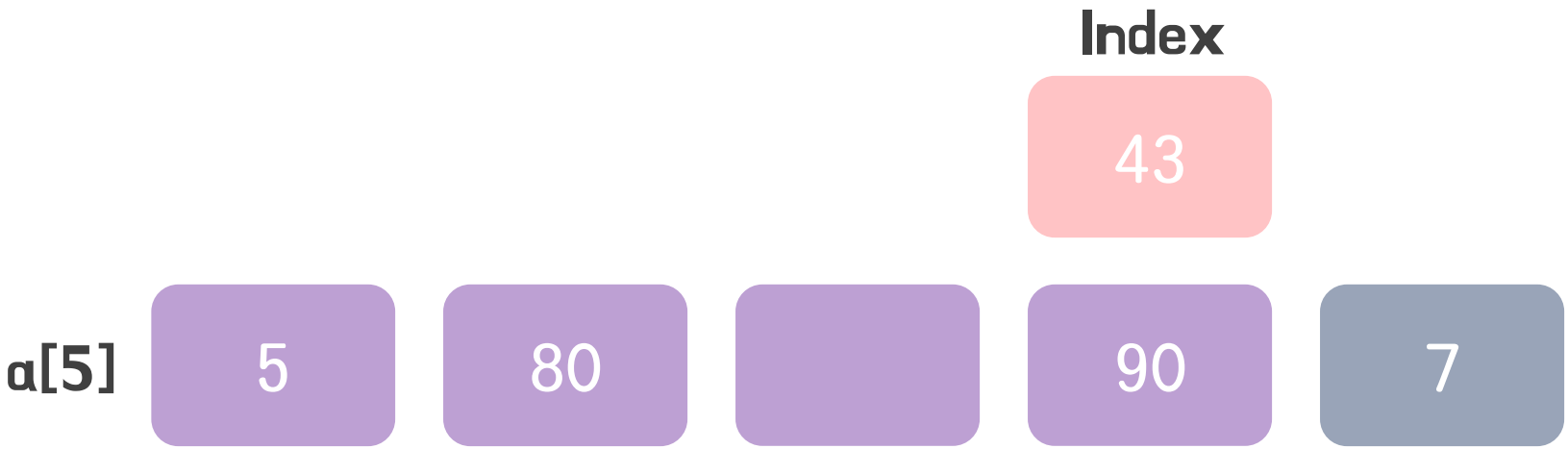
4 삽입정렬

오름차순 정렬일때, index를 왼쪽 방향으로 움직여
a [index-1] 에 a [index] 보다 작은 수가 올때까지 데이터를 옮긴다



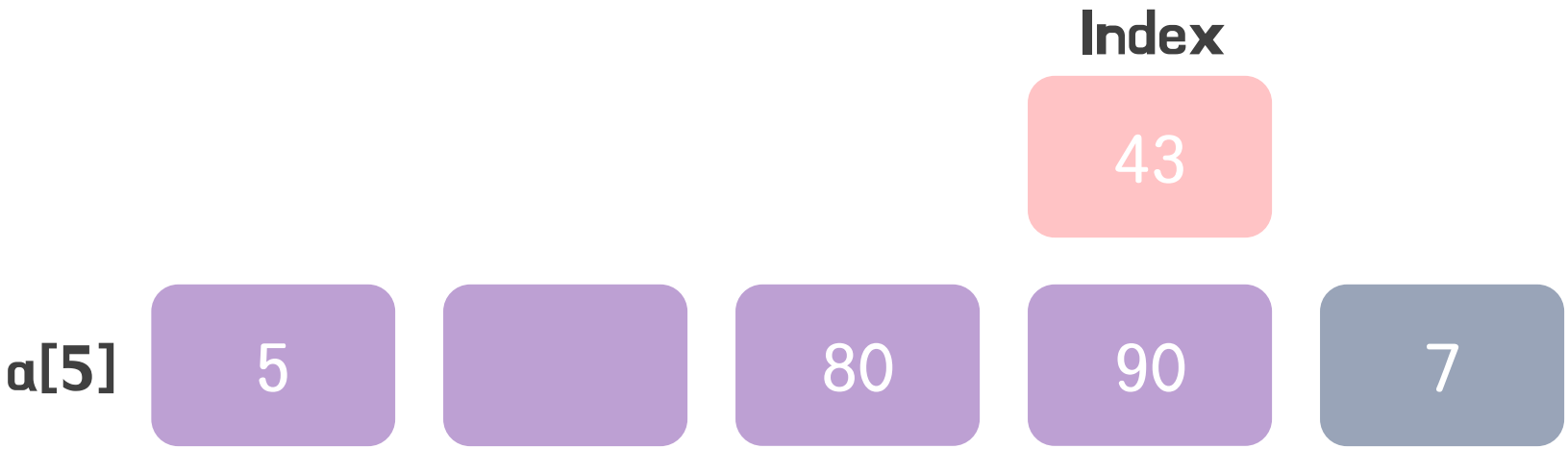
4 삽입정렬

오름차순 정렬일때, index를 왼쪽 방향으로 움직여
a [index-1] 에 a [index] 보다 작은 수가 올때까지 데이터를 옮긴다



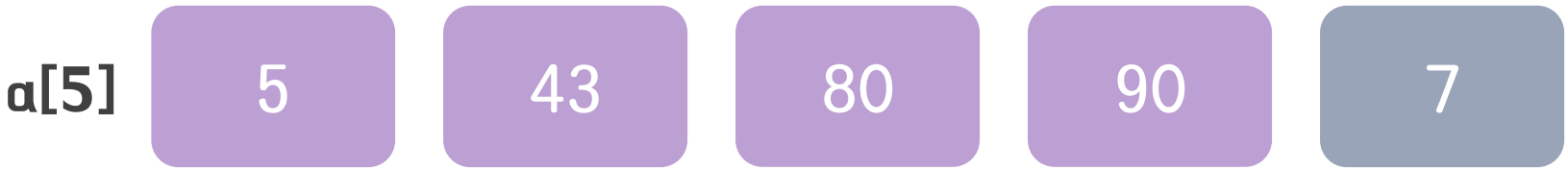
4 삽입정렬

오름차순 정렬일때, index를 왼쪽 방향으로 움직여
a [index-1] 에 a [index] 보다 작은 수가 올때까지 데이터를 옮긴다



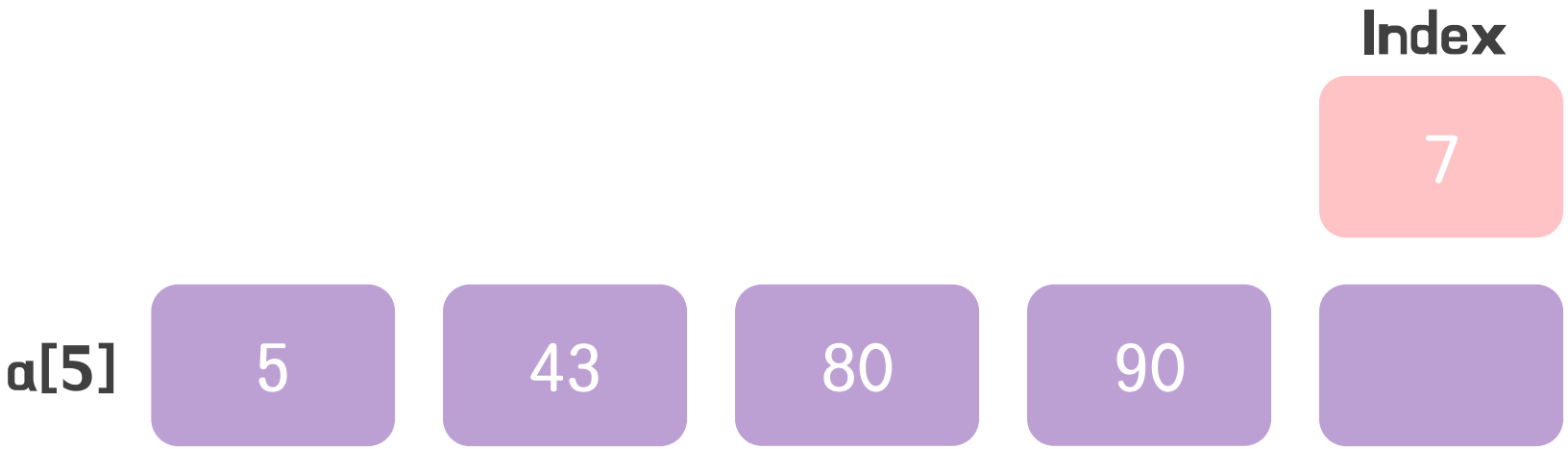
4 삽입정렬

오름차순 정렬일때, index를 왼쪽 방향으로 움직여
a [index-1] 에 a [index] 보다 작은 수가 올때까지 데이터를 옮긴다



4 삽입정렬

오름차순 정렬일때, index를 왼쪽 방향으로 움직여
a [index-1] 에 a [index] 보다 작은 수가 올때까지 데이터를 옮긴다

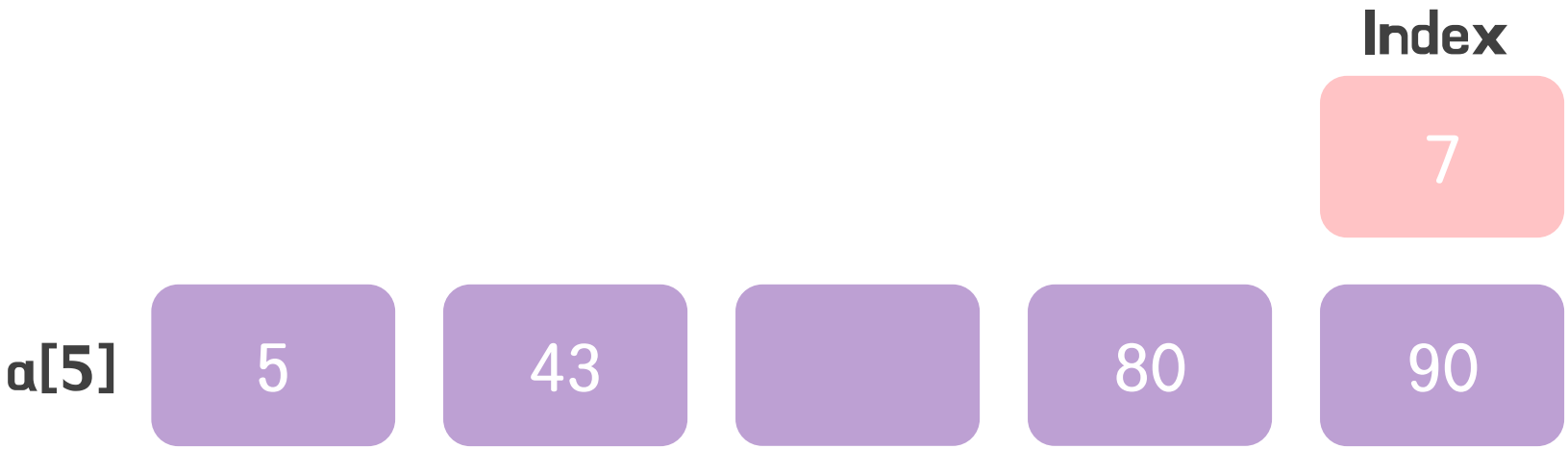


4 삽입정렬

오름차순 정렬일때, index를 왼쪽 방향으로 움직여
a [index-1] 에 a [index] 보다 작은 수가 올때까지 데이터를 옮긴다

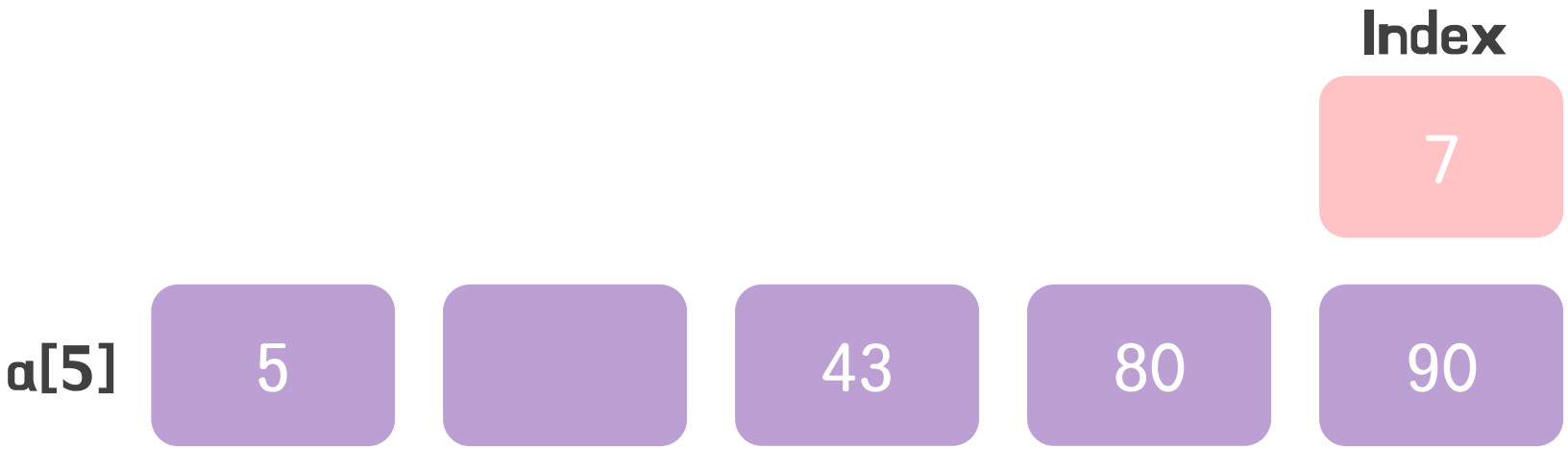
4 삽입정렬

오름차순 정렬일때, index를 왼쪽 방향으로 움직여
a [index-1] 에 a [index] 보다 작은 수가 올때까지 데이터를 옮긴다



4 삽입정렬

오름차순 정렬일때, index를 왼쪽 방향으로 움직여
a [index-1] 에 a [index] 보다 작은 수가 올때까지 데이터를 옮긴다



4 삽입정렬

오름차순 정렬일때, index를 왼쪽 방향으로 움직여
a [index-1] 에 a [index] 보다 작은 수가 올때까지 데이터를 옮긴다

4 삽입정렬

오름차순 정렬일때, index를 왼쪽 방향으로 움직여
a [index-1] 에 a [index] 보다 작은 수가 올때까지 데이터를 옮긴다

알고리즘 복잡도

시간복잡도

	최선의 경우	최악의 경우
비교횟수	N	N(N-1)/2
교환횟수	0	N(N-1)/2

평균 시간복잡도 $O(n^2)$

4 삽입정렬

오름차순 정렬일때, index를 왼쪽 방향으로 움직여
a [index-1] 에 a [index] 보다 작은 수가 올때까지 데이터를 옮긴다

알고리즘 복잡도

N개의 데이터 -> N+1개의 공간

5 그 외

Visual studio에는 algorithm헤더에 퀵소트를 이용한
정렬 라이브러리가 내장되어있다

```
#include <stdio.h>
#include <algorithm>
using namespace std;
#define max 5
int a[5];
Int main()
{
    int i;
    for(i=0;i<max;i++)
    {
        scanf("%d",&a[i]);
    }
    sort(&a[0],&a[max+1]);
    return 0;
}
```

```
template<class
RandomAccessIterator>
  void sort(
    RandomAccessIterator first,
    RandomAccessIterator last
  );
template<class
RandomAccessIterator, class
Predicate>
  void sort(
    RandomAccessIterator first,
    RandomAccessIterator last,
    Predicate comp
  );
```

5 그 외

질문 받아요