정렬Sorting

학습목표

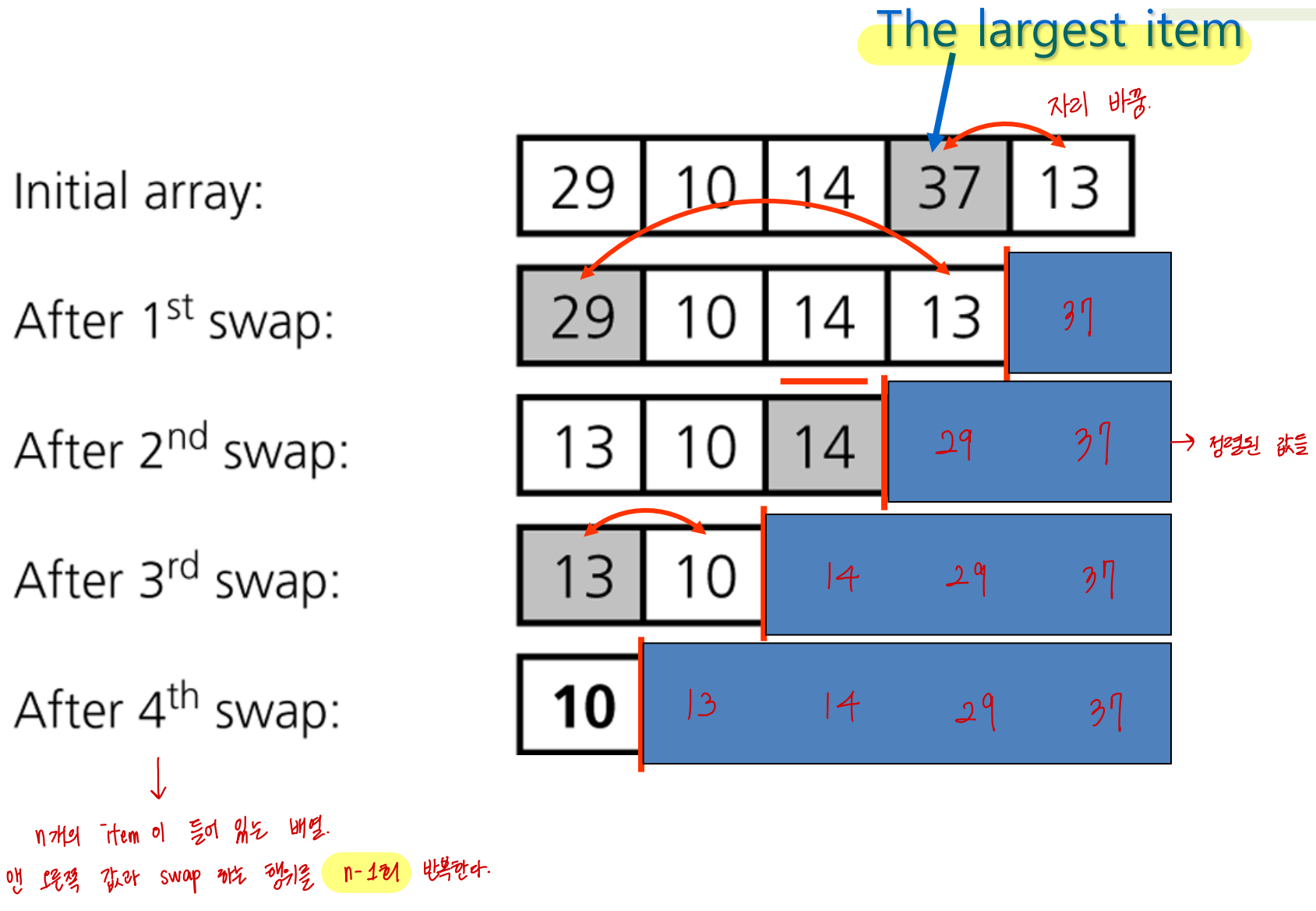
- **Computer Science에서 가장 많이 접하게 되는 문제 중의 하나인 Sorting (정렬)에 대해서 학습.**

  - **Selection Sort**
  - **Bubble Sort**
  - **Insertion Sort**
  - **Merge Sort**
  - **Quick Sort**
  - Heap Sort
  - Radix Sort
  - Counting Sort
  - …

→ loop를 이용해서 문제 해결

→ recursion을 이용해서 문제 해결

Selection Sort(선택 정렬)

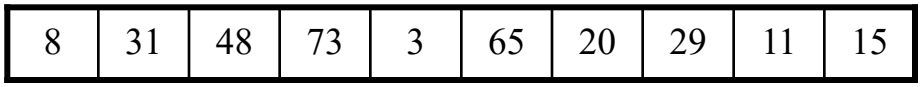

The largest item
자리 바꿈.
Initial array:
29 10 14 37 13
After 1st swap:
29 10 14 13 37
After 2nd swap:
13 10 14 29 37
→ 정렬된 값들
After 3rd swap:
13 10 14 29 37
After 4th swap:
10 13 14 29 37
↓
n개의 item 이 들어 있는 배열.
앤 오른쪽 값과 swap 하는 행위를 n-1회 반복한다.

Selection Sort

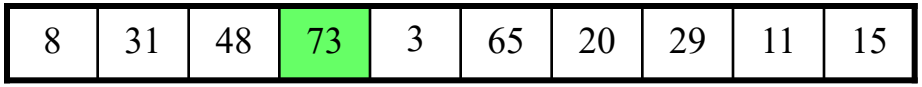
- 각 루프마다
  - 최대 원소를 찾는다
  - 최대 원소와 맨 오른쪽 원소를 교환한다
  - 맨 오른쪽 원소를 제외한다
- 하나의 원소만 남을 때까지 위의 루프를 반복

Selection Sort의 작동 예

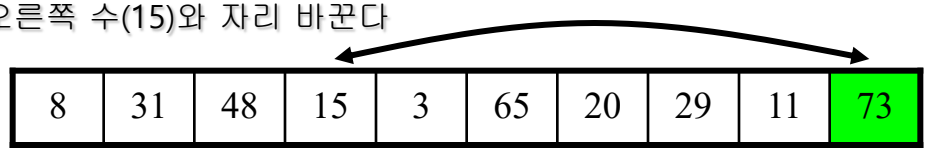
정렬할 배열이 주어짐



가장 큰 수를 찾는다 (73)

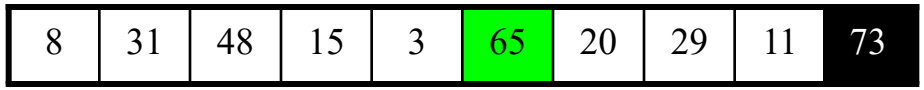


73을 맨 오른쪽 수(15)와 자리 바꾼다

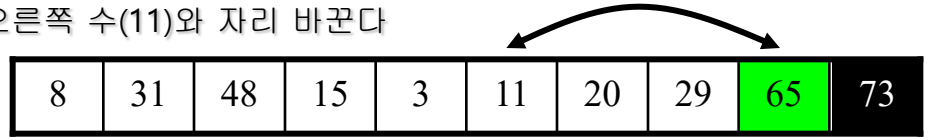


①의 첫번째 loop

맨 오른쪽 수를 제외한 나머지에서 가장 큰 수를 찾는다 (65)

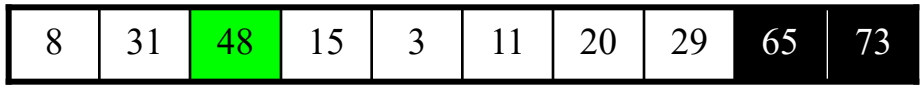


65를 맨 오른쪽 수(11)와 자리 바꾼다



①의 두번째 loop

맨 오른쪽 두 수를 제외한 나머지에서 가장 큰 수를 찾는다 (48)

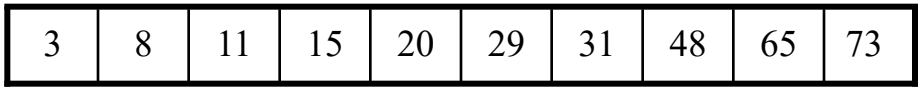


. . .

8을 맨 오른쪽 수(3)와 자리 바꾼다

→ 첫 번째 loop를 n-1회 반복

최종 배열



```
selectionSort(int* A, n)          ▷ 배열 A를 정렬한다
{
   for (int i = 0; i < n-1; i++ {          // n-1 반복
      A[0] ... A[n-1-i] 중 가장 큰 수 A[max_idx]를 찾는다;
      A[max_idx] ↔ A[n-1-i];  //두 배열 원소를 교환
   }
}
```

↱ selection

→ swap

i = 0 A[0] ... A[n-1] 에서 max 찾기
i = 1 A[0] ... A[n-2] 에서 max 찾기
⋮
i = n-2 A[0] ... A[1] 에서 max 찾기

→ 총 n-1 회를 반복하고,
마지막 2개를 조사해서 자리를 바꾸고 끝낸다.

<바깥 loop>
selection, swap을 n-1회 돌면서 진행.
→ 마지막 남은 1는 swap 할 필요가 없으므로.

<안쪽 loop>
selection 을 위해 max 값을 loop로 찾음.
sorted 된것 빼고 전체를 다 돌아야 max 값을 찾을수 있다.

```
selectionSort(int* A, n) // 배열 A를 정렬한다
{
    for (i = 0; i< n-1; i++ {     // n-1 반복
        A[0] ... A[n-1-i] 중 가장 큰 수 A[max_idx]를 찾는다;
        A[max_index] ↔ A[n-1-i]; //두 배열 원소를 교환
    }
}
```

배열에서 가장 큰 수 찾는 것도 loop

version1

첫 번째 값을 max로 할당하고, loop를 돌면서 max를 갱신한다.

```
max = A[0]; max_idx = 0;
for (j = 1; j < n - i; j++) {
        if( max < A[j] ) {   → 기존의 max보다 값이 크다면
                max = A[j];
                max_idx = j;   ← 갱신
        }
}
// loop가 끝나면 배열에서 가장 큰 수와 그 index를 알게 됨.
```

가장 큰 수를 찾는다 (73)

8	31	48	73	3	65	20	29	11	15

①의 첫번째 loop

맨 오른쪽 수를 제외한 나머지에서 가장 큰 수를 찾는다 (65)

8	31	48	15	3	65	20	29	11	73

①의 두번째 loop

맨 오른쪽 두 수를 제외한 나머지에서 가장 큰 수를 찾는다 (48)

8	31	48	15	3	11	20	29	65	73

...

version1

```
for (i = 0; i < n-1; i++) {
  max = A[0]; max_idx=0;
  for (j = 1; j < n-i; j++) {
    if(max < A[j] ) {
      max = A[j];
      max_idx = j;
    }
  }
}
// loop가 끝나면 배열에서 가장 큰 수와 그 index를 알게 됨.
```

n, n-1, n-2, n-3 ... 3, 2

i=0 j= n

i=1 j= n-1

i=2 j= n-2

⋮

i= n-2 j=2 → 마지막 2개만 비교

- 오름차순으로의 정렬을 위해 선택 정렬을 사용할 때
  - **큰 값을 찾아** 오른쪽으로 보내는 방법(우리가 조금 전 살펴본)과
  - **작은 값을 찾아** 왼쪽으로 보내는 방법도 가능하다. 다음 슬라이드는 두 번째 방법을 보여준다.

선택정렬(selection sort)

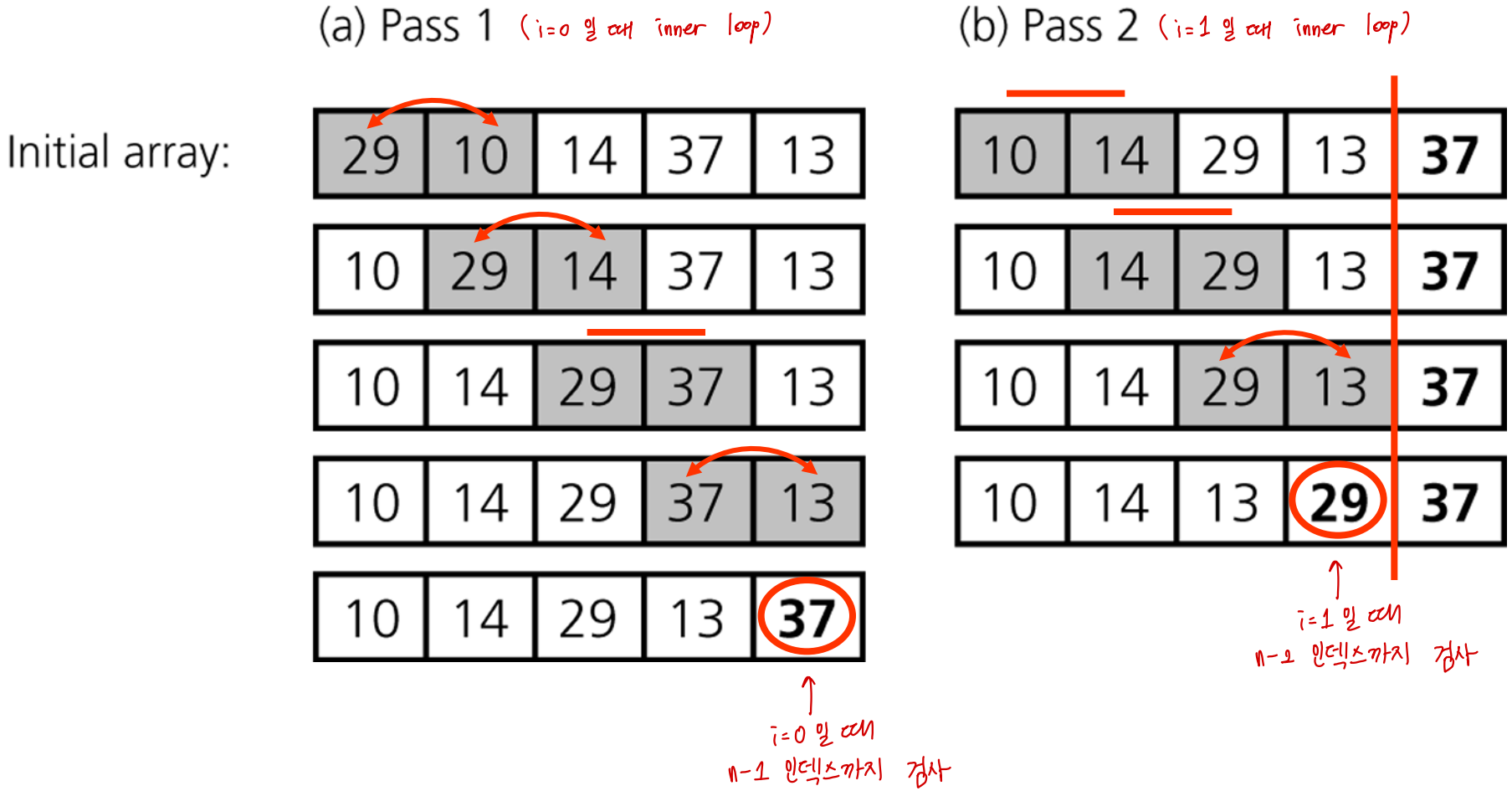
선택정렬 유사코드

```
selection_sort(A, n)

for i←0 to n-2 do
        least ← A[i], A[i+1],..., A[n-1] 중에서 가장 작은 값의 인덱스;
        A[i]와 A[least]의 교환;
        i++;
```

Bubble Sort(버블 정렬)

Bubble Sort → 단계마다 가장 큰 값을 오른쪽으로 이동시킴.



```
bubbleSort(int *A, n)      ▷ 배열 A를 정렬한다
{
   int i;
   for (i = 0; i < n-1; i++ {          // n-1 반복
      인접한 두 숫자의 SUB 배열의 끝까지 순서만 바꿈.
}
```

```
bubbleSort(int *A, n)    ▷ 배열 A를 정렬한다
{
    for (int i = 0; i < n-1; i++ {      // n-1 반복 (n개가 있으면 n-1회 반복)
        인접한 두 숫자의 SUB 배열의 끝까지 순서만 바꿈.
}
```

i=0 j = n-1
i=1 j = n-2
i=2 j = n-3
⋮
i=n-2 j=1

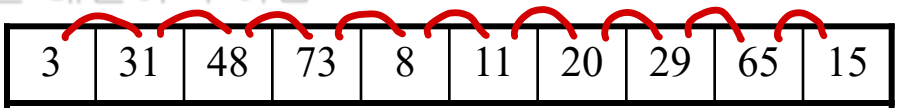
이것도 **loop**

```
int j;
for (j = 0; j < n-1-i; j++ {
  if( A[j] > A[j+1] ) {
     A[j]와 A[j+1]을 SWAP
  }
}
// loop가 끝나면 SUB 배열에서 가장 큰 수가 제일 뒤에
```

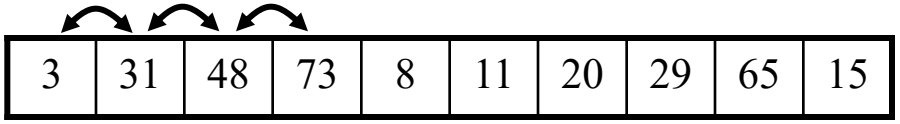
Bubble Sort의 작동 예

i=0 일 때 (첫 번째 검사 때)
for (j=0; j<9; j++)

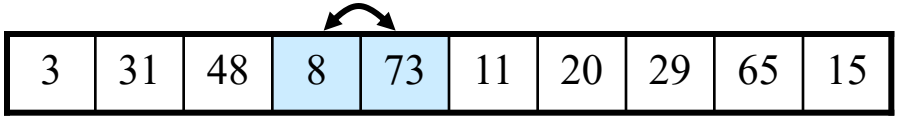
정렬할 배열이 주어짐

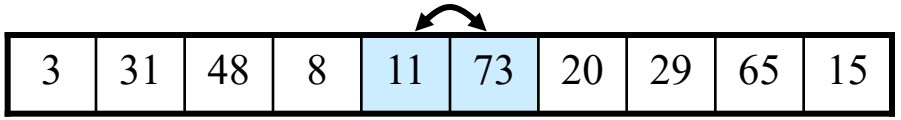


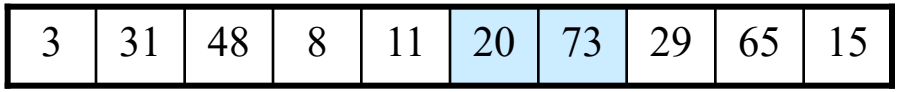
왼쪽부터 시작해 이웃한 쌍들을 비교해간다



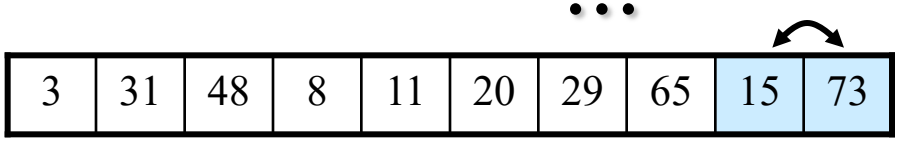
순서대로 되어 있지 않으면 자리 바꾼다



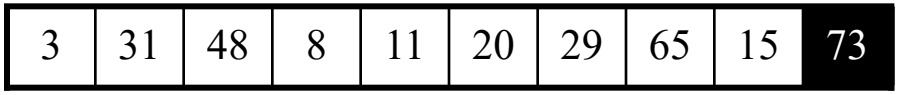




• • •



맨 오른쪽 수(73)를 대상에서 제외한다

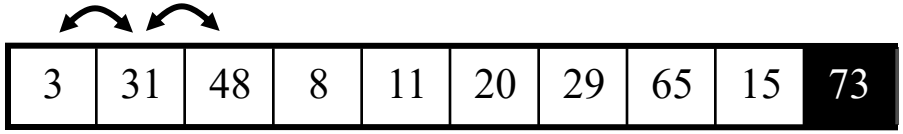


9번 검사하는
첫 번째 inner loop.

Pass 1

i=0
첫 번째 outer loop

왼쪽부터 시작해 이웃한 쌍들을 비교해간다



순서대로 되어 있지 않은 경우에는 자리 바꾼다

맨 오른쪽 수(65)를 대상에서 제외한다

8 번 검사하는
두 번째 inner loop

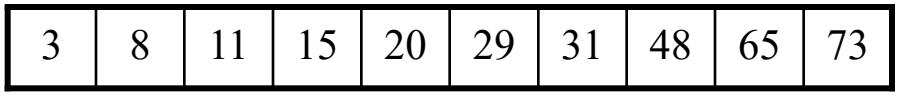
Pass 2

i = 1
두 번째 outer loop

앞의 작업을 반복하면서 계속 제외해 나간다

• • •

두개짜리 배열의 처리를 끝으로 정렬이 완료된다

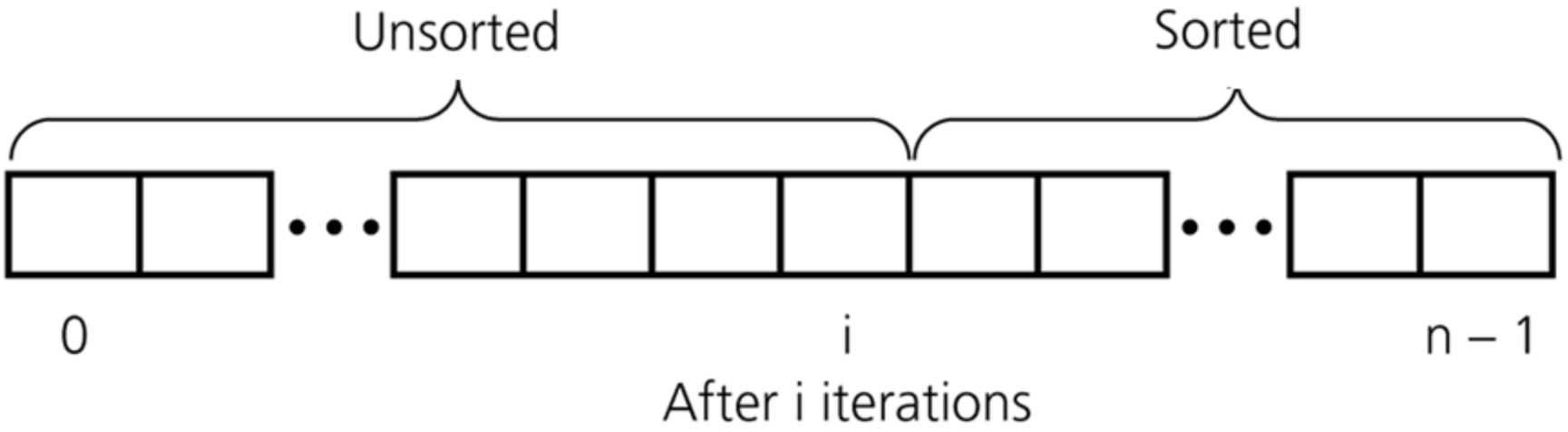


.

.

.

Pass n-1


Unsorted
Sorted
0
i
n – 1
After i iterations