이산수학

휴먼지능정보공학전공

알고리즘

- 알고리즘 기본 개념 및 특성
- 알고리즘의 종류

기본개념

- 알고리즘(Algorithm)
  - 문제를 해결하기 위한 체계적 단계
  - 순서도, 의사코드, 일반적인 언어로 작성

기본개념

- 알고리즘
  - 주어진 문제를 해결하기 위한 일련의 절차
    - 알고리즘 중 가장 효율적인 알고리즘을 찾는 것이 중요
    - 수학에서는 문제를 풀기 위해 정의나 정리들을 활용하는 데 비해 컴퓨터에서는 수행 가능한 효율적인 알고리즘을 사용
    - 순서도, 유사 코드 등 여러 가지 방법으로 표현
    - 누구나 이해할 수 있도록 명확하게 기술하는 것이 중요

기본개념

- 알고리즘 특성
  - (1) 입력(input) : 문제와 관련된 입력이 반드시 존재
  - (2) 출력(output) : 입력을 처리한 출력(결과)이 반드시 존재
  - (3) 정확성(correctness) : 입력을 이용한 문제 해결 과정과 출력은 논리적이고 정확
  - (4) 유한성(finiteness) : 입력은 제한된 개수의 명령 단계를 거쳐 출력을 내고 반드시 종료
  - (5) 효율성(effectiveness) : 문제 해결 과정이 효율적
  - (6) 일반성(generality) : 같은 유형의 문제에 대해 항상 적용 가능
  - (7) 확정성(definiteness) : 같은 입력에 대해 출력이 항상 확정적

기본개념

- 알고리즘 표현
  - 순서도(Flow Chart): 명령의 종류와 기능에 따라 도표를 만들고 명령들의 순서대로 도표를 나열해 표현한 방식
  - 의사코드(Pseudo Code): 일반적인 언어와 프로그램 코드를 적절히 이용해 명령들을 나열한 방식

기본개념

- 알고리즘 구현(파이썬)

```
1  algorithm maxnum(a, b, c){
2      x = a;
3      if b > x then
4          x = b;
5      else if c > x then
6          x = c;
7      endif
8      print x;
9   }
```

```
1  algorithm sum1to10(){
2      sum=0;
3      i=1;
4      while(i<=10)
5          sum = sum + i;
6          i=i+1;
7      endwhile
8      print sum;
9  }
```

기본개념

- 알고리즘 구현(파이썬)

```
1   algorithm A( ){
2       i = 0;
3       while(i<=20)
4           x = 1;
5           x = x + 1;
6           if(i < 20) then
7               x = 10;
8           else
9               x = 0;
10          endif
11          i = i+2;
12      endwhile
13      print i, x;
14  }
```

```
1   algorithm B( ){
2       i = 1;
3       x = 0;
4       while(i<=n)
5           x=x+1;
6           i=i+1;
7       endwhile
8       printf i, x;
9   }
```

기본개념

- 알고리즘 구현(파이썬)

```
1   algorithm A( ){
2       i = 0;
3       while(i<=20)
4           x = 1;
5           x = x + 1;
6           if(i < 20) then
7               x = 10;
8           else
9               x = 0;
10          endif
11          i = i+2;
12      endwhile
13      print i, x;
14  }
```

```
1   algorithm B( ){
2       i = 1;
3       x = 0;
4       while(i<=n)
5           x=x+1;
6           i=i+1;
7       endwhile
8       printf i, x;
9   }
```

알고리즘

- 복잡도(Complexity)
  - 알고리즘 수행 시 필요한 시간 또는 공간 비용
    - 시간복잡도(Time Complextiy) : 프로그램이 수행되는 시간
    - 공간복잡도(Space Complexity) : 프로그램이 차지하는 기억 공간

알고리즘 종류

- 유클리드 알고리즘
  - 최대공약수(GCD, Greatest Common Diviser)를 구하는 알고리즘
  - 최대공약수 GCD(a,b)
    - 양의 정수 a, b가 가지는 공약수 중 최댓값
    - 양의 정수 a, b가 어떤 양의 정수 k와 각각 k|a, k|b의 관계가 있을 때, k는 a, b의 공약수, 이 중 가장 큰 값이 최대 공약수

알고리즘 종류

- 유클리드 알고리즘

```
1   algorithm gcd(a, b){
2       if a < b then
3           tmp = a;
4           a = b;
5           b = tmp;
6       endif
7       while b ≠ 0
8           r = a mod b;
9           a = b;
10          b = r;
11      endwhile
12      return a;
13  }
```

알고리즘 종류

- 재귀 알고리즘(Recursive Algorithm)
  - 문제를 해결하면서 어떤 항목이 이전에 사용되었던 자기 자신을 다시 호출해 사용하는 방법

```
1  algorithm sum(int n){
2      s = 0;
3      for i = 0 to n
4          s = s + 1;
5      next i
6      return s;
7  }
```

```
1  algorithm sum(int n){
2      if n != 0 then
3          return (1+sum(n-1));
4      else
5          return 1;
6      endif
7  }
```

알고리즘 종류

- 재귀 알고리즘(Recursive Algorithm)
  - 문제를 해결하면서 어떤 항목이 이전에 사용되었던 자기 자신을 다시 호출해 사용하는 방법

```
1  algorithm fibonacci(int n){
2      if n = 1 then
3          return 1;
4      else if n = 2 then
5          return 1;
6      else
7          return fibonacci(n-1)+fibonacci(n-2);
8      endif
9  }
```

알고리즘 종류

- 탐색 알고리즘($\boldsymbol{Search\ Algorithm}$)
  주어진 원소의 집합에서 특정 원소를 찾는 작업을 체계적으로 명시해놓은 것
  (1) 순차 탐색(Sequential Search) 또는 선형 탐색(Linear Search) 알고리즘
  원소 집합의 처음부터 하나씩 비교하며 탐색하는 알고리즘
  (2) 이진 탐색(Binary Search) 알고리즘
  원소 집합을 반으로 나누어 키(key)를 정하고 키와 특정 원소를 비교하여 특정 원소가 속하는 영역에 대해서 탐색을 반복하는 방법으로 탐색 범위를 좁혀가는 알고리즘
  · 키를 정하는 규칙 : $\left\lfloor \frac{i+n}{2} \right\rfloor$
  · i : 시작 인덱스 또는 키 인덱스
  · n : 원소의 개수

알고리즘 종류

문제

▪ 다음 원소 집합에서 10의 인덱스를 찾으려고 할 때, 적당한 탐색 방법은 무엇인지 알아보자.

Index	0	1	2	3	4	5	6	7	8	9
value	1	7	3	8	3	10	6	2	9	4

- (1) 원소 집합의 원소들이 정렬되어 있지 않은 상태다. 이런 경우는 순차 탐색을
- 이용한다.
- ① 인덱스 0번의 1과 10을 비교하면 1 ≠ 10이므로 다음 인덱스 탐색
- ② 인덱스 1번의 7과 10을 비교하면 7 ≠ 10 이므로 다음 인덱스 탐색
- ③ 인덱스 2번의 3과 10을 비교하면 3 ≠ 10 이므로 다음 인덱스 탐색
- ④ 인덱스 3번의 8과 10을 비교하면 8 ≠ 10 이므로 다음 인덱스 탐색
- ⑤ 인덱스 4번의 3와 10을 비교하면 3 ≠ 10 이므로 다음 인덱스 탐색
- ⑥ 인덱스 5번의 10와 10을 비교하면 10 = 10 이므로 탐색을 끝낸다.
- ∴ 10의 인덱스는 5다.

알고리즘 종류

문제

- 다음 원소 집합에서 10의 인덱스를 찾으려고 할 때, 적당한 탐색 방법은 무엇인지 알아보자.

Index	0	1	2	3	4	5	6	7	8	9
value	1	2	3	3	4	6	7	8	9	10

2) 원소 집합의 원소들이 정렬되어 있으므로 이진 탐색을 이용해 찾는다.

① 0부터 9까지 10개의 원소들이 있으므로 10개의 원소 중 가운데 원소를 찾는다.

$\therefore \left\lfloor \frac{0+10}{2} \right\rfloor = 5$

② 인덱스 5의 6은 10보다 작으므로 남아 있는 다른 원소들과 다시 탐색

$\therefore \left\lfloor \frac{6+10}{2} \right\rfloor = 8$

□□

③ 인덱스 8의 9는 10보다 작으므로 남아 있는 다른 원소들과 다시 검색

$\therefore \left\lfloor \frac{8+10}{2} \right\rfloor = 9$

□□

④ 인덱스 9의 10은 10과 같으므로 탐색을 끝낸다.

∴ 10의 인덱스는 9다.

알고리즘 종류

- 탐색 알고리즘(***Search Algorithm***)
  (1) 순차 탐색(Sequential Search) 또는 선형 탐색(Linear Search) 알고리즘
  원소 집합의 처음부터 하나씩 비교하며 탐색하는 알고리즘

```
1   algorithm seq_search(int arr[], int key, int n){
2       int i;
3       for i = 0 to n
4           if arr[i] = key then
5               return i;
6           else
7               return -1;
8           endif
9       next i
10  }
```

알고리즘 종류

- 탐색 알고리즘(***Search Algorithm***)
  (2) 이진 탐색(Binary Search) 알고리즘

원소 집합을 반으로 나누어
키(key)를 정하고
키와 특정 원소를 비교하여
특정 원소가 속하는 영역에
대해서 탐색을 반복하는 방법으로
탐색 범위를 좁혀가는 알고리즘

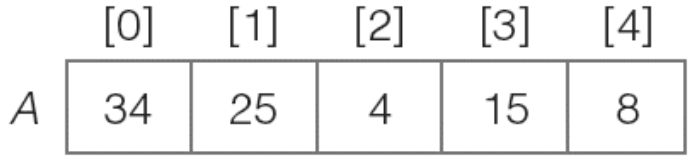
```
1   algorithm binary_search(int arr[], int key, int n){
2       int left = 0;
3       int right = n;
4       int mid;
5       while (left <= right)
6           mid = (left + right) / 2;
7           if arr[mid] < key then
8               left = mid + 1;
9           else if arr[mid] > key then
10              right = mid - 1;
11          else
12              return mid;
13          end if
14      endwhile
15      return -1;
16  }
```

알고리즘 종류

- 정렬 알고리즘(***Sort Algorithm***)
  - 원소 집합 내의 원소들이 임의로 나열되어 있을 때, 이 원소들을 일정 기준에 따라 다시 나열하는 방식"
  - (1) 버블 정렬(Bubble Sort)
    인접하는 두 개의 원소를 비교해 기준에 따라 순서를 바꾸는 방식
  - (2) 선택 정렬(Selection Sort)
    배열에서 가장 큰 값을 찾고 그 값을 A[n-1](배열의 마지막)의 값과 서로 교환
  - (3) 삽입 정렬(Insertion Sort)
    원소 집합 중에 가장 첫 번째 값을 정렬된 원소로 하여 그 다음 원소부터 정렬된 원소를 기준으로 적절한 위치에 삽입하는 방식
  - (4) 퀵 정렬(Quick Sort)
    피벗(pivot) 값을 기준으로 피벗보다 큰 집합과 작은 집합으로 나누어 각 집합을 정렬하는 방식"

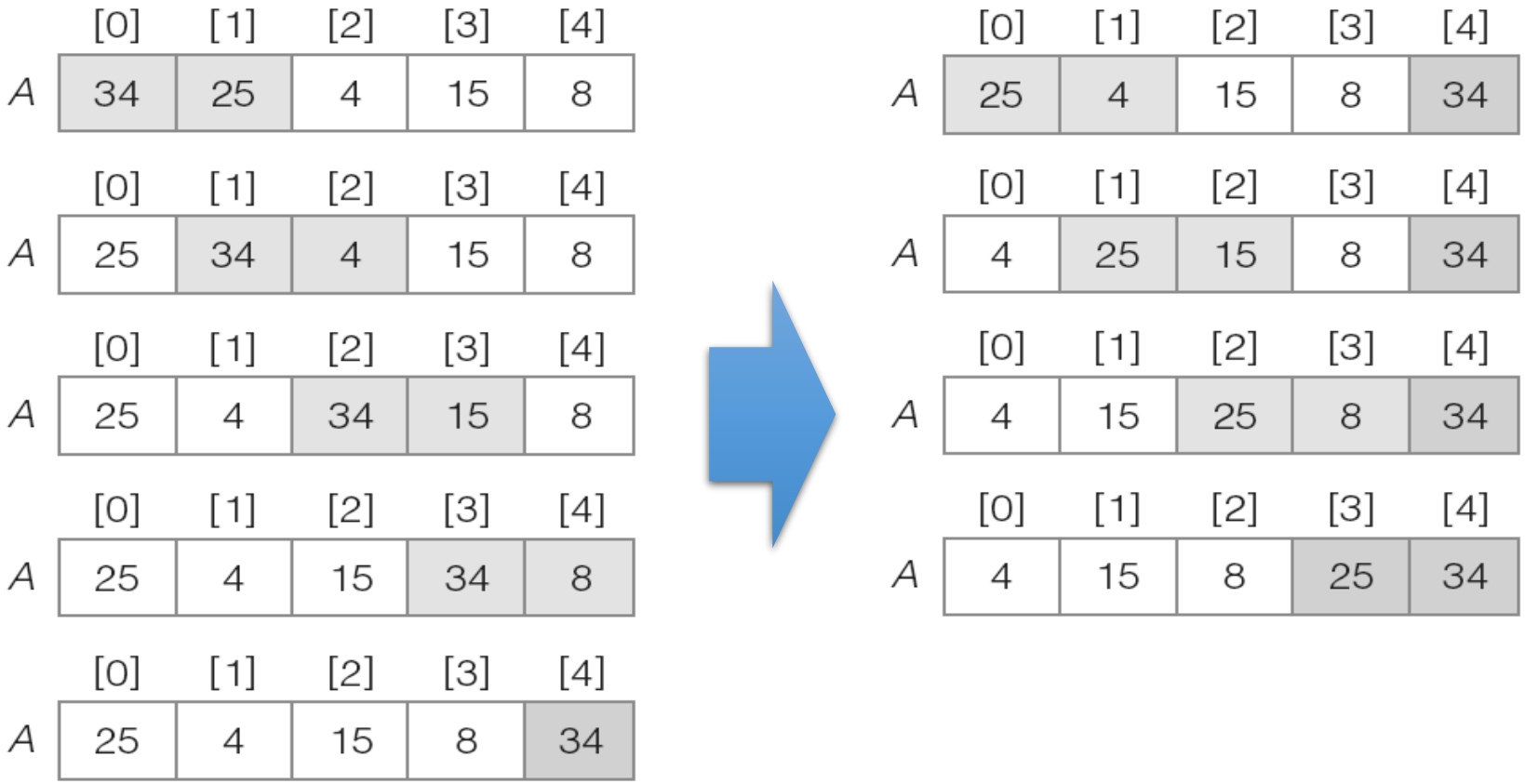
알고리즘 종류

- 버블 정렬
  - 인접한 두 데이터 A[i]와 A[i+1]의 값들을 비교
  - A[i+1]의 값이 A[i]의 값보다 작으면 두 데이터를 교환
  - 큰 데이터가 배열의



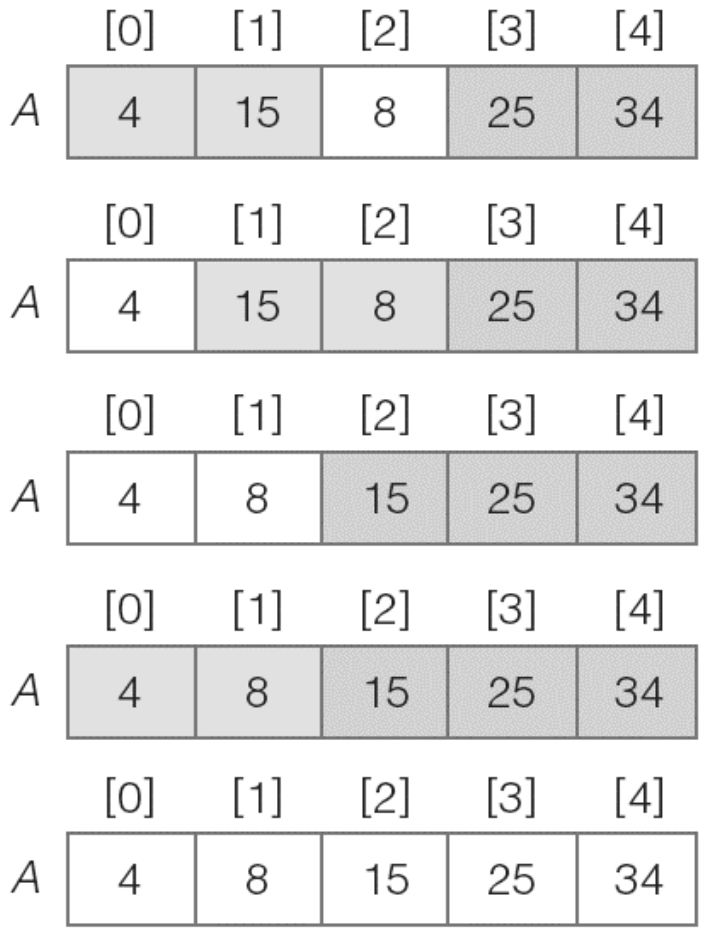
알고리즘 종류

- 버블 정렬


[0] [1] [2] [3] [4]
A 34 25 4 15 8
[0] [1] [2] [3] [4]
A 25 34 4 15 8
[0] [1] [2] [3] [4]
A 25 4 34 15 8
[0] [1] [2] [3] [4]
A 25 4 15 34 8
[0] [1] [2] [3] [4]
A 25 4 15 8 34
[0] [1] [2] [3] [4]
A 25 4 15 8 34
[0] [1] [2] [3] [4]
A 4 25 15 8 34
[0] [1] [2] [3] [4]
A 4 15 25 8 34
[0] [1] [2] [3] [4]
A 4 15 8 25 34

알고리즘 종류

- 버블 정렬


[0] [1] [2] [3] [4]
A 4 15 8 25 34
[0] [1] [2] [3] [4]
A 4 15 8 25 34
[0] [1] [2] [3] [4]
A 4 8 15 25 34
[0] [1] [2] [3] [4]
A 4 8 15 25 34
[0] [1] [2] [3] [4]
A 4 8 15 25 34

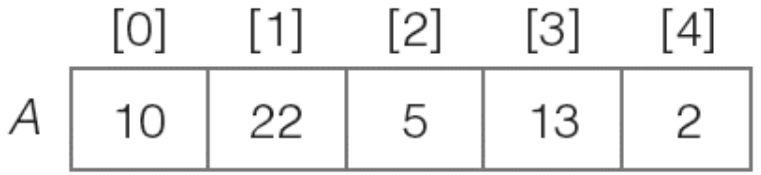
알고리즘 종류

- 버블 정렬

```
1   algorithm bubble(int arr[], int n){
2       int i, j;
3       int tmp;
4       for i = 1 to n-1
5           for j = 1 to n-i
6               if arr[j-1] > arr[j] then
7                   tmp = arr[j-1];
8                   arr[j-1] = arr[j];
9                   arr[j] = tmp;
10              endif
11          next j
12      next i
13  }
```

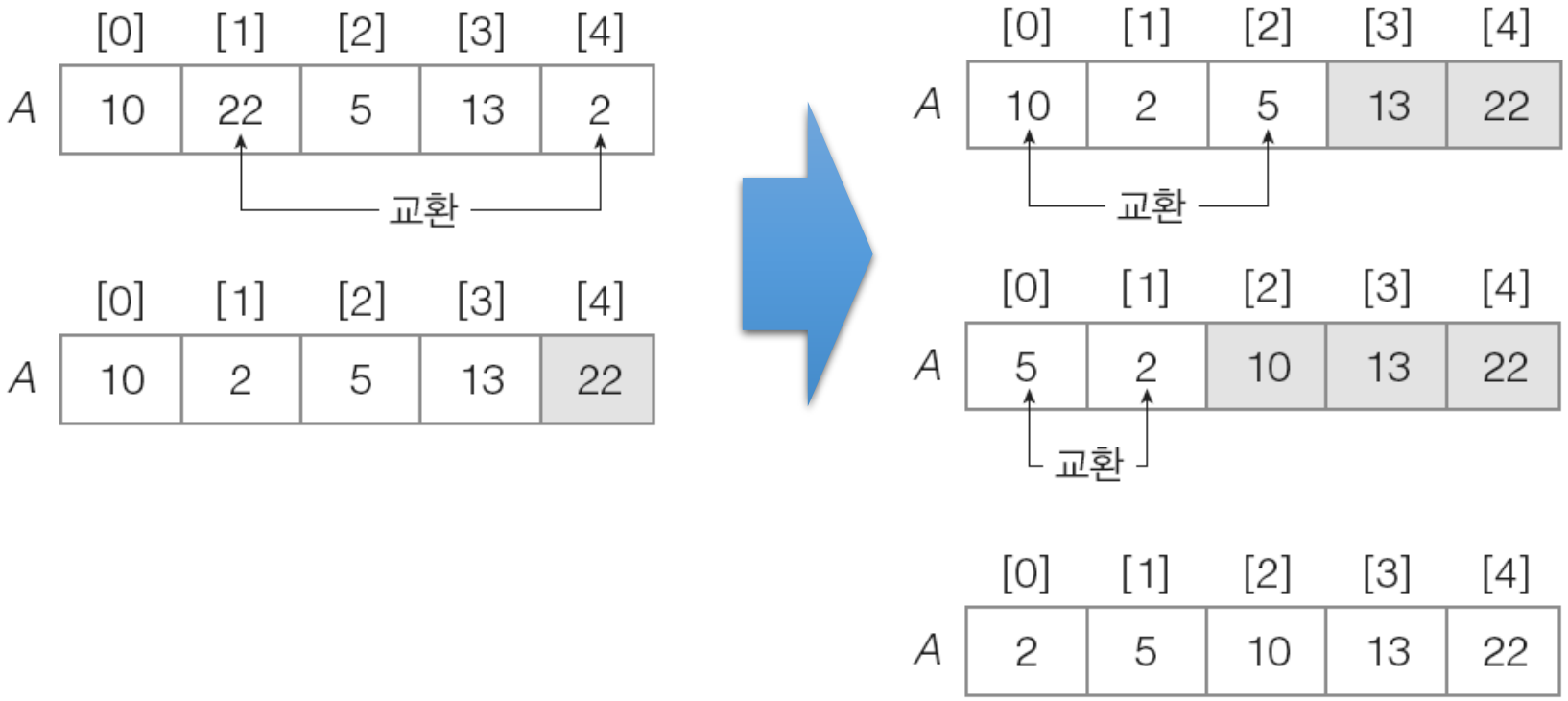
알고리즘 종류

- 선택 정렬
  - 배열에서 가장 큰 값을 찾음
  - 그 값을 A[n-1](배열의 마지막)의 값과 서로 교환
  - A[n-1]을 제외한 나머지 값들 중에서 가장 큰 값을 찾음
  - 그 값을 A[n-2]의 값과 서로 교환
  - 같은 과정을 정렬이 [illegible]



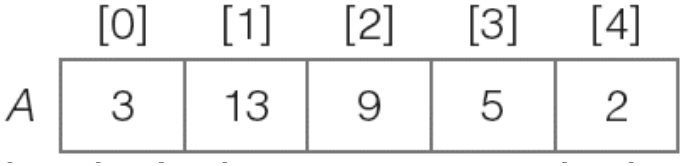
알고리즘 종류

- 선택 정렬


[0] [1] [2] [3] [4]
A 10 22 5 13 2
교환
[0] [1] [2] [3] [4]
A 10 2 5 13 22
[0] [1] [2] [3] [4]
A 10 2 5 13 22
교환
[0] [1] [2] [3] [4]
A 5 2 10 13 22
교환
[0] [1] [2] [3] [4]
A 2 5 10 13 22

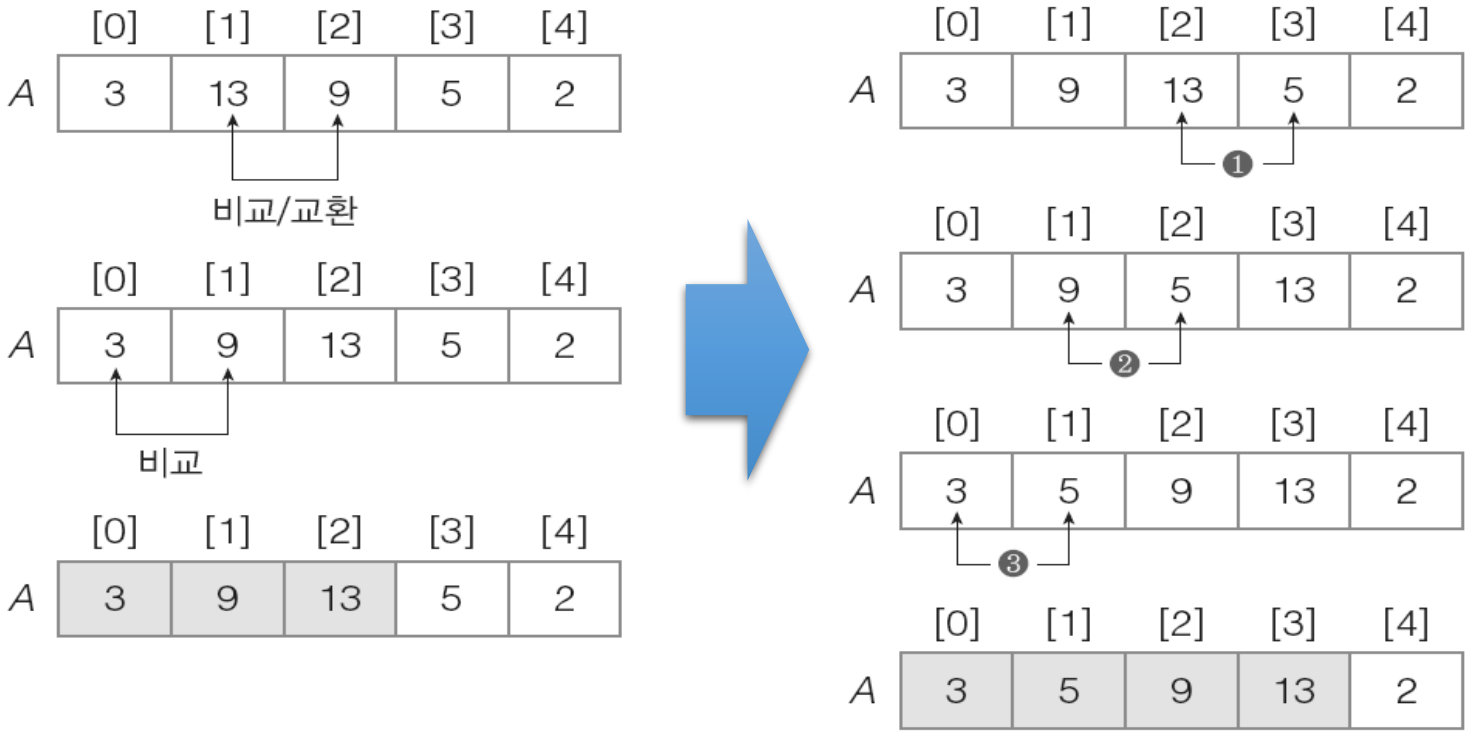
알고리즘 종류

- 삽입 정렬
  - 처음 A[0]은 정렬된 데이터로 취급
  - 다음 데이터 A[1]은 정렬된 데이터 A[0]와 비교하여 적절한 위치에 삽입
  - 다음 데이터 A[2]는 정렬된 데이터 A[0], A[1]과 비교하여 적절한 위치에 삽입
  - 같은 방식으로 나머지 정렬



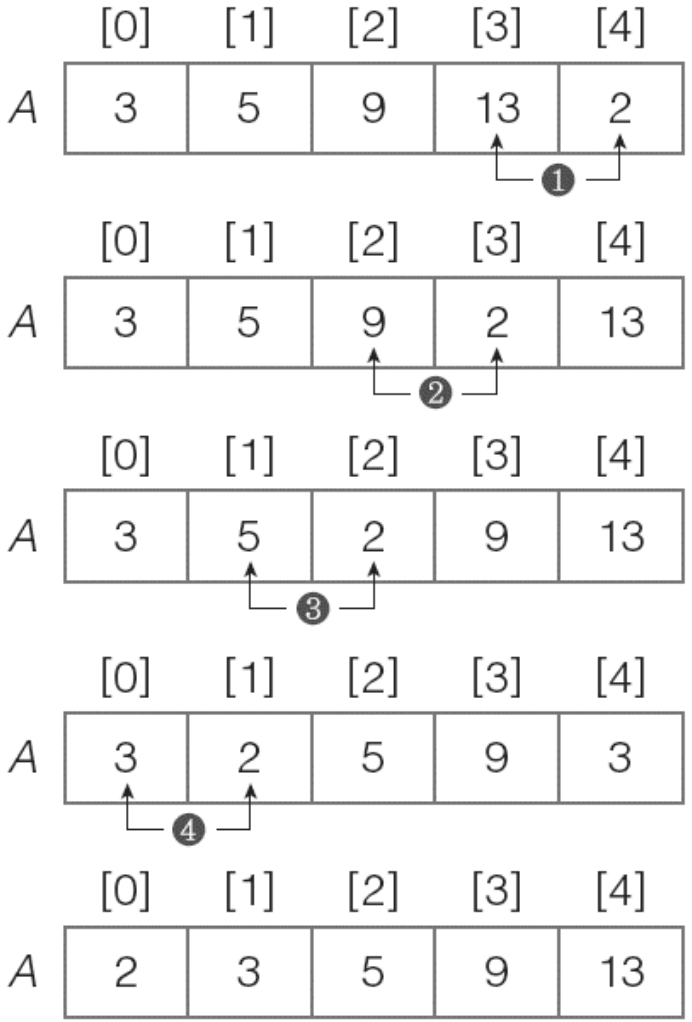
알고리즘 종류

- 삽입 정렬


[0] [1] [2] [3] [4]
A 3 13 9 5 2
비교/교환
[0] [1] [2] [3] [4]
A 3 9 13 5 2
비교
[0] [1] [2] [3] [4]
A 3 9 13 5 2
[0] [1] [2] [3] [4]
A 3 9 13 5 2
1
[0] [1] [2] [3] [4]
A 3 9 5 13 2
2
[0] [1] [2] [3] [4]
A 3 5 9 13 2
3
[0] [1] [2] [3] [4]
A 3 5 9 13 2

알고리즘 종류

- 삽입 정렬


[0] [1] [2] [3] [4]
A 3 5 9 13 2
1
[0] [1] [2] [3] [4]
A 3 5 9 2 13
2
[0] [1] [2] [3] [4]
A 3 5 2 9 13
3
[0] [1] [2] [3] [4]
A 3 2 5 9 3
4
[0] [1] [2] [3] [4]
A 2 3 5 9 13

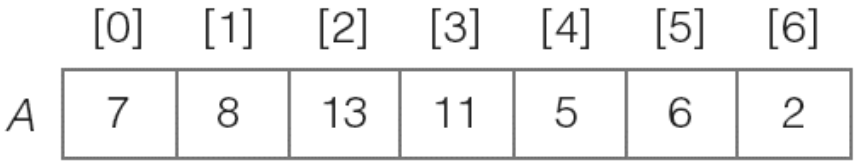
알고리즘 종류

- 삽입 정렬

```
1   algorithm insertion(int arr[], int n){
2       int i, j;
3       int t, tmp;
4       for i = 1 to n-1
5           tmp = A[i];
6           j = i - 1;
7           t = i;
8           while (j >= 0)
9               if A[j] > tmp then
10                  A[j+1]=A[j];
11                  t = j;
12              endif
13              j = j - 1;
14          endwhile
15          A[t] = tmp;
16      next i
17  }
```

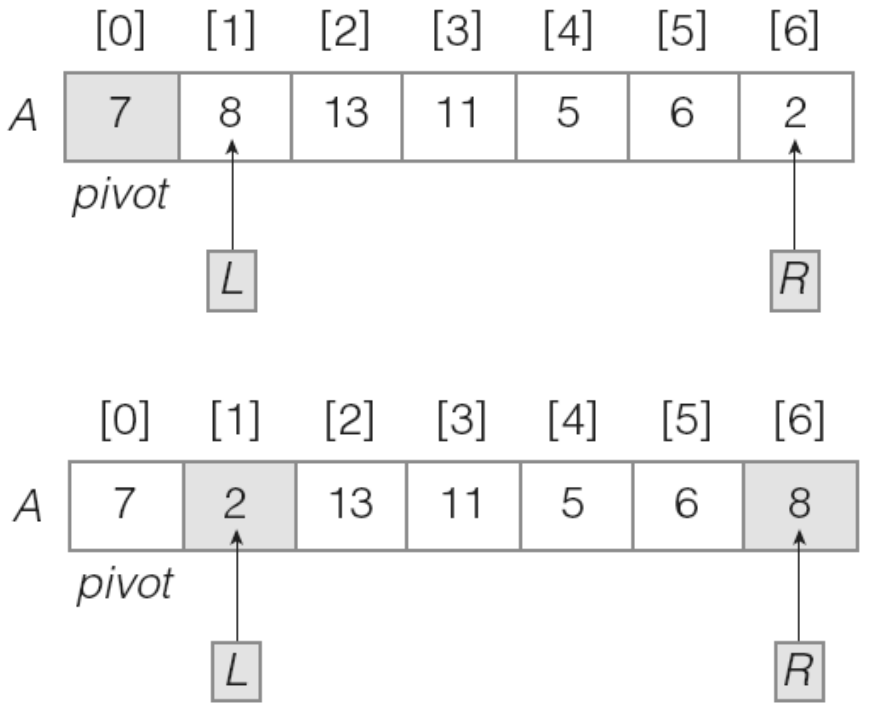
알고리즘 종류

- 퀵 정렬
  - 피벗(pivot)과 두 개의 포인터 지정
  - 두 개의 포인터를 이용하여 피벗보다 큰 데이터와 작은 데이터를 찾아 두 개의 포인터 값을 서로 교환
  - 피벗을 기준으로 작은 데이터들의 집합과 큰 데이터들의 집합으로 정렬
  - 같은 과정을 두 집



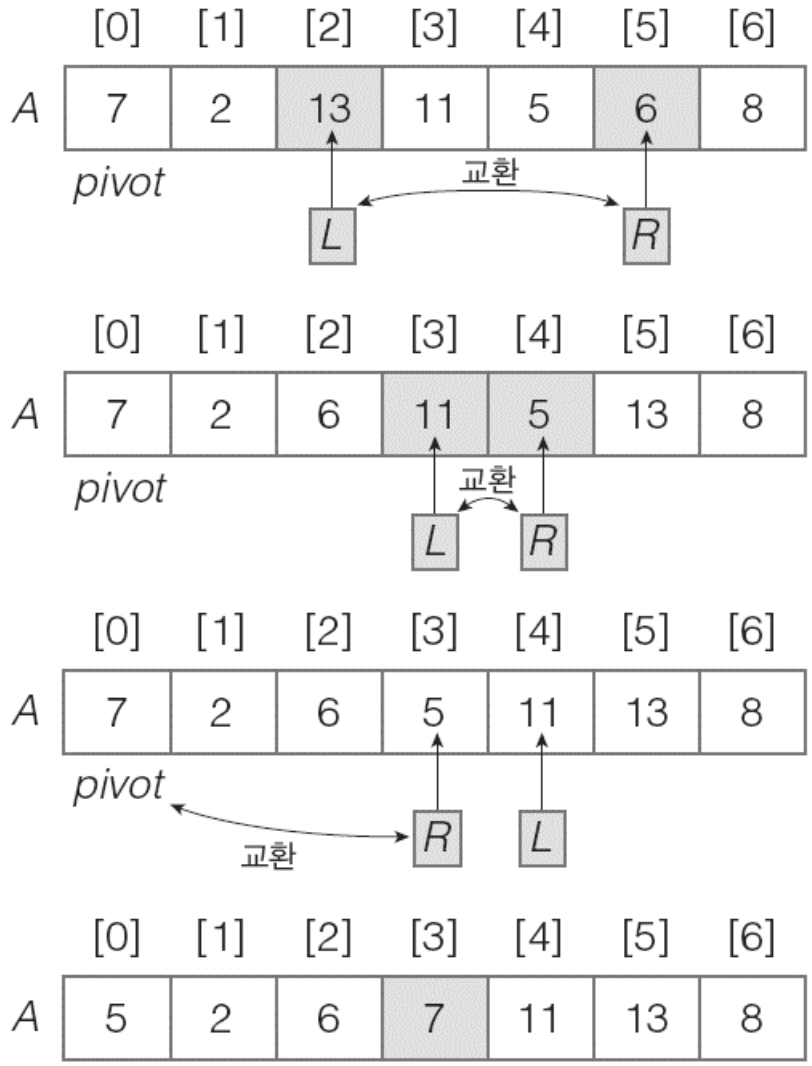
알고리즘 종류

- 퀵 정렬



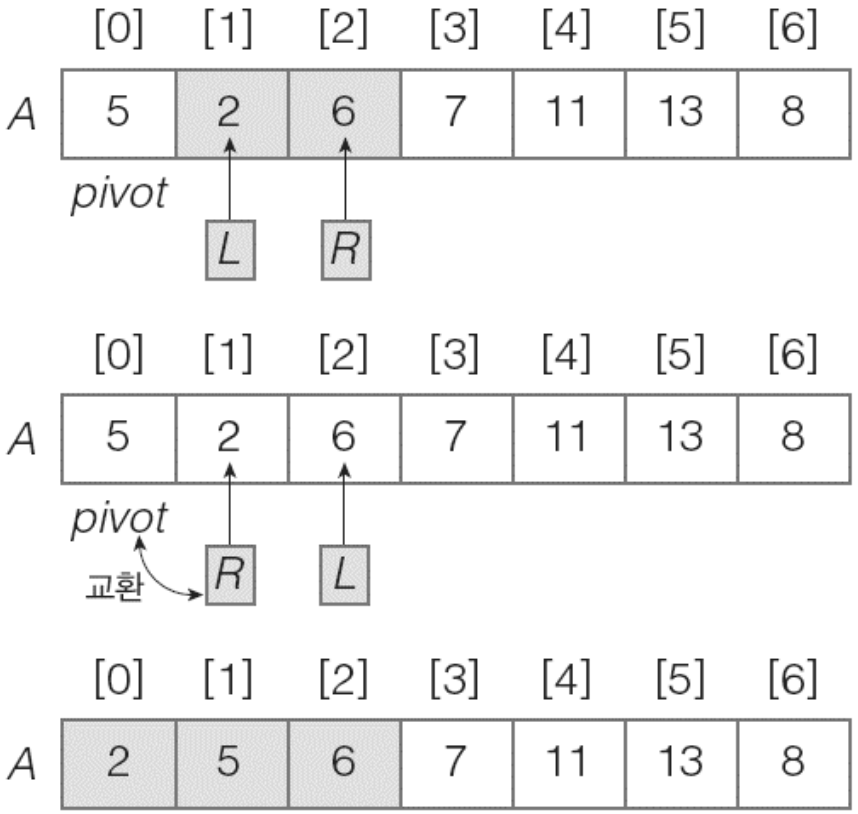
알고리즘 종류

- 퀵 정렬


[0] [1] [2] [3] [4] [5] [6]
A 7 2 13 11 5 6 8
pivot
교환
L
R
[0] [1] [2] [3] [4] [5] [6]
A 7 2 6 11 5 13 8
pivot
교환
L
R
[0] [1] [2] [3] [4] [5] [6]
A 7 2 6 5 11 13 8
pivot
교환
R
L
[0] [1] [2] [3] [4] [5] [6]
A 5 2 6 7 11 13 8

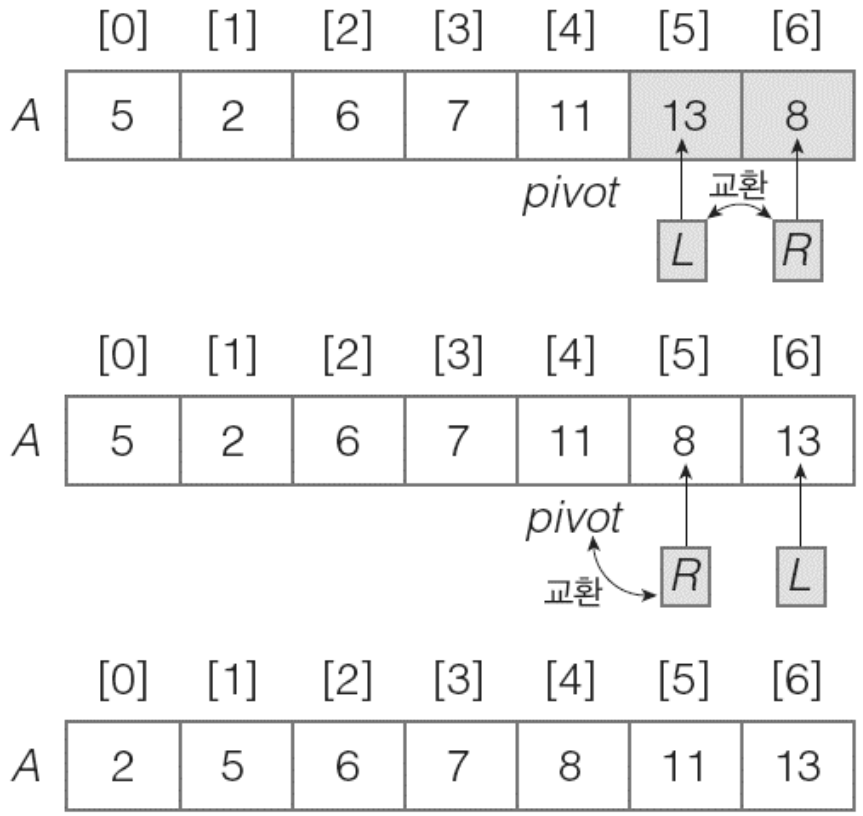
알고리즘 종류

- 퀵 정렬


[0] [1] [2] [3] [4] [5] [6]
A 5 2 6 7 11 13 8
pivot
L
R
[0] [1] [2] [3] [4] [5] [6]
A 5 2 6 7 11 13 8
pivot
교환
R
L
[0] [1] [2] [3] [4] [5] [6]
A 2 5 6 7 11 13 8

알고리즘 종류

- 퀵 정렬


[0] [1] [2] [3] [4] [5] [6]
A 5 2 6 7 11 13 8
pivot
교환
L
R
[0] [1] [2] [3] [4] [5] [6]
A 5 2 6 7 11 8 13
pivot
교환
R
L
[0] [1] [2] [3] [4] [5] [6]
A 2 5 6 7 8 11 13

알고리즘 종류

- 퀵 정렬

```
1   algorithm quick(int arr[], int left, int right){
2       int p;
3       int tmp;
4       if left < right then
5           p = arr[left];
6           i = left;
7           j = right + 1;
8           do
9               do
10                  i = i + 1;
11              while(arr[i] < p);
12              do
13                  j = j - 1 ;
14              while(arr[j] > p);
15              if i < j then
```

이산수학 – 문제해결 - 파이썬코딩

```
#선택정렬
def findmin(a):
    n = len(a)
    minpos = 0
    for i in range (1,n):
        if a[i] < a[minpos]:
            minpos=i;
    return minpos

def selsort(a):
    result =[]

    while a:
        minpos = findmin(a)
        value = a.pop(minpos)
        result.append(value)
    return result
d = [2,4,5,1,3]
print(selsort(d))
```

```
## 출력결과

[1, 2, 3, 4, 5]
```

이산수학 – 문제해결 - 파이썬코딩

```
#삽입정렬
def findpos(r,v):
    for i in range(0, len(r)):
        if v < r[i]:
            return i

    return len(r)

def insertsort(a):
    result=[]
    while a:
        value = a.pop(0)
        pos = findpos(result, value)
        result.insert(pos, value)
    return result
d = [2,4,5,1,3]
print(insertsort(d))
```

```
## 출력결과

[1, 2, 3, 4, 5]
```

이산수학 – 문제해결 - 파이썬코딩

```
#퀵정렬
def quicksort(a):
    n = len(a)
    if n<=1:
        return a
    pivot = a[-1]
    g1 = []
    g2 = []
    print("d = {0}".format(a))
    print("pivot: {0}".format(pivot))

    for i in range(0, n-1):
        if a[i]<pivot:
            g1.append(a[i])
        else:
            g2.append(a[i])
    return quicksort(g1) + [pivot] + quicksort(g2)

d = [9,7,5,1,3,6,8,2,4,10]
print(quicksort(d))
```

```
## 출력결과

d = [9, 7, 5, 1, 3, 6, 8, 2, 4, 10]
pivot: 10
d = [9, 7, 5, 1, 3, 6, 8, 2, 4]
pivot: 4
d = [1, 3, 2]
pivot: 2
d = [9, 7, 5, 6, 8]
pivot: 8
d = [7, 5, 6]
pivot: 6
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

맺음말

- 알고리즘 기본 개념 및 특성
- 알고리즘의 종류