이산수학

상명대학교 융합공과대학 휴먼지능정보공학전공
지능정보융합전공 / 인공지능융합전공 / 금융서비스AI융합전공
일반대학원 지능정보공학과
일반대학원 감성공학과
일반대학원 스포츠ICT융합학과
디지털 신기술 바이오헬스케어 혁신공유대학 사업단
지능정보기술연구소 (ai.smu.ac.kr)

강의개요

- 이산수학 개요
  - 이산수학 소개
- 논리와 명제
  - 기본개념, 논리연산자와 진리표, 논리적 동치, 한정기호, 명제함수, 추론, 파이썬 코딩
- 증명
  - 수학적 귀납법, 직접증명법, 간접증명법, 재귀법, 파이썬 코딩
- 집합
  - 기본개념 , 집합의 연산, 곱집합과 멱집합, 집합의 분할, 퍼지집합, 파이썬 코딩
- 관계
  - 기본개념, 관계의 표현, 관계의 성질, 관계의 연산, 파이썬 코딩
- 함수
  - 기본개념, 함수의 성질, 합성함수, 여러 가지 함수, 파이썬 코딩
- 중간고사

강의개요

- 행렬
  - 기본개념, 행렬의 연산, 여러 가지 행렬, 행렬식, 역행렬, 연립일차방정식, 파이썬 코딩
- 경우의 수
  - 기본개념, 순열과 조합, 이항계수, 확률, 파이썬 코딩
- 그래프
  - 기본개념, 오일러와 해밀턴 순환, 여러 가지 그래프, 그래프의 표현, 그래프의 탐색, 파이썬 코딩
- 트리
  - 기본개념, 이진트리, 신장트리, 파이썬 코딩
- 알고리즘
  - 기본개념, 정렬알고리즘, 탐색알고리즘, 파이썬 코딩
- 부울대수와 논리회로
  - 부울대수, 부울함수, 논리게이트, 논리회로, 조합회로의 최소화
- 유한상태기계와 오토마타
  - 오토마타, 유한상태기계
- 기말고사

학습목표

- 알고리즘 기본 개념 및 특성
- 알고리즘의 종류

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 문제해결 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

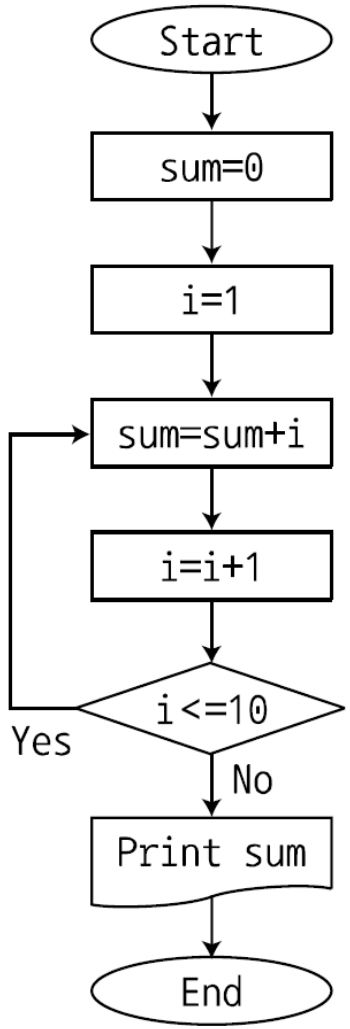
- 알고리즘

```
Algorithm max(a, b, c)
Begin
    x=a
    if b>x then
        x=b
    if c>x then
        x=c
End
```

기본개념

- 알고리즘
  - 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

```
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

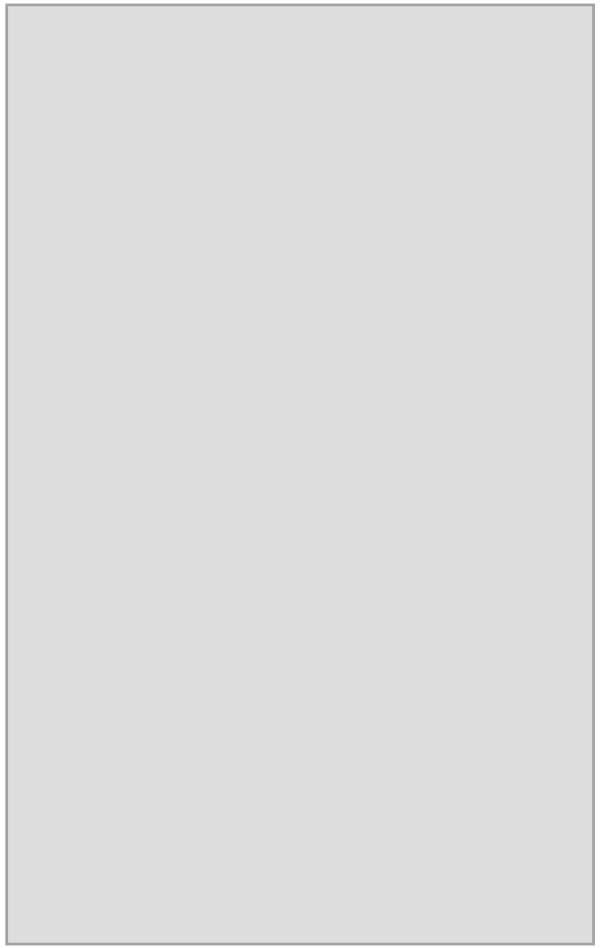
- 알고리즘

```
Algorithm gcd(a, b)
Begin
   if a<b then
   begin
      tmp=a
      a=b
      b=tmp
   endif
   while b≠0
   begin
      r=a mod b
      a=b
      b=r
   endwhile
   gcd(a, b)=a
End
```

GCD(150, 165) = GCD(165, 150)

= GCD(150, 15) $\because 165\ mod\ 150 = 15$

= GCD(15, 0) $\because 150\ mod\ 15 = 0$



기본개념

- 정렬 (sorting)
  - 기억 공간의 데이터 중에서 원하는 정보를 찾는 일련의 작업
- 정렬알고리즘
  - 원소 집합 내의 원소들이 임의로 나열되어 있을 때 이 원소들을 일정 기준에 따라 다시 나열하는 방식
- 정렬알고리즘 종류
  - 버블정렬
  - 선택정렬
  - 삽입정렬
  - 퀵정렬
  - 합병정렬

기본개념

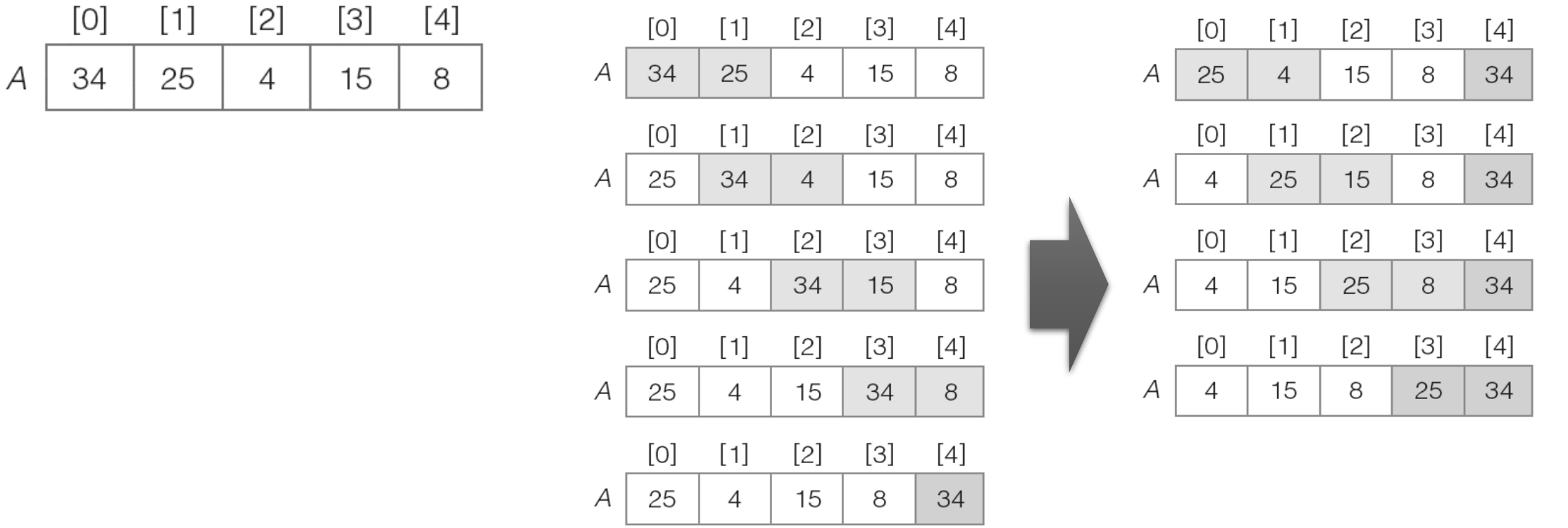
- 탐색(search)
  - 기억 공간의 데이터 중에서 원하는 정보를 찾는 일련의 작업
- 선형탐색(linear search)
  - 데이터의 조작없이 순서대로 데이터를 탐색
- 이진탐색(binary search)
  - 정렬되어 있는 데이터에서 원하는 값이 정렬된 데이터의 가운데 값보다 작은 쪽에 있는지 큰 쪽에 있는지를 판단하면서 어느 한쪽만을 탐색함으로써 탐색시간을 단축
- 이진탐색트리(binary search tree)
  - 데이터를 트리로 구현하여 탐색

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"

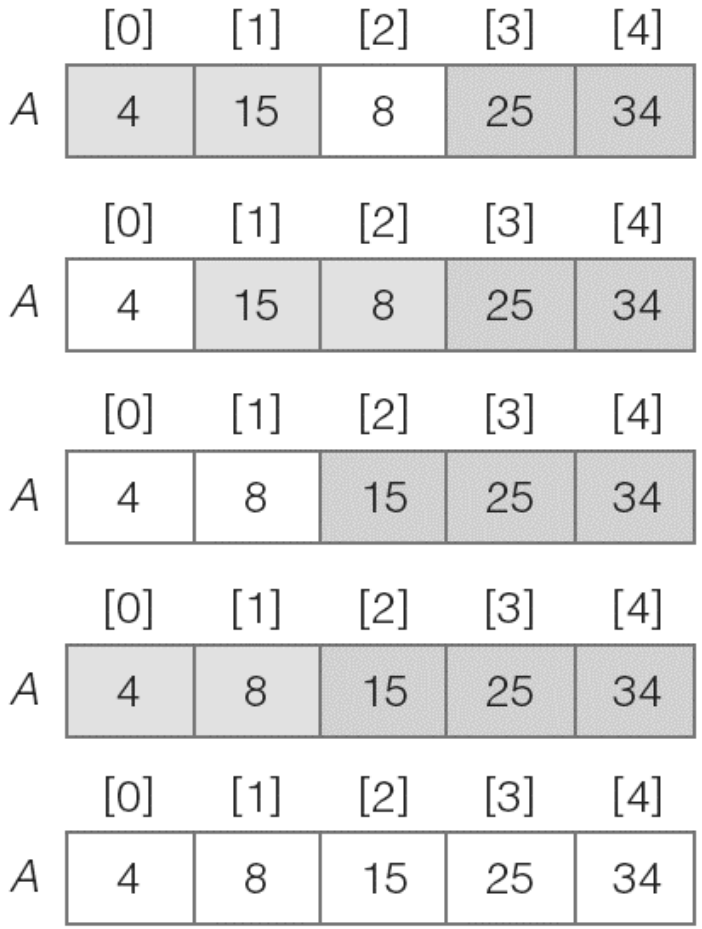
알고리즘 종류

- 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 끝에 오도록 정렬



알고리즘 종류

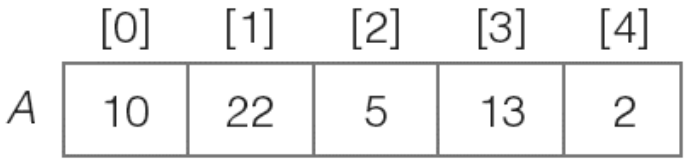
- 버블 정렬




```
Algorithm bubblesort(A, n)
/* A=(A[0], A[1], ···, A[n−1]) */
Begin
   for i=1 to n−1
      for j=1 to n−i
         if A[j−1]>A[j] then
         begin
            tmp=A[j−1]
            A[j−1]=A[j]
            A[j]=tmp
         endif
End
```

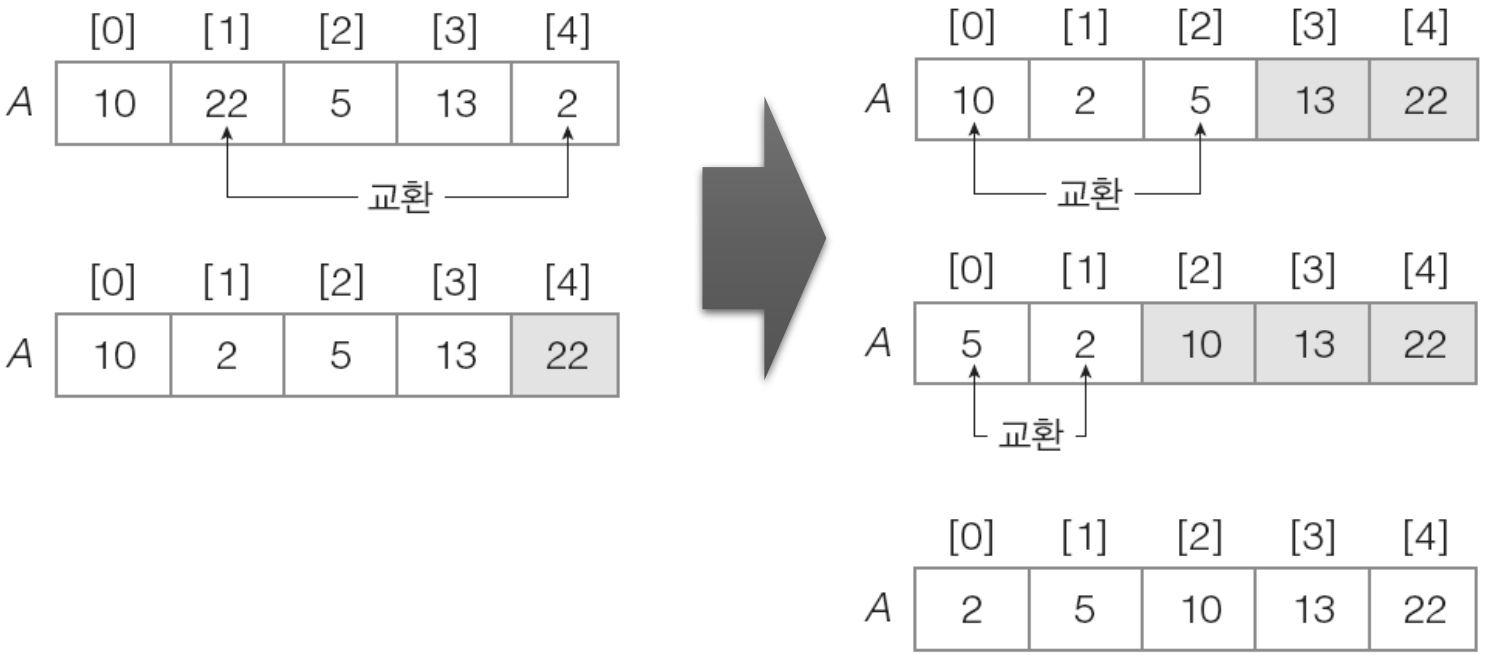
알고리즘 종류

- 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 완성 될 때까지 반복



알고리즘 종류

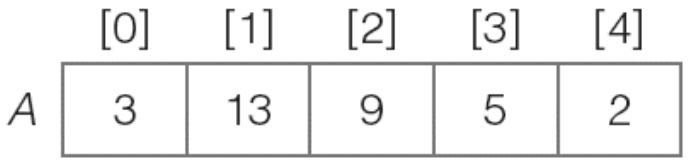
- 선택 정렬
  - 배열에서 가장 큰 값을 찾음
  - 그 값을 A[n-1](배열의 마지막)의 값과 서로 교환
  - A[n-1]을 제외한 나머지 값들 중에서 가장 큰 값을 찾음
  - 그 값을 A[n-2]의 값과 서로 교환
  - 같은 과정을 정렬이 완성 될 때까지 반복

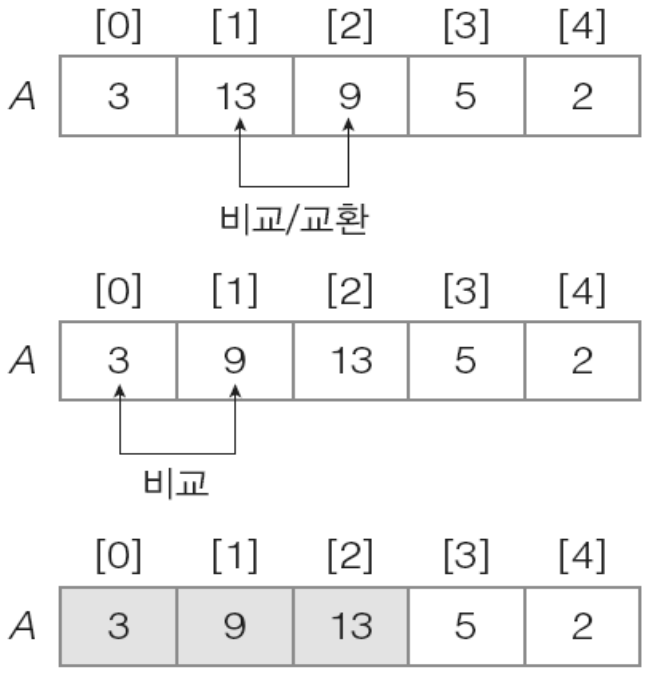


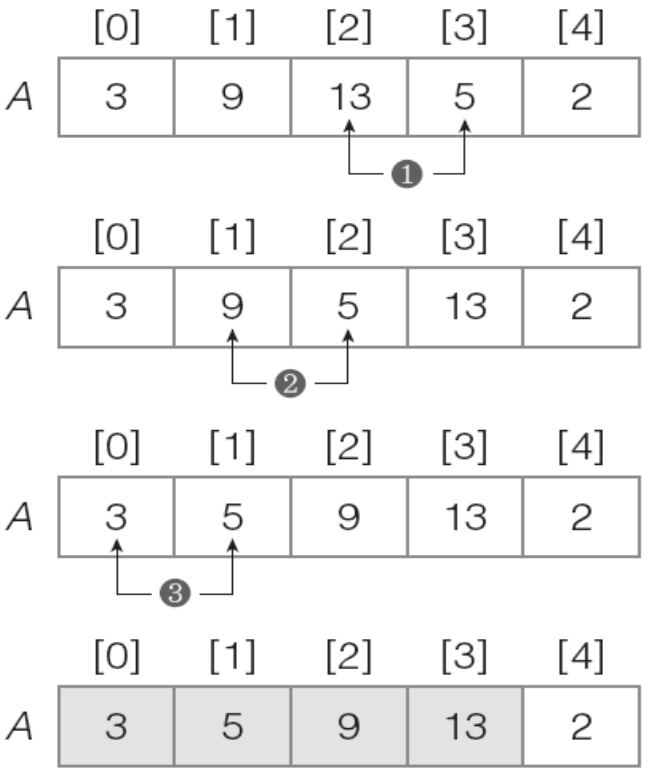

```
Algorithm selectionsort(A, n)
/* A=(A[0], A[1], ···, A[n−1]) */
Begin
   for i=1 to n−1
   begin
      max=n−i
      for j=0 to n−i−1
         if A[j]>A[max] then
            max=j
      if max≠n−i then
      begin
         tmp=A[n−i]
         A[n−i]=A[max]
         A[max]=tmp
      endif
   endfor
End
```

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 데이터들을 삽입하여 정렬

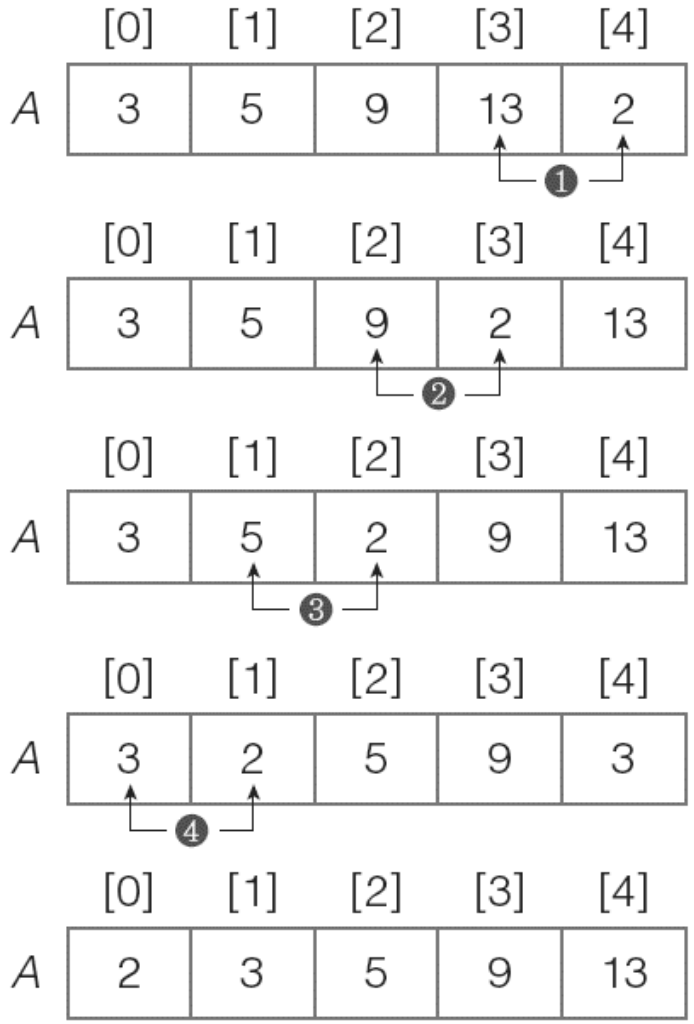






알고리즘 종류

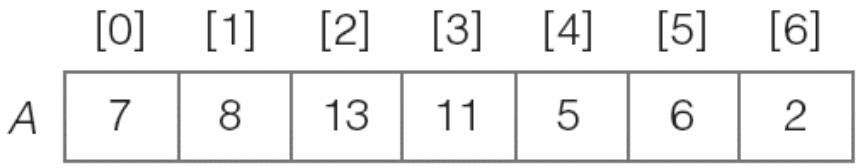
- 삽입 정렬

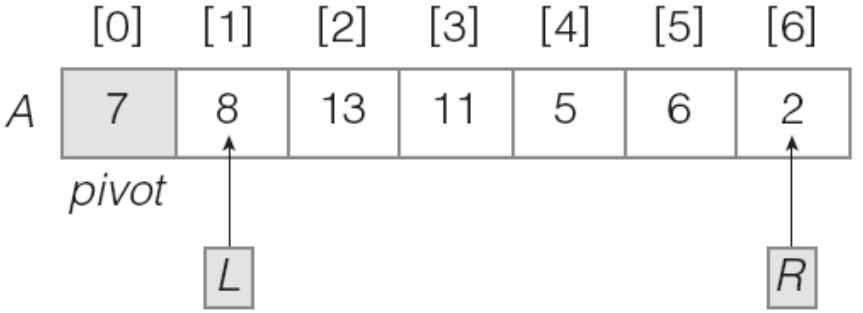


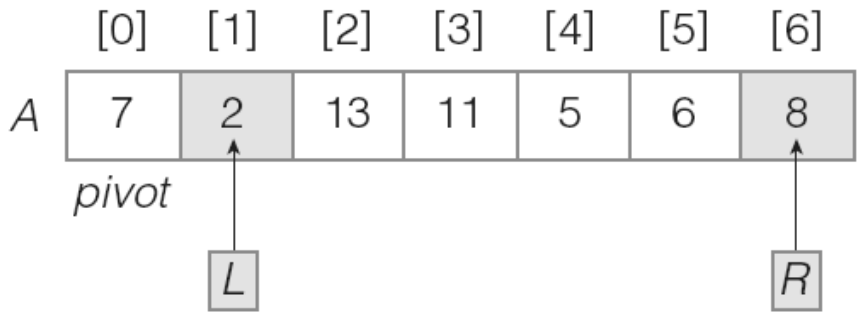

```
Algorithm insertionsort(A, n)
/* A=(A[0], A[1], ···, A[n−1]) */
Begin
   for i=1 to n−1
   begin
      tmp=A[i]
      j=i−1
      t=i
      while j≥0
      begin
         if A[j]>tmp then
         begin
            A[j+1]=A[j]
            t=j
         endif
         j=j−1
      endwhile
      A[t]=tmp
   endfor
End
```

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 집합에서 동일하게 진행

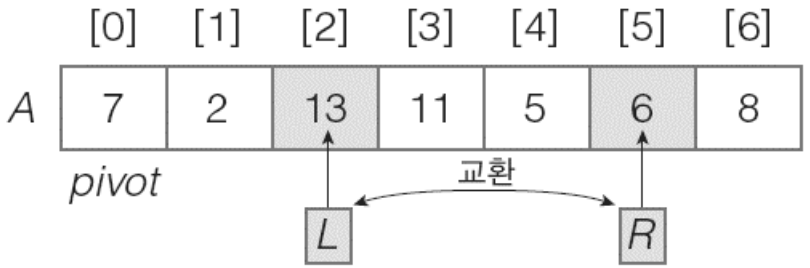


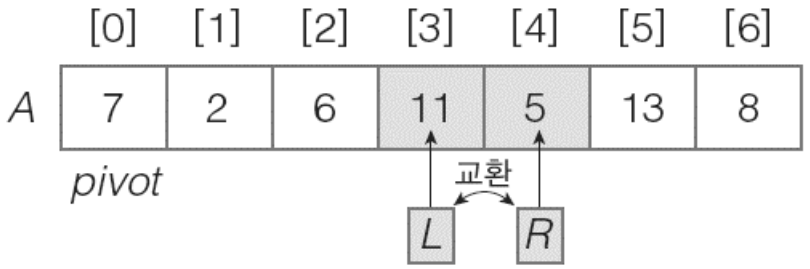


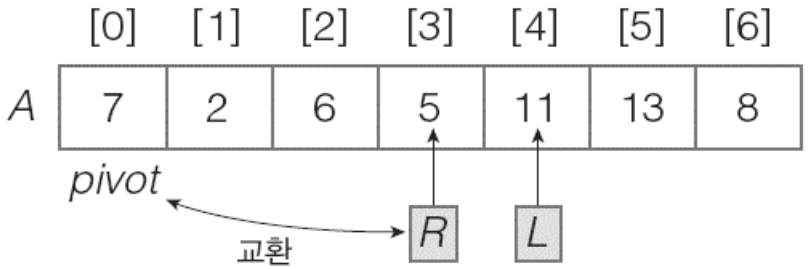


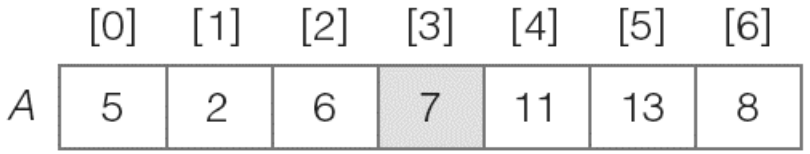
알고리즘 종류

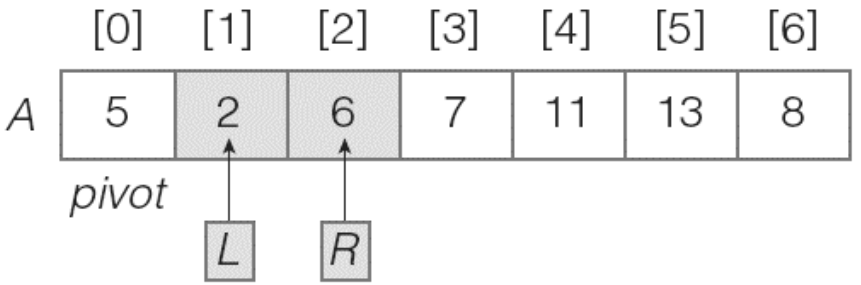
- 퀵 정렬

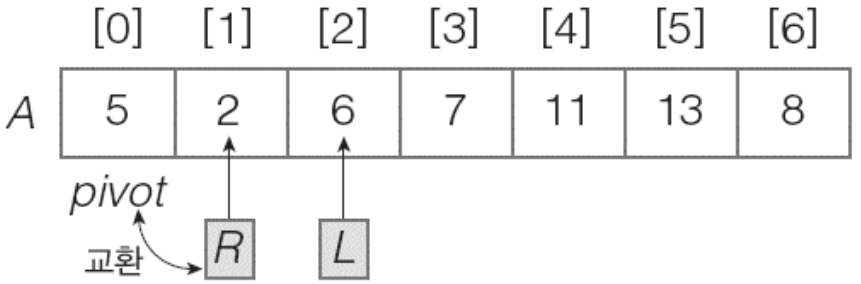

[0] [1] [2] [3] [4] [5] [6]
A 7 2 13 11 5 6 8
pivot
교환
L
R

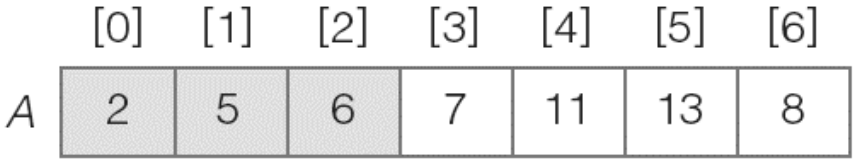

[0] [1] [2] [3] [4] [5] [6]
A 7 2 6 11 5 13 8
pivot
교환
L
R


[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

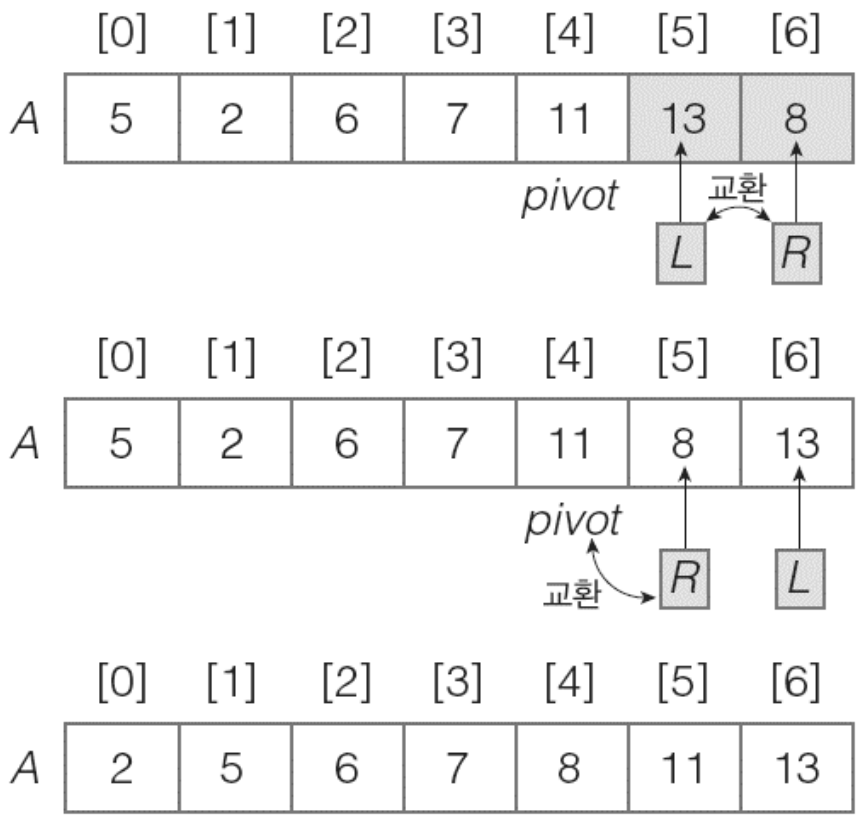

[0] [1] [2] [3] [4] [5] [6]
A 5 2 6 7 11 13 8
pivot
L
R


[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

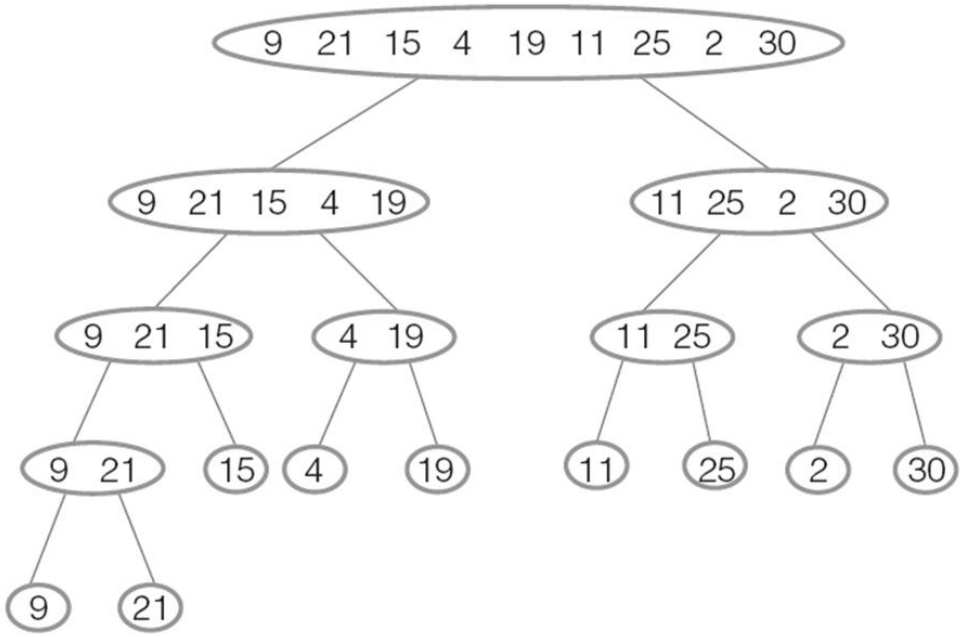



```
Algorithm quicksort(A, L, R)
/* A=(A[0], A[1], ···, A[n−1]) */
Begin
   if L<R then
   begin
      pivot=A[L]
      i=L
      j=R+1
      do
         do
            i=i+1
         while(A[i]<pivot)
         do
            j=j−1
         while(A[j]>pivot)
         if i<j then
         begin
            tmp=A[i]
            A[i]=A[j]
            A[j]=tmp
         endif
      while(i<j)
      tmp=A[L]
      A[L]=A[j]
      A[j]=tmp
      quicksort(A, L, j−1)
      quicksort(A, j+1, R)
   endif
End
```

알고리즘 종류

- 합병정렬(merge sort)
  - 정렬 방식
    - 정렬할 데이터들을 이등분
    - 하나의 원소집합이 될 때까지 이등분
    - 하나의 원소로 이등분된 집합을 정렬하면서 합병
    - 두 개의 정렬된 원소를 포함한 집합들을 다시 합병
    - 전체 데이터를 정렬할 때가지 합병 반복
  - 합병정렬 예

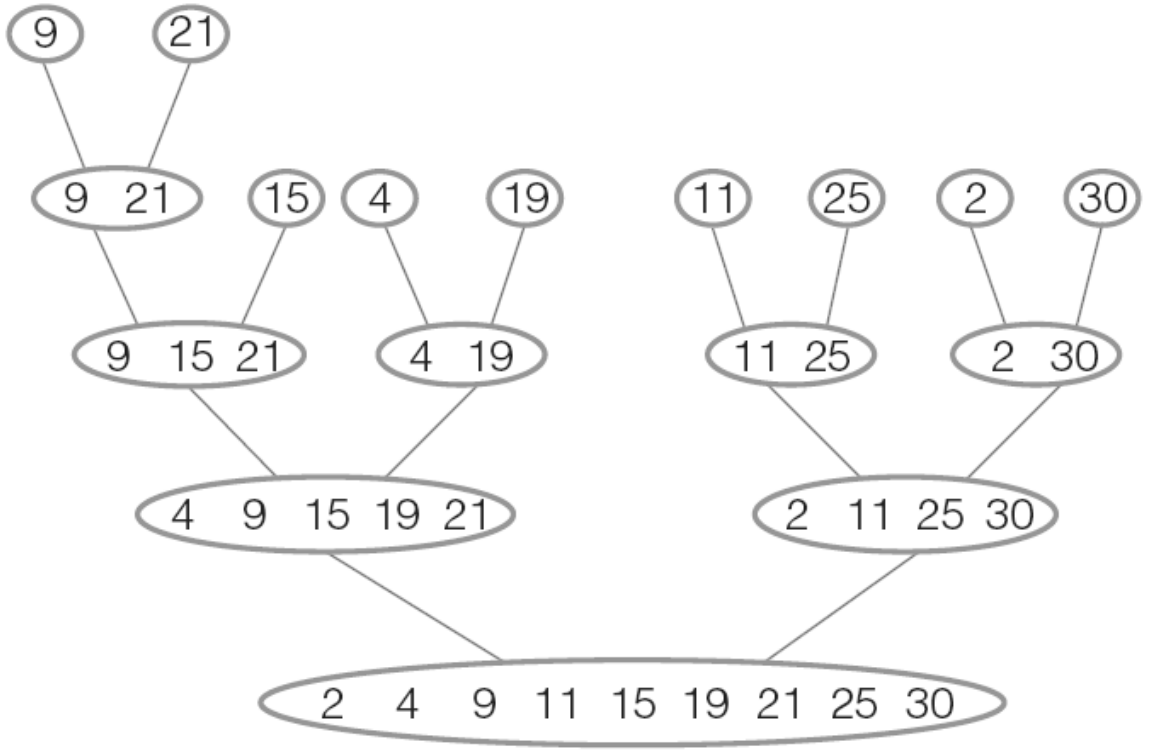
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]
A	9	21	15	4	19	11	25	2	30



알고리즘 종류

- 합병정렬(merge sort)

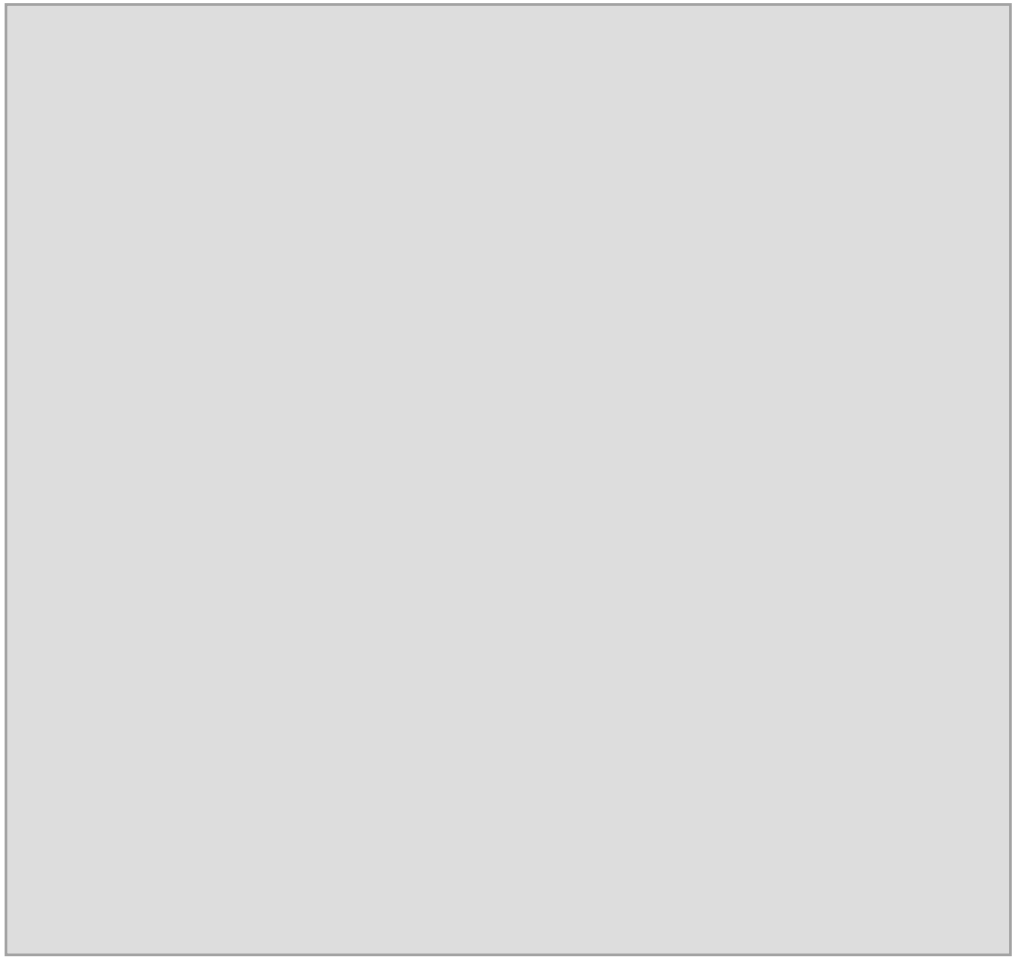
집합1	집합2	합병정렬된 집합
9 15 21	4 19	
9 15 21	19	4
15 21	19	4 9
21	19	4 9 15
21		4 9 15 19
		4 9 15 19 210



알고리즘 종류

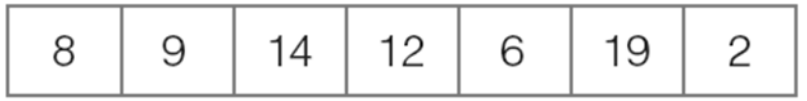
- 합병정렬(merge sort)

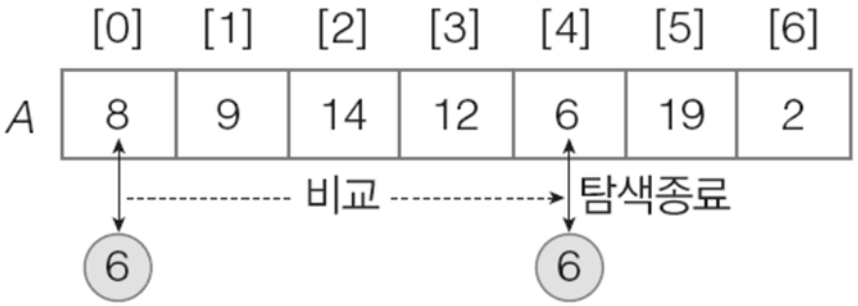
```
Algorithm mergesort(A, i, j)
/* A=(A[0], A[1], ···, A[n−1]) */
Begin
    if i=j then
        return
    m=⌊(i+j)/2⌋
    mergesort(A, i, m)
    mergesort(A, m+1, j)
    merge(A, i, m, j, C)
    for k=i to j
        A[k]=C[k]
End
```



알고리즘 종류

- 탐색 알고리즘(***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






```
Algorithm linearsearch(A, n, key, loc)
/* A=(A[0], A[1], ···, A[n-1]) */
Begin
    loc=-1
    for i=0 to n-1
        if A[i]=key then
        begin
            loc=i
            break
        endif
End
```

알고리즘 종류

- 탐색 알고리즘($\boldsymbol{Search\ Algorithm}$)

  주어진 원소의 집합에서 특정 원소를 찾는 작업을 체계적으로 명시해놓은 것

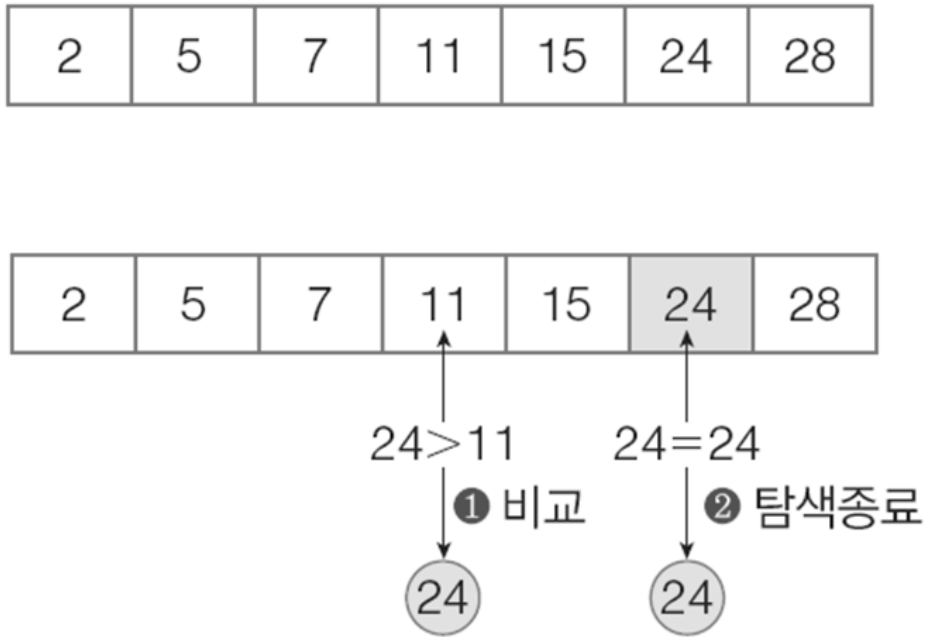
  (2) 이진 탐색(Binary Search) 알고리즘

  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

- 탐색 알고리즘(***Search Algorithm***)
  주어진 원소의 집합에서 특정 원소를 찾는 작업을 체계적으로 명시해놓은 것
  (2) 이진 탐색(Binary Search) 알고리즘

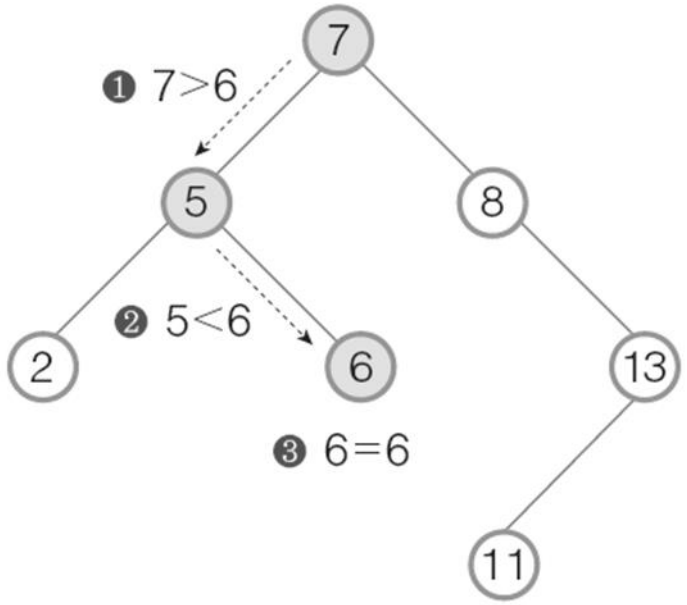



```
Algorithm linearsearch(A, n, key)
/* A=(A[0], A[1], ···, A[n−1]) */
Begin
    first=0
    last=n
    while first≤last
    begin
        mid=⌊(first+last)/2⌋
        if A[mid]=key then
            exit
        else
            if A[mid]>key then
                last=mid−1
            else
                first=mid+1
    endwhile
    if first>last then
        mid=−1
End
```

알고리즘 종류

- 탐색 알고리즘(***Search Algorithm***)
  주어진 원소의 집합에서 특정 원소를 찾는 작업을 체계적으로 명시해놓은 것
  (2) 이진 탐색(Binary Search) 알고리즘

7	8	13	11	5	6	2



알고리즘 종류

- 탐색 알고리즘

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

- 탐색 알고리즘

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.

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