알고리즘

5장 선택 알고리즘
(selection algorithm)

학습내용

1. 평균 선형시간 선택 알고리즘
2. 최악의 경우에도 선형시간을 보장하는 선택 알고리즘

학습목표

- 평균적으로 선형시간이 걸리는 선택 알고리즘의 원리를 이해한다.
- 평균 선형시간 선택 알고리즘의 수행시간 분석을 이해한다.
- 최악의 경우에도 선형시간을 보장하는 선택 알고리즘의 원리를 이해한다.

학습내용

1. 평균 선형시간 선택 알고리즘

2. 최악의 경우에도 선형시간을 보장하는 선택 알고리즘

Selection (i번째 작은 수 찾기)

- 배열 A[p…r]에서 i번째 작은 원소를 찾는 문제

예) 다음 A[1…10] 중 4번째 작은 원소는?

31	8	48	73	11	3	20	29	65	15

– n개의 원소 각각을 적어도 한번씩은 봐야 하므로 **$\Omega(n)$**

– $O(n \log n)$ 알고리즘으로 정렬한 후 i번째 원소를 고르면 되므로 **$O(n \log n)$**

➔ 선택 알고리즘을 사용하여 $n \sim n \log n$ 범위 내에서 얼마나 n에 근접할 수 있는가?

Selection (*i*번째 작은 수 찾기)

- 두 가지 선택(selection) 알고리즘
  1. 평균적으로 선형시간이 소요되는 선택 알고리즘 ➔ **select**
  2. 최악의 경우에도 선형시간이 보장되는 선택 알고리즘 ➔ **linearSelect**

✓ 선형시간이란 n에 비례한 시간을 말한다.

✓ 주의: 선택 알고리즘(selection algorithm)은 선택 정렬(selection sort)과는 무관

평균 선형시간 선택 알고리즘

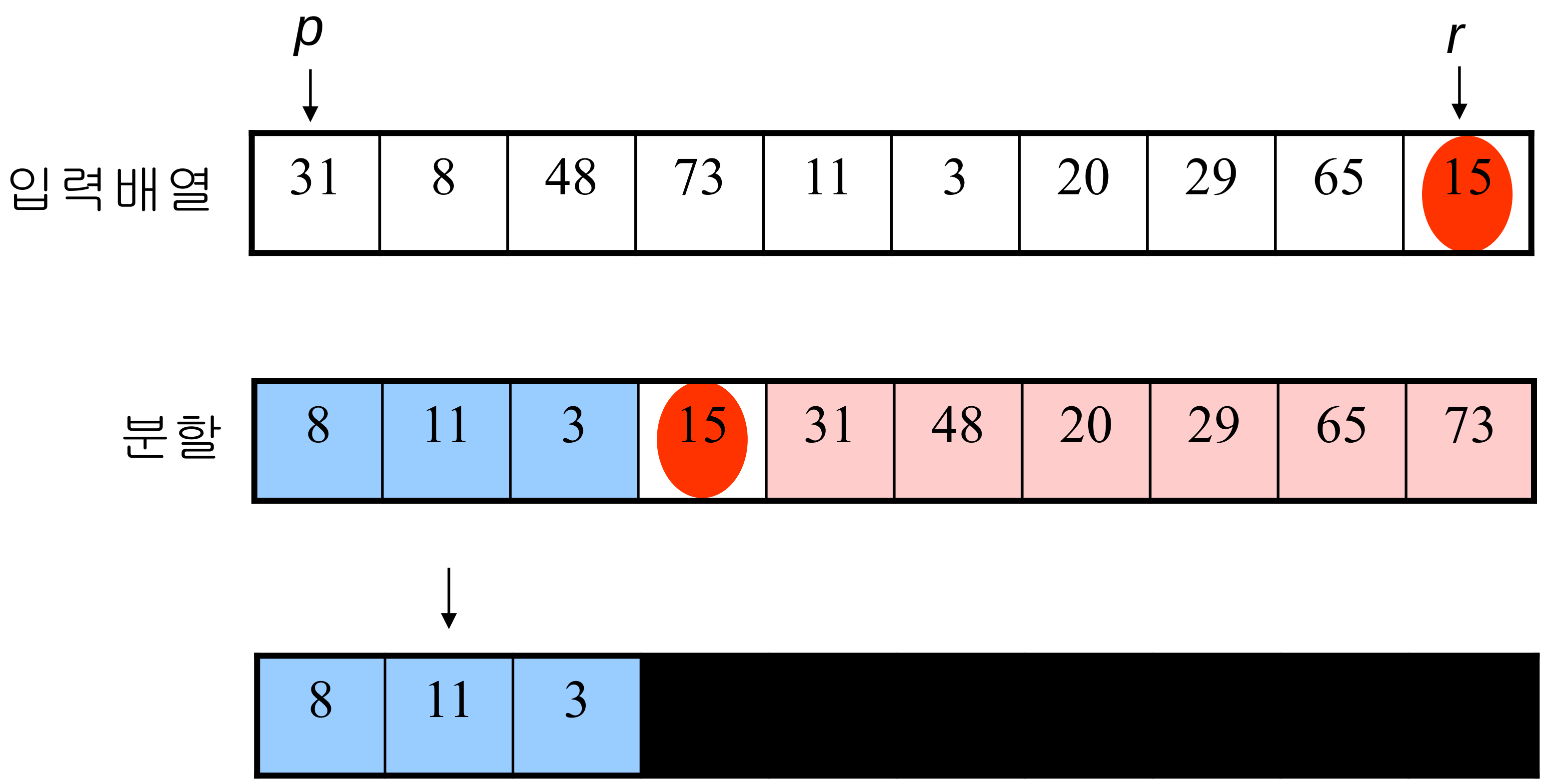
```
select(A[], p, r, i)            ▷ 배열 A[p...r]에서 i번째 작은 원소를 찾는다 .
{
    if (p = r) then return A[p] ;    ▷ 원소가 하나뿐인 경우. i는 반드시 1.

    q ← partition(A, p, r) ;  ▷ quickSort에서의 partition 알고리즘과 동일

    k ← q-p+1;       ▷ k : 기준원소가 A[p...r] 에서 k 번째 작은 원소임을 의미

    if (i < k) then return select(A, p, q-1, i) ; ▷ 왼쪽 분할로 범위를 좁힘
    else if (i = k) then return A[q] ;          ▷ 기준원소가 바로 찾는 원소임
    else return select(A, q+1, r, i-k) ;        ▷ 오른쪽 분할로 범위를 좁힘
}
```

✓평균 수행시간: **$\Theta(n)$**

✓최악의 경우 수행시간: **$\Theta(n^2)$**

select 작동 예 1

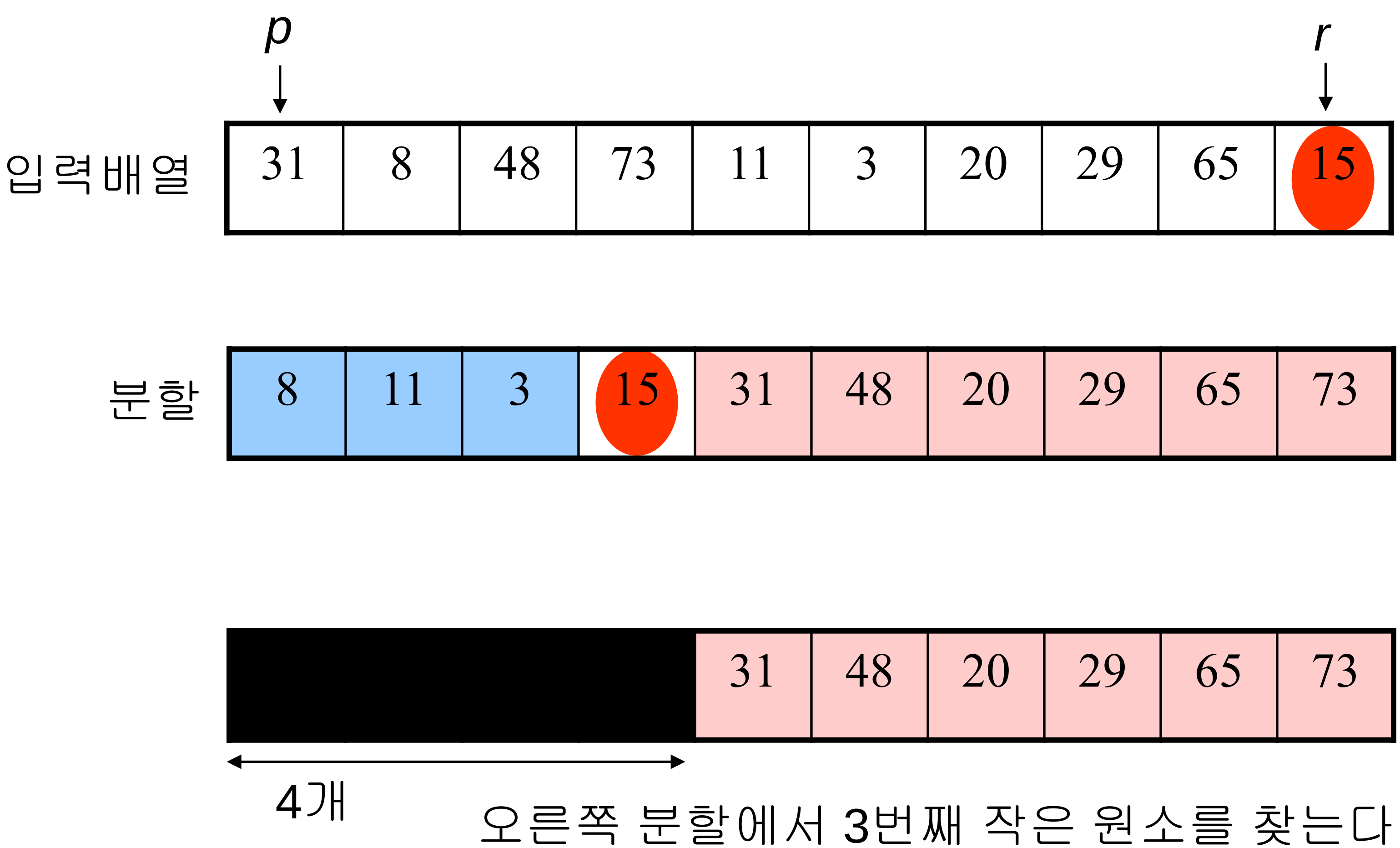
2번째 작은 원소 찾기



왼쪽 분할에서 2번째 작은 원소를 찾는다

select 작동 예 2

7번째 작은 원소 찾기



select의 평균 수행시간

```
select(A[], p, r, i)
{
    if (p = r) then return A[p] ;

    q ← partition(A, p, r) ;

    k ← q-p+1;

    if (i < k) then return select(A, p, q-1, i) ;
    else if (i = k) then return A[q] ;
    else return select(A, q+1, r, i-k)
}
```

select의 평균 수행시간

$$T(n) \le \frac{1}{n}\sum_{k=1}^{n} \max[T(k\text{-}1),\ T(n\text{-}k)] \quad + \quad \Theta(n)$$

분할된 양쪽 중 큰
쪽을 처리하는 비용

재귀호출을 제외한 오버헤드
(분할이 대부분)

이것은 $T(n) \le cn$임을 추정 후 증명법으로 증명할 수 있다.

$$\therefore\ T(n) = O(n)$$

$T(n) = \Omega(n)$임은 자명하므로 $T(n) = \Theta(n)$

select의 최악의 경우 수행시간

$$T(n) = T(n\text{-}1) + \Theta(n)$$

재귀호출을 제외한 오버헤드
(분할이 대부분)

분할이 0 : n-1로 되고 큰
쪽을 처리하는 비용

$$\therefore T(n) = \Theta(n^2)$$

학습내용

1. 평균 선형시간 선택 알고리즘
2. **최악의 경우에도 선형시간을 보장하는 선택 알고리즘**

최악의 경우 선형시간 선택 알고리즘

- 앞서 다룬 평균 선형시간 선택 알고리즘 **select**는
  - 수행시간이 **분할의 균형**의 영향을 받는다.
  - 최악의 경우 $\Theta(n^2)$
- 이번 알고리즘은
  - 최악의 경우에도 분할의 균형이 어느 정도 보장되도록 함으로써 수행시간이 $\boldsymbol{\Theta(n)}$이 되도록 한다.

  ➔ **최악의 경우 선형시간 선택 알고리즘 linearSelect**

  단, 분할의 균형을 유지하기 위한 오버헤드가 지나치게 크면 안 된다.

분할과 선택 알고리즘 수행시간

- 앞서 다룬 평균 선형시간 선택 알고리즘 **select**에서
  - 계속 1:9로 분할되고, 이 중 큰 분할에서 탐색을 하게 되는 경우 수행시간 T(n)은 다음과 같다.

$$T(n) = T(9n/10) + \Theta(n)$$

  - 마스터 정리를 이용하여 풀면

$$T(n) = \Theta(n)$$

  - 즉, 분할의 균형이 아주 나빠 보여도 일정한 상수비만 넘지 않으면 점근적 복잡도는 $\Theta(n)$
- 이번 알고리즘 **linearSelect**는
  - 분할의 균형을 어느 정도까지 보장함으로써 최악의 경우에도 $\Theta(n)$이 되는 것을 보장한다.
  - 단, 균형을 맞추는 오버헤드가 너무 커지면 목표를 이룰 수 없다.

linearSelect(**A**, ***p***, ***r***, ***i***) ▷ 배열 A[*p*...*r*]에서 *i*번째 작은 원소를 찾는다.
{

① 원소가 5개 이하이면 *i*번째 작은 원소를 찾고 알고리즘을 끝낸다.

② 전체 원소 n개를 원소 수 5인 작은 그룹들로 나눈다. (작은 그룹의 개수는 $\lceil n/5 \rceil$이며, n이 5의 배수가 아니면 이중 한 그룹은 원소 수가 5 미만이 된다.)

③ 각 그룹에서 중앙값(원소 수가 5이면 3번째 원소)을 찾는다. 이렇게 찾은 중앙값들을 $m_1, m_2, \ldots, m_{\lceil n/5 \rceil}$ 이라 하자.

④ $m_1, m_2, \ldots, m_{\lceil n/5 \rceil}$ 의 중앙값 M을 재귀적으로 구한다. 이 m_k 값들이 홀수개면 중앙값이 하나이므로 문제 없고, 짝수개면 두 중앙값 중 임의로 하나를 선택한다. ▷ call **linearSelect**( )

예) m_k가 15개면 linearSelect를 재귀 호출하여 8번째 작은 m_k 를 찾음 → 이를 M으로 삼음

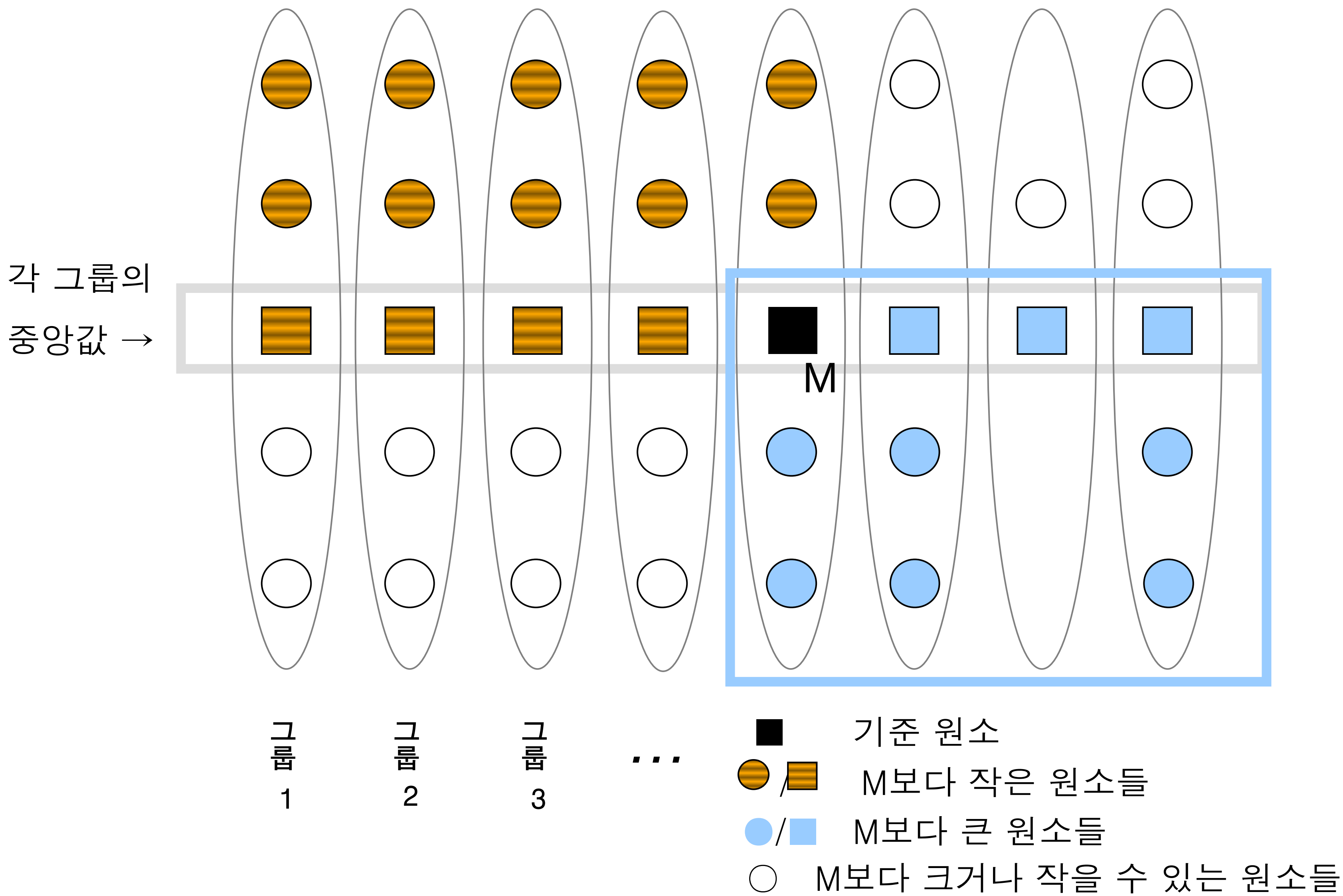
⑤ M을 기준원소로 삼아 전체 원소를 분할한다. (M보다 작거나 같은 것은 M의 왼쪽에, M보다 큰 것은 M의 오른쪽에 오도록)

⑥ 분할된 두 쪽 중 ***i***번째 작은 원소가 속해 있을 쪽을 선택하여 단계 ①~⑥을 재귀적으로 반복한다. ▷ call **linearSelect**( )

}

✓최악의 경우 수행시간: $\Theta(n)$

기준 원소를 중심으로 한 대소 관계



linearSelect 수행 예

다음과 같은 23 개의 값 중에서 18번째로 작은 원소를 찾으시오.

1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*	*16*	*17*	*18*	*19*	*20*	*21*	*22*	*23*
28	26	23	43	22	17	29	11	13	15	35	12	14	42	31	20	41	19	16	40	18	19	27

linearSelect의 최악의 경우 수행시간

$$T(n) \le T(\lceil n/5 \rceil) + T(7n/10 + 2) + \Theta(n)$$

④ ⑥ ①②③⑤

이것은 $T(n) \le cn$임을 추정 후 증명법으로 증명할 수 있다.

$$\therefore\ T(n) = O(n)$$

$T(n) = \Omega(n)$임은 명백하므로 $T(n) = \Theta(n)$

요약

- i번째 작은 수를 찾는 방법은?
  - 정렬한 다음 i번째 수를 찾는 방법 $\Theta(n \log n)$
  - 선택 알고리즘 : 퀵 정렬의 partition을 이용
    - select - 평균 $\Theta(n)$, 최악의 경우 $\Theta(n^2)$
      - 분할의 균형을 운에 맡기므로 평균 $\Theta(n)$, 최악의 경우 $\Theta(n^2)$
      - 분할의 균형이 일정한 상수 비율 이상으로 나빠지지 않는다면 최악의 경우 $\Theta(n)$
    - linearSelect – 평균 $\Theta(n)$, 최악의 경우 $\Theta(n)$
      - 분할의 균형이 일정 비율 이상으로 나빠지지 않도록 전처리하는 선택 알고리즘
      - 최악의 경우 $\Theta(n)$