분할정복 방법_2

2020년도 2학기 최 희석

목차

- 대표적 분할정복 방법
  - 선택문제
  - 최근접 점의 쌍 찾기
  - 분할 정복을 적용하는 데 있어 주의할 점

대표적 분할정복 방법

선택문제

선택 문제의 개념과 원리

- n개의 원소가 임의의 순서로 저장 배열 A[0..n-1]에서
  i번째로 작은 원소를 찾는 문제 -> 정렬되지 않은 데이터 집합에서 k 번째 element를 찾아라
  - i=1 → 최소값
  - i=n/2 → 중간값
  - i=n → 최대값
- 직관적인 방법
  - 오름차순으로 정렬한 후 i번째 원소를 찾는 방법 → O(nlogn)
  - 최솟값 찾는 과정을 i번 반복((i-1)번째까지는 최솟값을 찾은 후 삭제) → O(**i**n)
- 최악 $O(n^2)$, 평균 O(n) 알고리즘
- 최악 O(n), 평균 O(n) 알고리즘

최소값 찾기

- 각 데이터를 하나씩 모두 비교하는 방법
  - n개의 데이터에 대해서 최소한 (n-1)번의 비교가 필요

→ O(n)

```
FindMinimum (A[], n)
{
  min = A[0];
  for (i=1; i<n; i++)
    if (A[i]<min) min = A[i];
  return min;
}
```

최소값과 최대값 모두 찾기

- **최소값 찾은 후 최대값 찾는 방법(또는 최대값 찾은 후 최소값 찾기)**
  - n개의 데이터에서 최소값을 찾는데 (n-1)번의 비교

    \+ (n-1)개의 데이터에서 최대값을 찾는데 (n-2)번의 비교

    2n-3번의 비교

- **모든 원소를 두 개씩 짝을 이루어 동시에 최소값/최대값과 비교**
  - 2n-3번의 비교가 아닌 3/2n-2 번의 비교로 수행 가능

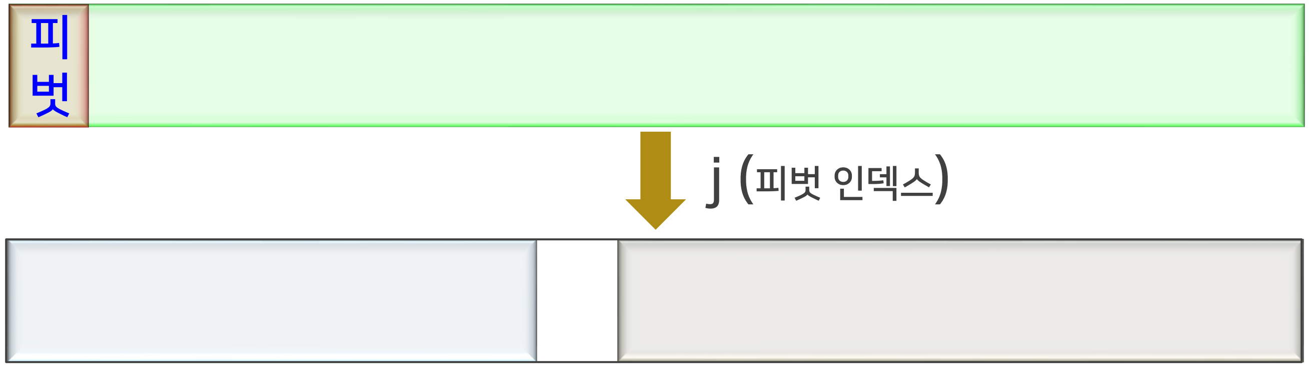
최소값과 최대값 모두 찾기

```
FindMinMax (A[], n, min, max)
{
  if (A[0]<A[1]) { min = A[0]; max = A[1]; }
  else { min = A[1]; max = A[0]; }
  for (i=2; i<n; i+=2) {
    if (A[i]<A[i+1]) { small = A[i];  large = A[i+1]; }
    else { small = A[i+1];  large = A[i]; }
    if (small < min) min = small;
    if (large > max) max = large;
  }
}
```

i번째로 작은 원소 찾기 최악 $O(n^2)$, 평균 O(n)

▶개념과 원리

- 퀵 정렬의 분할 함수 Partition()을 순환적으로 적용



- **i = j** → 피벗이 찾고자 하는 i번째 원소
- **i < j** → 왼쪽 부분배열에 대해 순환 적용
- **i > j** → 오른쪽 부분배열에 대해 순환 적용

i번째로 작은 원소 찾기 최악 O(n^2), 평균 O(n)

- 분할정복 방법을 적용한 알고리즘

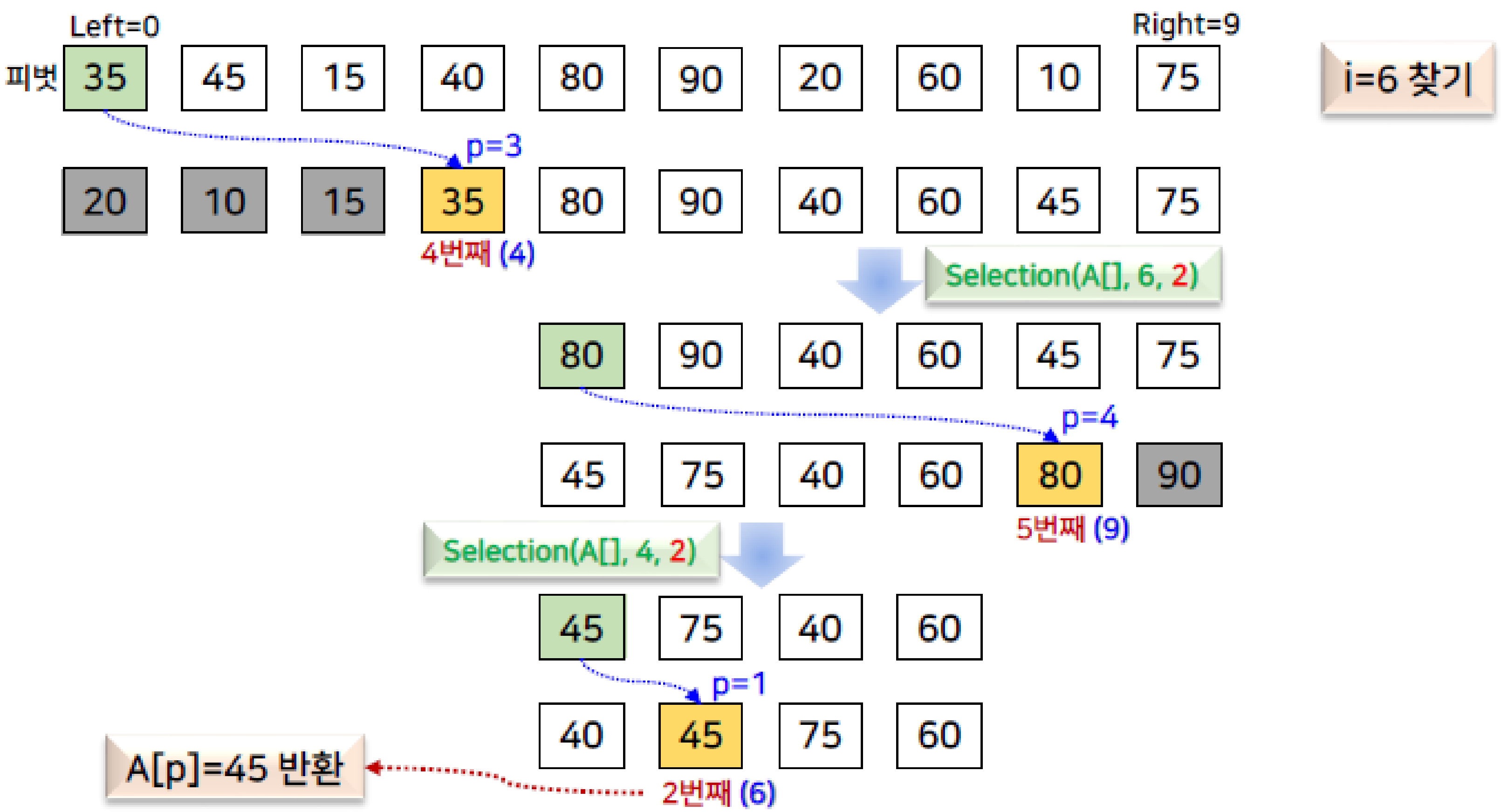
- **분할**
  - 피벗을 기준으로 주어진 배열을 두 부분배열로 분할,

    i 가 피벗의 인덱스와 같으면 피벗의 값을 반환하고 종료
- **정복**
  - 인덱스 i 가 포함된 부분배열에 대해서 선택알고리즘을 순환적으로 적용
- **결합**
  - 필요 없음

i번째로 작은 원소 찾기 최악 O(n^2), 평균 O(n)

```
int Selection (A[], n, i)
{
   Left = 0; Right = n-1;
   p = Partition(A, n);
   if (i == p+1) return A[p];
   else if ( i<p+1 ) Selection(A[Left..p-1], (p-1)-Left+1, i);
         else Selection(A[p+1..Right], Right-(p+1)-1, i-p-1);
}
```

i번째로 작은 원소 찾기 최악 O(n^2), 평균 O(n)



i번째로 작은 원소 찾기 최악 $O(n^2)$, 평균 O(n)

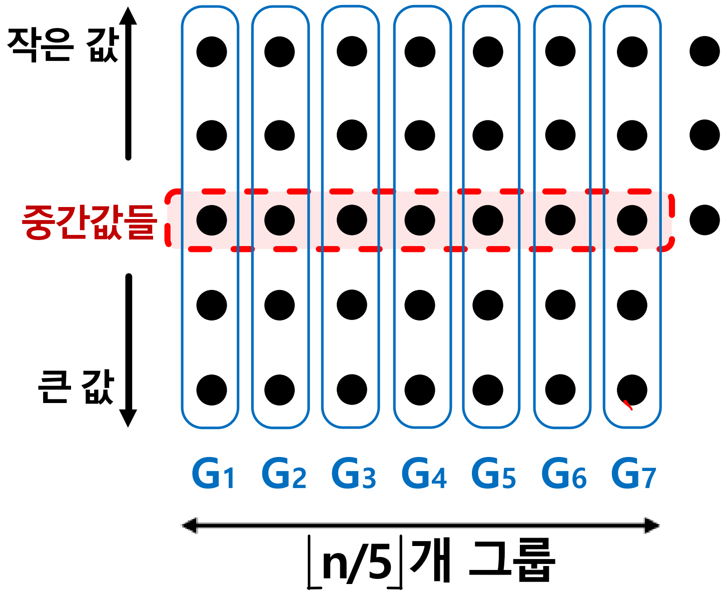
- **성능분석**
- 최악의 경우 = **퀵 정렬의 최악의 경우**
  - 분할 함수가 항상 하나의 부분배열만 생성하는 경우
  - 오름차순으로 정렬된 상태에서 i=n을 찾는 경우 → 분할 함수 호출할 때 마 피벗의 인덱스는 1씩 증가 → Partition()을 O(n)번 호출 ⇒ **O(n^2)**
- 해결책 → 항상 일정한 비율의 두 부분배열로 분할
  → 최악의 경우에도 O(n)
- 평균적인 경우
- **O(n)**

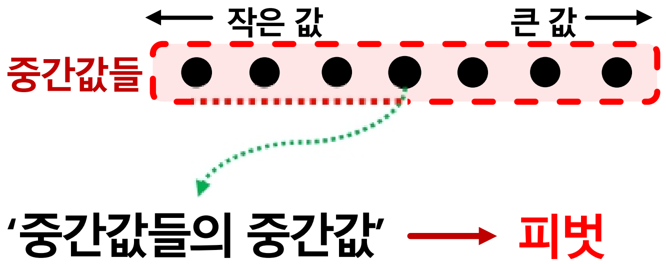
i번째로 작은 원소 찾기 최악 O(n), 평균 O(n)

- **개념과 원리**
- 특정한 성질을 만족하도록 피벗을 선택
  - 항상 일정한 비율의 두 부분배열로 분할
- **피벗 선택 방법**
- 1) 크기 n인 배열의 원소를 5개씩 묶어 ⌊n/5⌋개의 그룹을 형성
  - 5의 배수가 되지 않아 그룹을 형성하지 못한 채 남는 원소는 그대로 남겨 둠
- 2) 각 그룹에 대해서 중간 값을 찾음
- 3) ⌊n/5⌋개의 중간 값들을 대상으로 다시 중간 값을 찾음

→ "중간 값들의 중간 값" ⇒ **"피벗"**

i번째로 작은 원소 찾기 최악 O(n), 평균 O(n)





i번째로 작은 원소 찾기 최악 O(n), 평균 O(n)

A[]={ 9, 6, 35, 39, 15, 24, 70, 95, 50, 1, 97, 84, 77, 28, 10, 22, 27, 11, 31, 62,
54, 81, 5, 34, 4, 89, 60, 29, 2, 75, 18, 36, 80, 7, 53, 25, 66, 43 }

G1	G2	G3	G4	G5	G6	G7	
9	24	97	22	54	89	18	25
6	70	84	27	81	60	36	66
35	95	77	11	5	29	80	43
39	50	28	31	34	2	7	
15	1	10	62	4	75	53	

정렬

G1	G2	G3	G4	G5	G6	G7	
6	1	10	11	4	2	7	25
9	24	28	22	5	29	18	66
15	50	77	27	34	60	36	43
35	70	84	31	54	75	53	
39	95	97	62	81	89	80	

중간값들

15	27	34	36	50	60	77

중간값들의 중간값 → 피벗

i번째로 작은 원소 찾기 최악 O(n), 평균 O(n)

- 부분배열로의 분할 비율

	G1	G4	G5	G7	G2	G6	G3	
작은 값	6	11	4	7	1	2	10	25
	9	22	5	18	24	29	28	66
	15	27	34	36	50	60	77	43
	35	31	54	53	70	75	84	
큰 값	39	62	81	80	95	89	97	

S_1, S_2

9	24	97	22	54	89	18	25
6	70	84	27	81	60	36	66
35	95	77	11	5	29	80	43
39	50	28	31	34	2	7	
15	1	10	62	4	75	53	
G1	G2	G3	G4	G5	G6	G7	정렬

	6	1	10	11	4	2	7	25
	9	24	28	22	5	29	18	66
중간값들	15	50	77	27	34	60	36	43
	35	70	84	31	54	75	53	
	39	95	97	62	81	89	80	
	G1	G2	G3	G4	G5	G6	G7	

15	27	34	36	50	60	77

중간값들의 중간값 → 피벗

S_1과 S_2에 각각 속하는 원소의 개수 = $3 \times \lceil \lfloor n/5 \rfloor / 2 \rceil$

분할할 때마다 탐색 대상에서 제외되는 데이터 개수

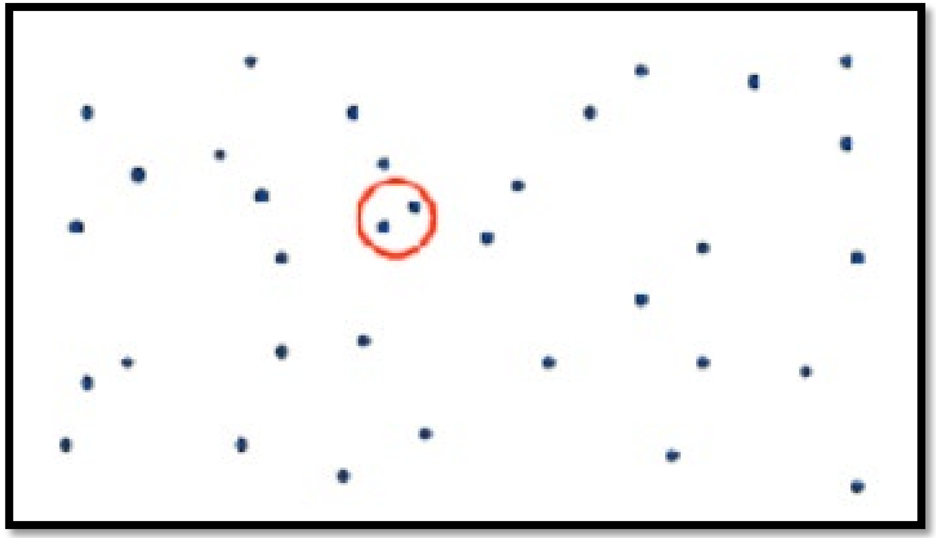
i번째로 작은 원소 찾기 최악 O(n), 평균 O(n)

```
int Selection_n (A[], n, i) {
  [단계1] if ( n<=5 ) 배열 A에서 i번째 원소를 찾아서 반환
          else [단계2]~[단계6]을 진행
  [단계2] A의 원소를 5개씩 묶어서 ⌊n/5⌋개의 그룹을 생성
  [단계3] 각 그룹의 중간값을 구하고, 이들을 모아 배열 M을 구성
  [단계4] 중간값들의 중간값을 계산하기 위해서 선택 함수를 순환 호출
          p = Selection_n ( M, ⌊n/5⌋, ⌈⌊n/5⌋/2⌉)
  [단계5] p를 피벗으로 사용하여 A를 분할 (피벗의 인덱스 = j라고 가정)
  [단계6] if ( i == j+1 ) return A[j]
          else if ( i <j+1 ) Selection_n(A[0..j-1], j, i)를 순환 호출
              else Selection_n(A[j+1..n-1], n-j-1,i-j-1)를 순환 호출
}
```

최근접 점의 쌍 찾기

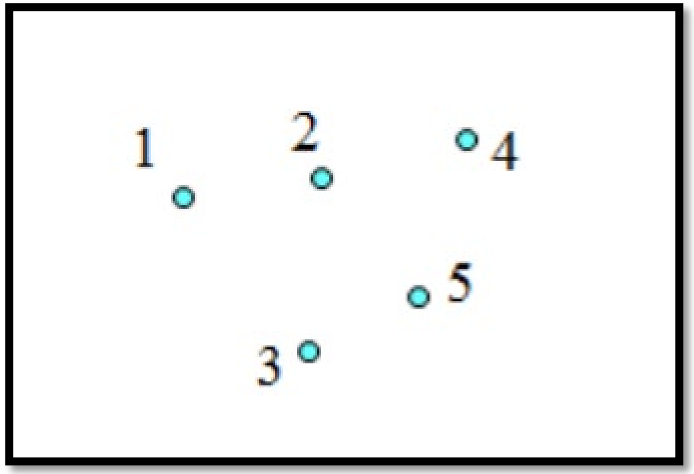
[최근접 점의 쌍 찾기]의 개념과 원리

- 최근접 점의 쌍 (Closest Pair)
- 2차원 평면상의 n개의 점이 입력으로 주어질 때, 거리가 가장 가까운 한 쌍의 점을 찾는 문제



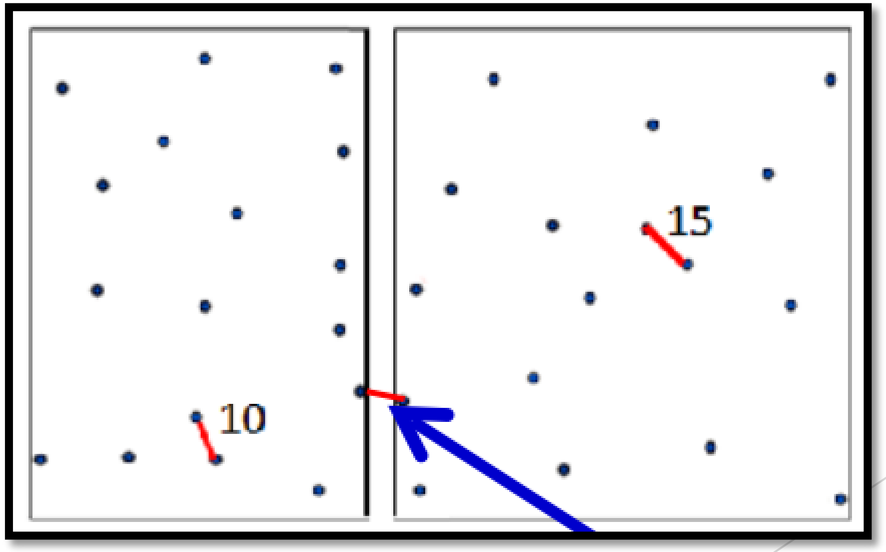
[최근접 점의 쌍 찾기]의 개념과 원리

- 가장 간단한 방법 – Brute-Force Method
- 모든 점에 대하여 각각의 두 점 사이의 거리를 계산하여 가장 가까운 점의 쌍을 찾음
- 거리 계산을 위한 쌍의 개수 구하기 -> 수학의 조합(**nCr**)개념
- 시간 복잡도는 O(n^2)



O(n^2)보다 효율적인 분할 정복 기법 사용

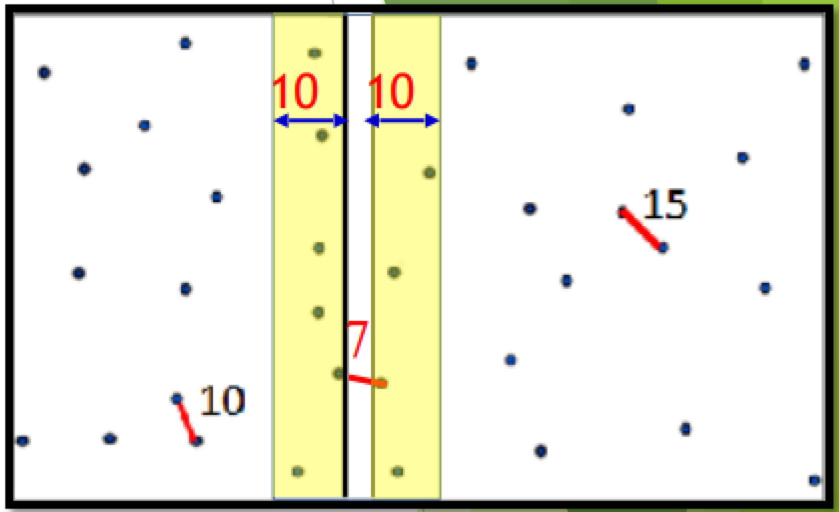
- N개의 점을 ½로 분할하여 각각의 부분 문제에서 최근접 점의 쌍을 찾고, 2개의 부분해 중에서 짧은 거리를 가진 점의 쌍을 일단 찾음
- 그리고 2개의 부분해를 취합할 경우, 반드시 중간영역 안에 있는 점들 확인 필요



O(n^2)보다 효율적인 분할 정복 기법 사용

- 중간 영역 내 점들을 찾는 방법

- d = min{왼쪽 부분의 최근접 점의 쌍 사이의 거리, 오른쪽 부분의 최근접 점의 쌍 사이의 거리}
- X좌표 오름차순 정렬, Y좌표 생략



왼쪽 부분 문제의 가장오른쪽 점

오른쪽 부분 문제의 가장 왼쪽 점

(1,-)	(13,-)	(17,-)	(25,-)	(26,-)	(28,-)	(30,-)	(37,-)	(45,-)	(56,-)

O(n^2)보다 효율적인 분할 정복 기법 사용

- d= 10이라면, 점 (17,-), (25,-), (26,-), (28,-), (30,-), (37,-) 이 중간 영역에 속함

d = 10

0	1	2	3	4	5	6	7	8	9
(1,-)	(13,-)	(17,-)	(25,-)	(26,-)	(28,-)	(30,-)	(37,-)	(45,-)	(56,-)

26-d = 16　　28+d = 38

중간영역 점들 사이의 거리가 d=10 보다 작은지 검사함

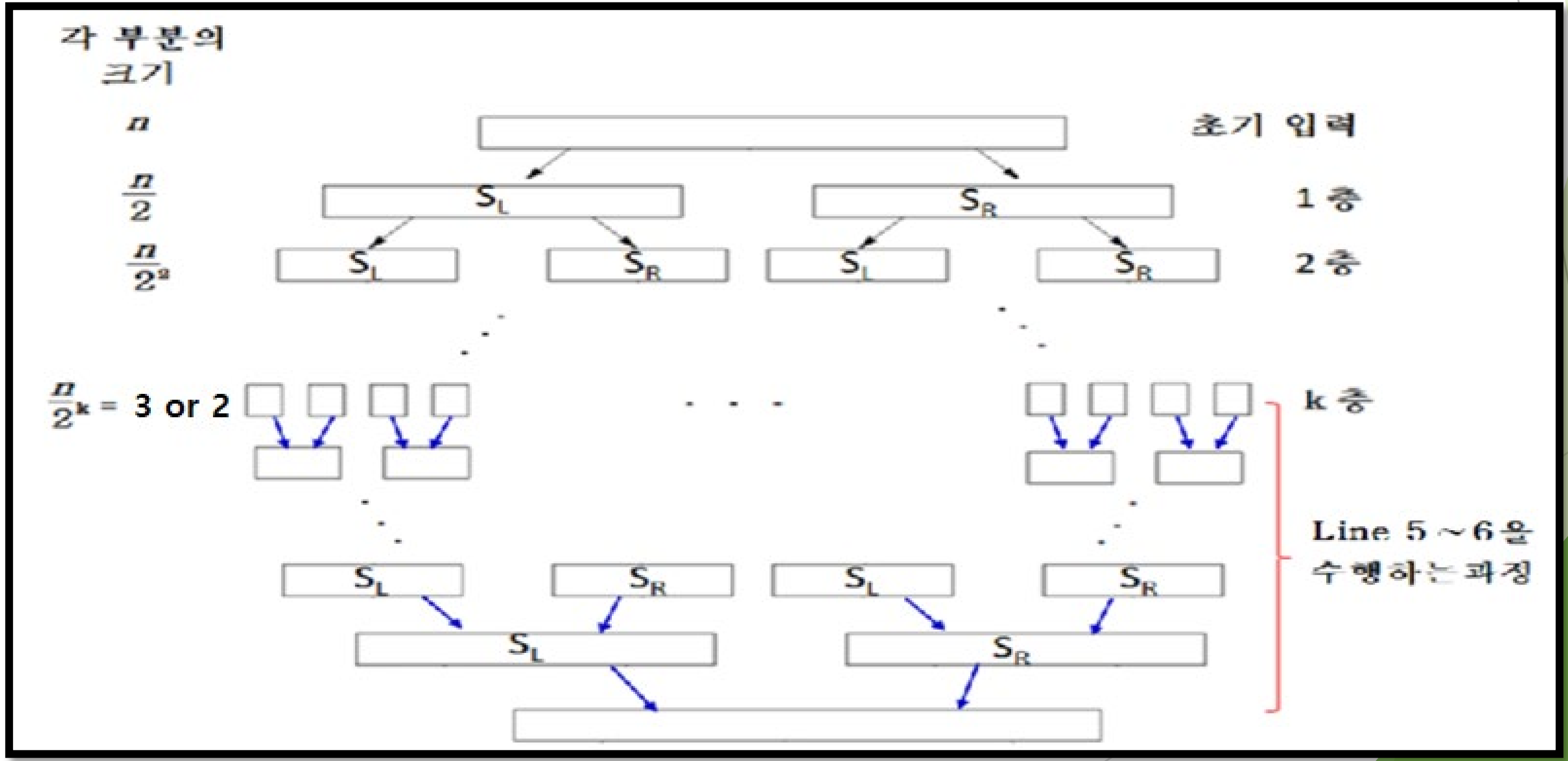
최근접 점의 쌍 분할 정복 알고리즘

ClosestPair(S)
입력: x-좌표의 오름차순으로 정렬된 배열 S내의 i개의 점 (단, 각 점은 (x,y)로 표현)
출력: S에 있는 점들 중 최근접 점의 쌍의 거리

1. if (i ≤ 3) return (2 또는 3개의 점들 사이의 최근접 쌍)
2. 정렬된 S를 같은 크기의 SL과 SR로 분할한다. |S|가 홀수이면, |SL| = |SR|+1이 되도록 분할.
3. CPL = ClosestPair(SL) // CPL은 SL에서의 최근접 점의 쌍
4. CPR = ClosestPair(SR) // CPR은 SR에서의 최근접 점의 쌍
5. d = min{dist(CPL), dist(CPR)}일 때, 중간 영역에 속하는 점들 중에서 최근접 점의 쌍을 찾아서 이를 CPC라고 가정. 단, dist()는 두 점 사이의 거리
6. return (CPL, CPC, CPR 중에서 거리가 가장 짧은 쌍)

최근접 점의 쌍 찾기

- 성능분석
- O($nlog^2n$)
- Y축을 미리 정렬해두고 이를 참조하여 사용할 경우에는 O($nlogn$)



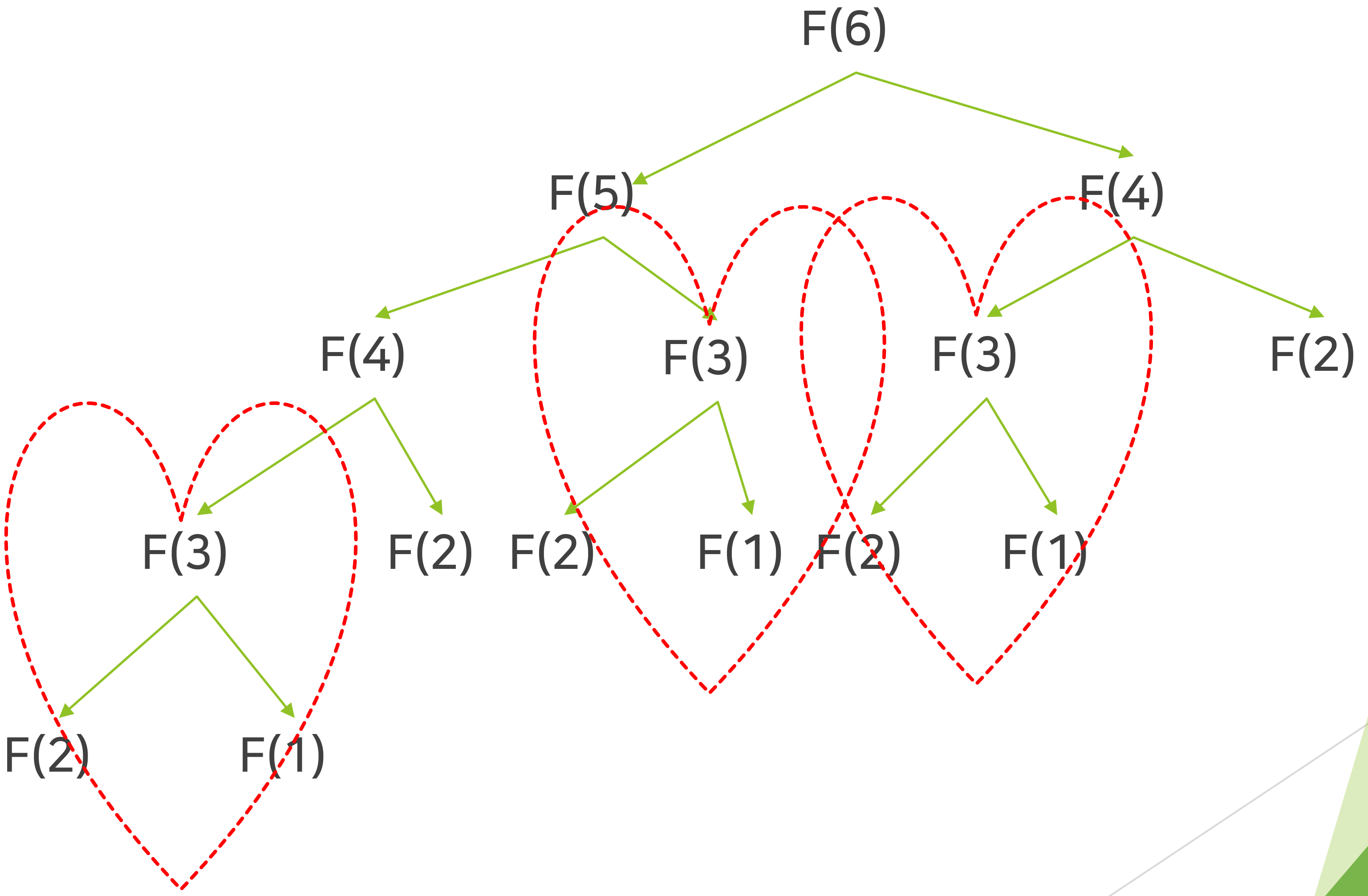
분할 정복을 적용하는 데 있어 주의할 점

분할 정복을 적용하는 데 있어 주의할 점

- 부적절한 경우
  - 입력이 분할될 때마다 분할된 부분문제의 입력 크기의 합이 분할되기 전의 입력 크기보다 매우 커지는 경우

- 예) 피보나치 수열
- N번째 피보나치의 수 구하기 - F(n) = F (n-1) + F(n-2)로 정의
- 입력은 1개이지만, 사실상 n의 값 자체가 입력의 크기

분할 정복을 적용하는 데 있어 주의할 점



분할 정복을 적용하는 데 있어 주의할 점

- **피보나치 수 계산을 위한 O(n) 시간 알고리즘**

```
FibNumber(n)
F[0]=0
F[1]=1
For i=2 to n
    F[i] = F[i-1] +F[i-2]
```

- 취합(정복)과정도 고려사항
  - 입력을 분할만 한다고 해서 효율적인 알고리즘은 아님

과제 안내

선택 문제

Microsoft Visual Studio 디버그 콘솔

```
Sorted array:
11 12 22 25 64
```

최근접 점의 쌍 찾기

Microsoft Visual Studio 디버그 콘솔

```
The closest distance is 1.41421
between (0,0) and (1,1)
```

4주차 과제

- 선택문제, 최근접 점의 쌍 찾기 구현하기
  - e-Class 업로드

- 양식 (한글, 워드,PDF -> 자유)

- 파일명 (이름_학번_전공)
  - 예) 최희석_2014182009_게임공학

- 질의 응답은 e-Class 질의응답 게시판에 남겨 주시길 바랍니다.