

제3장 분할 정복 알고리즘 (II)

과 목 명 정 보 처 리 알 고 리 즘

담당교수 김 성 훈

경북대학교 과학기술대학 소프트웨어학과

이 장에서 배울 내용

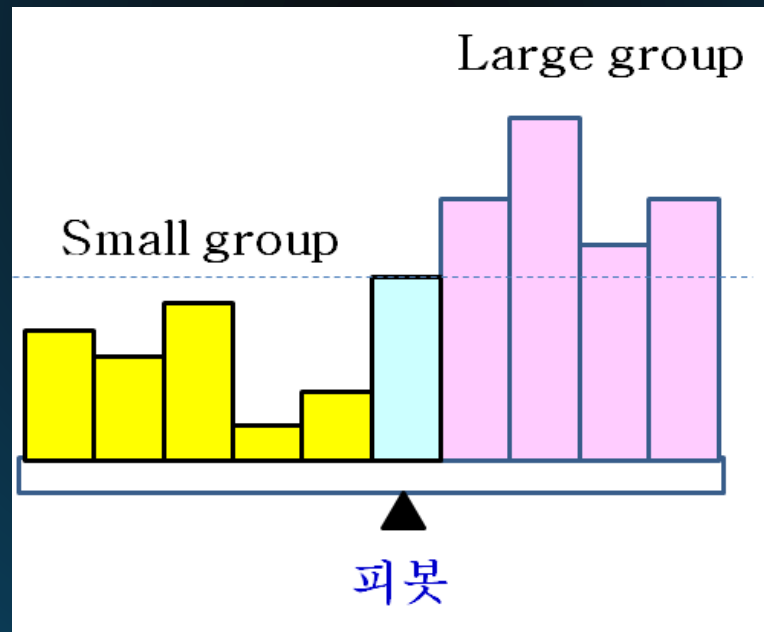
1. 분할정복 개념
2. 합병정렬
3. 퀵정렬
4. 선택문제
5. 최근접 점의 쌍 찾기
6. 분할정복을 적용하는데 있어서 주의할 점

3.3 선택 문제

- 선택(Selection) 문제는 n 개의 숫자 중에서 k 번째로 작은 숫자를 찾는 문제
 - 선택 문제를 해결하기 위한 간단한 방법은,
 방법1: 최소 숫자를 k 번 찾는다. 단, 최소 숫자를 찾은 뒤에는
 입력에서 최소 숫자를 제거한다.
 방법2: 숫자들을 정렬한 후, k 번째 숫자를 찾는다.
 - 위의 알고리즘들은 각각 최악의 경우,
 - 방법1은 $O(kn)$,
 - 방법2는 $O(n\log n)$ 의 수행 시간이 걸린다.
- 더 나은 방법이 있을까? $O(n)$ 의 수행시간

아이디어

- **이진탐색**은 정렬된 입력의 중간에 있는 숫자와 찾고자 하는 숫자를 비교함으로써, 입력을 1/2로 나눈 두 부분 중에서 한 부분만을 검색
- 선택 문제는 입력이 정렬되어 있지 않으므로, 입력 숫자들 중에서 (퀵 정렬에서와 같이) **피벗**을 선택하여 아래와 같이 분할



- Small group은 피봇보다 작은 숫자의 그룹이고, Large group은 피봇보다 큰 숫자의 그룹이다.
- 분할 후, 각 그룹의 크기를 알아야, k 번째 작은 숫자가 어느 그룹에 있는지를 알 수 있고, 그 다음엔 그 그룹에서 몇 번째로 작은 숫자를 찾아야 하는지를 알 수 있다.

- Small group에 k 번째 작은 숫자가 속한 경우: k 번째 작은 숫자를 Small group에서 찾는다.
- Large group에 k 번째 작은 숫자가 있는 경우: $(k - |\text{Small group}| - 1)$ 번째로 작은 숫자를 Large group에서 찾아야 한다. 여기서 $|\text{Small group}|$ 은 Small group에 있는 숫자의 개수이고, 1은 피봇에 해당된다.

선택 문제 알고리즘

Selection(A, left, right, k)

입력: A[left]~A[right]와 k, 단, $1 \leq k \leq |A|$, $|A| = \text{right} - \text{left} + 1$

출력: A[left]~A[right]에서 k 번째 작은 원소

1. 피벗을 A[left]~A[right]에서 랜덤하게 선택하고,
피벗과 A[left]의 자리를 바꾼 후,
피벗과 배열의 각 원소를 비교하여
피벗보다 작은 숫자는 A[left]~A[p-1]로 옮기고,
피벗보다 큰 숫자는 A[p+1]~A[right]로 옮기며,
피벗은 A[p]에 놓는다.
2. $S = (p-1) - \text{left} + 1$ // S = Small group의 크기
3. if ($k \leq S$) Selection(A, left, p-1, k) // Small group에서 찾기
4. else if ($k = S + 1$) return A[p] // 피벗 = k번째 작은 숫자
5. else Selection(A, p+1, right, k-S-1) // large group에서 찾기

알고리즘 설명

- Line 1은 피벗을 랜덤하게 선택하는 것을 제외하고는 퀵 정렬 알고리즘의 line 2와 동일
- Line 2에서는 입력을 두 그룹으로 분할된 후, $A[p]$ 가 피벗이 있는 것이기 때문에 Small group의 크기를 알 수 있다. 즉, Small group의 가장 오른쪽 원소의 인덱스가 $(p-1)$ 이므로, Small group의 크기 $S = (p-1) - \text{left} + 1$ 이다.
- Line 3은 k 번째 작은 수가 Small group에 속한 경우이므로 $\text{Selection}(A, \text{left}, p-1, k)$ 호출
- Line 4는 k 번째 작은 수가 피벗인 $A[p]$ 와 같은 경우이므로 해 발견

- Line 5에서는 k 번째 작은 수가 Large group에 속한 경우이므로 Selection(A, p+1, right, k-S-1) 호출
- 이때에는 (k-S-1) 번째 작은 수를 Large group에서 찾아야; 왜냐하면 피벗이 k번째 작은 수보다 작고, S는 Small group의 크기이므로

Selection(A, left, right, k)

입력: A[left]~A[right]와 k, 단, $1 \leq k \leq |A|$, $|A| = \text{right} - \text{left} + 1$

출력: A[left]~A[right]에서 k 번째 작은 원소

1. 피벗을 A[left]~A[right]에서 랜덤하게 선택하고,
 피벗과 A[left]의 자리를 바꾼 후,
 피벗과 배열의 각 원소를 비교하여
 피벗보다 작은 숫자는 A[left]~A[p-1]로 옮기고,
 피벗보다 큰 숫자는 A[p+1]~A[right]로 옮기며,
 피벗은 A[p]에 놓는다.
2. $S = (p-1) - \text{left} + 1$ // S = Small group의 크기
3. if ($k \leq S$) Selection(A, left, p-1, k) // Small group에서 찾기
4. else if ($k = S + 1$) return A[p] // 피벗 = k번째 작은 숫자
5. else Selection(A, p+1, right, k-S-1) // large group에서 찾기

Selection 알고리즘의 수행 과정

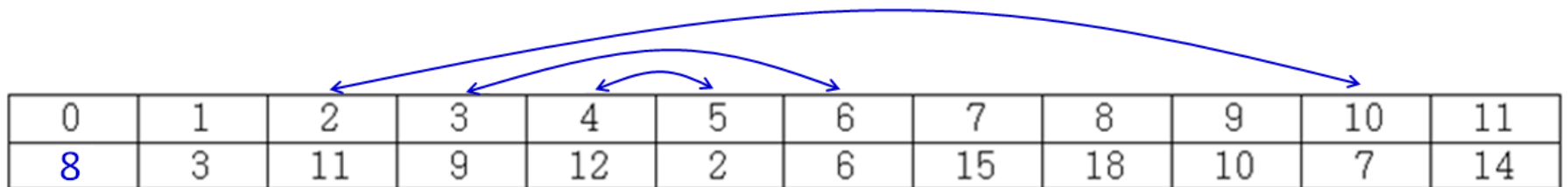
- $k=7$

0	1	2	3	4	5	6	7	8	9	10	11
6	3	11	9	12	2	8	15	18	10	7	14

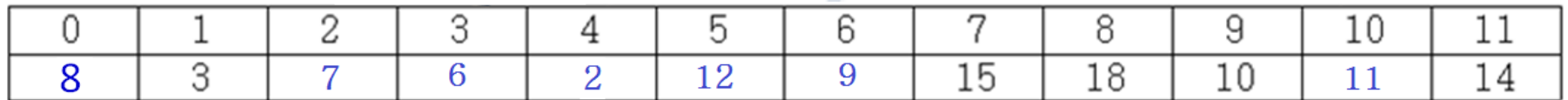
- 최초로 Selection(A,0,11,7) 호출
- $k=7$, left=0, right=11
- Line 1에서 피벗 $A[6]=8$ 이라면, 피벗이 A[0]에 오도록 A[0]과 A[6]을 서로 바꾼다.

0	1	2	3	4	5	6	7	8	9	10	11
8	3	11	9	12	2	6	15	18	10	7	14

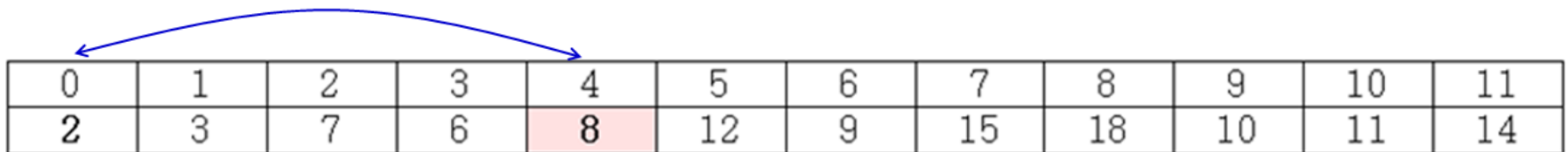
- 아래와 같이 차례로 원소들이 자리를 서로 바꾼다.



0	1	2	3	4	5	6	7	8	9	10	11
8	3	11	9	12	2	6	15	18	10	7	14



0	1	2	3	4	5	6	7	8	9	10	11
8	3	7	6	2	12	9	15	18	10	11	14



0	1	2	3	4	5	6	7	8	9	10	11
2	3	7	6	8	12	9	15	18	10	11	14

- Line 2: Small group의 크기 계산 (즉, $S = (p-1) - \text{left} + 1 = (4-1) - 0 + 1 = 4$)
- 따라서 Small group에는 7 번째 작은 수가 없고, line 4의 if-조건 $(7=S+1) = (7=4+1) = (7=5)$ 가 '거짓'이 되어, line 5에서 $\text{Selection}(A, p+1, \text{right}, k-S-1) = \text{Selection}(A, 4+1, 11, 7-4-1) = \text{Selection}(A, 5, 11, 2)$ 호출
- 즉, Large group에서 2 번째로 작은 수를 찾는다.


- Selection(A,5,11,2) 호출
- k=2, left=5, right=11

5	6	7	8	9	10	11
12	9	15	18	10	11	14

- Line 1에서 피벗 A[11]=14라면, 피벗이 A[5]에 오도록 A[5]와 A[11]을 서로 바꾼다.


5	6	7	8	9	10	11
14	9	15	18	10	11	12

- 그 다음에는 아래와 같이 차례로 원소들이 자리를 서로 바꾼다.



5	6	7	8	9	10	11
14	9	15	18	10	11	12

5	6	7	8	9	10	11
14	9	12	11	10	18	15



5	6	7	8	9	10	11
10	9	12	11	14	18	15

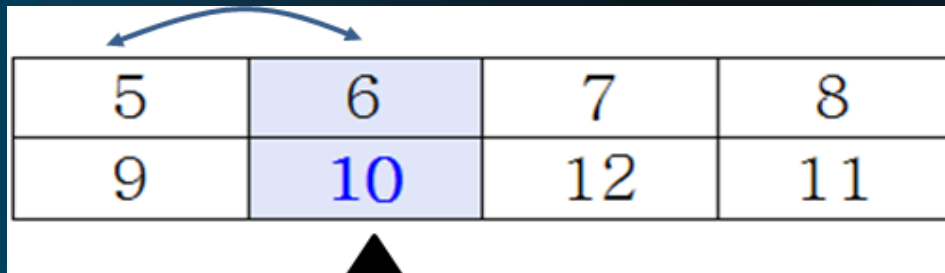


- Line 2: Small group의 크기 계산
(즉, $S = (p-1)-\text{left}+1 = (9-1)-5+1 = 4$)
- 따라서 $k=2$ 번째 작은 수를 찾아야 하므로
line 3의 if-조건인 $(k \leq S) = (2 \leq 4)$ 가 '참'이 되어
 $\text{Selection}(A, \text{left}, p-1, k) = \text{Selection}(A, 5, 9-1, 2)$
 $= \text{Selection}(A, 5, 8, 2)$ 호출
- 즉, Small group에서 2 번째로 작은 수를 찾는다.

- Selection(A, 5, 8, 2) 호출
- k=2, left=5, right=8

5	6	7	8
10	9	12	11

- Line 1에서 피벗 $A[5]=10$ 이라면, 원소 간 자리바꿈 없이 아래와 같이 된다.



5	6	7	8
9	10	12	11

- Line 2: Small group의 크기 계산 (즉, $S = (p-1)-\text{left}+1 = (6-1)-5+1 = 1$)
- 따라서 $k=2$ 번째 작은 수를 찾아야 하고, line 3의 if-조건 ($k \leq S$) = $(2 \leq 1)$ 이 '거짓'이 되지만, line 4의 if-조건 $(2 = S+1) = (2 = 1+1) = (2 = 2)$ 가 '참'이 되므로, 최종적으로 $A[6]=10$ 을 $k=7$ 번째 작은 수로서 해를 리턴한다.

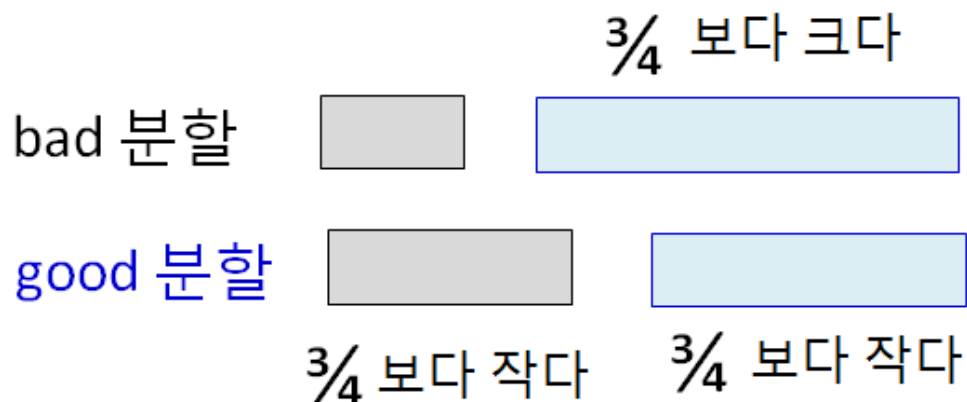
Selection 알고리즘의 무작위성

- Selection 알고리즘은 분할 정복 알고리즘이기도 하지만 **랜덤 (random) 알고리즘**이기도 하다.
- 왜냐하면 선택 알고리즘의 line 1에서 피벗을 랜덤하게 정하기 때문이다.
- 만일 피벗이 입력 리스트를 너무 한 쪽으로 치우치게 분할하면, 즉, $|Small\ group| \ll |Large\ group|$ 또는 $|Small\ group| \gg |Large\ group|$ 일 때에는 알고리즘의 수행 시간이 길어진다.
- 선택 알고리즘이 호출될 때마다 line 1에서 입력을 한쪽으로 치우치게 분할될 확률은 마치 **동전을 던질 때 한쪽 면이 나오는 확률과 같다.**



good/bad 분할

- 분할된 두 그룹 중의 하나의 크기가 입력 크기의 $\frac{3}{4}$ 과 같거나 그 보다 크게 분할하면 **나쁜 (bad) 분할**이라고 정의해보자.
← 최선은 $\frac{1}{2}$ 씩 나누는 것이고, 최악은 0과 모두로 나누는 것이므로.
- **좋은 (good) 분할**은 그 반대의 경우이다.

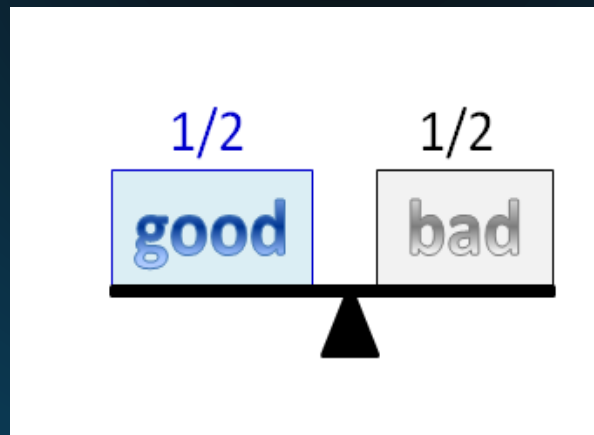


- 아래의 예를 살펴보면 good 분할이 되는 피봇을 선택할 확률과 bad 분할이 되는 피봇을 선택할 확률이 각각 $1/2$ 로 동일함을 확인할 수 있다.

다음과 같이 16개의 숫자가 있다고 가정하자.

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

- 16개 숫자들 중에서 5~12 중의 하나가 피봇이 되면 good 분할.
- 1~4 또는 13~16 중 하나가 피봇으로 정해지면 bad 분할.



시간복잡도

- 피봇을 랜덤하게 정했을 때 **good 분할이 될 확률이 $1/2$** 이므로 평균 2회 연속해서 랜덤하게 피봇을 정하면 good 분할을 할 수 있다.
- 즉, 매 2회 호출마다 good 분할이 되므로, good 분할만 연속하여 이루어졌을 때만의 시간복잡도를 구하여, 그 값에 2를 곱하면 평균 경우 시간복잡도를 얻을 수 있다.

시간복잡도(2)

1 번째 good 분할



$\frac{3}{4}$ 보다 작다



$\frac{3}{4}$ 보다 작다

2 번째 good 분할



$(\frac{3}{4})^2$ 보다 작다

3 번째 good 분할



$(\frac{3}{4})^3$ 보다 작다

...

...

i 번째 good 분할



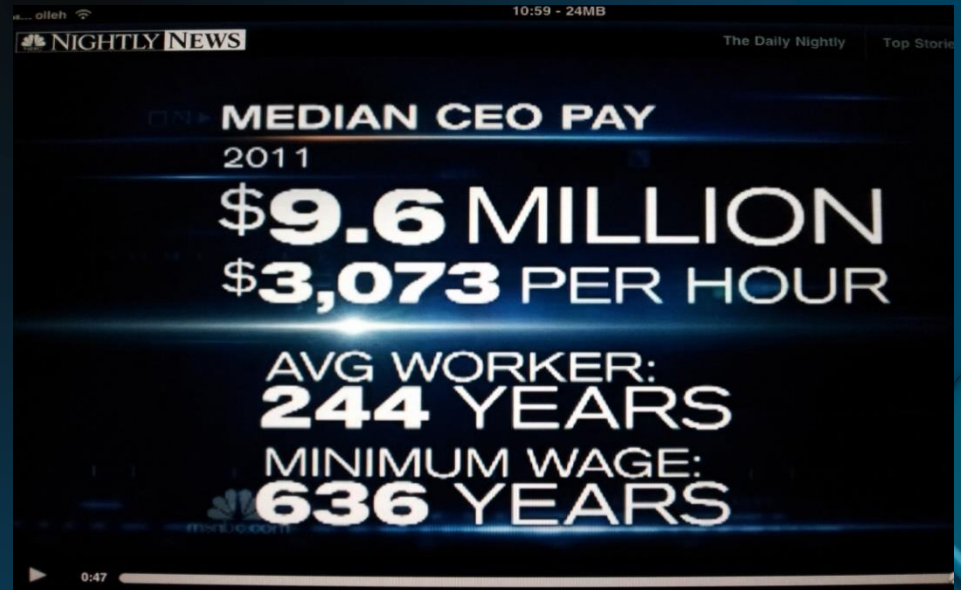
$(\frac{3}{4})^i$ 보다 작다

시간복잡도(3)

- 리스트 크기가 n 에서부터 $3/4$ 배로 연속적으로 감소되고, 리스트 크기가 1일 때에는 더 이상 분할할 수 없게 된다. 그러므로 Selection 알고리즘의 평균 경우 시간복잡도는 다음과 같다.
- $O[n + 3/4n + (3/4)^2n + (3/4)^3n + \dots + (3/4)^{i-1}n + (3/4)^in] = O(n)$
- Selection 알고리즘의 평균 경우 시간복잡도는 $2 \times O(n) = O(n)$

선택알고리즘의 응용

- 선택알고리즘은 데이터분석을 위한 **중앙값 (median)**을 찾는데 활용된다.
 - 데이터 분석에서 평균값도 유용하지만, 중앙값이 더 설득력 있는 데이터 분석을 제공하기도 한다. 예를 들어, 대부분의 데이터가 1이고, 오직 1개의 숫자가 매우 큰 숫자 (노이즈 (noise), 잘못 측정된 데이터)이면, 평균값은 매우 왜곡된 분석이 된다.
 - 실제로 대학 졸업 후 바로 취업한 직장인의 연간 소득을 분석할 때에 평균값보다 중앙값이 더 의미 있는 분석 자료가 된다.

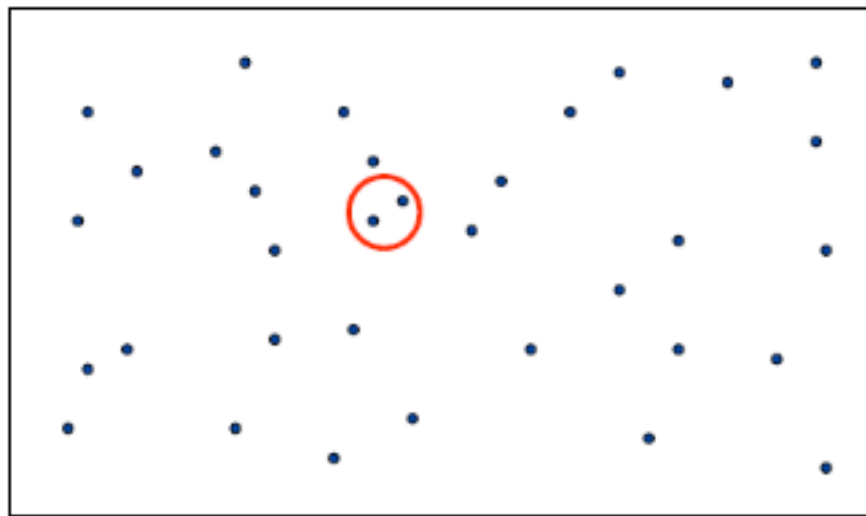


BREAK TIME



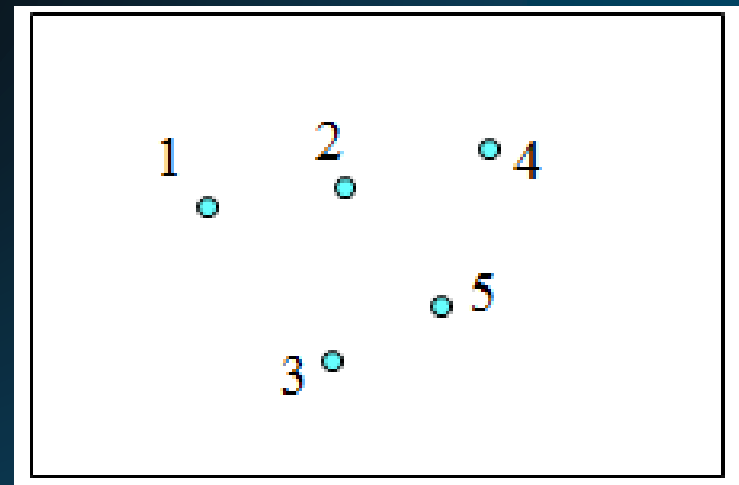
3.4 최근접 점의 쌍 찾기

- 최근접 점의 쌍 (Closest Pair) 문제는 2차원 평면상의 n 개의 점이 입력으로 주어질 때, 거리가 가장 가까운 한 쌍의 점을 찾는 문제이다.



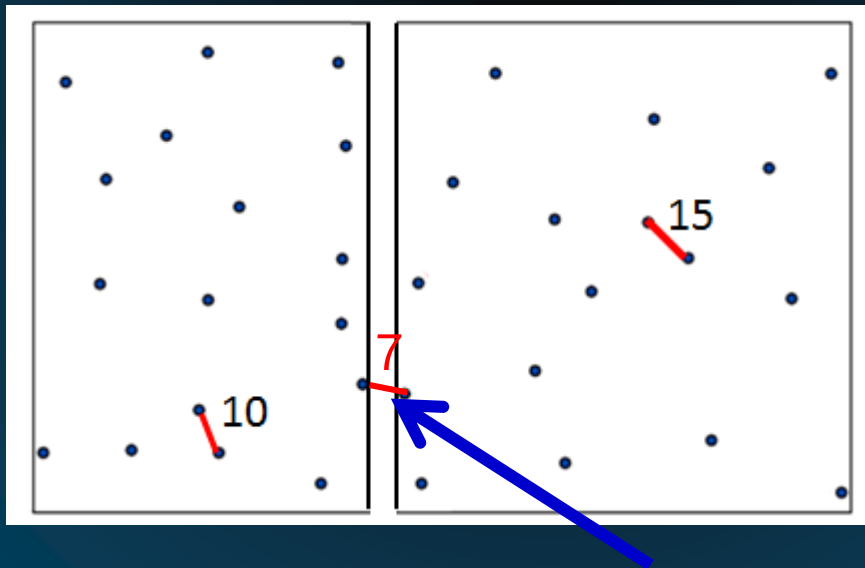
간단한 방법

- 간단한 방법:
 - 모든 점에 대하여 각각의 두 점 사이의 거리를 계산하여 가장 가까운 점의 쌍을 찾다.
 - 예를 들어, 5개의 점이 아래의 [그림]처럼 주어진다면, 1-2, 1-3, 1-4, 1-5, 2-3, 2-4, 2-5, 3-4, 3-5, 4-5 사이의 거리를 각각 계산하여 그 중에 최소 거리를 가진 쌍이 최근접 점의 쌍이 되는 것이다. 그러면 비교해야 할 쌍은 몇 개인가?
 - ${}_nC_2 = n(n-1)/2$
 - $n=5$ 이면, $5(5-1)/2 = 10$
 - $n(n-1)/2 = O(n^2)$
 - 한 쌍의 거리 계산은 $O(1)$ 시간
 - 시간복잡도는 $O(n^2) \times O(1) = O(n^2)$



분할정복 아이디어

- $O(n^2)$ 보다 효율적인 분할 정복 이용:
 - n 개의 점을 $1/2$ 로 분할하여 각각의 부분 문제에서 최근접 점의 쌍을 찾고, 2개의 부분해 중에서 짧은 거리를 가진 점의 쌍을 일단 찾는다.
 - 그리고 2개의 부분해를 취합할 때에는 반드시 다음과 같은 경우를 고려해야 한다.

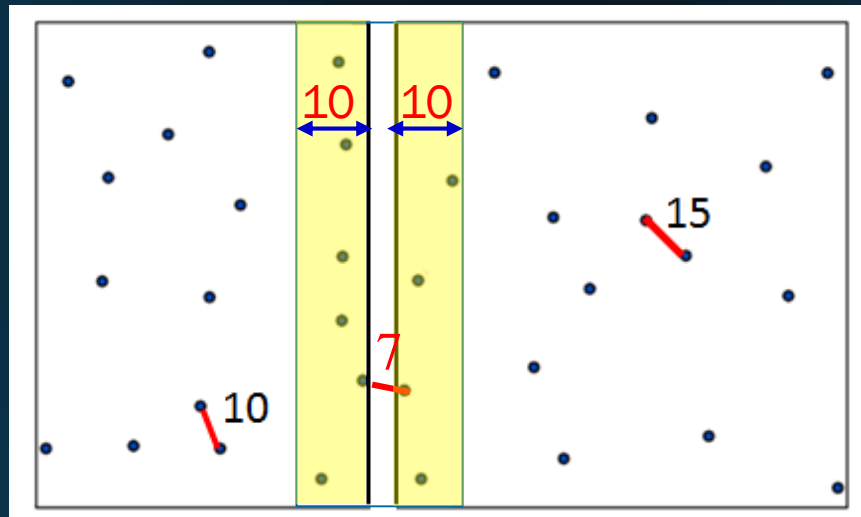


예시 분석

- 왼쪽 부분 문제의 최근접 쌍의 거리가 10이고, 오른쪽 부분 문제의 최근접 쌍의 거리가 15이다,
- 왼쪽 부분 문제의 가장 오른쪽 점과 오른쪽 부분 문제의 가장 왼쪽 점 사이의 거리가 7이다.
- 따라서 2개의 부분 문제의 해를 취합할 때 단순히 10과 15 중에서 짧은 거리인 10을 해라고 할 수 없는 것이다.

예시 분석(2)

- 그러므로 아래의 그림에서와 같이 각각 거리가 10이내의 중간 영역 안에 있는 점들 중에 최근접 점의 쌍이 있는지도 확인해보아야 한다.

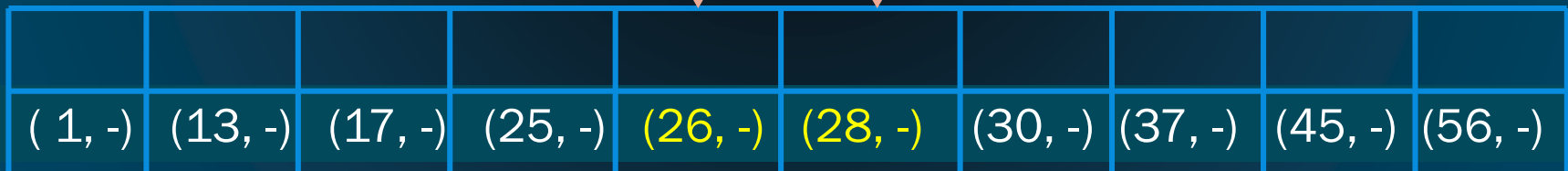


예시 분석(3)

- 배열에 점의 좌표가 저장되어 있을 때, 중간 영역에 있는 점들을 찾는 방법
- 단, $d = \min\{\text{왼쪽 부분의 최근접 점의 쌍 사이의 거리}, \text{오른쪽 부분의 최근접 점의 쌍 사이의 거리}\}$ 이다.
- 아래의 배열에는 점들이 x-좌표의 오름차순으로 정렬되어 있고, 각 점의 y-좌표는 생략되었다.

왼쪽 부분 문제의
가장오른쪽 점

오른쪽 부분 문제의
가장왼쪽 점



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

예시 분석(4)

- 중간 영역에 속한 점들은 왼쪽 부분 문제의 가장 오른쪽 점 (왼쪽 중간 점)의 x-좌표에서 d 를 뺀 값과 오른쪽 부분 문제의 가장 왼쪽 점 (오른쪽 중간점)의 x-좌표에 d 를 더한 값 사이의 x-좌표 값을 가진 점들이다.
- $d=10$ 이라면, 점 $(25,-)$, $(26,-)$, $(28,-)$, $(30,-)$, $(37,-)$ 이 중간 영역에 속한다.

$$d = 10$$

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

$$26-d = 16$$

$$28+d = 38$$

최근접 점의 쌍 분할 정복 알고리즘

ClosestPair(S)

입력: x-좌표의 오름차순으로 정렬된 배열 S에는 i 개의 점 (단, 각 점은 (x,y) 로 표현된다.)

출력: S에 있는 점들 중 최근접 점의 쌍의 거리

1. if ($i \leq 3$) return (2 또는 3개의 점들 사이의 최근접 쌍)
2. 정렬된 S를 같은 크기의 S_L 과 S_R 로 분할한다. $|S|$ 가 홀수이면, $|S_L| = |S_R| + 1$ 이 되도록 분할한다.
3. $CP_L = \text{ClosestPair}(S_L)$ // CP_L 은 S_L 에서의 최근접 점의 쌍
4. $CP_R = \text{ClosestPair}(S_R)$ // CP_R 은 S_R 에서의 최근접 점의 쌍
5. $d = \min\{\text{dist}(CP_L), \text{dist}(CP_R)\}$ 일 때, 중간 영역에 속하는 점들 중에서 최근접 점의 쌍을 찾아서 이를 CP_C 라고 하자. 단, $\text{dist}()$ 는 두 점 사이의 거리이다.
6. return (CP_L, CP_C, CP_R 중에서 거리가 가장 짧은 쌍)

알고리즘 설명

- Line 1에서는 S에 있는 점의 수가 3개 이하이면 더 이상 분할하지 않는다. S에 2개의 점이 있으면 S를 그대로 리턴하고, 3개의 점이 있으면 3개의 쌍에 대하여 최근접 점의 쌍을 리턴한다.
- Line 2에서는 x-좌표로 정렬된 S를 왼쪽과 오른쪽에 같은 개수의 점을 가지는 S_L 과 S_R 로 분할한다. 만일 S의 점의 수가 홀수이면 S_L 쪽에 1개 많게 분할한다.
- Line 3~4에서는 분할된 S_L 과 S_R 에 대해서 재귀적으로 최근접 점의 쌍을 찾아서 각각을 CP_L 과 CP_R 이라고 놓는다.

ClosestPair(S)

입력: x-좌표의 오름차순으로 정렬된 배열 S에는 i개의 점 (단, 각 점은 (x,y)로 표현된다.)

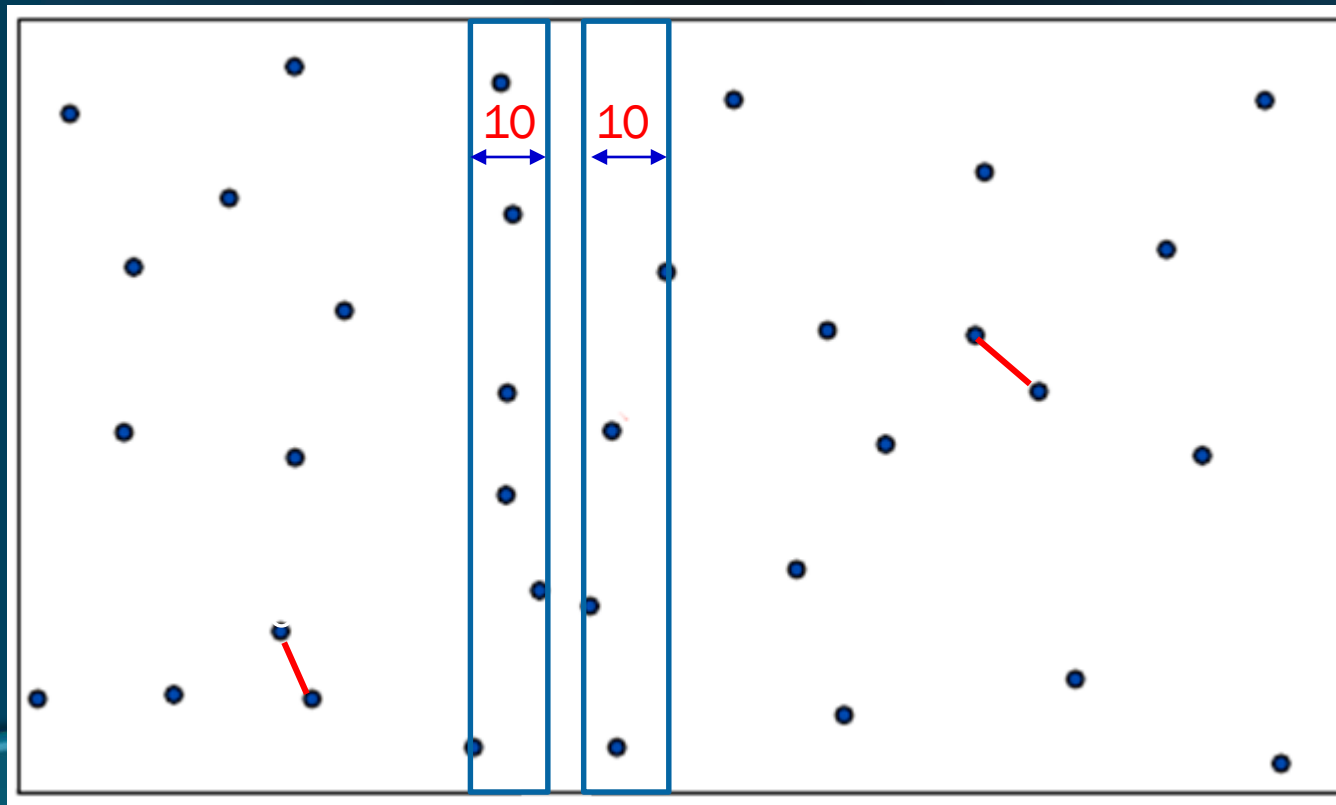
출력: S에 있는 점들 중 최근접 점의 쌍의 거리

1. if ($i \leq 3$) return (2 또는 3개의 점들 사이의 최근접 쌍)
2. 정렬된 S를 같은 크기의 S_L 과 S_R 로 분할한다. |S|가 홀수이면, $|S_L| = |S_R| + 1$ 이 되도록 분할한다.
3. $CP_L = \text{ClosestPair}(S_L)$ // CP_L 은 S_L 에서의 최근접 점의 쌍
4. $CP_R = \text{ClosestPair}(S_R)$ // CP_R 은 S_R 에서의 최근접 점의 쌍
5. $d = \min\{\text{dist}(CP_L), \text{dist}(CP_R)\}$ 일 때, 중간 영역에 속하는 점들 중에서 최근접 점의 쌍을 찾아서 이를 CP_C 라고 하자. 단, $\text{dist}()$ 는 두 점 사이의 거리이다.
6. return (CP_L, CP_C, CP_R 중에서 거리가 가장 짧은 쌍)

알고리즘 설명(2)

- Line 5는 d 를 이용하여 중간 영역에 속하는 점들을 찾고, 이 점들 중에서 최근접 점의 쌍을 찾아서 이를 CP_C 라고 놓는다.

$$d = \min \{CP_L, CP_R\} = \min \{10, 15\} = 10$$



알고리즘 설명(3)

- Line 6에서는 line 3~4에서 각각 찾은 최근접 점의 쌍 CP_L 과 CP_R 과 line 5에서 찾은 CP_C 중에서 가장 짧은 거리를 가진 쌍을 해로서 리턴한다.

ClosestPair(S)

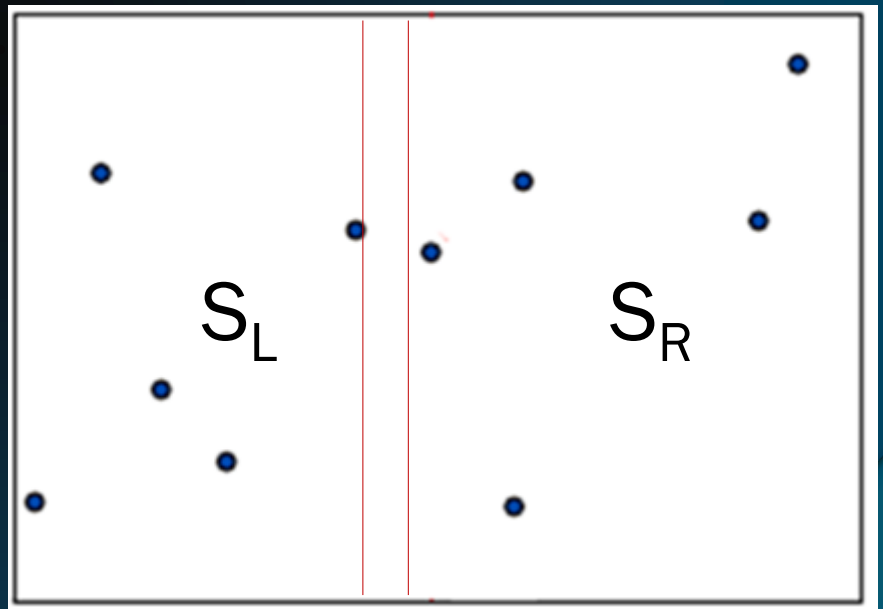
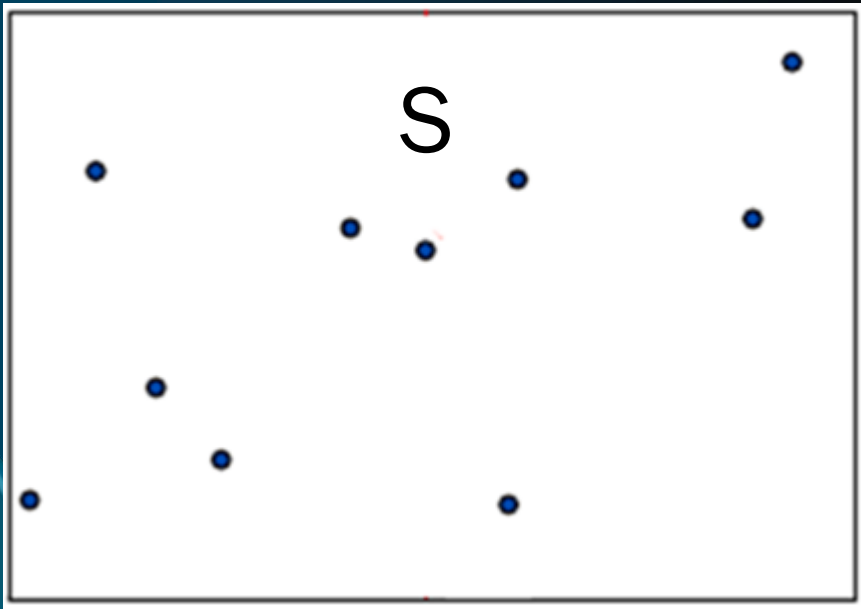
입력: x-좌표의 오름차순으로 정렬된 배열 S에는 i개의 점 (단, 각 점은 (x,y)로 표현된다.)

출력: S에 있는 점들 중 최근접 점의 쌍의 거리

1. if ($i \leq 3$) return (2 또는 3개의 점들 사이의 최근접 쌍)
2. 정렬된 S를 같은 크기의 S_L 과 S_R 로 분할한다. |S|가 홀수이면, $|S_L| = |S_R| + 1$ 이 되도록 분할한다.
3. $CP_L = \text{ClosestPair}(S_L)$ // CP_L 은 S_L 에서의 최근접 점의 쌍
4. $CP_R = \text{ClosestPair}(S_R)$ // CP_R 은 S_R 에서의 최근접 점의 쌍
5. $d = \min\{\text{dist}(CP_L), \text{dist}(CP_R)\}$ 일 때, 중간 영역에 속하는 점들 중에서 최근접 점의 쌍을 찾아서 이를 CP_C 라고 하자. 단, $\text{dist}()$ 는 두 점 사이의 거리이다.
6. return (CP_L, CP_C, CP_R 중에서 거리가 가장 짧은 쌍)

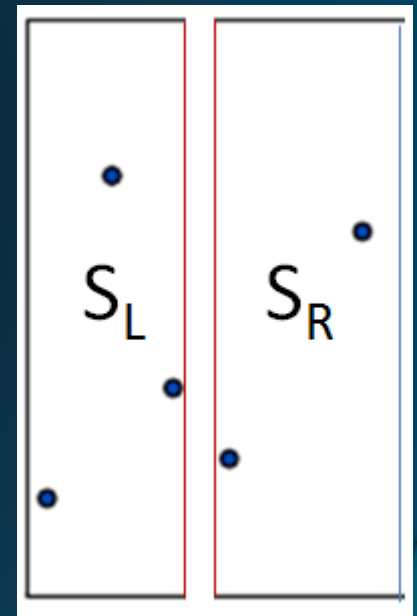
ClosestPair의 수행 과정

- ClosestPair(S)로 호출 [1]: 단, S는 아래의 그림
- Line 1: S의 점의 수 > 3이므로 다음 line을 수행
- Line 2: S를 S_L 과 S_R 로 분할



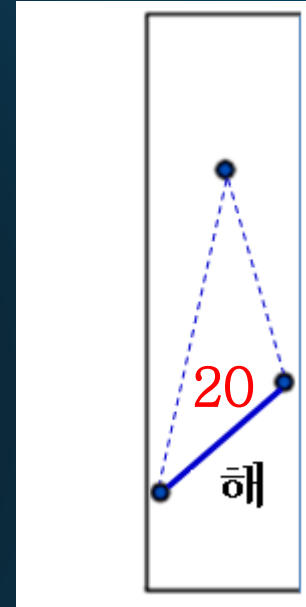
ClosestPair의 수행 과정(2)

- Line 3: ClosestPair(S_L) 호출: ClosestPair(S_L)을 수행한 후 리턴된 점의 쌍을 CP_L 이라고 놓은 후에 line 4~6을 차례로 수행
- ClosestPair(S_L) 호출 [2]:
- Line 1의 if-조건이 '거짓'이므로 line 2에서 다시 분할
- Line 3: ClosestPair(S_L) 호출 (여기서의 S_L 은 처음 S_L 의 왼쪽 반)
- ClosestPair(S_L)을 수행한 후, 리턴된 점의 쌍을 CP_L 이라고 놓은 후에 line 4~6을 차례로 수행



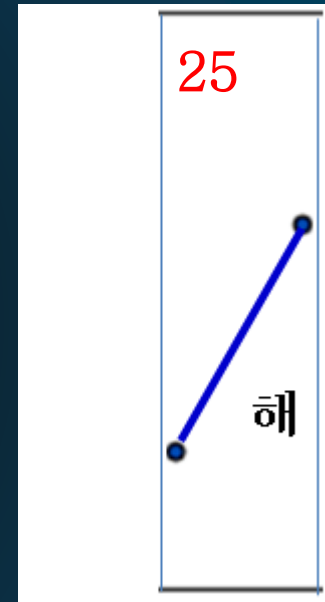
ClosestPair의 수행 과정(3)

- ClosestPair(S_L) 호출:
- Line 1의 if-조건이 '참'; S_L 의 3개의 점들에 대해서 최근접 점의 쌍을 찾는다. 옆의 그림과 같이 3개의 쌍에 대해 거리를 각각 계산하여 최근접 쌍을 해로서 리턴
- 최근접 점의 쌍의 거리를 20이라고 가정



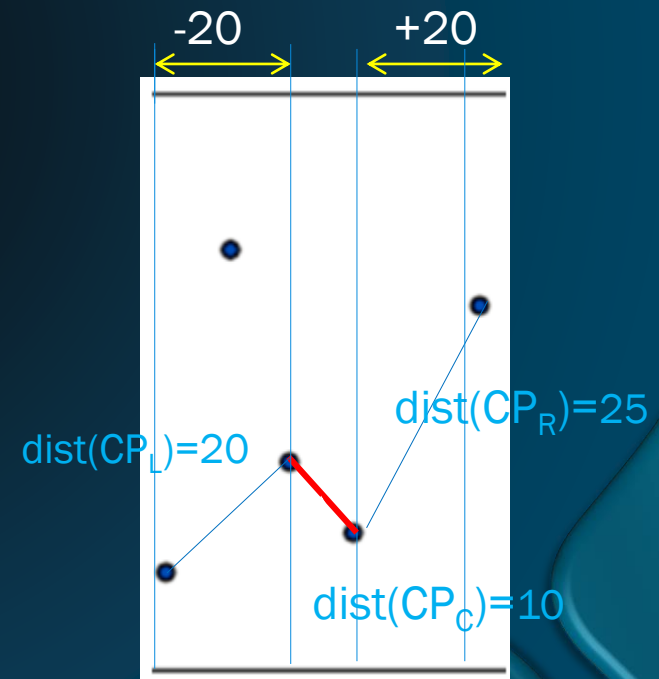
ClosestPair의 수행 과정(4)

- ClosestPair(S_R) 호출
- Line 1에서 점의 수가 2이므로, 이 두 점을 최근접 점의 쌍으로 리턴
- 최근접 점의 쌍의 거리를 **25**라고 가정



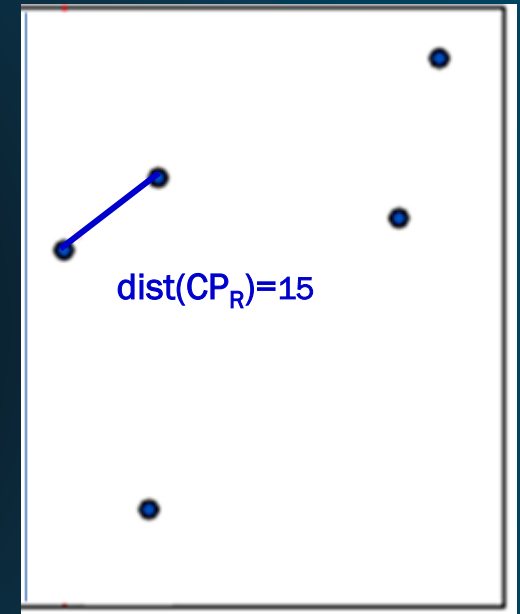
ClosestPair의 수행 과정(5)

- [2]의 ClosestPair(S_L) 호출 당시 line 3~4가 수행되었고, 이제 line 5가 수행된다.
- Line 3~4에서 찾은 최근접 점의 쌍 사이의 거리인 $\text{dist}(CP_L)=20$ 과 $\text{dist}(CP_R)=25$ 중에 작은 값을 $d=20$ 라고 놓는다.
- 그리고 왼쪽 중간점의 x-좌표에서 20을 뺀 값과 오른쪽 중간점의 x-좌표에 20을 더한 값 사이의 x-좌표 값을 가진 점들 중에서 CP_C 를 찾는다.
- 옆의 그림과 같이 거리가 10인 CP_C 를 찾는다.



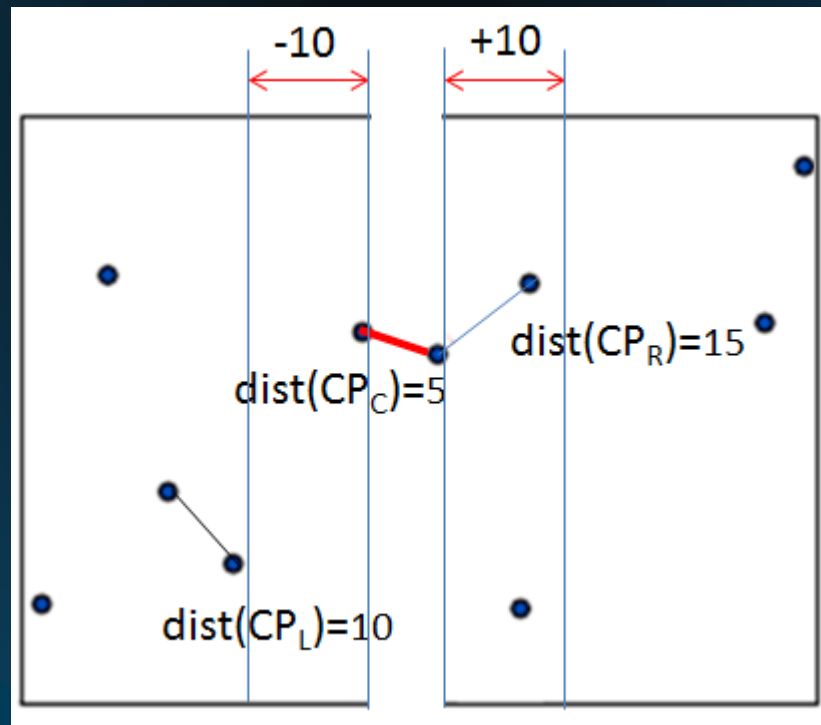
ClosestPair의 수행 과정(6)

- Line 6: $\text{dist}(\text{CP}_L)=20$, $\text{dist}(\text{CP}_C)=10$, $\text{dist}(\text{CP}_R)=25$ 중에서 가장 거리가 짧은 쌍인 $\text{CP}_C=10$ 을 리턴
- [1]의 ClosestPair(S) 호출 당시 line 3이 수행되었고, 이제 line 4에서는 ClosestPair(S_R)을 호출한다. 여기서 S_R 은 초기 입력의 오른쪽 반인 영역이다. ClosestPair(S_R) 호출 결과로 옆 그림의 최근접 점의 쌍을 리턴하고 이를 CP_R 로 놓는다.
- 이때 $\text{dist}(\text{CP}_R)=15$ 라고 하자.



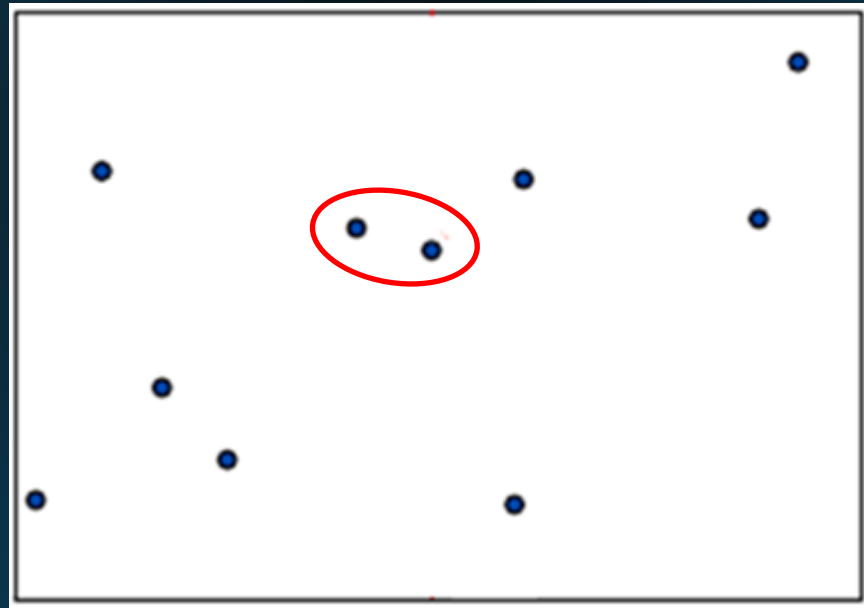
ClosestPair의 수행 과정(7)

- Line 5: line 3~4에서 찾은 최근접 점의 쌍 사이의 거리인 $\text{dist}(\text{CP}_L)=10$ 과 $\text{dist}(\text{CP}_R)=15$ 중에 작은 값을 $d=10$ 이라고 놓는다. 그리고 중간 영역에 있는 점들 중에서 CP_C 를 찾는다. 여기서는 아래의 그림과 같이 거리가 5인 CP_C 를 최종적으로 찾는다.



ClosestPair의 수행 과정(8)

- Line 6: $\text{dist}(\text{CP}_L)=10$, $\text{dist}(\text{CP}_C)=5$, $\text{dist}(\text{CP}_R)=15$ 중에서 가장 거리가 짧은 쌍인 CP_C 를 최근접 쌍의 점으로 리턴(최종해)

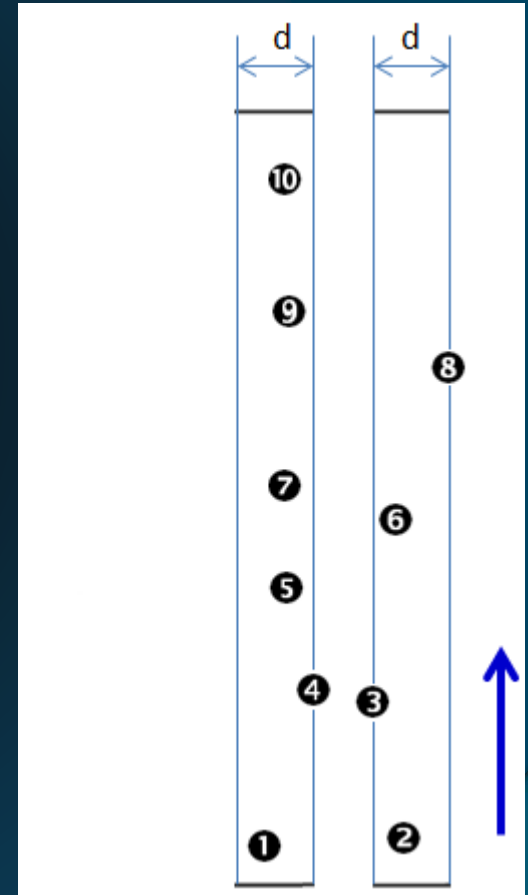


시간복잡도

- 입력 S 에 n 개의 점이 있다면 전처리 (preprocessing) 과정으로서 S 의 점을 x -좌표로 정렬: $O(n\log n)$
- Line 1: S 에 3개의 점이 있는 경우에 3번의 거리 계산이 필요하고, S 의 점의 수가 2이면 1번의 거리 계산이 필요하므로 $O(1)$ 시간이 걸린다.
- Line 2: 정렬된 S 를 S_L 과 S_R 로 분할하는데, 이미 배열에 정렬되어 있으므로, 배열의 중간 인덱스로 분할하면 된다. 이는 $O(1)$ 시간 걸린다.
- Line 3~4: S_L 과 S_R 에 대하여 각각 ClosestPair를 호출하는데, 분할하며 호출되는 과정은 합병 정렬과 동일

시간복잡도(2)

- Line 5: $d = \min\{\text{dist}(CP_L), \text{dist}(CP_R)\}$ 일 때 중간 영역에 속하는 점들 중에서 최근접 점의 쌍을 찾는다.
 - 이를 위해 먼저 **중간 영역에 있는 점들을 y-좌표 기준으로 정렬**한 후에, 아래에서 위로 (또는 위에서 아래로) 각 점을 기준으로 거리가 d 이내인 주변의 점들 사이의 거리를 각각 계산하며, 이 영역에 속한 점들 중에서 최근접 점의 쌍을 찾는다.
 - 따라서 y-좌표로 정렬하는데 $O(n \log n)$ 시간이 걸리고, 그 다음에는 아래에서 위로 올라가며 각 점에서 주변의 점들 사이의 거리를 계산하는데 $O(1)$ 시간이 걸린다. 왜냐하면 각 점과 거리 계산해야 하는 주변 점들의 수는 $O(1)$ 개이기 때문이다.(왜? → 연습문제 17, 18)



시간복잡도(3)

- Line 6: 3개의 점의 쌍 중에 가장 짧은 거리를 가진 점의 쌍을 리턴하므로 $O(1)$ 시간이 걸린다.
- ClosestPair 알고리즘의 분할과정은 합병 정렬의 분할과정과 동일
- 그러나 ClosestPair 알고리즘에서는 해를 취합하여 올라가는 과정인 line 5~6에서 $O(n\log n)$ 시간이 필요
- (다음 슬라이드에서) k층까지 분할된 후, 층별로 line 5~6이 수행되는 (취합) 과정을 보여준다. 이때 각 층의 수행 시간은 $O(n\log n)$ 이다.
- 여기에 층 수인 $\log n$ 을 곱하면, **$O(n\log^2 n)$** 이 된다.

(ClosestPair 알고리즘의 시간복잡도)

시간복잡도(4)

각 부분의
크기

n

$\frac{n}{2}$

$\frac{n}{2^2}$

$\frac{n}{2^k} = 1 \text{ or } 2$

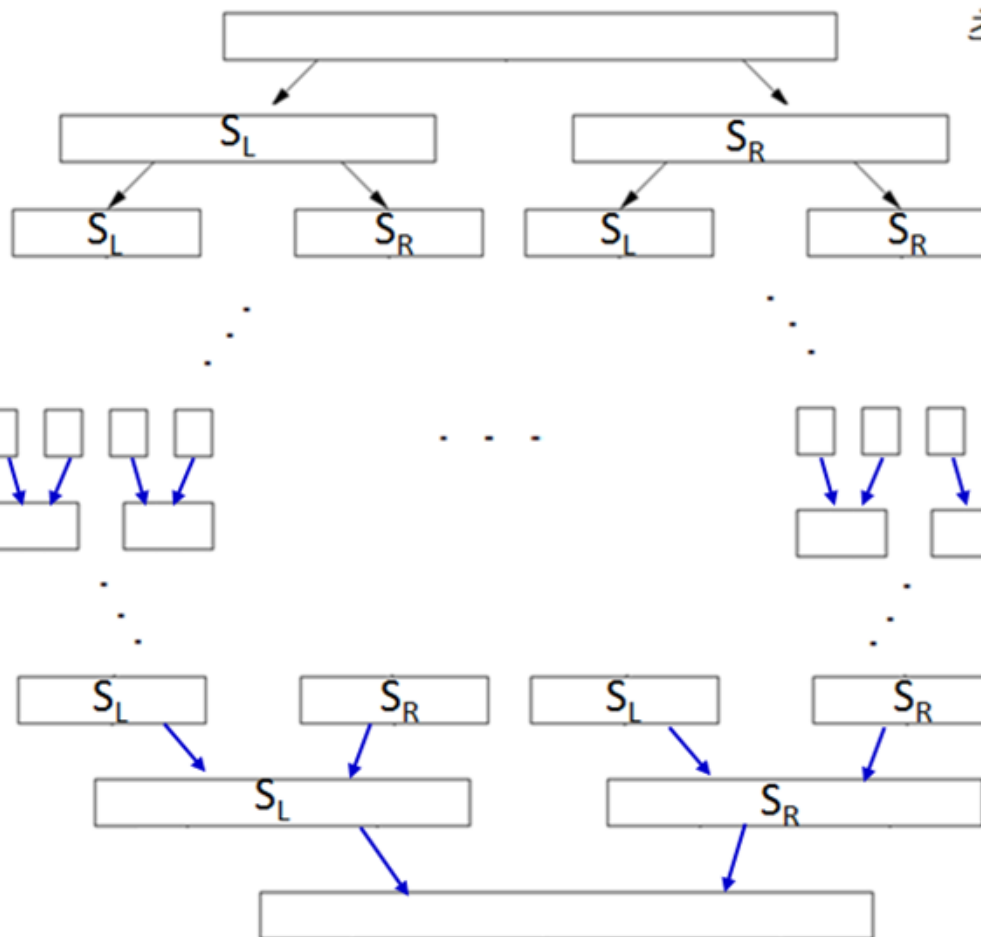
초기 입력

1 층

2 층

k 층

Line 5 ~ 6 을
수행하는 과정

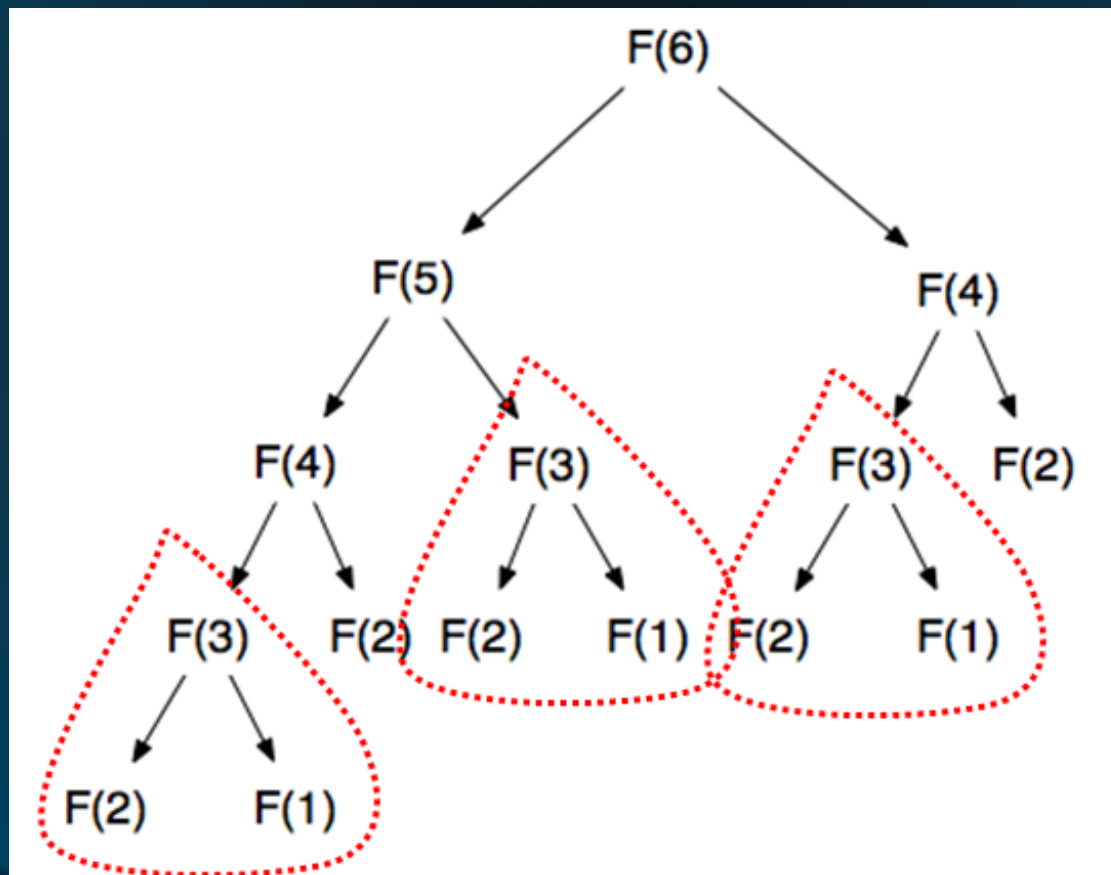


응용 분야

- 컴퓨터 그래픽스
- 컴퓨터 비전 (Vision)
- 지리 정보 시스템 (Geographic Information System, GIS)
- 분자 모델링 (Molecular Modeling)
- 항공 트래픽 조정 (Air Traffic Control)
- 마케팅 (주유소, 프랜차이즈 신규 가맹점 등의 위치 선정) 등

3.5 분할 정복을 적용하는데 있어서 주의할 점

- 분할 정복이 부적절한 경우는 입력이 분할될 때마다 분할된 부분문제의 입력 크기의 합이 분할되기 전의 입력 크기보다 매우 커지는 경우이다.



예시

- 예를 들어, n 번째의 피보나치 수를 구하는데 $F(n) = F(n-1) + F(n-2)$ 로 정의되므로 재귀 호출을 사용하는 것이 자연스러워 보이나, 이 경우의 입력은 1개이지만, 사실상 n 의 값 자체가 입력 크기인 것이다.
- 따라서 n 이라는 숫자로 인해 2개의 부분 문제인 $F(n-1)$ 과 $F(n-2)$ 가 만들어지고, 2개의 입력 크기의 합이 $(n-1) + (n-2) = (2n-3)$ 이 되어서, 분할 후 입력 크기가 거의 2배로 늘어난다. 이전 슬라이드의 그림은 피보나치 수 $F(5)$ 를 구하기 위해 분할된 부분 문제들을 보여준다. $F(2)$ 를 5번이나 중복하여 계산해야 하고, $F(3)$ 은 3번 계산된다.

피보나치 수 계산을 위한 $O(n)$ 시간 알고리즘

FibNumber(n)

1. $F[0]=0$
2. $F[1]=1$
3. for $i=2$ to n
4. $F[i] = F[i-1] + F[i-2]$

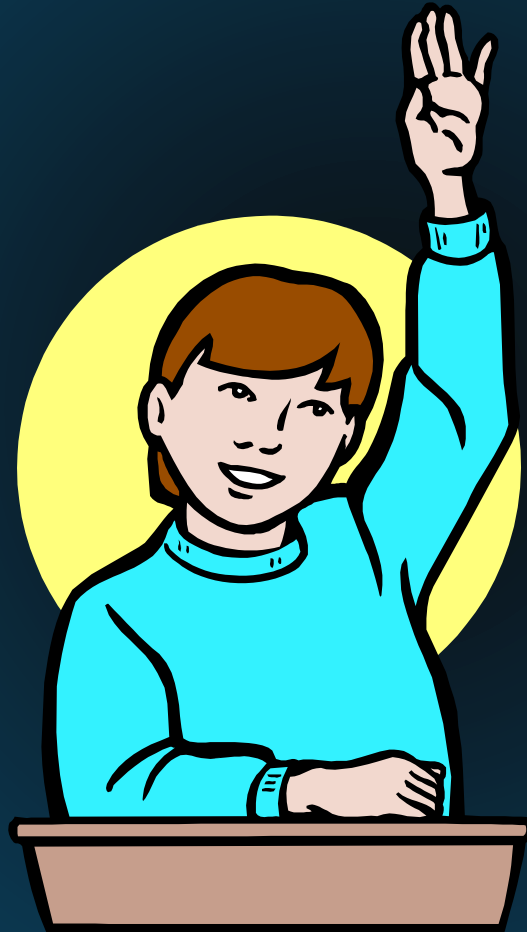
또 하나의 주의할 점

- 주어진 문제를 분할 정복 알고리즘으로 해결하려고 할 때에 주의해야 하는 또 하나의 요소는 취합(정복) 과정이다.
 - 입력을 분할만 한다고 해서 효율적인 알고리즘이 만들어지는 것은 아니다. 3장에서 살펴본 문제들은 취합 과정이 간단하거나 필요 없었고, 최근접 점의 쌍을 위한 알고리즘만이 조금 복잡한 편이었다.
 - 또한 기하 (geometry)에 관련된 다수의 문제들이 효율적인 분할 정복 알고리즘으로 해결되는데, 이는 기하 문제들의 특성상 취합 과정이 문제 해결에 잘 부합되기 때문이다.

Summary

- 선택 (Selection) 문제: k 번째 작은 수를 찾는 문제로서, 입력에서 퀵 정렬에서와 같이 피벗을 선택하여 피벗보다 작은 부분과 큰 부분으로 분할한 후에 k 번째 작은 수가 들어있는 부분을 재귀적으로 탐색한다. 평균 경우 시간복잡도는 $O(n)$ 이다.
- 최근접 점의 쌍 (Closest Pair) 문제: n 개의 점들을 $1/2$ 로 분할하여 각각의 부분 문제에서 최근접 점의 쌍을 찾고, 2개의 부분해 중에서 짧은 거리를 가진 점의 쌍을 일단 찾는다. 그리고 2개의 부분해를 취합할 때, 반드시 중간 영역 안에 있는 점들 중에 최근접 점의 쌍이 있는지도 확인해보아야 한다.
시간복잡도는 $O(n \log^2 n)$ 이다.
- 분할 정복이 부적절한 경우는 입력이 분할될 때마다 분할된 부분 문제들의 입력 크기의 합이 분할되기 전의 입력 크기보다 커지는 경우이다. 또 하나 주의해야 할 요소는 취합 (정복) 과정이다.

Q&A



BREAK TIME



실습시간

1. Selection 알고리즘을 파이선 구현하기
2. ClosestPair 알고리즘을 파이선 구현하기

- 숙제:

1. Selection 알고리즘 성능 측정 실험
2. ClosestPair 알고리즘 성능 측정 실험
3. 3장 연습문제 17번과 18번의 아이디어를 반영한 ClosestPair 알고리즘의 구현