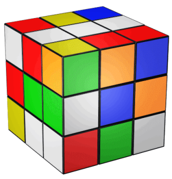
분할 정복

한국기술교육대학교 컴퓨터공학부 김상진



```
while(!morning){
    alcohol++
    dance++
} #party
```

```
while(!sleep){
    think++
    solve++
} #cse-mode
```

교육목표

- 분할 정복(divide-and-conquer)
  - 문제의 사례를 2개 이상의 더 작은 사례(같은 종류의 문제)로 나누어(divide) 각 작은 사례에 대한 해답(conquer)을 얻고, 이들을 결합(combine)하여 원 문제에 대한 해답을 얻는 방식
  - 보통 다차시간 알고리즘의 성능을 개선하기 위해 사용함
- 살펴볼 문제
  - 역쌍 개수 구하기
  - 연속 부분 구간의 최댓값 구하기
  - 1부터 n까지 합
  - 수의 거듭제곱
  - 행렬 곱셈
  - 가장 가까운 좌표 구하기

합병정렬
    왼쪽 절반 정렬
    오른쪽 절반 정렬
    정렬된 양쪽 결합

- $O(n^2)$을 $O(n\log n)$으로 개선
- 핵심은 비재귀적인 부분이 $O(n)$ 트리 높이는 $\log n$

분할정복

- 합병 정렬에 사용한 전략
- 기본 알고리즘
  - 단계 1. 문제를 같은 유형의 작은 문제로 나눔
  - 단계 2. 작은 문제를 재귀적으로 해결
  - 단계 3. 해결 결과를 결합하여 원 문제에 대한 해결책 제시
- 재귀 깊이와 단계 3의 시간 복잡도가 전체 성능에 가장 큰 영향을 줌
- 문제를 나누는 과정은 해답을 쉽게 얻을 수 있는 수준까지 반복적으로 적용함
  - 보통 재귀적으로 구현
  - 작은 문제는 반드시 같은 유형이지만 규모가 작은 문제임
    - 재귀를 종료하는 시점도 알고리즘 설계에서 중요 검토 사항
  - 하향식 접근 방법(top-down)
    - 참고. 동적 프로그래밍은 상향식 접근 방법

분할 정복으로 해결할 수 있는 문제

- 문제 1. 정렬된 배열에서 특정 요소 찾기
  - 이진 검색
- 문제 2. 배열을 오름차순으로 정렬하기
  - 합병 정렬, 빠른 정렬 (Note 05)

역쌍 개수 구하기 (1/4)

- 입력. n개의 수로 구성된 배열
- 출력. 역쌍(inversion)의 개수 구하기
  - 역쌍이란 $i < j$에 대해 $A[i] > A[j]$
  - 예) [1, 3, 5, 2, 4, 6]
    - (3, 2), (5, 2), (5, 4): 3개

- Brute Force 알고리즘: $O(n^2)$

```
count := 0
for i := 1 to n − 1 do
    for j := i + 1 to n do
        if A[i] > A[j] then ++count
return count
```

Can we do better?

역쌍 개수 구하기 (2/4)

- 관찰
  - 정렬되어 있으면 역쌍의 개수는?
  - n 크기 배열에서 최대 역쌍의 개수는?
    - 내림차순으로 정렬되어 있는 경우: $\frac{n(n-1)}{2}$
- 분할 정복으로 접근
  - 왼쪽 역쌍, 오른쪽 역쌍, 쪼개진 역쌍으로 역쌍 구분
  - 왼쪽 역쌍: 배열 왼쪽 절반에 있는 역쌍
  - 오른쪽 역쌍: 배열 오른쪽 절반에 있는 역쌍
  - 쪼개진 역쌍: 쌍에 요소가 하나는 왼쪽에 다른 하나는 오른쪽에 있는 역쌍
  - 예) [1, 5, 3, 4, 2, 6]
    - 왼쪽 역쌍: (5, 3), 오른쪽 역쌍: (4, 2), 쪼개진 역쌍: (5, 4), (5, 2), (3, 2)
  - 전체 역쌍: 왼쪽 역쌍 + 오른쪽 역쌍 + 쪼개진 역쌍

역쌍 개수 구하기 (3/4)

- 분할 정복 알고리즘

countInversions(A[], n)

if $n = 1$ *then return 0*
else
    L $:=$ *countInversion(left half of A,* $n/2$*)*
    R $:=$ *countInversion(right half of A,* $n/2$*)*
    S $:=$ *countSplitInversion(A,* n*)*
    return L+R+S

sort_countInv(A[], n)

if $n = 1$ *then return 0*
else
    L $:=$ *sort_countInv (left half of A,* $n/2$*)*
    R $:=$ *sort_countInv (right half of A,* $n/2$*)*
    S $:=$ *merge_countSplitInv(A,* n*)*
    return L+R+S

- 여기서 countSplitInversion이 $O(n)$이면 전체는 $O(n\log n)$임
  - Why? 합병 정렬과 동일
- countSplitInversion은 실제 합병정렬과 동일한 과정을 수행
  - Why? 합병하는 과정에서 역쌍을 발견할 수 있음
    - 합병과 동일한 과정을 수행하기 위해서는 정렬되어 있어야 함
  - How?

- Side Effect. A[]가 알고리즘 수행 결과 바뀜

역쌍 개수 구하기 (4/4)

- Merge: [1,3,5], [2,4,6]
  - 쪼개진 역쌍이 없으면 왼쪽 배열과 오른쪽 배열의 특성은?
    - 왼쪽 배열에 있는 모든 요소가 오른쪽 배열에 있는 모든 요소보다 작음

1	3	5

2	4	6

1	2				

- 오른쪽에서 옮길 때 왼쪽에 남아 있는 요소의 개수만큼 역쌍이 존재

역쌍 개수 구하기 응용

- Collaborative Filtering
  - 주어진 10개의 영화에 대한 사용자들의 정한 순위 확보
    - A, B, C가 정한 순위가 있을 때, B와 C 중 A와 취향이 누구가 더 유사한가?

아바타: 물의 길	마녀2	6	블랙 팬서2	4
탑건: 매버릭	범죄도시2	7	탑건: 매버릭	2
닥터 스트레인지2	한산: 용의 출현	8	아바타: 물의 길	1
블랙 팬서2	아바타: 물의 길	1	신비한 동물들3	5
신비한 동물들3	탑건: 매버릭	2	닥터 스트레인지2	3
마녀2	블랙 팬서2	4	범죄도시2	7
범죄도시2	닥터 스트레인지2	3	한산: 용의 출현	8
한산: 용의 출현	신비한 동물들3	5	마녀2	6
토르4	토르4	9	미니언즈2	10
미니언즈2	미니언즈2	10	토르4	9

- 역쌍은 A와 의견이 다르다는 것을 의미함

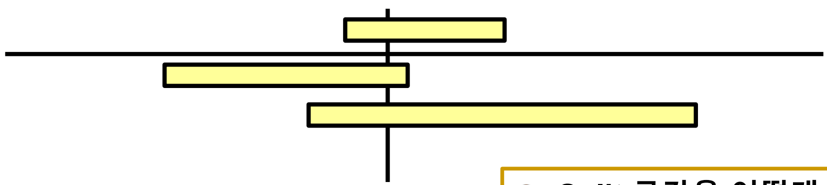
- 역쌍: 16개
- 역쌍: 8개

응용 문제

- 연속 부분 구간의 최댓값 구하기: LeetCode 53

```
max := MIN
for i := 1 to n do
    sum := 0
    for j := i to n do
        sum += A[j]
        max := max(sum, max)
return max
```

- $O(n^2)$을 $O(n\log n)$으로
- 참고. 이 문제는 $O(n)$으로 해결가능



- Split 구간을 어떻게 $O(n)$?

빠른 합

- 1부터 n까지의 합을 분할 정복으로?
  - $1+2+\cdots+n = \left(1+2+\cdots+\frac{n}{2}\right) + \left(\left(\frac{n}{2}+1\right)+\left(\frac{n}{2}+2\right)+\cdots+\left(\frac{n}{2}+\frac{n}{2}\right)\right)$
  - sum(n) = sum($n/2$)+...
  - $\left(\left(\frac{n}{2}+1\right)+\left(\frac{n}{2}+2\right)+\cdots+\left(\frac{n}{2}+\frac{n}{2}\right)\right) = \frac{n}{2}\times\frac{n}{2}+\left(1+2+\cdots+\frac{n}{2}\right)$
  - $\mathrm{sum}(n) = 2\times\mathrm{sum}\left(\frac{n}{2}\right)+\frac{n^2}{4}$
    - n이 짝수일 때만 위 식이 성립함
    - 그러면 n이 홀수일 때는?
- 시간 복잡도
  - 원래 합: $n-1$개의 합
  - 분할 정복 합? $\log n$번 호출. 내부적으로 곱셈 2번, 나눗셈 2번
    - 곱하기 2, 나누기 2는 일반 곱셈, 나눗셈은 아님
- 실제 합을 이렇게 분할 정복하지는 않음. 하지만 개념은?

수의 거듭제곱 (1/2)

- 일반 수의 거듭제곱
  - $n^m = n\times n\times\cdots\times n$: 기본은 $m-1$번 곱셈
  - Square-and-multiply
    - 예) $3^{53} = 3^{32}\cdot 3^{16}\cdot 3^4\cdot 3$ (0b110101)
      - 3^2, $3^4 = 3^2\cdot 3^2$, $3^8 = 3^4\cdot 3^4$, $3^{16} = 3^8\cdot 3^8$, $3^{32} = 3^{16}\cdot 3^{16}$
      - 총: 8번 (≪ 52번)

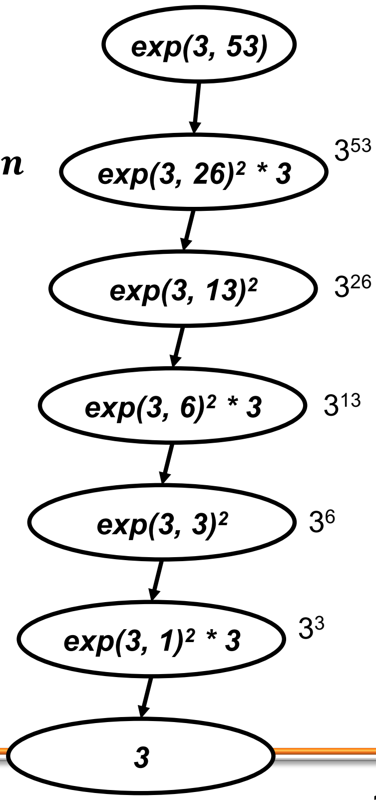
```
ret := 1
while m > 0 do
    if m is odd then
        ret := ret * n
    n := n * n
    m := m/2
return ret
```

ret	n	m
1	3	53
3	3^2	26
3	3^4	13
3^5	3^8	6
3^5	3^{16}	3
3^{21}	3^{32}	1
3^{53}	3^{64}	0

수의 거듭제곱 (2/2)

- 분할 정복은?
  - $n^m = n^{m/2} \times n^{m/2}$
  - exp(n, m) = exp(n, $m/2$) * exp(n, $m/2$)
  - n이 홀수이면? exp(n, m) = exp(n, $m-1$) $\times n$
  - 기저 사례:
    - $m = 0$: 1
    - $m = 1$: n

```
if m = 0 then return 1
if m = 1 then return n
if m is even then
     ret := exp(n, m/2)
     ret := ret * ret
else
     ret := n * exp(n, m − 1)
return ret
```



행렬 곱셈

- 두 개의 $n \times n$ 행렬의 곱 구하기
  - 입력. $n \times n$ 행렬 2개 X, Y
  - 출력. $Z = X \times Y$
  - 가정. n은 2의 거듭제곱
- 기본 알고리즘
  - $z_{ij} = \sum_{k=1}^{n} x_{ik} y_{kj}$
  - 기본 알고리즘의 복잡도: $O(n^3)$
- 참고. 입력 크기: $O(n^2)$, 출력 크기: $O(n^2)$
  - $O(n^2)$보다 빠르게 처리하기는 …
- 분할 정복을 배우고 있으니, 행렬 곱셈도 분할 정복이 가능???

```
for i := 1 to n do
    for j := 1 to n do
        Z[i][j] := 0
        for k := 1 to n do
            Z[i][j] := Z[i][j] + X[i][k] * Y[k][j]
return Z
```

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} ae+bg & af+bh \\ ce+dg & cf+dh \end{bmatrix}$$

Can we do better?

분할 정복 행렬 곱셈

- X와 Y를 부분 행렬로 나눈 후 곱셈을 진행함

$$XY = \begin{bmatrix} A & B \\ C & D \end{bmatrix}\begin{bmatrix} E & F \\ G & H \end{bmatrix} = \begin{bmatrix} AE+BG & AF+BH \\ CE+DG & CF+DH \end{bmatrix}$$

  - 분할 정복을 하였지만 성능 향상은???
    - 총 8개의 재귀호출이 필요함 $\Longrightarrow O(n^3)$
- Strassen 알고리즘
  - 위 8개의 재귀 호출을 7개로 축소함
    - $P_1 = A(F-H)$
    - $P_2 = (A+B)H$
    - $P_3 = (C+D)E$
    - $P_4 = D(G-E)$
    - $P_5 = (A+D)(E+H)$
    - $P_6 = (B-D)(G+H)$
    - $P_7 = (A-C)(E+F)$

$$XY = \begin{bmatrix} A & B \\ C & D \end{bmatrix}\begin{bmatrix} E & F \\ G & H \end{bmatrix}$$
$$= \begin{bmatrix} P_5+P_4-P_2+P_6 & P_1+P_2 \\ P_3+P_4 & P_1+P_5-P_3-P_7 \end{bmatrix}$$

- 재귀 호출 수의 감소는 전체 호출 수를 지속하여 줄이는 효과가 있음

가장 가까운 쌍 찾기 (1/8)

- 입력. $n(\geq 2)$개의 $p_i = (x_i, y_i)$ 좌표
- 출력. 가장 가까운 두 개의 좌표 p_i, p_j
  - 가장 가까운 좌표 $\Longrightarrow$ Euclidean distance
    - $d(p_i, p_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$
- Brute-force 방법

```
min ≔ max
ret ≔ []
for i ≔ 1 to n − 1 do
    for j ≔ i + 1 to n do
        d ≔ euclideanDistance(A[i], A[j])
        if min>d then
            min ≔ d
            ret[0] ≔ A[i]
            ret[1] ≔ A[j]
return ret
```

- $O(n^2)$을 분할 정복을 이용하여 $O(n\log n)$으로

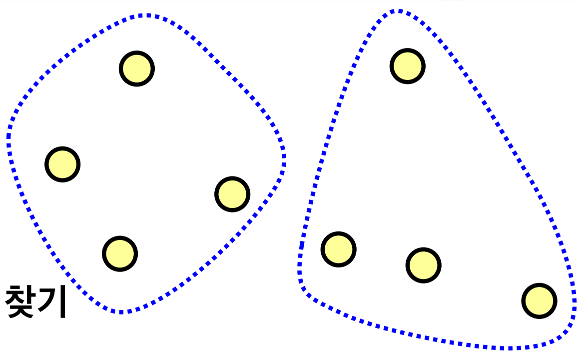
가장 가까운 쌍 찾기 (2/8)

- 좌표들이 1차원이면
  - 좌표들을 정렬
  - 인접 좌표들을 비교
  - $O(n\log n) + O(n) = O(n\log n)$
    - Brute-force보다는 성능이 좋은 알고리즘

- 분할 정복
  - 역쌍 개수 구하기와 비슷하게 접근
  - 전체 좌표를 왼쪽, 오른쪽으로 나눔 (기준(x, y)은 상관 없음)
  - 가장 가까운 쌍은 왼쪽, 오른쪽, 양쪽에 있을 수 있음
    - 왼쪽, 오른쪽은 재귀적으로 구함
    - 양쪽에 있는 쌍을 찾는 것만 $O(n)$에 할 수 있으면 전체 비용은 $O(n\log n)$

가장 가까운 쌍 찾기 (3/8)

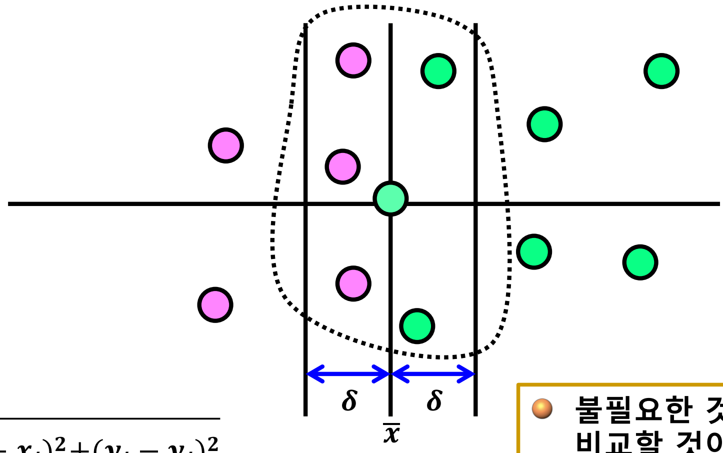
- 왼쪽, 오른쪽 나누기는 어떻게?
  - x 좌표를 이용하여 정렬: $O(n\log n)$
- 알고리즘
  - 단계 1. 왼쪽에서 가장 가까운 쌍 찾기
  - 단계 2. 오른쪽에서 가장 가까운 쌍 찾기
  - 단계 3. 양쪽으로 분리된 가장 가까운 쌍 찾기
- 단계 3을 어떻게 $O(n)$에 할 수 있을까?
  - 서로 반대쪽에 있는 쌍 간의 비교만 필요함
    - 전체 필요한 비교 수: $\frac{n}{2} \times \frac{n}{2}$
- 비교해야 하는 쌍을 어떻게 줄일 수 있을까?



- 알고리즘이 매우 직관적인 경우도 있지만 이 문제처럼 매우 clever thinking이 필요한 경우도 있음

가장 가까운 쌍 찾기 (4/8)

- 아이디어 1. (두 좌표의 x 좌표를 이용한 배제) 단계 1과 단계 2에서 구한 쌍 중 가장 가까운 쌍의 거리 δ를 이용하여 그 거리보다 큰 것들은 배제함
  - 어떻게? $|x_i - x_j| > \delta$이면 고려 대상에서 제외
  - x 좌표를 이용하여 정렬되어 있으므로 $\bar{x}$가 x 좌표를 이용하여 정렬하였을 때 중간 좌표의 x 좌표일 때, $\bar{x} - \delta$와 $\bar{x} + \delta$ 사이에 있는 좌표들만 고려함



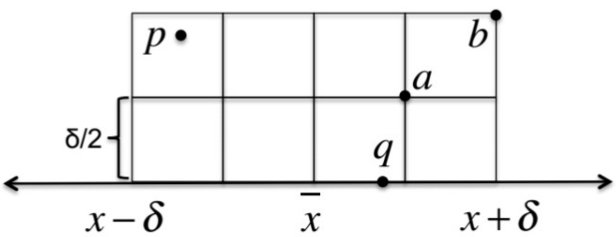
$$d(p_i, p_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

- 불필요한 것은 제거하였지만 비교할 것이 어떻게 $O(n)$?

가장 가까운 쌍 찾기 (5/8)

- 아이디어 2. (두 좌표의 y 좌표를 이용한 배제) $\bar{x} - \delta$와 $\bar{x} + \delta$ 사이에 있는 좌표로만 구성된 집합이 있을 때, 이 좌표들 중 하나를 선택하였을 때, 이 좌표와 비교해야 하는 좌표는 총 7개만 존재함???
  - 아래 그림에서 각 박스에는 두 개의 좌표가 존재할 수 없음
    - Why? 모순. 존재하면 왼쪽, 오른쪽에서 가장 가까운 좌표 거리가 δ보다 작아짐
      - 그림에서 좌표 a와 b를 생각하면 이 두 좌표의 거리는 $\frac{\delta}{\sqrt{2}} < \delta$임
  - 해당 7개 좌표를 구하기 위해서는 y 좌표를 기준으로 정렬되어 있어야 함

q가 y 좌표가 가장 작은 좌표
이 좌표와 비교할 좌표는 최대 7개



가장 가까운 쌍 찾기 (6/8)

- 집합 P로부터 x 좌표를 기준으로 정렬된 P_x와 y 좌표를 기준으로 정렬된 P_y를 확보 → $O(n\log n)$
- closestPair 알고리즘(P_x, P_y)
  - 단계 1. 다음 4개를 확보 $\Longrightarrow O(n)$
    - L_x: P_x의 왼쪽 절반
    - L_y: P_x의 왼쪽 절반, y 좌표를 기준으로 정렬
    - R_x: P_x의 오른쪽 절반
    - R_y: P_x의 오른쪽 절반, y 좌표를 기준으로 정렬
  - 단계 2. (base case ≤ 3: brute-force 방법)
    - $(l_1, l_2) = \text{closestPair}(L_x, L_y)$
    - $(r_1, r_2) = \text{closestPair}(R_x, R_y)$
    - $(s_1, s_2) = \text{closestSplitPair}(P_x, P_y) \Longrightarrow O(n)$
  - 단계 3. (l_1, l_2), (r_1, r_2), (s_1, s_2) 중 가장 가까운 쌍 반환

- P_x, P_y로부터 L_x, L_y, R_x, R_y를 $O(n)$에 확보해야 함. How?

- 비교해야 하는 좌표의 수가 3 이하이면 전수조사하여 가장 가까운 쌍을 찾음 (더 이상 재귀호출하지 않음)

가장 가까운 쌍 찾기 (7/8)

- 예) (2, 4), (4, 6), (1, 1), (4, 4), (6, 3), (5, 2)
  - P_x: (1, 1), (2, 4), (4, 4), (4, 6), (5, 2), (6, 3)
  - P_y: (1, 1), (5, 2), (6, 3), (2, 4), (4, 4), (4, 6)
  - L_x: (1, 1), (2, 4), (4, 4)
  - L_y: (1, 1), (2, 4), (4, 4)
  - R_x: (4, 6), (5, 2), (6, 3)
  - R_y: (5, 2), (6, 3), (4, 6)

가장 가까운 쌍 찾기 (8/8)

- closestSplitPair 알고리즘

closestSplitPair(P_x, P_y, δ)

$\bar{x} := P_x$ 중 중간 x 값 (median)
$S_y := \{q_1, q_2, \ldots, q_l\}$, $q_i = (x_i, y_i)$일 때 $\bar{x} - \delta < x_i < \bar{x} + \delta \rightarrow O(n)$
best $:= \delta$
bestPair $:=$ *null*
for $i := 1$ *to* $l - 1$ *do*
    for $j := 1$ *to* $min(7, l - i)$ *do*
        if $d(q_i, q_{i+j}) < best$ then
            best $:= d(q_i, q_{i+j})$
            bestPair $:= (q_i, q_{i+j})$
return *bestPair*

- $O(n)$: 2중 for loop이지만 내부 for loop는 항상 7이하: $7n$

분할 정복이 적합하지 않는 경우

- 다음 두 가지 경우에는 분할 정복을 사용하지 않아야 함
  - 경우 1. 입력 크기가 n인 것이 입력 크기가 거의 n인 두 개 이상의 사례로 분할되는 경우
    - 예) 피보나찌 수: $f(n) = f(n-1) + f(n-2)$
      - 지수시간 알고리즘
  - 경우 2. 입력 크기가 n인 것이 거의 n개의 입력 크기가 n/b인 사례로 분할되는 경우