Lecture 4 : 분할정복(Divide and Conquer) 알고리즘 개발모형

이제부터는 어떤 실제적인 문제에 대한 알고리즘을 개발하는 방법론(methodology)에 대하여 하나씩 살펴볼 예정이다. 비유하자면 이런 것이다. 우리가 특정 요리에 관한 조리법을 공부하여 그것을 만들 수도 있지만, 대부분의 학원에서는 특정 요리가 아니라 요리법 자체에 대한 교육을 시킨다.

예를 들어, 찜을 하는 방법, 튀김을 위한 일반적인 방법과 주의사항, 또는 굽기, 찌기(steaming) 등은 특정 요리법이 아니라 요리의 방법론이다. 따라서 앞서 말한 일반적인 요리법에 익숙한 조리사라면 특정 재료가 주어졌을 때 어떤 조리법이 가장 적절한지를 이전의 경험을 바탕으로 찾아낼 수 있다. 예를 들어 큰 크기의 흰 살 생선의 맛을 가장 잘 살리는 방법은 회를 뜨는 것인지 또는 찌는 것인지 또는 튀기는 것인지에 대해 고민을 할 것이다. 그 고민 이전에는 이러한 조리법에 대하여 충분한 경험과 지식이 존재해야 한다.

지금 배우는 분할정복은 바로 이와 같은 알고리즘의 개발법 중 가장 대표적인 하나의 접근 방법론이다. 어떤 계산 문제가 있을 때 분할정복법에 기초한 알고리즘이 좋은지 다음에 배울 동적계획법으로 개발하는 것이 더 좋은지[1], 또는 백트래킹이 좋은지는 알 수 없다. 마치 어떤 요리 재료가 주어졌을 때 어떤 조리법이 가장 좋은 것인지는 아무도 알 수 없다. 하지만 이전이 경험을 동원하며 시행착오의 횟수를 훨씬 줄일 수 있다. 알고리즘 역시 이와 같다.

실전에서 주어진 문제의 알고리즘으로 분할정복법이 가장 좋은지는 알 수 없지만, 이전에 이 문제와 비슷한 문제를 분할정복법을 적용하여 좋은 성능의 알고리즘을 개발할 수 있었다면 이 대상 문제 역시 시작은 분할정복법으로 개발을 시작하는 것이 합당한 접근법이라 할 수 있다. 즉 주어진 실전 문제를 해결하는 가장 좋은 알고리즘의 구조는 아무도 알 수 없지만, 이전의 경험을 통하여 그 접근법을 잘 선택하면 시행착오의 횟수를 크게 줄일 수 있다. 따라서 앞으로 배울 알고리즘 개발방법론에서 여러분은 최대한 많은 사례를 익혀야 한다는 것이다. 그것이 충분히 쌓이면 주어진 문제를 어떻게 풀어야 하는지 좋은 직관을 가질 수 있다.

4.1 왜 다양한 알고리즘 설계 방법론(methodology)[2]을 공부해야 할까요 ?

전형적인 접근 방법이며 개발 시간을 단축시켜 줄 수 있다.

요리법 = { 굽기, 튀기기, 삶기, 조리기, 으깨기.... } 카테고리로 분리

a) 분할정복 (Divide and Conquer)

---

1) 성능면에서 더 우수, 즉 더 빠르고, 더 적은 메모리를 사용하고

2) 방법은 method, 이러한 방법에 대한 상층 단계 즉 meta method가 mdtdology가 된다.

b) 욕심쟁이법 (Greedy Method)
c) 동적계획법 (Dynamic programming)
d) 되살펴보기 기법 (Backtracking)
e) 전 구역 탐색법 (Exhaustive searching)
f) 대충해보기 (Heuristics)

4.2 분할정복법(Divide and Conquer)의 개념 및 실생활 활용
a) 아주 큰 크기의 수박을 먹기
b) 친구 사이를 이간질
c) 큰 건물 전체 청소하기
d) 월급으로 32평 아파트 구입하기
e) 나폴레옹의 전투 전략, 한니발의 전투 전략

4.3 **[분할정복법]**을 이용하는 알고리즘 구조

Step 1. 분할과정 – 만일 바로 해결가능하면 그 자리에서 해결
else 아니라면 적절한 크기로 쪼갠다. (divide)

Step 2. 정복과정 – 각각 잘려진 부분 문제를 해결한다. (solve or conquer)
만일 그 크기가 크다고 생각하면 다시 쪼갠다.
그리고 Goto Step 1.

Step 3. 통합과정 : 각 <부분 해(partial solution)>를 종합하여 해를 구성.

Divide → Conquer → Merge → Solve and Complete

적용의 예) Merge Sorting에 적용

5	43	22	7	98	50	36	9	89	11	17	50	8	90	17

5	43	22	7	98	50	36								

							9	89	11	17	50	8	90	17

각각을 sorting하면 안됨. Divide하는 과정이 필요함.
분할정복이 의미가 있도록, 일단 전체를 먼저 분할해야 함.
Quick Sort의 Pivot과정이 필요함.

4.4 분할정복법을 이용한 전형적인 알고리즘의 예

예 1) Array[n]에서 가장 큰 수 구하기 (Find Max)

예 2) Array[n]에서 두 번째로 큰 수 구하기 (Find Second max)

반으로 나눈 두 구간에서 각각 가장 큰 원소를 구한 다음

이를 비교해서 작은 값의 원소가 2번째 원소인가 ? 반례?

예 3) Array[n]에서 k-th 번째로 큰 수 구하기 (Find k-th element)

4.5 분할정복법 알고리즘의 분석 예(1) : 이분 탐색(binary Search)

정렬된 data N개가 1차원 배열에 저장되어 있다.[3)]

- Time complexity 분석
- Space Complexity 분석

Q) 만일 3분 검색을 하면 그 복잡도는 어떻게 될 것인가 ?

Q) k-ary 검색을 생각해보고 k가 얼마일 때 비교 횟수에서 가장 유리한가 ?

4.6 1000개의 원소가 순서 없이 저장되어 있다. 이 중에서 300번째 원소를 찾아라.

방법1) 전체를 정렬 Sorting하여 그 중에서 X[299]을 선택함.

방법2) Sorting은 너무 과한 작업. 더 작은 비교 횟수로 해결하는 방법

Q <	V	< P

다음 실제 배열에 들어있는 원소를 이용해서 해보자. 각각의 칸을 채우시오.

t=0	56	83	13	89	43	12	34	9	50	98	34	11

4.7 분할정복법 알고리즘 아이디어 - Do not recompute !

- 기본동작은 곱셈. (이 문제에서 덧셈은 포함하지 않는다.)
- 복소수 X=(a + bi) (c + di) 의 계산, 꼭 4번의 곱셈이 필요한가 ?
  뭔가 좀 더 비겁한(?) 방법이 없을까 ?

3) STL algorithm에서 제공한다. 제발.. STL을 사용하자. 이상하게 짜지말고..

X= R + Ii = (ac – bd) + (bc + ad)i

$m1 = (a + b) \cdot (c - d) = a \cdot c - a \cdot d + b \cdot c - b \cdot d$
$m2 = b \cdot c$
$m3 = a \cdot d$
$R = m1 - m2 + m3$
$I = m2 + m3$

- 덧셈은 더 많이 사용하지만 arithmetic
곱셈의 횟수는 한 번더 줄일 수 있다.[4)]

4.8 행렬 곱에서 단위 곱셈의 횟수를 최대한으로 줄이기

일반적으로 $n \times n$ 두 개의 행렬을 곱할 때 필요한 곱셈의 수 = n^3

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}$$
$$B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$
$$AB = \begin{pmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{pmatrix}$$

8번의 곱셈을 7번으로 줄이는 처절한(?) 쉬트라센(Strassen)의 알고리즘

$$\begin{aligned} P_1 &= (A_{11} + A_{22})(B_{11} + B_{22}) \\ P_2 &= (A_{21} + A_{22})(B_{11}) \\ P_3 &= (A_{11})(B_{12} - B_{22}) \\ P_4 &= (A_{22})(-B_{11} + B_{21}) \\ P_5 &= (A_{11} + A_{12})(B_{22}) \\ P_6 &= (-A_{11} + A_{21})(B_{11} + B_{12}) \\ P_7 &= (A_{12} - A_{22})(B_{21} + B_{22}) \end{aligned}$$

$$T(n) = 8 \cdot T(n/2) + O(n^2) = O(n^{\log 8}) = O(n^3)$$

그러나 쉬트라센의 공식을 이용하면... Voila !!!!

$$AB = \begin{pmatrix} P_1 + P_4 - P_5 + P_7 & P_3 + P_5 \\ P_2 + P_4 & P_1 - P_2 + P_3 + P_6 \end{pmatrix}$$

$$T(n) = 7 \cdot T(n/2) + O(n^2) = O(n^{\log 7}) = O(n^{2.373})$$

4) 현대 컴퓨터는 곱셈도 하드웨어로 하기 때문에 곱셈에 대한 부담이 전혀 없다.

Lecture 5 : 분할정복법의 변형 - Doubling 기법

많은 알고리즘 교재에서는 전통적인 분할정복법 예제를 사용하고 있지만 실제 현실에서는 이 더블링 방법이 더 많이 요구된다. 이 방법을 정확하게 분할정복법과 일치하지는 않지만 한 쪽이 열린 공간에서 분할정복의 취지를 사용하는 전형적인 기법이므로 잘 익혀두어야 한다. 실제 예제로 다루어보지 않으면 생각이 잘 떠오르지 않는 개발방법론이다.

분할정복법의 기본 철학은 탐색 공간의 크기를 줄이는 것이다. 즉 전체가 아니라 어떻게든 탐색해야 할 공간의 크기를 한정함으로서 빠른 계산을 도모할 수 있다. 그런데 앞서의 한 작업의 결과를 그대로 원용하면 우리가 원하는 목표점까지 더 빨리 달릴 수 있다. 더블링 대신 트리플링(tripling)도 가능하지만 더블링 방식이 분석에 더 용이하므로 이 방법이 선호된다.

5.1 재미있는 문제, x가 주어져 있을 때 이 값의 100승 즉 x^{100}을 구해보자.
단 조건은 사용하는 곱셈의 수를 최소로 하는 것이다.

방법 1. 짐승의 방법	**방법 2. 사람의 방법**

S=1 ;
for(i=1; i <= 100; i++)
{
  S *= x ;
} // 100번의 곱셈
	// 8번의 곱셈
	a = x*x ; // a=x^2
	w= a = a*a ; // w= x^4, 저장
	a = a*a ; // a= x^8
	a = a*a ; // a= x^16
	t= a = a*a ; // t= x^32
	a = a*a ; // a= x^64
	S = a*t*w // 64+32+4=100

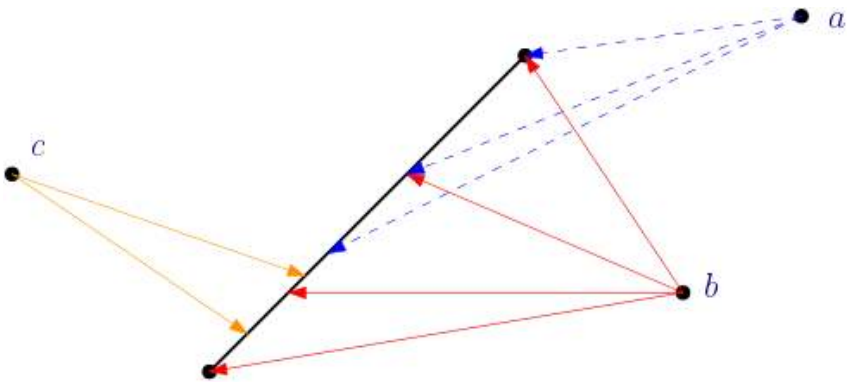
방법.2에서 사용한 곱셈의 수는 그 안에 표시된 연산자 '*'와 같다. 즉 8번이다. 100번과 8번은 엄청난 차이가 있다.

Q) x^{1000} 일때 방법-2와 같이 하면 몇 번의 곱셈이 필요한지 계산해보시오.

5.1 분할정복법 알고리즘의 변형- Doubling :
어떤 수, d와 M이 있다. 우리는 d를 거듭제곱해서 M을 넘지 않으면서 가장 가깝게 만들고 싶다. 즉 $d^e \le M$ 을 만족하는 e중에서 가장 큰 수를 구하고 싶다. 즉 만일 d=2 이고 M=1500 이라면 e=10 이 된다. 이 문제를 분할정복의 변형인 doubling을 이용해서 구해보자.

단 우리는 매우 좋은 정수 곱셈 연산 소자(GPU?) 가 있어 Multiple(X,Y)는 빠른 시간에 구할 수 있다. 여러분은 이 소자인 Multiple( )의 사용횟수를 최소화시켜야 한다.

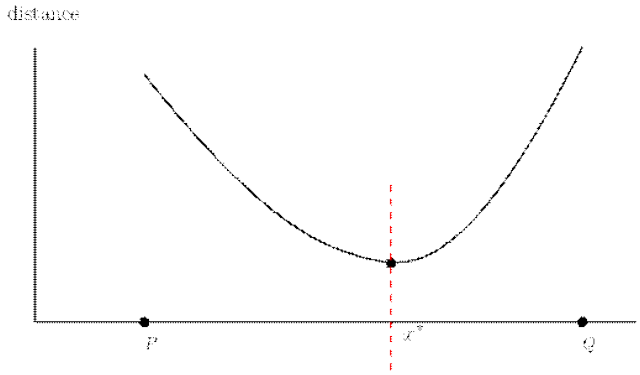
5.2 어떤 3차원 공간상에 있는 점 $X(x,y,z)$에서 유한한 길이의 선분 (P,Q)을 연결하는 최소 정수 길이. 즉 X에서 선분 (P,Q)까지 가는 어떤 통로를 건설하려고 하는 데 그 통로를 건설하는 길이는 정수만 가능하다.



이 문제는 중고등학교 때 수학을 이용한 수식으로도 구할 수 있지만 이 장에서 설명하는 계산적 방식으로 아주 우아하게 찾을 수 있다. 이 문제의 핵심을 공간 상의 한 점과 직선상의 점과의 거리는 monotone 하다. X에서 PQ상의 한 점과의 거리를 측정해보는 과정을 이렇게 생각해보자. PQ상에 있는 점을 편의상 w라고 하자. 우리는 X에서 w까지의 거리를 계속 계산하여 그 값이 최저일 때는 찾아야 한다. 그 점 w가 P에서 출발하여 등속으로 움직여 Q에 도착한다고 가정하고 두 점 X와 w의 거리를 측정해보면 그 길이의 증가 감소는 3가지 경우로 나타난다.

case 1: 계속 길어지거나(단조 증가, monotonic increasing) , case 2: 계속 짧아지거나 (monotonic decreasing), case 3) 짧아졌다 최저점에 도달한 뒤에 다시 길어지는, 이 3가지 경우만 가능하다. 이 말은 전체에서 X와의 길이가 최저인 점은 하나만 존재한다는 것이다. 이 경우 점 X와 선분 PQ에 있는 w의 최저점을 이분 탐색으로 찾을 수 있다.

PQ의 중앙점 m을 찾는다. 이 m을 기준으로 어느 쪽으로 더 기울어지는지를 조사한다. 만일 m와 w의 길이가 $m+\epsilon$의 위치의 길이와 $m-\epsilon$와의 길이보다 작으면 바로 m위치가 local minimum이 된다. 아래 그림을 참고해보자.



도전문제) 만일 한 점과 선분이 아니라, 두 개의 선분이라고 할 때 두 선분의 거리[5]

5.3 밑 빠진 배열에서 원소 찾기 :

아주 큰 어떤 배열 X[ ]이 있고 그 안에 원소는 모두 크기 순서대로 정렬되어 있다. 이 배열에서 특정한 원소 K가 있는지를 계산하고자 하는데, 문제는 이 배열의 끝이 어디인지를 모른다는 것이다. 따라서 우리가 이전에 배운 binary searching을 제대로 사용할 수가 없다. 즉 이 밑 빠진 배열의 모양은 아래와 같다. 아래 표에서 위는 index이고 아래는 값이다. 이 경우 주어진 원소 Q가 있는지 있다면 어디(index)에 있는지 알아보고자 한다면 어떻게 해야 할 것인지 설명하시오.

10245	10246	10247	10248	10249	10250	10251	10252	10253
179187	390239	490786	10983675	0	0	0	0	0

Doubling의 시간 복잡도 분석 (주어진 배열에서 마지막 원소 찾기)

a. 만일 for loop을 달리면... $\Theta(n)$의 시간을 피할 수 없다.

b. 점프하는 간격을 2배씩 늘여가면서 첫 번째 0을 찾음,

$$T(n) = \log_2 n + T(n/2)$$

$$= T(n/2) + \log_2 n$$

$$= T(n/4) + \log n/2 + \log n$$

$$= T(n/8) + \log n/4 + \log n/2 + \log n$$

$n = 2^k$ 라고 가정한다. 일단은. 그러면 다음의 식을 구할 수 있다.

5) 개체 A, B의 거리는 가장 짧게 연결 거리를 의미한다. 즉 min distance(a,b), where $a \in A$, $b \in B$

$$T(n) = T(n/8) + \log n/4 \ + \log n/2 \ + \log n$$

$$= T(n/2^k) + \log\frac{n}{2^{k-1}} + \log\frac{n}{2^{k-2}} \cdot \cdot \cdot + \log 2^k$$

$$= T(1) + \log 2 + \log 2^2 + \log 2^3 + \cdots + \log 2^{k-1} + \log 2^k$$

$$= T(1) + 1 + 2 + 3 \cdots + (k-1) + k$$

$$= 1 + \frac{k(k+1)}{2} = \Theta(k^2)$$

$$= \Theta(\log^2 n)$$

Q) 만일 2배씩이 아니라 3배씩 Jump 구간을 증가시키면서 진행한다면 시간복잡도는 어떻게 될지 생각해보자. (이 문제는 Binary search와 Tenary search의 비교와 같다.)

5.4 피보나치(Fibonacci) 수열을 Matrix 곱으로 빠르게 구하기

$$\begin{pmatrix} F_n \\ F_{n-1} \end{pmatrix} = \begin{pmatrix} 1\,1 \\ 1\,0 \end{pmatrix} \begin{pmatrix} F_{n-1} \\ F_{n-2} \end{pmatrix} = \begin{pmatrix} 1\,1 \\ 1\,0 \end{pmatrix} \begin{pmatrix} 1\,1 \\ 1\,0 \end{pmatrix} \begin{pmatrix} F_{n-2} \\ F_{n-3} \end{pmatrix} = M^k$$

F_{100}을 구하려면 M= $\begin{pmatrix} 1\,1 \\ 1\,0 \end{pmatrix}$을 99제곱을 한 뒤 $\begin{pmatrix} F_1 \\ F_2 \end{pmatrix}$을 곱하면 된다. 따라서 문제는 M^{100}을 빨리 구하는 것이다. 이것은 다음과 같이 하면 빠르게 할 수 있다.

$$M^2 \rightarrow M^4 \rightarrow \ldots M^{64}$$

5.5 (2018년 중간고사) 각 코드에서 basic_operation( )의 1회 수행을 기본동작으로 할 때 이 동작이 몇 번 수행되는지 N의 함수로 나타내시오. 만일 closed form이 가능하면 closed form으로, 만일 어려우면 가장 유용한 점근적 함수로 나타내시오.

code 1	복잡도 분석 (위 5.3번 문제)

`read N ;  # Python-style`
 	
`pos = 0 ; jump = 1`
 	
`while( pos <= N  ) :`
`  **basic_operation( ) ;**`
`  if pos + jump > N :`
`  jump = 1 # jump reset`
`  pos += jump`
`  else :`
`  pos += jump`
`  jump = jump *2`	

code 2	복잡도 분석

`read N ;  # Python-style`
 	
`k = 1`
`while( k <= N ) :`
`  S = N`
`  while( S > 0 ):`
`  S = S - k`
`  **basic_operation( )**`
`  k = k+1`	

5.6 Quick Sort와 Merge Sort의 같은 점, 다른 점

$$Q(n) = \underset{\text{분할}}{n} + \underset{\text{왼쪽 정복}}{Q(n-k)} + \underset{\text{오른쪽 정복}}{Q(k)}$$

Quick Sort 순서는 분할을 먼저, 그 다음 각각을 정복함.

$$M(n) = \underset{\text{분할}}{1} + \underset{\text{왼쪽 정복}}{M(n/2)} + \underset{\text{오른쪽 정복}}{M(n/2)} + \underset{\text{합병}}{n}$$

Merge Sort는 분할을 할 필요가 없다.

Lecture 6 : 분할정복법 알고리즘 구현 주의사항

분할정복법은 개념상으로 매우 간단한 모형이지만 이것을 실제 코딩으로 구현하는 것은 간단하지 않다. 어슬프게 이해한 분할정복법으로 구현할 경우 단순한 "막코딩"보다 더 긴 시간이 걸린다. 대부분의 코딩 교재를 보면 resursive function을 설명하기 위하여 일반적으로 피보나치 수열, 즉 $F_n = F_{n-1} + F_{n-2}$의 계산을 간단한 코딩을 설명한다. 아래 Python code의 예를 보자.

```
def Fibo( N ) :
      if N == 1 : return(1)
      if N < 0  : raise( ERROR )
      return( Fibo(N-1) + Fibo(N-2) )
```

그러나 실제 위와 같이 작성을 하면 실제 Fibonacci 수만큼의 연산이 소요되어 Fibo(100)정도만 호출하여도 오류가 발생할 수 있다.[6] 이와 같이 수식으로 제시된 대로 그대로 분할정복법 기반 알고리즘을 구현하면 큰 문제가 발생할 수 있다. 이 장에서는 그 여러 주의사항을 실제 문제와 함께 설명해본다.

6.1 분할정복법으로 구성된 알고리즘의 복잡도 분석하기

- 대부분은 recursion 수식 P로 표현된다.
- 만일 P가 closed form이 나오면 다행으로 진행하면 된다.
- 그러나 closed form이 없다면 할 수 없이 Asymptotic notation으로 계산하여 우리는 그 Order에 대해서만 짐작할 수 있다.

6.2 분할정복법 알고리즘들의 시간복잡도 분석과 Recursion 풀기

- Recursion 식의 구조 (수식의 몸체와 기본 조건(base condition))
- Recursion 식과 Closed form의 구성
- Example 모든 Boundary condition의 값은 1, All(1) = 1 ;

$W(n) = W(n-1) + n-2$;	$Q(n) = 2\ Q(n-1) + 1$;
$T(n) = T(n/3) + n$;	$P(n) = P(\sqrt{n}) + 2$;
$R(n) = 2R(n/2) + n$;	$S(n) = 3\ S(n/2) + n$;

6) Fibonacci 수열의 차수는 exponential이다. N=100일 경우 100 : 354224848179261915075 = 3 x 5^2 x 11 x 41 x 101 x 151 x 401 x 3001 x 570601로 계산된다.

6.3 Recurrence 수식에서 Asymptotic Order 구하기

- Master Theorem $T(n) = aT(n/b) + f(n)$
- Recursive 식은 손으로 몇 번 전개해보는 것이 제일 좋다.
- D&C의 시간복잡도 분석하기

a) 전체 구조를 보고 재귀식을 구한다. 이때는 combine time에 유의한다.
b) base condition T(1), T(2)를 결정한다.
c) 반드시 수식을 손으로 몇 단계 전개해 다.
d) 데이터의 개수 n은 특성에 따라서 적절히 한정한다.
$n = 2^k$, $n = 3^k$과 같이 예쁘게 만든다. (이것이 가능한 이유 ?)
e) 만일 closed form이 있으면 우리는 아주 happy !
아니면 $\Theta(\ )$, 그도 아니면 $O(\ )$notation을 구해본다.

Recurrence MASTER THEOREM

if $T(n) = aT(n/b) + O(n^d)$

$$T(n) = \begin{cases} O(n^d) & , \text{ if } d > \log_b a \\ O(n^d \log n) & , \text{ if } d = \log_b a \\ O(n^{\log_b a}) & , \text{ if } d < \log_b a \end{cases}$$

6.4 **분할정복법으로 구현할 때의 주의할 점 (아주 중요)**

- 주어진 문제를 언제, 어떻게, 얼마나 잘게 쪼갤 것인가 ?
예) 수박을 분자단위, 소립자 단위까지 쪼갤 수는 없다. ?
- 적절한 단위가 되면 "무식하게" 바로 해결하는 것이 더 나은 방법이다.
- Improved Quick Sort
// N<16인 경우에는 insertion sort가 가장 빠르다고 알려져 있다.

- Improved Binary Search
- Binary(10)과 Linear(10)중에서 어떤 것이 빠른지 측정할 필요 있음
이 경우 분할작업을 중지할 데이터 크기의 임계점을 찾아야 한다.

6.5 문제를 더 이상 분할하지 않는 크기의 임계값 (Critical Value)을 구하는 문제

Q) 주어진 거리(S)가 있을 때 가장 빨리 갈 수 있는 매체를
차례대로 나열해 보고, 그 임계값을 정하시오.

- 걸어가기, 뛰어가기, 자전거, 자동차, 전철, 택시, 기차, 비행기
- 항상 비행기가 가장 좋은 수단인가 ?

- 자전거와 승용차의 우월 임계값(거리)를 구해보시오.

6.6 분할 정복법, 또는 재귀적 프로그래밍 방법을 사용해서는 안 되는 경우

- 쪼갤수록 그 가짓수가 늘어나는 경우, 실익이 없다.
  예) $T(N) = T(N-1) + T(N/2) + 1$
- Recursion overhead가 심할 경우
- 직접 해결할 방법이 존재하는 경우

Q) $F(n) = F(n-1) + F(n-2)$ 를 recursion으로 짜보기

6.7 실제 분할정복 프로그램으로 구현하여 수행 시간 측정해 보기

- Merge Sorting (병합정렬)의 임계 상수 (critical constant)값 N_0는 얼마 ?
- Quick Sorting (퀵 정렬)을 구현할 때
  임계 상수를 다르게 하여 성능을 측정해본다.

6.9 Nearest Points-Pair : N개의 점 중에서 가장 가까운 두 점을 찾아라.

a) 그냥 "쌩"으로 구하면 최악의 경우 $O(n^2)$ 복잡도를 결코 피할 수 없다.

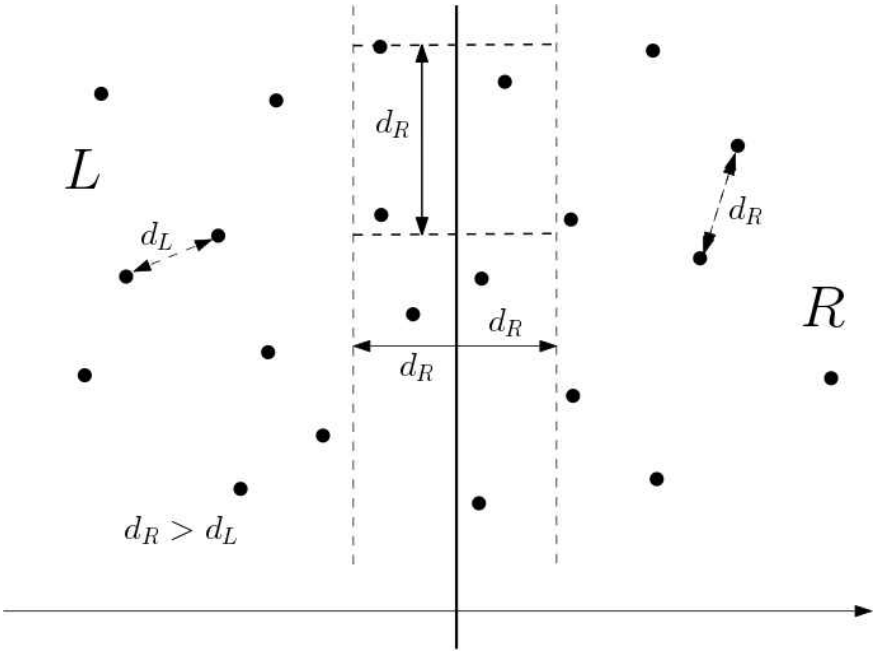
b) 그리드(grid)로 분할하여 그리드 단위로 나눠서 계산한다?

c) 분할정복법을 사용하여 nearest pair를 찾는다.

Step 1) $P_i(x_i, y_i)$을 x_i 좌표 기준으로 sorting하여 저장. $\Theta(n \log n)$
$P_i(x_i, y_i)$을 y_i 좌표 기준으로 sorting하여 저장. $\Theta(n \log n)$
가운데 점 $x_{n/2}$을 중심으로 전체를 L과 R로 두 집합으로 나눈다.
전체에서 가장 가까운 두 점은 (L,L), (L,R), (R,R) 중 하나가 된다.

Step 1) 왼쪽 집합에서 가장 가까운 쌍의 거리를 찾는다. 그 거리가 d_L
Step 2) 오른쪽 집합에서 가장 가까운 쌍의 거리를 찾는다. 그 거리가 d_R
Step 3) 구분 경계선을 기준으로 max{ d_L, d_R }의 두께 공간을 따로

잘라내서 이 벨트(belt) 안에서 가장 가까운 점을 찾는다.

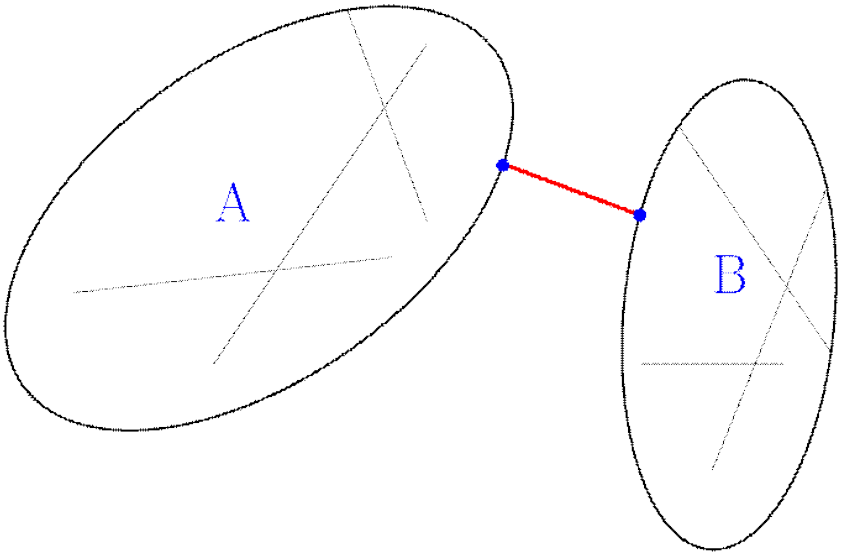


Step 4) 벨트 안에 있는 w개의 점을 y축을 중심으로 정렬한다. $\Theta(w \log w)$.

위에서부터 scan하면서 y축으로 w안에 있는 점집합에서 가장 가까운 점을 찾는다. y축으로 모든 점을 scan하면서 벨트 안에 들어있는 점을 순서대로 저장한다.

$$N(n) = N(n/2) + N(n/2) + O(n) = 2N(n/2) + O(n)$$

6.10 (도전문제) 2차원 공간에 2개의 convex object A, B가 있다. 이 둘의 거리를 구하는 알고리즘을 제시하시오. 아래 그림에서 붉은 색 segment가 그 거리를 의미한다. convex object란 그 내부의 두 점을 연결한 선분 위의 모든 점이 그 안에 존재하는 도형이다. 중요한 포인트는 대상 물체가 convex하다는 것이다.


A
B