Chapter 2. Divide and conquer

분할정복
1. 문제를 subproblems 로 나누고
2. 재귀적(Recursively)으로 subproblem 들을 해결한다
3. 해결한 답들을 combine

2.1 Multiplication

n-bit 짜리 x 와 y, n 이 2 의 배수 --> x 와 y 를 곱할때 divide & conquer 을 사용한다면

x 와 y 를 n/2-bit 의 길이로 자르면

$$x = \boxed{x_L}\,\boxed{x_R} = 2^{n/2}x_L + x_R$$
$$y = \boxed{y_L}\,\boxed{y_R} = 2^{n/2}y_L + y_R.$$

1) $xy \ = \ (2^{n/2}x_L + x_R)(2^{n/2}y_L + y_R) \ = \ 2^n\, x_L y_L \ + \ 2^{n/2}\,(x_L y_R + x_R y_L) \ + \ x_R y_R.$

여기서 significant operation: 4 개의 n/2-bit multiplication $x_L y_L,\ x_L y_R,\ x_R y_L,\ x_R y_R$

→ 크기가 n/2 인 4 개의 subproblems 로 분할

→ 따라서, recurrence relation 은

$$T(n) \ = \ 4T(n/2) + O(n).$$

T(n): overall running time

O(n): combining 하는데 소요되는 시간

→ subproblem 을 줄일 수는 없을까?

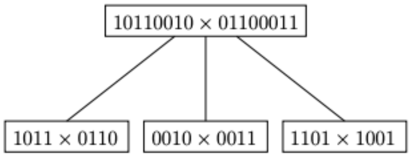
2) $(x_L+x_R)(y_L+y_R)$ 식을 이렇게 변경 - $x_L y_R + x_R y_L = (x_L+x_R)(y_L+y_R) - x_L y_L - x_R y_R$

여기서 significant operation: 3 개

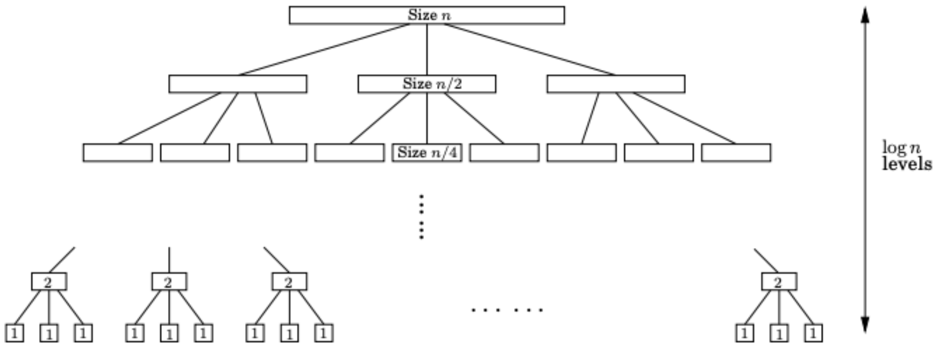
$$T(n) \ = \ 3T(n/2) + O(n).$$

```
function multiply(x, y)
Input:  Positive integers x and y, in binary
Output:  Their product

n = max(size of x, size of y)
if n = 1:  return xy

x_L, x_R = leftmost ⌈n/2⌉, rightmost ⌊n/2⌋ bits of x
y_L, y_R = leftmost ⌈n/2⌉, rightmost ⌊n/2⌋ bits of y

P_1 = multiply(x_L, y_L)
P_2 = multiply(x_R, y_R)
P_3 = multiply(x_L + x_R, y_L + y_R)
return P_1 × 2^n + (P_3 − P_1 − P_2) × 2^(n/2) + P_2
```

위의 알고리즘을 tree 로 표현하면,



→ 3 개의 subproblem 으로



→$\log_2 n$ 번째 단계에서 subproblem 의 크기가 1 이 된다.

→ depth 가 k 라면, 크기가 n/2^k 인 3^k 의 subproblem 들을 가지게 된다

$$3^k \times O\left(\frac{n}{2^k}\right) = \left(\frac{3}{2}\right)^k \times O(n)$$

- top level : k=0 → O(n)
- bottom level : k = $\log_2 n$ → $O(n^{\log_2 3})$

→ top 에서 bottom 까지 geometrically 증가

→ 증가하는 geometric series(급수)들의 합은 마지막 항: $O(n^{\log_2 3})$

* 따라서 총 running time = $O(n^{\log_2 3})$

2.2 Recurrence relations

Master theorem

: divide and conquer 방식의 알고리즘에서, 시간복잡도 계산을 편리하게 해주는 theorem

If $T(n) = aT(\lceil n/b \rceil) + O(n^d)$ ***for some constants*** $a > 0$, $b > 1$, ***and*** $d \geq 0$,

a : 몇개의 subproblem 으로 나뉘는지

b : subproblem 에 들어가는 input size 가 줄어드는 비율의 역수

c : combine 하는데 드는 시간 복잡도

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

2.3 Mergesort

List 를 두개의 sublist 로 분할하고, 각각을 recursively 정렬한 후 그들을 merge 한다.

- 만약 두개의 sorted array x,y 가 있을때 이 둘을 single sorted array z 로 merge 하려면?

  : x[1]과 y[1]을 비교하여 z[1]를 선택하고..

```
function merge(x[1...k], y[1...l])
if k = 0:   return y[1...l]
if l = 0:   return x[1...k]
if x[1] ≤ y[1]:
   return x[1] ∘ merge(x[2...k], y[1...l])
else:
   return y[1] ∘ merge(x[1...k], y[2...l])
```

- 하나의 list 를 merge sort 하는 경우

  $T(n) = 2T(n/2) + O(n)$ → 시간복잡도 : O(nlogn) -split 한 횟수 만큼 n 번의 루프를 도니까

2.3 Medians

List 를 sort 하고 medians 을 찾으면 쉽지만 시간 복잡도가 O(nlogn)이고, 필요하지 않는 부분도 sort 한다

* Selection

Input: list of numbers S, integer K

Output: the kth smallest element of S

- A randomized divide-conquer algorithm for selection (divide and conquer approach to selection)

: 숫자 v 가 있을때, S 를 “v 보다 작은 수들” ,“v 와 같은 수들”, “v 보다 큰 수들”로 나누고 각각을 S_L, S_v, S_R 라고 할 때

S : | 2 | 36 | 5 | 21 | 8 | 13 | 11 | 20 | 5 | 4 | 1 | 에서 v 가 5 라면,

S_L : | 2 | 4 | 1 | S_v : | 5 | 5 | S_R : | 36 | 21 | 8 | 13 | 11 | 20 |

: 만약 8 번째로 작은 수를 찾는 것이 문제라면…

: S_L의 크기가 3, S_v의 크기가 2 이기 때문에 찾고자 하는 8 번째 작은 수는 S_R의 3 번째로 큰 수이다

  즉, selection(S,8) = selection(S_R , 3)

$$\text{selection}(S,k) = \begin{cases} \text{selection}(S_L,k) & \text{if } k \le |S_L| \\ v & \text{if } |S_L| < k \le |S_L| + |S_v| \\ \text{selection}(S_R, k - |S_L| - |S_v|) & \text{if } k > |S_L| + |S_v|. \end{cases}$$

  → 새로운 memory 할당이 필요없다

  → 가장 이상적인 상황: ½로 나눌 수 있는 v 를 뽑은 경우

  $|S_L|, |S_R| \approx \frac{1}{2}|S|$.

  → 이상적인 상황에서 running time

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

  → 여기서 v 는 S 에서 랜덤하게 뽑는다

- Efficiency analysis

Running time 은 v 를 선택하는 시간에 많이 의존한다.

Worst case: v 를 list 에서 가장 큰 수나 가장 작은 수를 선택해서, split 될때 [원소하나][나머지 원소]로 되는 경우

$n + (n-1) + (n-2) + \cdots + \frac{n}{2} = \Theta(n^2)$ → worst case 에서 median 을 구할때

Best case: O(n) → 모든 n 에 대해 split 하는 연산이 끝인 경우
→ 평균적으로는 best-case !

- 50%의 확률로 good 을 뽑을 수 있다고 한다면, good 을 뽑기 위해 얼마나
 V 가 25^{th}~75^{th} 에 포함된 경우 good 이라고 하자.
 Good v 를 뽑은 경우 S_L 과 S_R 의 최대 크기: S 의 3/4
 * Lemma: On average a fair coin needs to be tossed two times before a "heads" is seen
 → 평균적으로 2 번에 한번은 good v 가 뽑힌다 = 평균적으로 2 번 split operation 을 하면 리스트의 크기는 ¾로 줄어든다.

2.5 Matrix multiplication

nxn matrices X, Y 가 있을 때, Z = XY 이면

$$Z_{ij} = \sum_{k=1}^{n} X_{ik} Y_{kj}$$

시간 복잡도: O(n)을 n^2 번 수행하므로 → $O(n^3)$

* matric multiplication 을 divide and conquer 로!
- n x n 행렬을 n/2 x n/2 행렬로

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

 : 8 개의 subproblems
 : $O(n^2)$번의 더하기 연산
 → $T(n) = 8T(n/2) + O(n^2)$ → 기존의 연산법과 동일

- 7 개의 subproblem 들로

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

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

 → $T(n) = 7T(n/2) + O(n^2)$
 → $O(n^{\log_2 7}) \approx O(n^{2.81})$.

2.6 The fast Fourier transform

이제 polynomials 에 대해 divide and conquer

$A(x) = a_0 + a_1 x + \cdots + a_d x^d$, $B(x) = b_0 + b_1 x + \cdots + b_d x^d$

$C(x) = A(x) \cdot B(x) = c_0 + c_1 x + \cdots + c_{2d} x^{2d}$

C(x)에서 각각의 원소를 c_k 라고 할 때,

$$c_k = a_0 b_k + a_1 b_{k-1} + \cdots + a_k b_0 = \sum_{i=0}^{k} a_i b_{k-i}$$

→ c_k 를 계산할 때 O(k)

→ 2d+1 는 $\theta(d^2)$ 왜 d^2??

→ Fast Fourier transform 을 이용해서 줄여보자! FFT 를 사용하면 O(nlogn)이 된다

2.6.1 An alternative representation of polynomials

Polynomials 의 두가지 표현

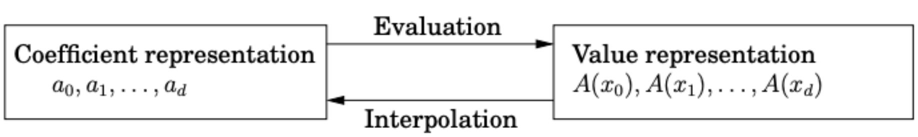
$A(x) = a_0 + a_1 x + \cdots + a_d x^d$ 이런 polynomials 가 있을 때,

1) Coefficient representation

$a_0, a_1, \ldots, a_d$

2) Value representation

$A(x_0), A(x_1), \ldots, A(x_d)$



→ 이렇게 변환이 가능해야 한다