Design Techniques

① incremental approach - Insertion Sort $O(n)$ ↑ $O(n^2)$ ↗ in place

② divide and conquer - merge sort & quick sort memo~ in place ↑ Randomized Algorithm ↑ $O(n^2)$ ↙ 7.4.1

③ using data structure - heap sort $O(n \lg n)$ in place

④ Dynamic programming - Assembly s. problem (linear time) ↑

⑤ greedy Algorithms

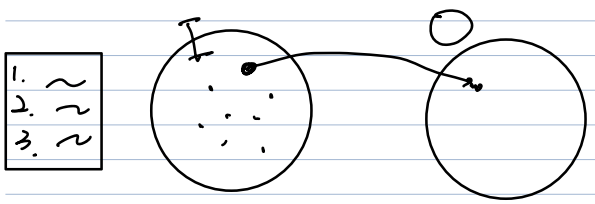
Sorting problem ↑ $O(n^2)$ Alg

Chapter 1

Definition for computational problem

Given { input – an instance (infinity many)
output

Alg → clear specification of compute output



- finite # of steps
- correctness

randomized Algorithm
quicksort – 7.4.2
selection of a pivot.

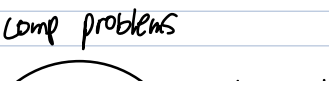
can be a error but exceptionally small possibly

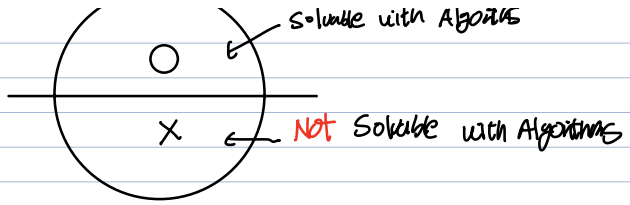
잘못된 결과는 낼 수도 있다 매우 작다.

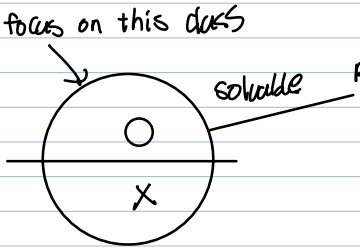
Halting problem

Input: P: Arbitrary written program with any language (Java, C, Python etc)
X: input for P

output: yes if P(X) halts
no otherwise







Anything that can help Algorithms to solve problem
Time, space, random bits, interaction, etc

→ Time
# of steps

1. worst case
2. best case
3. average case

} 3 가지 라인공부

Space
main memory

1 Insertion - In place sorting
2. merge - for 100 element [ ]
for 1000 elemat [ ] a lot more.
3. heap - In place
4. quick - In place

Random bits
2. Randomized quick-sort [n] $n = 10 \rightarrow p(n) = \frac{1}{10}$

↳ It is becomes possible to solve problem with rand(1, n) $\simeq \frac{1}{n}$

Randomized Quick sort with rand → $O(n \lg n)$ // average case complexity

Interaction

1. 2 coloring problem
2. Non-isomorphism problem

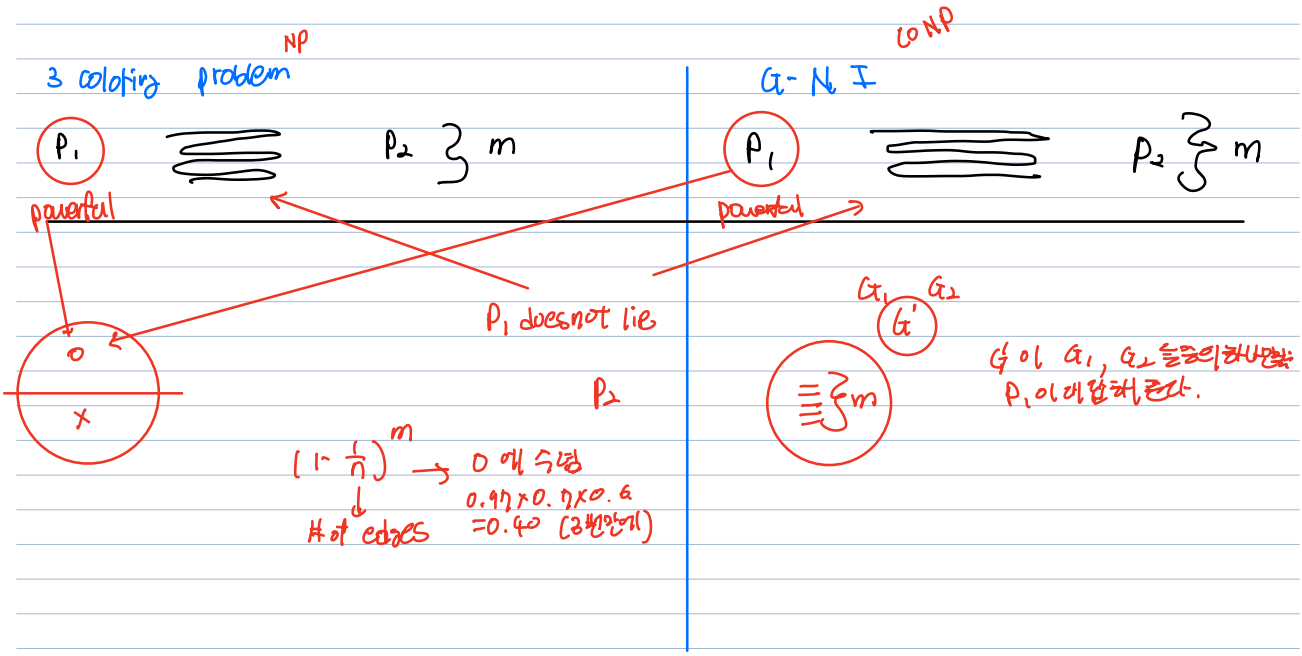
$P_1 \leftrightarrow P_2 \rightarrow P_2$ can get some information from P_1

P_1: can compute anything that soluble

Network band width

→ How much data can be sent trough network

Do not deal with it.



chapter 2

Insertion sort (A)

{1, 2, 3, 4, 5}

5! ways of input is given with the same elements

1, 2, 3, 4, 5 ⋮ 5, 4, 3, 2, 1 } 5! → $\frac{1}{n!}$

best case

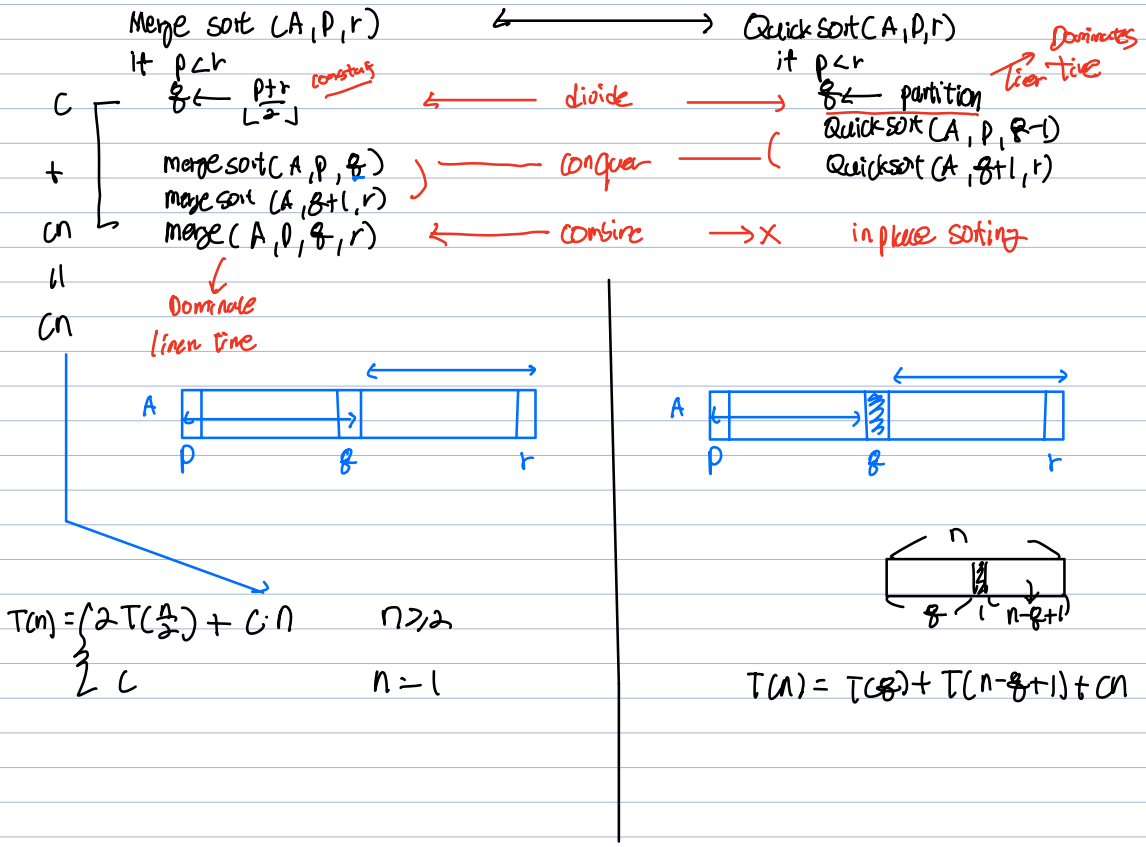
worst case

Insertion sort 같은 randomized - insertion sort 라고.

Design an algorithm

Input : How this input is given (Distribution of input)

output



$$T(n) = \begin{cases} 2T(\frac{n}{2}) + c \cdot n & n \geq 2 \\ c & n = 1 \end{cases}$$

$$T(n) = T(q) + T(n-q+1) + cn$$

Chapter 3

$n \to \infty$

Asymtotic notation

$O(\quad) = \{ \quad \}$

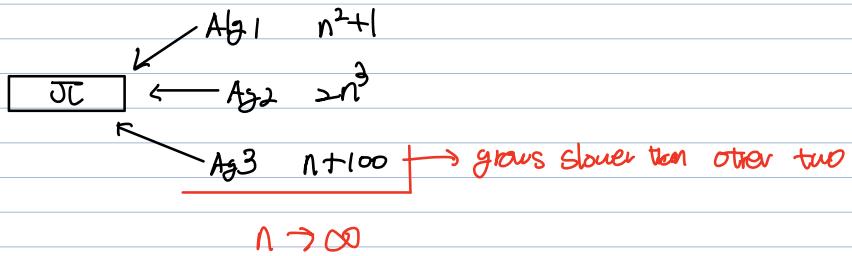
$\Omega(\quad) = \{ \quad \}$

$\Theta(\quad) = \{ \quad \}$

$o(\quad) = \{ \quad \}$

$\omega(\quad) = \{ \quad \}$

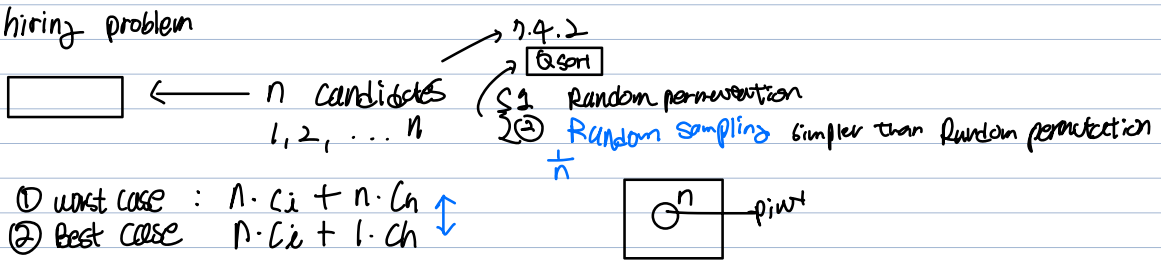
} set of runtime



ch29. simplex Alg → cpp
$O(2^n)$ → $O(n^3)$
worst

Chapter 5

hiring problem



① worst case : $n \cdot c_i + n \cdot c_h$ ↕
② best case $n \cdot c_i + 1 \cdot c_h$

① c_h : hiring cost
② c_i : interview cost

$c_i < c_h$

# of times an assistant is hired

n ——— 1

Expected [hired]

$= O(\lg n)$

Impose a distribution → random permutation → $\frac{1}{n!}$ ~ $O(\lg n)$

Lemma 5.1 I : indicator random variable

n $n!$
ex) 1, 2, 3
$n=3$ 2, 1, 3

$Exp(I_{(A3)}) = pr\{A3\}$

2,3,1
total 1,3,2
2,1,2
3,2,1

$\int_1^n \frac{dx}{x} = \log n$

$\{1 + \frac{1}{2} + \frac{1}{3} + \dots \frac{1}{n}\} \hookrightarrow \log n$

O(lgn)

Assumptions
① distinct different completeness
② rand permutation
any sequence out of n! possibility → $\frac{1}{n!}$

count the total # of times that an assistant is hired
X

E[X] = ?

=> 만일 X를 회사에서 최대로 고용하는 횟수라고 놓으면 우리가 구하려는 값은 X의 기대값이며 X는 확률 변수입니다.
그런데 X는 다음 방식으로 계산할 수 있습니다.
만일 i번째 인터뷰한 사람이 고용되는 경우 Xi가 1, 그렇지 않은 경우 Xi가 0이라고 한다면
X는 X1+X2+X3+....+Xn이 됩니다.
따라서 X의 기대값은 X1+X2+X3+...+Xn의 기대값과 같은데 여기서 linearity of expectation에 의하면
X1+X2+X3+...+Xn의 기대값은 X1의 기대값 + X2의 기대값 + X3의 기대값 +.... + Xn의 기대값과 같습니다.
그런데 lemma 5.1에 의하면 indicator random variable의 기대값은 해당 indicator random varialbe과 연관된
사건이 일어날 확률과 같습니다. 따라서 X1의 기대값은 처음 인터뷰한 사람이 고용될 확률인데 회사 입장에서
보면 처음에는 누가 오든지 고용하므로 그 확률은 1입니다. X2는 두 번째 인터뷰한 사람이 고용될 확률인데
그것은 처음에 누가 인터뷰 했는가에 따라 다릅니다, 예를 들어 네 명의 지원자들이 있다고 할 경우 모두 4!인
24가지 경우, 즉 어떤 순서로 인터뷰하는가의 경우들이 있는데 이들 중 12번이 X2 값이 1이 되는 경우입니다.
그 이유는 예를 들어 rank 1,2,3,4 순서대로 인터뷰 했다면 두 번째 사람보다 첫 번째 사람이 우수하므로 X2=0입니다
반면 2,1,3,4 라면 두번 째 사람의 rank는 1이고 첫 번째 사람의 rank는 2이므로 두 번째 사람이 회사 입장에서
첫 번째 사람보다 우수하므로 고용됩니다. 이와 같은 방식으로 분석한다면 n명이 인터뷰 할 경우 X1=1
X2=1/2, X3=1/3... 이라는 것을 알 수 있습니다. 이제 이들을 모두 더하면 1+1/2+1/3+1/4... 이 되고 대략 log n에
수렴 (미적분한 공식을 이용)합니다.

Chapter 6

Heapsort

Alg ↔ DS

Ins Sort
merge sort
Quicksort
w/o DS

Heapsort w DS

Binary Heap [Max Heap / min Heap

O(n)

① Build - max heap
② max-heapify O(h)
③ Heapsort ← O(nlgn)

~~FIFO~~ - queue.

→ bottom up 방식 may work

Chapter 15 DP — opt. prop

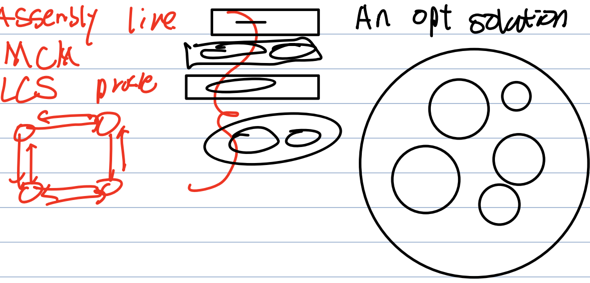
table

① optimal substructure property
② overlapping subproblems property (a table)
③ greedy choice property

①, ② → DP will work
③ → GA ch16.

① Assembly line
② MCM
③ LCS prove
④

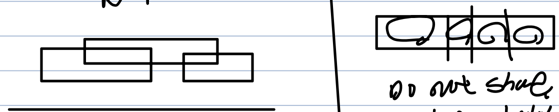
An opt solution "shape"



Overlapping subproblem property

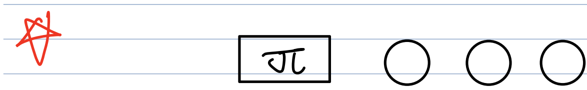
$$ {}_{n}C_{r} = {}_{n-1}C_{r} + {}_{n-1}C_{r-1} $$

DP | D & C



DP: structure rule, overlapping subproblem

D & C: do not share, independent



check overlapping part