**4.3-1**



**1. Substitution method**

If. T(n) < c\*n^2

T(n) < c\*(n-1)^2 + n

< c\* n^2 – 2cn + c + n

= c\*n^2 – ( 2c\*n - n –c) … desired–residual

<= c\*n^2.

Whenever c + n <= 2cn. For example, if c>=1, n>=1.

**2. recursion tree**

T(n) = T(n - 1) + n

= T(n – 2) + (n – 1) + n

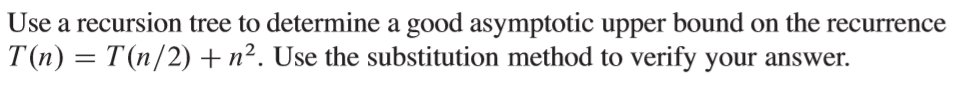
….

= T(0) + 1 + 2+ .. n-2 + n-1 + n ….. arithmetic series

= T(0) + n(n+1) / 2

= O(n^2)

**4.4-2**



n^2

(n/2)^2

(n/4)^2

….

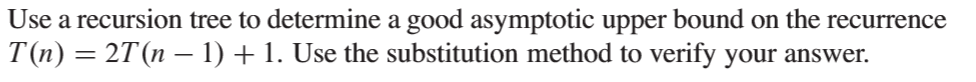
Therefore

= O()

**Substiution**

Whenever = . T(n) = O()

**4.4-4**



**Recursion tree**

Tree is binary tree.

Height: n

Cost: 1

Max Num of Tree node: 2^h+1 – 1

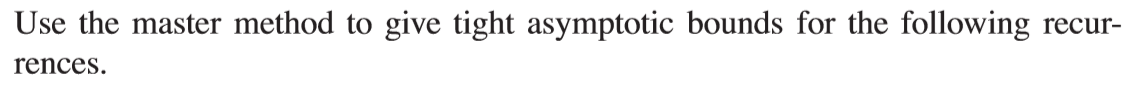
Therefore T(n) = O(2^n)

**Substiution**

– c

Whenever = . T(n) = O()

**4.5-1**





-> T(n) = Θ(



-> T(n) = Θ( lg n)



-> T(n) = Θ(



-> T(n) = Θ(

**7.1-1**



a. 13, 19, 9, 5, 12, 8, 7, 4, 21, 2, 6, 11

b. 13, 19, 9, 5, 12, 8, 7, 4, 21, 2, 6, 11

c. 13, 19, 9, 5, 12, 8, 7, 4, 21, 2, 6, 11

d. 9, 19, 13, 5, 12, 8, 7, 4, 21, 2, 6, 11

e. 9, 5, 13, 19, 12, 8, 7, 4, 21, 2, 6, 11

f.. 9, 5, 13, 19, 12, 8, 7, 4, 21, 2, 6, 11

g. 9, 5, 8, 19, 12, 13, 7, 4, 21, 2, 6, 11

h. 9, 5, 8, 7, 12, 13, 19, 4, 21, 2, 6, 11

i.. 9, 5, 8, 7, 4, 13, 19, 12, 21, 2, 6, 11

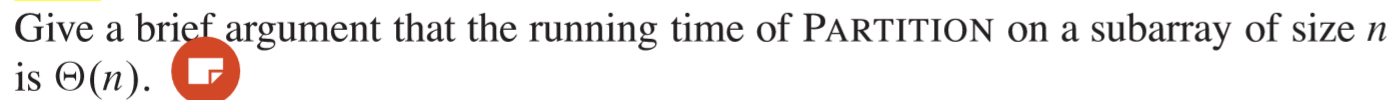
j.. 9, 5, 8, 7, 4, 13, 19, 12, 21, 2, 6, 11

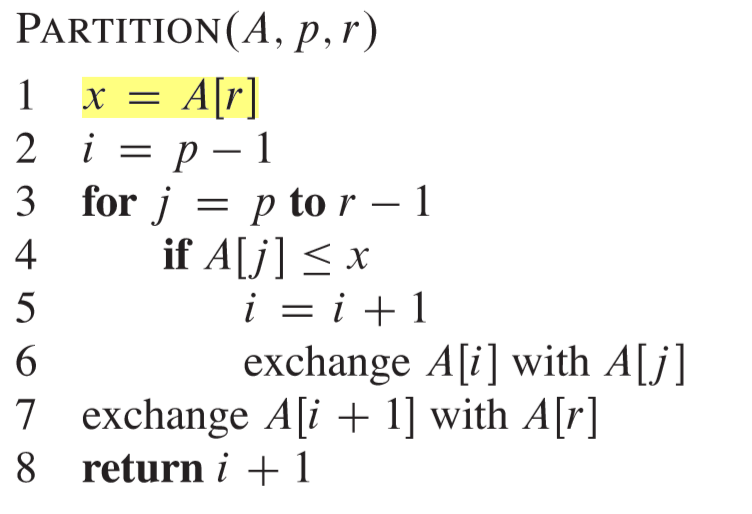
k. 9, 5, 8, 7, 4, 2, 19, 12, 21, 13, 6, 11

l. 9, 5, 8, 7, 4, 2, 6, 12, 21, 13, 19, 11

m. 9, 5, 8, 7, 4, 2, 6, 11, 12, 21, 13, 19

**7.1-3**



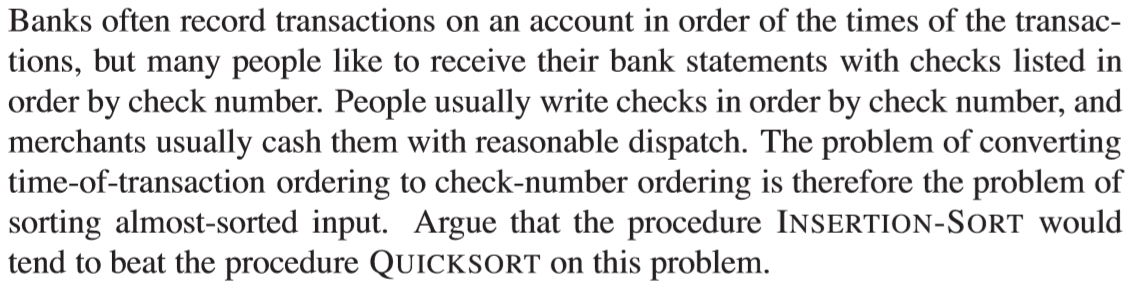


Subsrray of size = n = r – p. +1

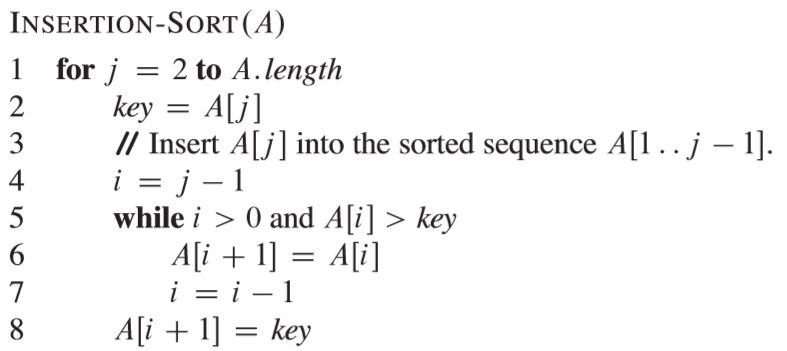
For loop iteration: r-p = n – 1

Therefore PARTITION = O(n)

**7.2-4**



Insertion sort:

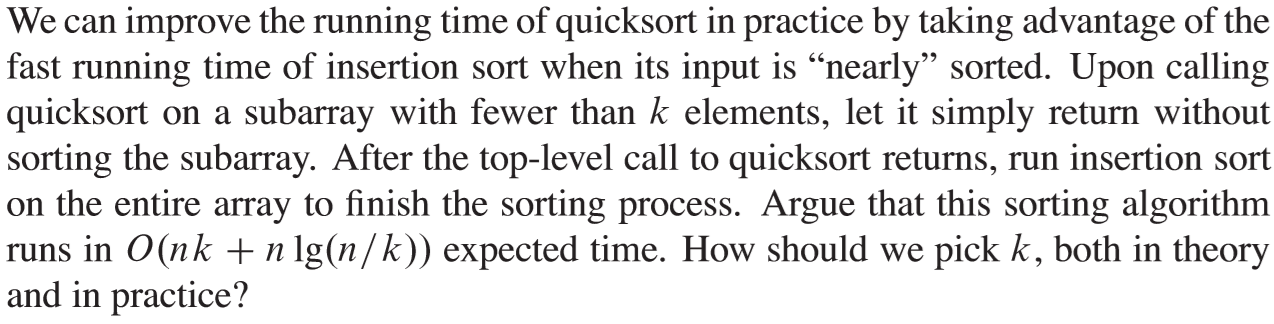


만일 배열 A가 작은 상수 c개를 제외한 모든 원소들이 잘 정렬 되어있고, 전체가 거의 정렬되어 있다면, while문은 c\*d 번 실행된다. (d는 아주 작은 상수) 따라서 이 문제는 Θ(n + c\*d), 즉 Θ(n)의 시간이 걸린다.

Quick sort:

배열 A가 전체적으로 거의 정렬되어있다면, partition은 n-1 : 1 로 나뉘어지는 경우의 수가 많아진다. 결국 worst case인 Θ(n^2) 이거나, expected case인 Θ(nlogn)룰 벗아나지 못한다.

**7.4-5**



**Quick sort**

부분 배열의 크기가 k개 보다 작은 부분 배열에서 리턴되면

PARTITION의 수행시간은 그대로 O(N),

TREE의 높이는 lg n/k 이 므로 (n/k개의 잎들)

Quic sort는 총 O(n\*lg(n/k))의 복잡도를 가진다.

**Insertion sort**

크기가 k이하인 부분배열을 n/k개를 가지는 배열을 insertion sort하는 것은

각각의 부분 배열을 insertion sort하는것과 동일한 문제이다.

(부분배열 s1, s2, ….sn/k, 일때 i<j, si<sj 이므로)

각각의 insertion sort는 O(k^2)의 복잡도를 가지고 n/k번 실행된다

따라서 O(nk)의 복잡도를 가진다.

Or

전체의 배열의 원소들각각은 최대 k번 비교만 일어나므로 O(n\*k)이다.

**Theory**

Quick sort가 O(nlgn) Insertion sort와 Quick sorts는 O(nk + n lg(n/k)) 시간에 동작하므로 nk + n\* lg(n.k) <= nlgn, k + lgn – lgk <= lgn, k <= lgk 인 k를 고르면되지만 k > 1이 되어야 하므로 불가능하다.

**Practical**

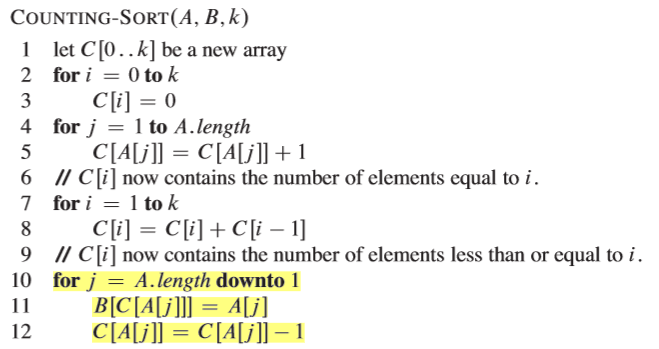
상수인자들을 아래와 같이 무시하지않거나 여러 번 시행 착오를 거쳐 k값을 선택해야한다..

Cq \* n lg n >= Ci \* n\*k + Cq \* n lg n/k

Lg k >= (Ci/Cq )\* k

**8.2-2**





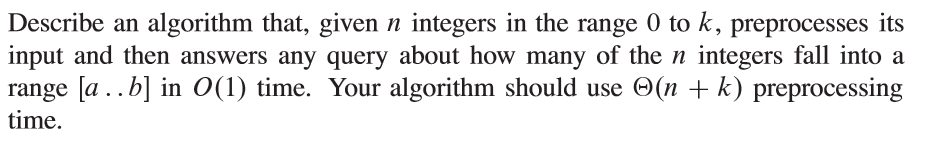
3번째 for문에서 배열 C에 i가 올 수 있는 최대 인덱스를 저장하고, 4번째 for문에서 A를 마지막 인덱스부터 루프를 돌면서 B에 C에서 저장했던 최대 인덱스를 참조하여 A값을 집어 넣고 C에서 저장했던 최대 인덱스를 감소시킨다.

결국 A배열에서 나중에 나타나는 것이 B배열에서도 나중에 나타나게되고 먼저 나타나는 것이 먼저 나타나 같은 숫자가 입력 배열에 있던 것과 같은 순서로 출력 배열에 정렬된다. 그러므로 COUNTING-SORT는 안정성을 가진다.

OR

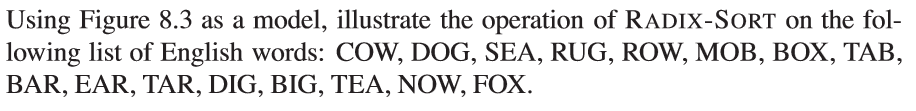
Assume A[i] =A[j], i < j -> 마지막 인덱스부터 집어 넣으므로 그 순서가 유지된다.

**8.2-4**



Counting sort의 1~9라인으로 입력을 전처리한 후 C[b] – C[a-1] (C[0] = 0)을 호출하면 range(a,b)안의 개수를 구할 수 있다.

**8.3-1**



COW SEA TAB BAR

DOG TEA BAR BIG

SEA MOB EAR BOX

RUG TAB TAR COW

ROW RUG SEA DIG

MOB DOG TEA DOG

BOX DIG DIG EAR

TAB BIG BIG FOX

BAR BAR MOB MOB

EAR EAR DOG NOW

TAR TAR COW ROW

DIG COW ROW RUG

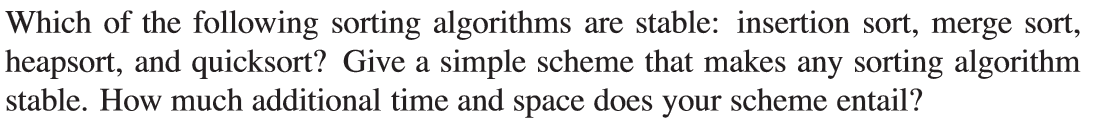
BIG ROW NOW SEA

T EA NOW BOX TAB

NOW BOX FOX TAR

FOX F OX RUG TEA

**8.3-2**



Insertion sort는 루프를 돌고 있는 반복자이하의 인덱스는 다 정렬되어 있고 반복자에서 1씩 감소하여 비교하므로 fifo형태가 나온다. 그러므로 stable하다.

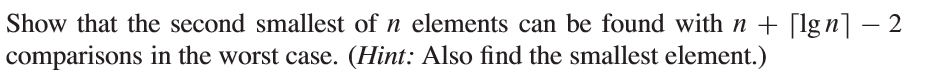
Merge sort에서 두 배열을 merge가 실행되어 두 값을 비교할 때 같은값이면 앞의 인덱스가 앞에오므로 stable하다.

Heap sort에서는 배열을 binary tree로 나타내므로 어느 쪽 트리에서 삽입될지 모르므로 unstable하다.

Quick sort에서 partition이 실행될 때 pivot과 작은 배열 부분은 stable하나 그것보다 큰 부분 배열은 뒤의 원소와 교체가 빈번하게 일어나므로 unstable하다.

index를 저장해논 배열을 생성하거나 (key, index)형태로 배열을 만든다. 비교가 일어날때 key값을 먼저 고려한후 key값이 같을 때 index를 비교하면 stable하게 만들 수 있다. Index를 저장해논 배열을 생성하므로 O(n)의 공간이 더 필요하지만 수행시간은 상수배 될뿐이므로 변함 없다.

**9.1-1**

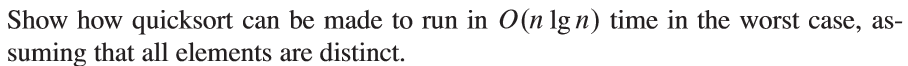


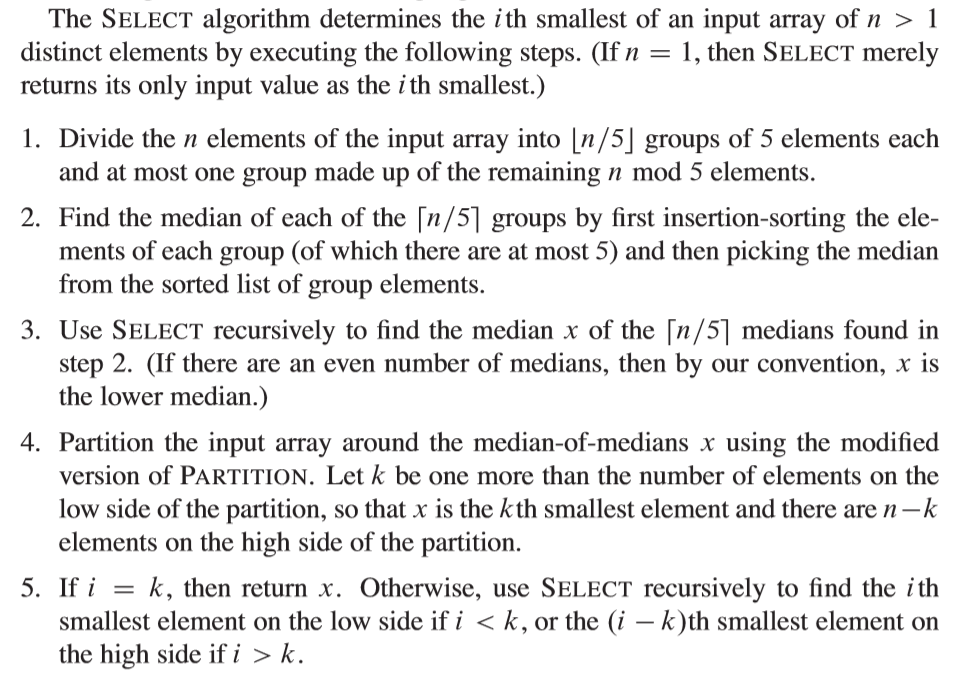
1. 배열의 원소를 두개씩 묶어 비교하고 작은값끼리 다시 묶어 비교하여

2. 최솟값을 찾는다. (n-1)

3. 마지막 최솟값과 비교한 값과 최솟값이 상대한 값들끼리 비교하여 얻은 최솟값을 비교하면 2번째 작은값을 구할 수 있다. (lgn – 1) (1은 처음 최솟값 비교)

**9.3-3**



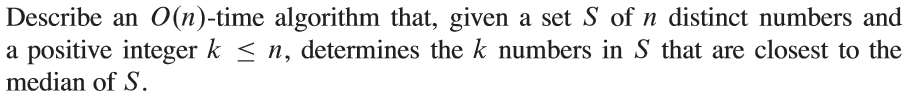


피벗을 고르는데 Select를 사용하면 평균적으로 고르게 나눌 수 있다.

최악의 경우 T(n) = T(7n/10 + 6) + T(3n/10 – 6) + O(n) 시간에 정렬된다.

Quick sort가 T(n) = T(n-1) + T(0) + O(n) 으로 파티션되는 경우를 제외하하면 시간 복잡도가 O(nlgn)임을 7장에서 배웠다. 그러므로 select을 사용하는 Quick sort는 최악의 경우는 O(nlgn)이다. (or ex 4.4-9)

**9.3-7**



1. selec을 사용하여 O(n)만에 중간값을 구한다.(중앙값기준으로 partiotion됨)

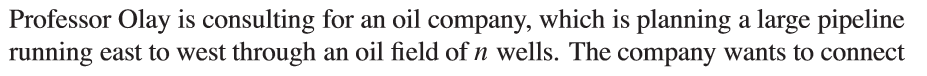
2. 배열 전체를 중앙값을 뺀값에 절대값을 취한다, +,-인지 저장 (O(n))

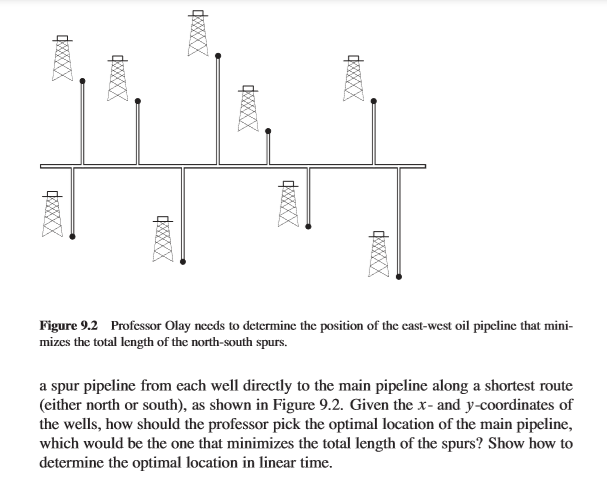
(ex. (3,+))

3. 중앙값 기준으로 파티션 다시 O(n)

4. subarray에서 k-1번째값을 찾고 파티션 (1은 중앙값제외함) O(n)

5. subarray에 k-1까지 루프돌면서 차이값 더하거나 뺌

**9.3-9**



N개의 y좌표들의 평균(중앙값)을 구하면 O(n)만에 최적의 위치를 고를 수 있다.