알고리즘	시간 복잡도
Sum Array	$T(n) = n + 2$
Brute Force Largest Zero-Sum	$T(n, m) = \Theta(nm) + \Theta(n^2 m^2) \in \Theta(n^2 m^2)$ $T(n) = (n+1)(m+1) \in \Theta(nm)$
BF-String Matching	$T(n) = (n+1)(m+1) \in \Theta(nm)$
BF-Closest Pair	$T(n) = \frac{n(n-1)}{2} \in \Theta(n^2)$
BF-Convex Hull	$T(n) = \Theta(n^3)$
Merge Sort	nlogn
Decrease & Conquer (power set)	$T(n) \in \Theta(n2^n)$
Divide & Conquer (binary search)	$T(n) = \lg n + 1$
Transform & Conquer	$T(n, m) \in \Theta(s.log_2(s), s = max(n, m)$
Buble Sort	$T(n) = \frac{n(n-1)}{2} \in \Theta(n^2)$
Insertion Sort	$T_b(n) = \sum_{i=1}^{n-1} 1 = n - 1 \in \Theta(n)$ $T_w(n) = \sum_{i=2}^{n} \sum_{j=1}^{i-1} 1 = \frac{n^2 - 3n + 2}{2} \in \Theta(n^2)$ $T_a(n) = \sum_{i=2}^{n} \frac{i-1}{2} = \frac{n(n-1)}{4} - \frac{n-1}{2} \in \Theta(n^2)$
Heap Sort	$T_{buildheap}(n) = \Theta(n)$ $T_{heapify}(n) = \Theta(h) = \Theta(logn)$ $T_{heapSort}(n) = \Theta(nlogn)$

Merge Sort	$T(n) \in \Theta(n \log_2 n)$
Quick Sort	$T(n) = \sum_{i=2}^{n} 1 = n - 1$ (avg) $T(n) = 2nlogn \in O(nlogn)$ (worst) $T(n) = \frac{n(n-1)}{2} \in \Theta(n^2)$
Lower Bounds by Comparison (Exchange Sort)	$T(n) = \frac{n(n-1)}{2} \in \Theta(n^2)$
Counting Sort	$T(n) = 2k + 2n - 1$ $\in \Theta(n + k)$
Radix Sort	$T(d, n, k) = d[2k + 2n - 1]$ $\in \Theta(d(n + k))$
Binary Search	$T(n) \in \Theta(log_2 n)$
Sequential Search	$T(n) \in \Theta(n)$
Interpolation Search	$\Theta(n)$
Partition	$T(n) = \sum_{i=2}^{n} 1 = n - 1$
Fake Coin Problems	$T(n) = \begin{cases} T(1) = 1 & n = 1 \\ T(\lceil \frac{n}{3} \rceil) + 2 & n > 1 \end{cases}$ $T(n) = log_3(n) + 2$
Post Office Location	
Closest Pair	$T(n) = \frac{n(n-1)}{2} \in \Theta(n^2)$ (in k-dimensional space) $T(n) = k\left[\frac{n(n-1)}{2}\right] \in \Theta(kn^2)$ (by divide conquer) $T(n) \in \Theta(nlogn)$

Convex Hull	(brute force) $\Theta(n^3)$ (divide and conquer) $T(n) \in \Theta(nlogn)$

week02

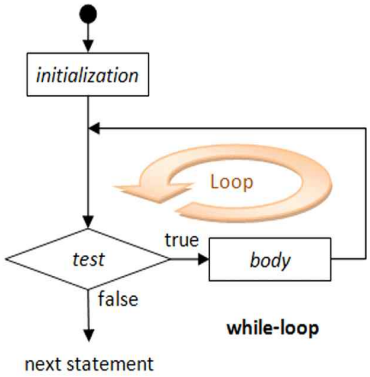
1. Interative vs Recursive

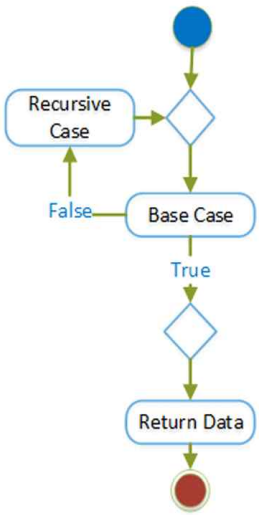
Interative(순환)

: 조건 초기화문이 거짓이 될 때까지 명령 세트를 반복 실행하는 프로세스이다.
대화형 알고리즘은 알고리즘을 실행하는 동안 사용자 입력을 수반하는 알고리즘입니다. 이러한 유형의 알고리즘은 사용자로부터 입력을 받아 계산을 수행한 다음 사용자에게 출력을 제공합니다. 대화형 알고리즘의 예로는 사용자에게 데이터를 입력하거나 프롬프트에 응답하도록 요청하는 프로그램이 있습니다.

Recursive(재귀)

: 문제 자체를 하나 이상의 간단한 버전으로 분할한다. base case가 없으면 재귀 알고리즘은 끝없이 실행된다.
해결책을 찾을 때까지 더 작은 문제로 분해하여 문제를 해결하는 알고리즘





Recurrence Relation	closed form Solution
$\begin{cases} T(1)=1 \\ T(n)=T\left(\frac{n}{2}\right)+1, \ n>1, \ n \text{ is power of } 2 \end{cases}$	$T(n)=\log n+1$
$\begin{cases} T(1)=1 \\ T(n)=7T\left(\frac{n}{2}\right), , n>1, \ n \text{ is power of } 2 \end{cases}$	$T(n)=n^{\log 7} \approx n^{2.81}$
$\begin{cases} T(0)=0 \\ T(1)=1 \\ T(n)=T(n-2)+T(n-1), \quad n>1 \end{cases}$	$T(n)=\frac{\left[\frac{(1+\sqrt{5})}{2}\right]^n-\left[\frac{(1-\sqrt{5})}{2}\right]^n}{\sqrt{5}}$
$\begin{cases} T(0)=0 \\ T(n)=T(n-1)+n-1, \quad n>0 \end{cases}$	$T(n)=\frac{n(n-1)}{2}$

$\begin{cases} T(1) = 0 \\ T(n) = 7T\left(\frac{n}{2}\right) + 18\left(\frac{n}{2}\right)^2, \quad n > 1,\ n \text{ is power of } 2 \end{cases}$	$T(n) = 6n^{\log 7} - 6n^2$

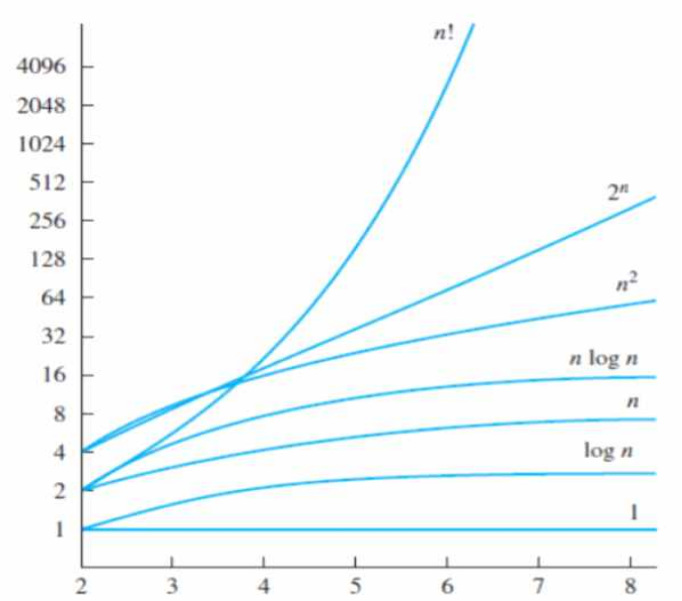
week03

order of growth

시간 복잡도:시간 복잡도는 알고리즘이 입력 크기의 함수로 실행되는 데 걸리는 시간 알고리즘 분석법은 경험적/실험적 분석과 이론적/형식적 분석으로 나뉜다.

이론적/형성적 분석

: 이 분석법은 알고리즘의 입력 크기와 관련된 시간복잡도를 결정하는 것을 포함한다. 점근적 의미에서 순서표기법, Big-O표기법을 사용해시간 복잡성을 추정하는 것이 일반적이다. 빅 오메가(Δ) 표기법 및 빅 세타() 표기 또한



알고리즘의 점근적 분석은 실행시간을 점근적 표기법, Big-O표기법, Big-Omega표기법, Big-Seta 표기법으로 나타낸다.

1. Big-O 표기법

: 알고리즘의 점근적 상한선을 설명하기 위한 표기법 (최대치)(최악의 경우)

- **Example-1:** We will show that $g(n) = T(n) = n^2 + 10n \in O(n^2)$

$$
\begin{aligned}
T(n) &= n^2 + 10n \\
f(n) &= n^2 \qquad\qquad \text{Show that } T(n) \leq c \times f(n) \text{ for } n \geq N \\
n^2 + 10n &\leq 2n^2 \qquad\qquad \text{for } c = 2,\ N = 10 \text{ holds} \\
T(n) = n^2 + 10n &\in O(n^2) \qquad\qquad T(n) \in O(f(n))
\end{aligned}
$$

2. Big- 표기법

: 알고리즘의 점근적 하한선을 설명하기 위한 표기법(최소치)("너는 언젠가 내 안의 함수보다 커질거야")

- **Example-1:** we will show that $T(n) = g(n) = n^2 + 10n \ \in \Omega(n^2)$

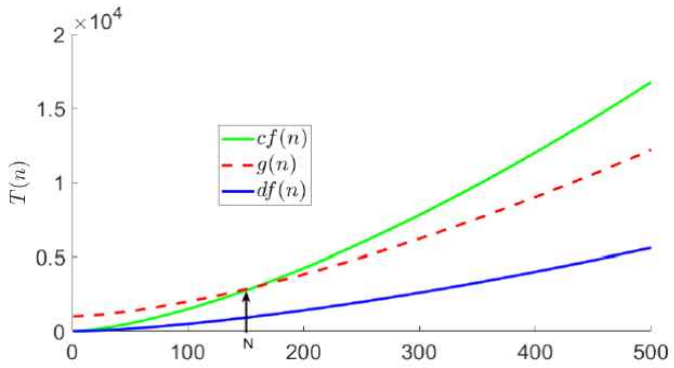
$$
\begin{aligned}
T(n) &= n^2 + 10n \\
f(n) &= n^2 \qquad\qquad \text{need to show } T(n) \geq c \times f(n) \text{ for } n \geq N \\
n^2 + 10n &\geq n^2 \qquad\qquad \text{for } c = 1,\ N = 0 \text{ holds} \\
T(n) = n^2 + 10n &\in \Omega(n^2) \qquad\qquad \text{as } T(n) \in \Omega(f(n))
\end{aligned}
$$

- **Example-2:** we will show $T(n) = g(n) = \frac{n(n-1)}{2} \in \Omega(n^2)$

$$
\begin{aligned}
T(n) &= \frac{n(n-1)}{2} \\
f(n) &= n^2 \qquad\qquad \text{need to show } T(n) \geq c \times f(n) \text{ for } n \geq N \\
\frac{n(n-1)}{2} &\geq \frac{1}{4}n^2 \qquad\qquad \text{for } c = \frac{1}{4},\ N = 2 \text{ holds} \\
T(n) = \frac{n(n-1)}{2} &\in \Omega(n^2) \qquad\qquad \text{as } T(n) \in \Omega(f(n))
\end{aligned}
$$

3. Big-

: 상한선과 하한선의 중간이 평균치("너는 내 안의 함수와 동등한 비율로 증가해")



$$\lim_{n\to\infty}\frac{g(n)}{f(n)}=\begin{cases}0, & g(n)\in O(f(n))\\ \infty, & g(n)\in\Omega(f(n))\\ c, c>0 & g(n)\in\Theta(f(n))\end{cases}$$

- It means:
  - if $\lim_{n\to\infty}\frac{g(n)}{f(n)}=0$: then intuitively $g(n)<f(n)\Longrightarrow g(n)=O(f(n))$ or $g(n)\in O(f(n))$ and $g(n)\neq\Theta(f(n))$.
  - if $\lim_{n\to\infty}\frac{g(n)}{f(n)}=\infty$: then intuitively $g(n)>f(n)\Longrightarrow g(n)\in\Omega(f(n))$ and $g(n)\neq\Theta(f(n))$.
  - if $\lim_{n\to\infty}\frac{g(n)}{f(n)}=c, c>0$: then intuitively $g(n)=c\cdot f(n)\Longrightarrow g(n)\in\Theta(f(n))$.

- Example-3 Let $T(n)=g(n)=log_2 n$, $f(n)=\sqrt{n}$

$$\begin{aligned}\lim_{n\to\infty}\frac{g(n)}{f(n)}&=\lim_{n\to\infty}\frac{log_2 n}{\sqrt{n}}=\lim_{n\to\infty}\frac{(log_2 n)'}{(\sqrt{n})'}\\&=\lim_{n\to\infty}\frac{\frac{log_2 e}{n}}{\frac{1}{2\sqrt{n}}}=2log_2 e\lim_{n\to\infty}\frac{1}{\sqrt{n}}\\&=0\end{aligned}$$

Hence $T(n)=g(n)=log_2 n\in O(\sqrt{n})$

- Example-4: Let $T(n)=g(n)=n!$, $f(n)=2^n$

$$\begin{aligned}\lim_{n\to\infty}\frac{T(n)}{f(n)}&=\lim_{n\to\infty}\frac{n!}{2^n}\\&=\lim_{n\to\infty}\frac{\sqrt{2\pi n}\left(\frac{n}{e}\right)^n}{2^n}\\&=\lim_{n\to\infty}\sqrt{2\pi n}\left(\frac{n}{2e}\right)^n\\&=\infty\end{aligned}$$

Hence $T(n)=g(n)=n!\in\Omega(2^n)$

4. Master Theorem

: 특정 유형의 반복 관계에 대한 솔루션을 얻을 수 있는 직접적인 방법
recursive(재귀 알고리즘)에서 계산하기 편한 방법

$$\begin{cases}T(1)=d, & n=1\\ T(n)=aT\left(\frac{n}{b}\right)+cn^k, & n>1\end{cases}$$

$$\begin{cases}T(n)\in\Theta\left(n^k\right) & a<b^k \quad (case-1)\\ T(n)\in\Theta\left(n^k\lg n\right) & a=b^k \quad (case-2)\\ T(n)\in\Theta\left(n^{\log_b a}\right) & a>b^k \quad (case-3)\end{cases}$$

Recurrence Relation	Solution Using Master Theorem
$\begin{cases} T(1) = 3 \\ T(n) = 8T\left(\frac{n}{4}\right) + 5n^2, n > 1 \end{cases}$	Since, $a = 8,\ b = 4,\ k = 2$ $\Rightarrow 8 < 4^2 \Rightarrow a < b^k \Rightarrow (case - 1)$ $\Rightarrow T(n) \in \Theta\left(n^2\right)$
$\begin{cases} T(1) = 7 \\ T(n) = 9T\left(\frac{n}{3}\right) + 5n^1, n > 1 \end{cases}$	Since, $a = 9,\ b = 3,\ k = 1$ $\Rightarrow 9 > 3^1 \Rightarrow a > b^k \Rightarrow (case - 3)$ $T(n) \in \Theta\left(n^{\log_3(9)}\right) \Rightarrow \Theta\left(n^2\right)$
$\begin{cases} T(64) = 200 \\ T(n) = 8T\left(\frac{n}{2}\right) + 5n^3, n > 64 \end{cases}$	Since, $a = 8,\ b = 2,\ k = 3$ $\Rightarrow 8 = 2^3 \Rightarrow a = b^k \Rightarrow (case - 2)$ $T(n) \in \Theta\left(n^3 \lg n\right)$

사례기반 분석 유형

1. 최상의 경우

: 알고리즘이 가장 빠르게 실행되는 입력값을 정의한 것으로 최소 입력이 아닌 입력 크기를 말한다.

2. 최악의 경우

: 알고리즘에 최대 시간이 걸리는 입력값을 정의한 것으로 알고리즘의 최악의 효율성을 만든다.
알고리즘을 분석하여 어떤 종류의 입력이 기본 연산의 가장 큰 값을 산출하는지 확인한다.

3. 평균

: 입력이 무작위라고 가정하고 알고리즘의 실행시간을 예측한다. 평균사례 효율성이 최악의 경우 효율성보다 훨씬 나은 알고리즘이 많이 있다.

상각분석

week 04

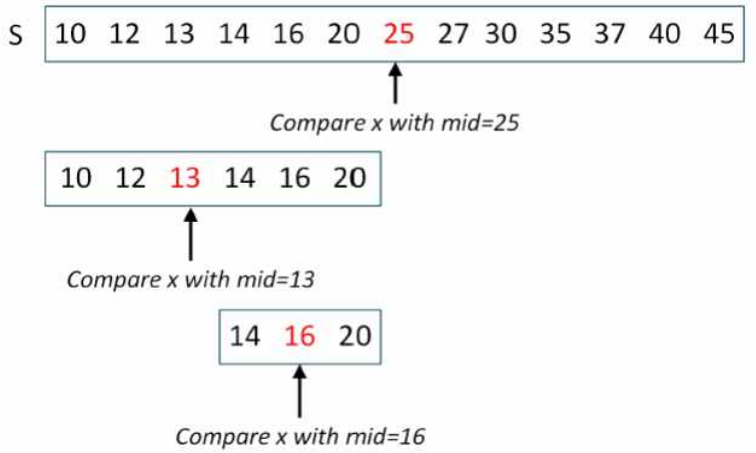
Algorithm Design Techniques

1. Brute Force

: 문제 진술과 관련된 정의에 직접 기초하여 문제를 해결하는 간단한 접근 방법이다. 만족스러운 해결책이 발견될 때까지 모든 가능성을 시도한다.

- 거의 모든 문제를 해결할 수 있는 유일한 일반적 접근방법

2. Divide and Conquer(분할정복)



ex)이진탐색

3. Decrease and Conquer

: 만약 우리가 더 작은 하위 문제를 정복할 수 있다면 결국 우리는 원래 문제를 해결할 수 있다. 주어진 문제 인스턴스에 대한 솔루션과 더 작은 인스턴스에 대한 솔루션 간의 관계를 이용하는 것을 기반으로 한다.

ex) $^{0} = (2^{5})^{2}$

- Decrease by a constant (usually by 1)

: 상수만큼 빼면서 문제를 축소시키는 방식

(ex. n -> n−1 -> n−2 -> ...)

- Decrease by a constant factor (usually by half)

: 상수만큼 나누면서 문제를 축소시키는 방식

(ex. n -> n/2 -> n/2^2 -> ...)

- Variable-size Decrease

: 문제를 축소시킬 때의 크기가 정해져있지 않은 방식

(ex. 유클리드 알고리즘 - 최대공약수 구하기)

4. Transform and Conquer
- 인스턴스 단순화
 : 동일한 문제의 더 단순하거나 편리한 인스턴스로 변환
- 표현 변경
 : 동일한 인스턴스의 다른 표현으로 변환
- 문제 감소
 : 이미 사용 가능한 알고리즘이 있는 다른 문제의 인스턴스로 변환

week05

Sorting

- 비교 기반 정렬 알고리즘

 : 비교 연산자를 사용하여 두 숫자 사이의 순서를 찾는 경우 정렬 알고리즘은 비교 기반입니다. 예: 머지, 퀵, 힙 정렬, 버블 정렬

- 논비교 기반 정렬 알고리즘

 : 이러한 알고리즘은 요소를 비교하지 않고 정렬을 수행합니다. 카운트 정렬, 기수 정렬, 버킷 정렬 예제

- 스왑 기반 정렬 알고리즘

 : 이러한 알고리즘은 요소를 스왑하여 입력을 정렬합니다. 예를 들어 교환 정렬의 경우 정렬을 선택하면 최소 스왑 수가 필요합니다.

- 재귀 정렬 알고리즘

 : 빠른 정렬, 병합 정렬과 같은 일부 정렬 알고리즘은 재귀를 사용하여 입력을 정렬합니다.

- 반복 정렬 알고리즘

 : 정렬 또는 삽입 정렬 선택과 같은 정렬 알고리즘은 중첩된 루프 또는 while-loop을 사용하여 요소를 반복적으로 정렬합니다

- 안정성

 : 정렬 후 중복 요소가 동일한 상대 위치에 남아 있으면 정렬 알고리즘이 안정적이다.
안정적 : 삽입, 병합, 버블 정렬
불안정 : 힙, 빠른 정렬

-제자리 알고리즘

: 추가 메모리를 사용하지 않고 입력을 변환(정렬)한다. 알고리즘이 실행될 때 입력은 일반적으로 출력에 의해 덮어쓰며, 이 작업에는 추가 공간이 필요하지 않다.

ㄴ 삽입, 선택, 빠른, 버블 정렬

-자리맞추기 알고리즘

: 제자리에 있지 않은 알고리즘을 제자리에 있지 않거나 자리에 있지 않은 알고리즘이다.

ㄴ 표준정합병렬 알고리즘

1. Buble Sort

: 버블 정렬 알고리즘은 배열의 각 요소 쌍을 비교하여 전체 배열이 정렬될 때까지 순서가 맞지 않으면 요소 쌍을 바꾼다.

2. Insertion Sort

: 삽입 정렬은 제자리, 안정 알고리즘이다.

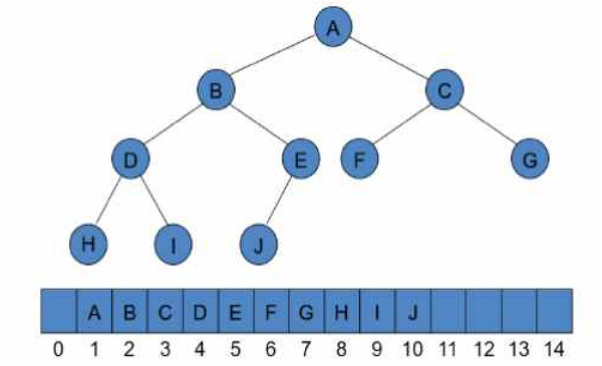
3. Heap Sort

: 힙은 힙 순서를 따르는 완전한 이진트리이다. 힙 데이터 구조는 (최대/최소) 우선순위에 따라 요소를 제거할 때 사용된다.

- 힙 순서 속성: (최소) 힙에서 모든 노드 X에 대해 상위에 있는 키가 X에 있는 키보다 작거나 같습니다.
- 힙 순서 속성: (최대) 힙에서 모든 노드 X에 대해 부모의 키가 X의 키보다 크거나 같습니다.

- 완전 이진트리

: 완전이진트리는 맨 아래 레벨을 제외하고 전부 채워진 트리이다.(맨 아래 레벨은 왼쪽에서 오른쪽으로 채워짐). 트리 레벨=log 이다.

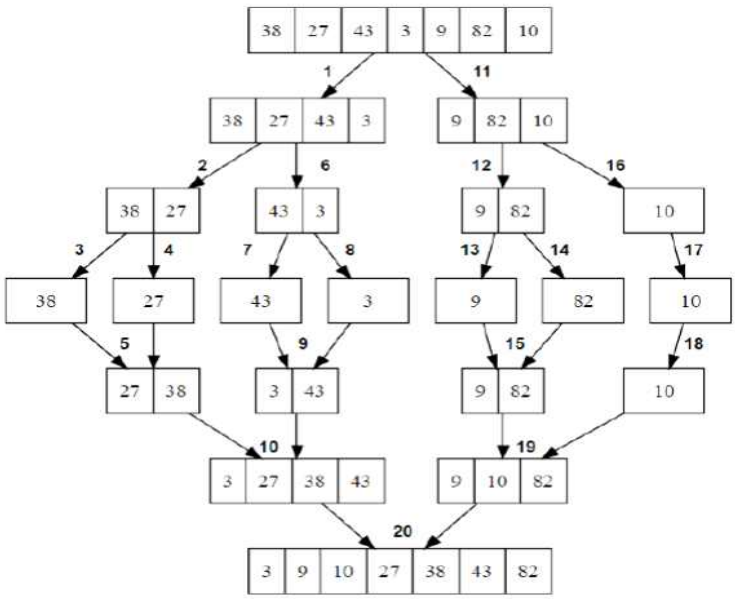


- 주어진 키에대해 힙 구성 방법

: 상향식 접근 방식에 따라 배열에서 형성된 전체 이진 트리를 역 레벨 순서로 힙화한다. 노드 p에서 키를 아래쪽으로 이동하는 hapify() 또는 percolateDown() 메서드가 있다고 가정한다. 일반적인 알고리즘은 배열에 N개의 키를 배치하고 정렬되지 않은 이진 트리로 간주하는 것이다.

- 힙 성질에 맞으면 왼쪽 안맞으면 오른쪽

4. Merge Sort



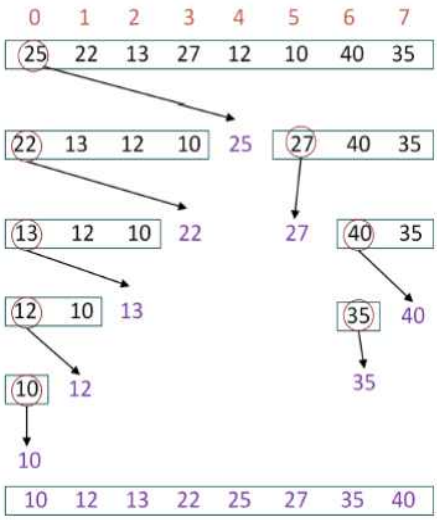
: 분할 및 정복 방식을 사용하여 배열의 요소를 정렬하는 효율적인 정렬 알고리즘

- Complexity Function

$$T(h,m) = \begin{cases} 0 & n = 1 \\ \underbrace{T(h)}_{\text{sort } U} + \underbrace{T(m)}_{\text{sort } V} + \underbrace{T(h,m)}_{\text{merge } U,V} & n > 1 \end{cases}$$

5. Quick Sort

: Quicksort는 배열을 두 개의 파티션으로 나눈 다음 각 파티션을 재귀적으로 정렬하여 정렬한다는 점에서 Mergesort와 유사하다. 피벗은 첫 번째,중간, 마지막 중에 고른다.



6. Radix Sort

: 정수 키로 데이터를 정렬하는 정수 정렬 알고리즘으로, 키를 같은 중요한 위치와 값(장소 값)을 공유하는 개별 숫자로 그룹화한다. 키 값이 동일한 요소의 상대적 순서를 유지한다. 논비교, 안정 정렬 알고리즘이다.

week06

Search

1. Binary Search
 : should be in order
2. Sequential Search
 : may not be in order
3. Interpolation Search
 : 항상 검색 키를 주어진 정렬된 배열의 중간 값과 비교하는 이진 검색과 달리, 보간 검색은 검색 키와 비교할 배열의 요소를 찾기 위해 검색 키의 값을 고려한다.
따라서 배열 크기가 동일한 두 개의 절반으로 나뉘지 않고 가변 크기 감소 메커니즘을 통해 배열이 감소한다.

week07

Convex Hull

 : 평면의 점 집합(유한 또는 무한)은 집합에서 p와 q에 끝점이 있는 전체 선 세그먼트가 집합에 속할 경우 볼록이라