

자료구조와 알고리즘

자료: data → 저장공간(memory) + 읽기, 쓰기, 삽입, 삭제, 탐색
 ↓
 입출력
 ↓
 구현한 함수 연산

알고리즘: 정답 출력

자료구조 예: ① 변수(variable) $a = 5$ ← 초기연산
 $\text{print}(a)$ ← 입출력 연산

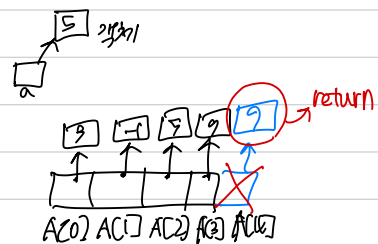
② 배열(array) $A = [3, -1, 5, 7]$

리스트(list) 접근: 원소의 index

읽기, 쓰기: $A[3]$

삽입: $A.append(9)$

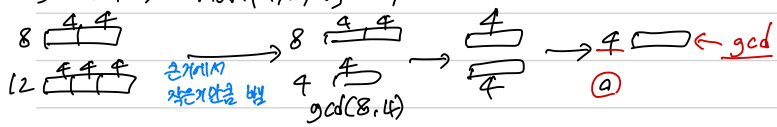
삭제: $A.pop()$, $A.pop(2)$



알고리즘 예:
 room의 정수 배열 A: 정렬
 ↓
 정렬된 정렬: 출력

• 인위적/초기 알고리즘: 최대공약수(GCD) 계산 알고리즘

• $\text{gcd}(8, 12) = \max\{1, 2, 4\} = 4$



• $\text{gcd}(a, b)$:

while $a \neq 0$ and $b \neq 0$:
 if $a > b$: $a = a \div b$
 else: $b = b \div a$
 return a or b

$\Rightarrow \text{gcd}(2, 100) \rightarrow \text{gcd}(2, 98) \rightarrow \text{gcd}(2, 96) \dots \rightarrow \text{gcd}(2, 0)$

숫자 ↑ → 숫자 ↑ → p
 7 % 2 = 1
 2 % 2 = 0

$\text{gcd}(2, 100) = \text{gcd}(2, 2)$
 1번 반복(5) 100/2 = 50

최종
 계산법

⇒ gcd-sub , gcd-mod , gcd-rec
 (비효율) (효율) (재귀)

$\text{gcd}(a, b) = \text{gcd}(a, b \% a)$ or
 $\text{gcd}(a \% b, b)$

알고리즘 시간복잡도 1

가상언어 (Pseudo/virtual Languages)

- 배열: 산술, 비교, 논리, bit 논리: 기본연산 표현
- 비교: if, if-else, if-elif-else
- 반복: for, while
- 함수: 정의, 호출, return

가상코드 (Pseudo Code)

algorithm Arraymax(A, n)

input: 배열의 원소를 갖는 배열 A

output: A의 수중에 최대값 리턴

currentMax = A[0]

for i in range(1, n):

if currentMax < A[i]:

 currentMax = A[i]

return currentMax

A = [3, -1, 9, 2, 5] n=5

C.M 1 1+1 1+1
비교 비교 비교

기본연산: 25

무한히 많은 입력 } → 시간 고려한 필요 0
무한히 많은 n
입력하기

자료구조, 알고리즘의 시간복잡도 (time complexity)

algorithm max_max(A, n):
 currentMax = A[0]

A = [2, 5, 7, 9, 15, 26] n: input size

① 모든 입력에 대해 기본연산 횟수를 더한 후 평균

① → for i in range(1, n):
 { if currentMax < A[i]:
 currentMax = A[i] (*)
 }
 return currentMax

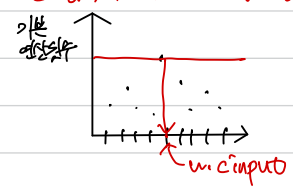
현실적으로 불가능

② 가장 안 좋은 입력 (worstcase input) 에 대한

기본 연산 횟수를 측정! worstcase time complexity

⇒ 어떤 입력에 대해서도 W.T.C. 값의 수행시간이 크지 않다!

알고리즘 = 최악의 입력에 대한
 수행시간 기본연산 횟수



(C.M. 값 → T(n) = 2n-1 → n 크면 무시할 수 있음)

예!

algorithm sum1(A, n):

① sum = 0
 for i in range(0, n):
 { ② if A[i] > 0: ③ sum += A[i] (*)
 }
 return sum

최악의 경우 = 최악의 입력에 대한 수행시간 → A: 모든 값이 양수 : W.C. 입력

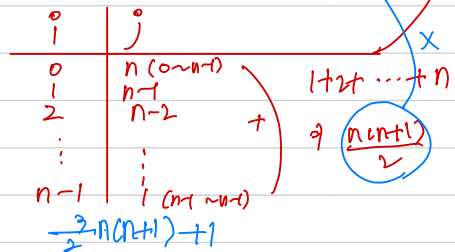
$T(n) = 4n + 1$

⇒ 알고리즘 = n에 대한
 수행시간 밀착성.

algorithm sum2(A, n)

① sum = 0
 for i in range(0, n):
 { for j in range(i, n):
 sum += A[i] * A[j]
 }
 return sum

최악의 경우 (첫 번째 입력)
 수행시간은 2이고
 그냥 수행시간
 수행하면 됨



$T(n) = \frac{n^2}{2} + \frac{n}{2} + 1$

→ n에 관한 이차함수

알고리즘 수행시간

표현가능

Big-O 표기법 : 알고리즘의 수행시간 = 최악의 경우의 입력에 대한 기본연산 횟수

Algorithm 1 : $T_1(n) = 2n - 1$
(arrayMax)

Algorithm 2 : $T_2(n) = 4n + 1$
(sum)

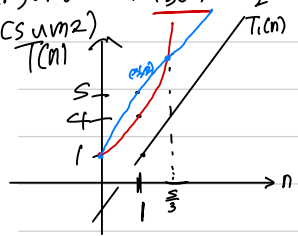
Algorithm 3 : $T_3(n) = \frac{3}{2}n^2 + \frac{3}{2}n + 1$
(sum2)

① Algorithm 2와 Algorithm 1보다 2배 느리다. (수행시간 ↑)

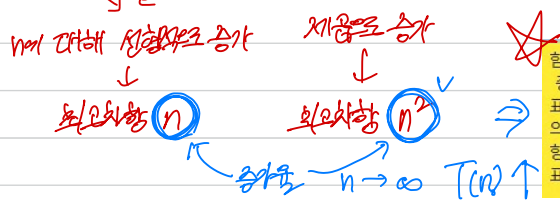
② Algorithm 3은 $n < \frac{5}{2}$ 면 Algorithm 2보다 빠르다. (수행시간 ↓)

모든 n 에 대해서 Algorithm 1보다 느리다. (수행시간 ↑)

③ Algorithm 3은 $n > \frac{5}{2}$ 면 항상 Algorithm 2보다 느리다!



$T_1(n), T_2(n), T_3(n)$



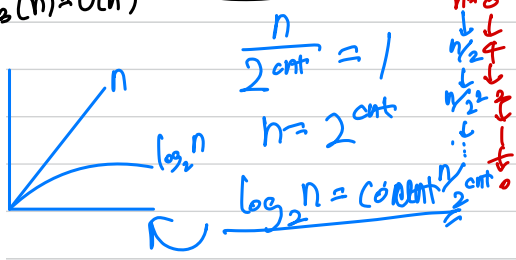
수행시간 $T(n)$ = 함수값을 결정하는 최고치향만으로 간단하게 표기: **Big-O 표기법**

$T_1(n) = 2n - 1$ $T_1(n) = O(n)$
 $T_2(n) = 4n + 1$ $T_2(n) = O(n)$
 $T_3(n) = \frac{3}{2}n^2 - \frac{3}{2}n + 1$ $T_3(n) = O(n^2)$

- ① 최고치향만 남긴다.
- ② 최고치향 계수(상수)는 생략
- ③ Big-O (최고치향)

가장함수 아래에서

$T_1(n) = O(n)$
 $T_2(n) = O(n)$
 $T_3(n) = O(n^2)$



예 1 def increment-one(a):
 return a + 1 $T(n) = 1$ $O(1)$

예 2 def number-of-bits(n):
 ① count = 0
 while n != 0:
 n = n // 2 ② $2^3 = 8 \rightarrow 4 \rightarrow 2 \rightarrow 1$
 count += 1
 return count

알고리즘
 수행시간
 함수로 표현하는 것
 이라한 함수의 최고치향만을
 수행시간을 표현할 수 있다
 ⇒ **Big-O 표기법**