

자료구조와 알고리즘

자료구조: data \rightarrow 저장장소 (memory) + 접근, 쓰기, 삽입, 삭제, 탐색 구조
 ↓
 접근: 접근 횟수의 연산

알고리즘: 정답 찾기

자료구조 예: ① 변수 (variable) $a = 5$ ← 초기화
 변수 이름

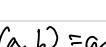
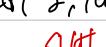
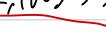
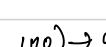
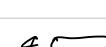
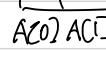
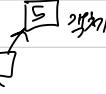
② 배열 (array) $A = [3, -1, 5, 7]$

리스트 (list) 접근: 원소의 index

입기, 쓰기: $A[3]$

삽입: $A.append(9)$

삭제: $A.pop(1), A.pop(2)$



가상언어 (Pseudo/virtual Languages)

- 반복, 산술, 비교, 논리, 복합논리: 기본연산 표현
- 비교: if, if-else, if-elif-else
- 반복: for, while
- 험수: 정의, 초기, return

가상코드 (Pseudo Code)

algorithm Arraymax (A, n)

(input: 배열의 원소를 갖는 배열 A

(output: A의 수중에서 최댓값 리턴

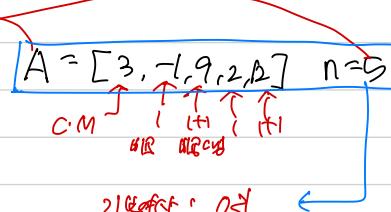
currentMax = A[0]

for i in range (1, n):

if currentMax < A[i]:

 L currentMax = A[i]

return CurrentMax



무한히 끝나지 않아?
무한히 끝나지 않아?
→ 시간고려할 필요!

자료구조, 알고리즘의 시간복잡도 (time complexity)

algorithm arraymax (A, n):

 CurrentMax = A[0]

 ① \rightarrow for i in range (1, n):

 if currentMax < A[i]:

 CurrentMax = A[i] (*)

 return currentMax

 +
 $\frac{1}{2}n-1$

 알고리즘 = 최악의 입력에 대한
 수행시간

 기본연산 횟수

$A = [2, 5, 7, 9, 15, 26]$ n: inputsize

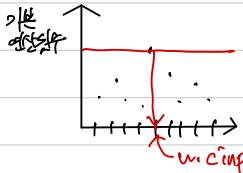
① 모든 입력에 대해서 기본연산 횟수를 더한 후 평균

 현실적으로 불가능

② 가장 안 좋은 입력 (worstcase input) 를 찾기

 기본연산 횟수를 측정: worstcase time. complexity

 → 어떤 입력에 대해서도 W.T.C.는 수행시간이 크지 않다!



$$(C \cdot M \text{번} \rightarrow T(n) = 2n-1 \rightarrow n \text{번의 } \frac{1}{2}(n-1))$$

예!

algorithm sum1 (A, n):

 ① sum = 0

 for i in range (0, n):

 if A[i] < 0: \rightarrow ② 0이면

 n \times 4 \rightarrow ② sum += A[i] (*)

 sum = sum + A[i]

 return sum

 수행시간 = 최악의 경우에 대한
 기본연산 횟수 \rightarrow A: 모든 값이 징수: W.C. 0

$$T(n) = 4n+1$$

 알고리즘 = n번의 단계
 수행시간 = 일차식.

algorithm sum2 (A, n)

 ① sum = 0

 for i in range (0, n):

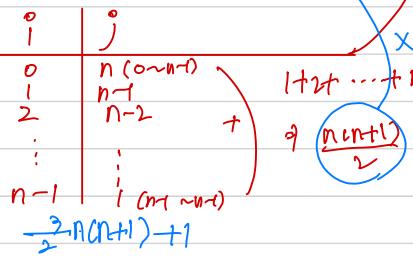
 for j in range (i, n):

 sum += A[i] * A[j]

 return sum

 최악
 수행시간과는无关

 그냥 수행시간
 제한될



$$\Rightarrow T(n) = \frac{3}{2}n^2 + \frac{3}{2}n + 1$$

\rightarrow n의 2차
 수행시간

 알고리즘 수행시간

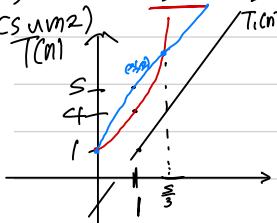
 표현 가능

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

Algorithm1: $T_1(n) = 2n - 1$
(arrayMax)

Algorithm2: $T_2(n) = 4n + 1$
(sum)

Algorithm3: $T_3(n) = \frac{3}{2}n^2 + \frac{3}{2}n + 1$
(sum2)

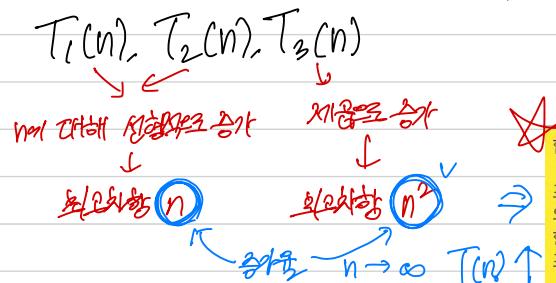


① Algorithm2가 Algorithm1보다 2배 느리다. (수행시간↑)

② Algorithm3는 n^2 을면 Algorithm2보다 빠르다. (수행시간↓)

모든 n에 대해서 Algorithm1보다 느리다. (↑↑)

③ Algorithm3은 n^2 을면 항상 Algorithm2보다 느리다!



수행시간 $T(n)$ = 할수 값을 결정하는 최고차항만으로 간단하게 표기: Big-O 표기법

$$T_1(n) = 2n - 1$$

$$T_1(n) = O(n)$$

① 최고차항만 남기자.

$$T_2(n) = 4n + 1$$

$$T_2(n) = O(n)$$

② 최고차항 계수(F)는 생략

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

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

③ Big-O(최고차항)

+합으로 아까운이

$$T_1(n) = O(n)$$



$$T_2(n) = O(n)$$

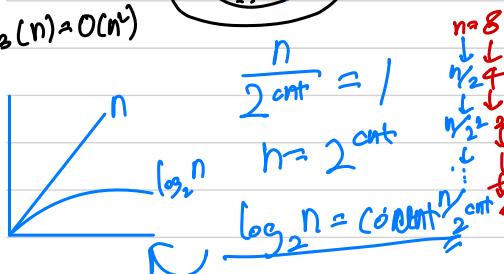
$$T_3(n) = O(n^2)$$

예 1 def increment_one(a):

$$\quad \quad \quad \text{return } a + 1 \quad T(n) = 1$$

예 2 def number_of_bits(n):

① count = 0
while $n \neq 0$:
 n = n // 2
 count += 1
return count



알고리즘
수행시간은
수행되는 횟수

이제는 수행시간은 최고차항으로
수행되는 횟수로 표현할 수 있다

⇒ Big-O 표기법