2021 KCA Algorithm Study

Day 1: What is “Algorithm”?

KU201711424

KCA Mentor **Pt J**

멘토 소개

peeeeeter_j
peeeeeter_j
peeeeeter_j
peeeeeterj
edenjint3927

- 2018년 2학기KCA 가입
- 2019년 1학기
  - KCA 자율 스터디 – <운영체제 랜선 스터디> 진행
- 2019년 여름방학
  - KCA비공식 스터디 <다전공생/편입생/전과생을 위한 방학특강☆ 컴공학생 따라잡기 프로젝트> 진행
- 2019년 2학기
  - KCA 공식 스터디 - <C언어 초보반> 진행
- 2020년 겨울방학
  - KCA 비공식 스터디 <방학특강 revise> 진행
  - KCA 비공식 스터디 <포인터부터 시작하는☆ 자료구조 스터디> 진행
- 2020년 1학기 KCA 스터디부장
- 2021년 1학기 KCA 총무
- 2021년 1학기
  - KCA 교육 프로그램 – <C 초급반> <Flutter 초급반> 진행

목차

알고리즘의 정의

- ✔ 문제를 해결하는 행동 및 논리의 절차
- ✔ 문제를 해결하는 전략

일반적인 의미에서의 알고리즘:
"주어진 문제를 어떻게 해결하는가"에 대한 답
컴퓨터과학에서의 알고리즘:
컴퓨터를 이용하여 무언가를 수행하는 방법

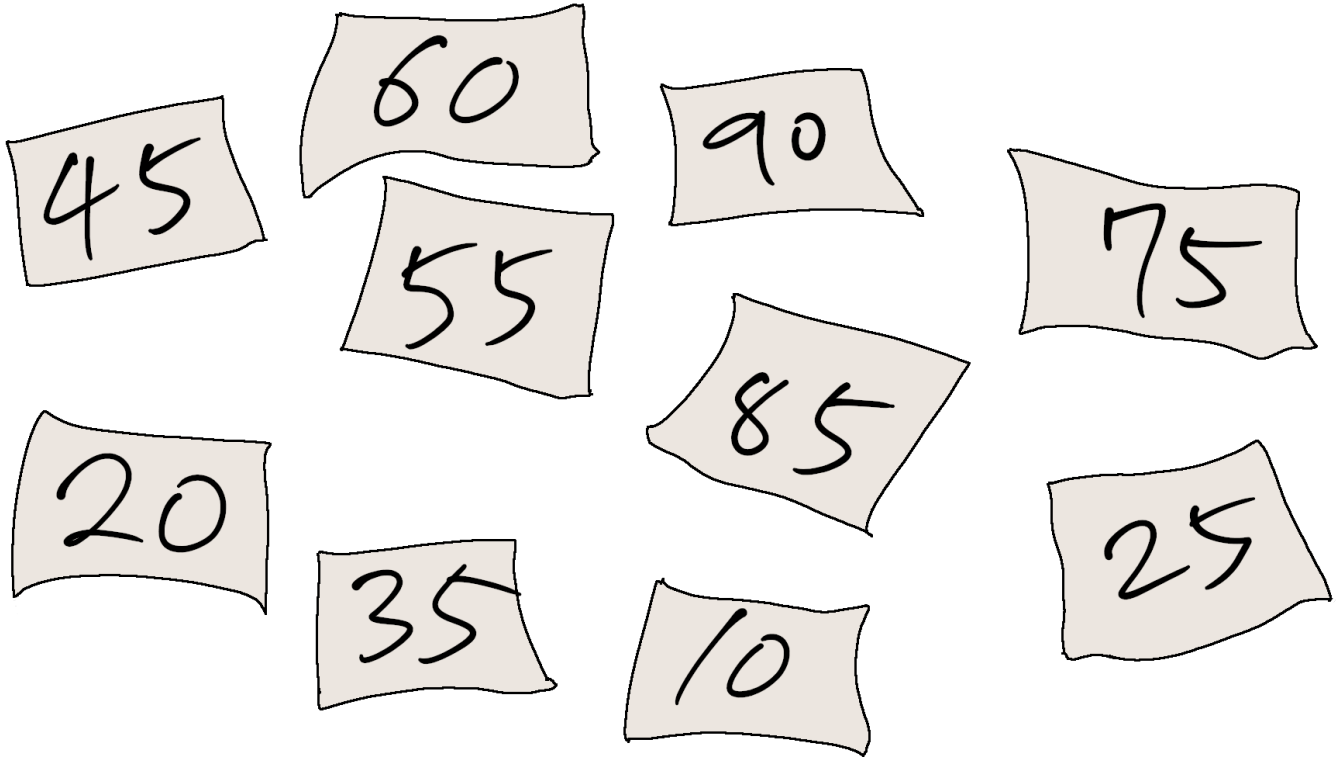
고려해야 할 점

- ✔ 각 Step마다 무엇이 필요한가
- ✔ 행동을 어떻게 배열하는가
- ✔ 이 절차와 행동을 선택하였을 때의 효과는 무엇인가
- ✔ 이 절차가 문제를 올바르게 해결하는가
- ✔ 효율적인 방법인가

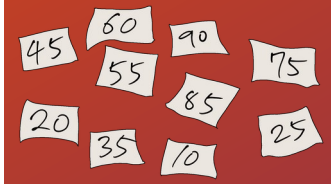
예시

첫번째 예시: 가장 큰 숫자 찾기

Q. 주어진 숫자 중 가장 큰 숫자는 무엇인가?



예시

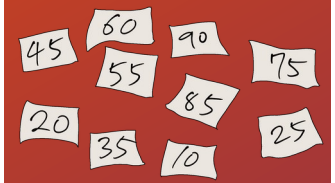


첫번째 예시: 가장 큰 숫자 찾기

Q. 주어진 숫자 중 가장 큰 숫자는 무엇인가?

Step 1 | 확인한 카드 중 가장 큰 수를 기억한다.
Step 2 | 확인하지 않은 새 카드를 한 장 확인한다.
Step 3 | Step 1의 가장 큰 수와 Step 2의 새 카드를 비교한다.
Step 4 | 새 카드가 더 클 경우 `가장 큰 수`를 갱신하고,
그렇지 않다면 방금 뽑은 카드를 버린다.
Step 5 | 모든 카드를 다 확인할 때까지 위 과정을 반복한다.

예시



첫번째 예시: 가장 큰 숫자 찾기

- ✔ 검색 알고리즘 (search)
- ✔ 순차 탐색 알고리즘 (sequential search)
- ✔ 브루트 포스 알고리즘 (brute-force)

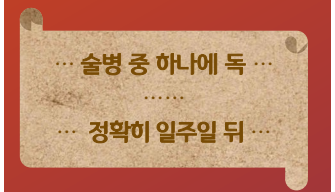
예시

두번째 예시: 독이 든 술병 찾기

술을 매우 몹시 좋아하는 왕이 있다 .
왕의 창고에는 매우 많은 양의 술병이 보관되어 있다 .
어느 날 , 간첩이 그 **술병 중 하나에 독**을 타고 잡혀 왔다 .
잡혀온 간첩이 말하길 ,
“ 어느 술병에 독을 탔는지 까먹었습니다 .”
덧붙여 , “ 한 모금이라도 마시면 **정확히 일주일 뒤**에 죽을 겁니다 .”
이에 왕이 신하들에게 명령하였다 .
“ 일주일 줄테니 독이 든 술병을 찾아내라 !”

Q. 이 때, 가장 적은 수의 신하가 위험을 감수하고
독이 든 술병을 찾아내기 위한 방법은?

예시



두번째 예시: 독이 든 술병 찾기

Q. 가장 적은 수의 신하가 위험을 감수하고
독이 든 술병을 찾아내기 위한 방법은?

N병의 술병과 M명의 신하가 있다고 하자.

N≤M이라면, N명의 신하가 한 병씩 맡아 한 모금씩 마셔 보고
일주일 뒤에 누가 죽는지 확인함으로써
독이 든 술병이 어느 것인지 확인할 수 있다.

⇒ N명의 신하가 1/N 확률로 죽을 위험을 감수해야 한다!

예시

··· 술병 중 하나에 독 ···
·····
··· 정확히 일주일 뒤 ···

두번째 예시: 독이 든 술병 찾기

Q. 가장 적은 수의 신하가 위험을 감수하고
독이 든 술병을 찾아내기 위한 방법은?

✔ 만약 신하가 1명이라면?
하나의 술병에 든 술을 마셔볼 수 있다.
일주일 뒤 그 신하가 죽으면 그 술병에 독이 든 것이고,
죽지 않는다면 다른 술병에 독이 든 것이다.

⇒ 술병이 2병이라면 신하 1명으로 충분

예시

··· 술병 중 하나에 독 ···
·····
··· 정확히 일주일 뒤 ····

두번째 예시: 독이 든 술병 찾기

Q. 가장 적은 수의 신하가 위험을 감수하고
독이 든 술병을 찾아내기 위한 방법은?

✔ 신하 1명당 술병 2병?

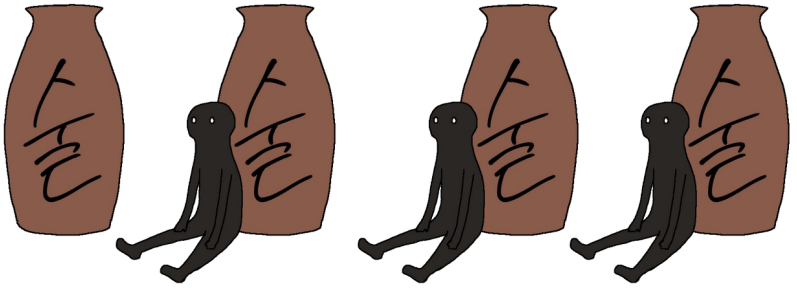
예시

··· 술병 중 하나에 독 ···
·····
··· 정확히 일주일 뒤 ···

두번째 예시: 독이 든 술병 찾기

Q. 가장 적은 수의 신하가 위험을 감수하고
독이 든 술병을 찾아내기 위한 방법은?

✔ 술병 N병에 대하여 신하 N−1명?



예시

··· 술병 중 하나에 독 ···
·····
··· 정확히 일주일 뒤 ···

두번째 예시: 독이 든 술병 찾기

Q. 가장 적은 수의 신하가 위험을 감수하고
독이 든 술병을 찾아내기 위한 방법은?

✔ 그룹화하여 술을 마시다면?
A가 죽으면 A만 마신 술병
B가 죽으면 B만 마신 술병
둘다 죽으면 둘다 마신 술병
안죽으면 아무도 안마신 술병

예시

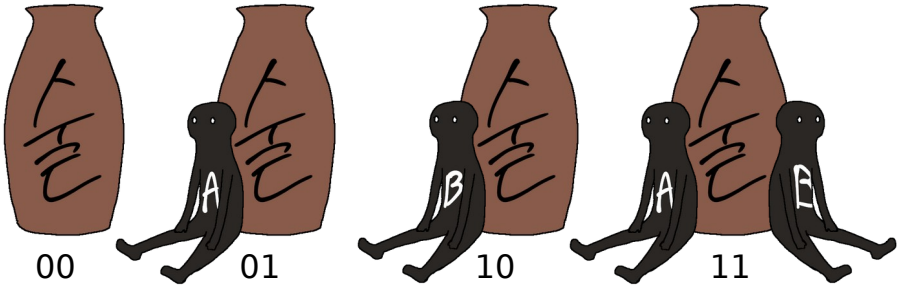
··· 술병 중 하나에 독 ···
·····
··· 정확히 일주일 뒤 ···

두번째 예시: 독이 든 술병 찾기

Q. 가장 적은 수의 신하가 위험을 감수하고
독이 든 술병을 찾아내기 위한 방법은?

- ✔ N병의 술병에 대해 일반화하면,
  N자리 이진수 값으로 표기할 수 있다.
- ✔ 신하 하나당 하나의 bit를 맡아 해당 bit가 1인 술병만 마신다.
- ✔ 일주일 후 신하의 생존 여부에 따라 bit를 확인한다.

낮은 자리부터 순서대로 맡았다고 하면



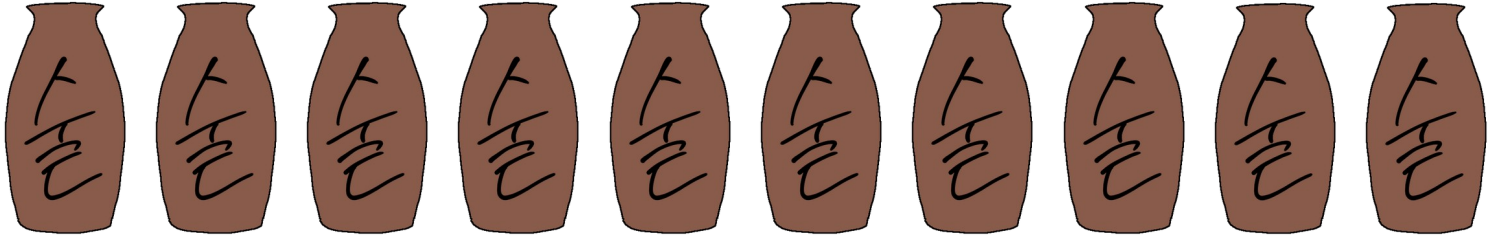
예시

··· 술병 중 하나에 독 ···
·····
··· 정확히 일주일 뒤 ···

두번째 예시: 독이 든 술병 찾기

Q. 가장 적은 수의 신하가 위험을 감수하고
독이 든 술병을 찾아내기 위한 방법은?

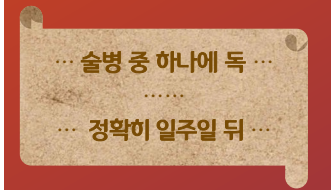
✔ 예를 들어, 술병이 10병이라면,



0000	0001	0010	0011	0100	0101	0110	0111	1000	1001
----	---A	--B-	--BA	-C--	-C-A	-CB-	-CBA	D---	D--A

⇒ A와 C가 죽었다면 독이 든 건 △ 요 녀석, 0101

예시



두번째 예시: 독이 든 술병 찾기

Q. 가장 적은 수의 신하가 위험을 감수하고
독이 든 술병을 찾아내기 위한 방법은?

- ✔ k명의 신하가 있을 때 최대 2^k병의 술병 확인 가능
- ✔ N병의 술병을 확인하기 위해선 최소 $\sqrt{N}$ 명의 신하 필요
(정수가 아닐 경우 올림)

예시

- ✔ 이와 같이 다양한 문제에 대해 적합한 알고리즘을 찾아내 그것을 이용하여 문제를 해결해야 한다.
- ✔ 아무도 힌트를 주지 않는 상황에서(...) 최적의 방식 찾아내기

알고리즘의 특성

- ✔ 정확성 (Correctness)
- ✔ 수행성 (Runnable)
- ✔ 유한성 (Finiteness)
- ✔ 효율성 (Efficiency)

알고리즘의 표현

다양한 표현방식 존재

- ✔ 자연어
- ✔ 순서도
- ✔ 수식
- ✔ 의사코드 (pseudo-code)
- ✔ 프로그래밍 언어
- ✔ ...등등

예시

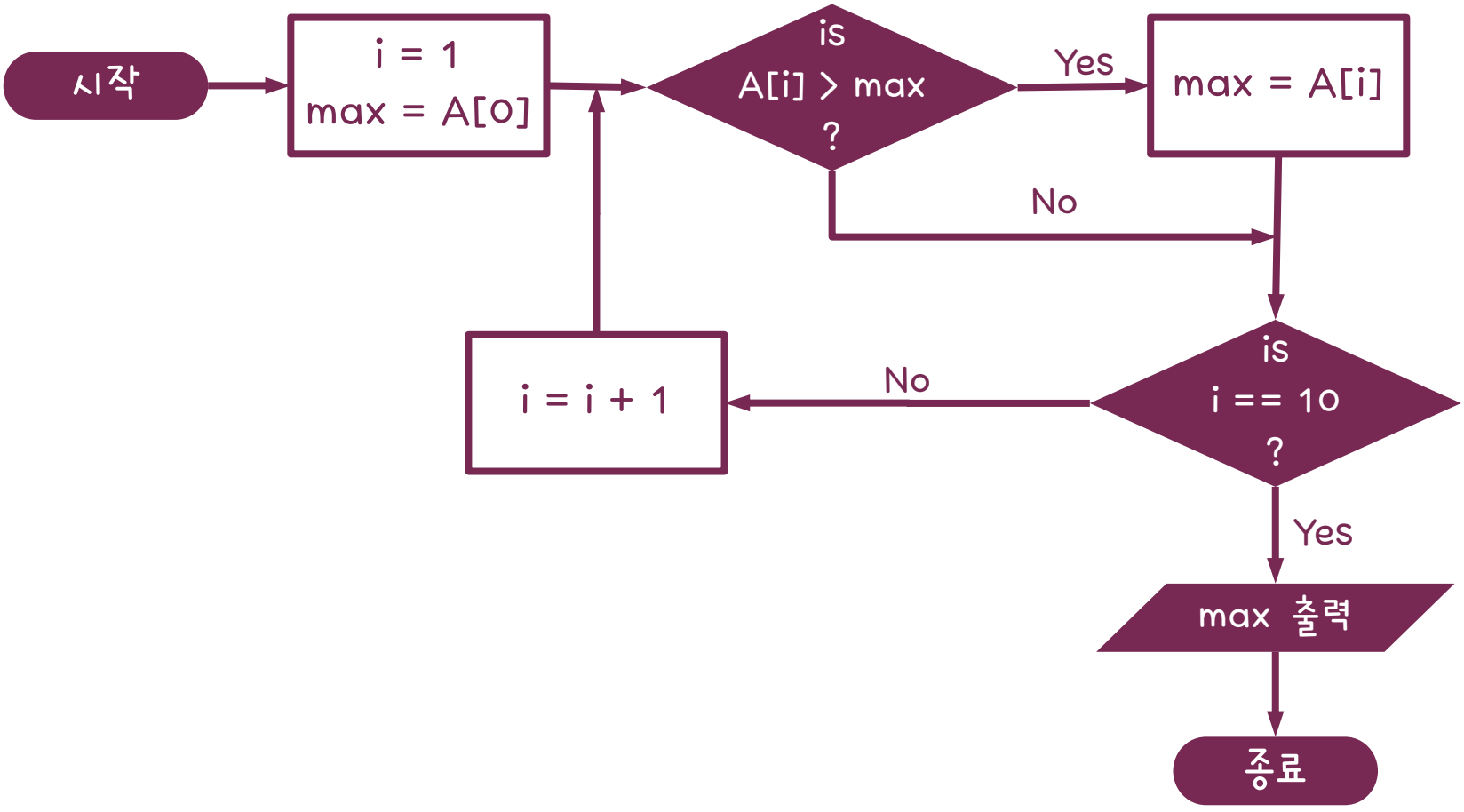
의사코드 표현 – 가장 큰 숫자 찾기

- ✔ input: 숫자 10개가 들어있는 배열 A
- ✔ output: 가장 큰 숫자 max

```
max = A[0]
for i = 1 to 9
    if (A[i] > max)
        max = A[i]
return max
```

예시

순서도 표현 – 가장 큰 숫자 찾기



알고리즘의 분류

전략에 따른 알고리즘의 분류

- ✔ 분할-정복 (Divide and Conquer)
- ✔ 탐욕 (Greedy)
- ✔ 동적 프로그래밍 (Dynamic Programming)
- ✔ 근사 (Approximation)
- ✔ 탐색 (Search)
  - ✔ 백트래킹 (Backtracking) / 분기한정 (Branch and Bound)

알고리즘의 분류

문제 유형에 따른 알고리즘의 분류

- ✔ 정렬 (Sorting problem)
- ✔ 그래프 (graph problem)
- ✔ 기하 (geometric problem)
- ✔ ...등등

알고리즘의 분류

연산 환경에 따른 알고리즘의 분류

- ✔ 병렬 (Parallel)
- ✔ 분산 (Distributed)
- ✔ 양자 (Quantum)
- ✔ ...등등

알고리즘의 효율성

- ✔ 시간 복잡도 (time complexity)
  연산 시간이 얼마나 걸리는가
- ✔ 공간 복잡도 (space complexity)
  메모리를 얼마나 사용하는가

주로 공간 복잡도보다는 시간 복잡도를 논한다
그런데 어떻게 측정할까?

예시

시간복잡도 계산 – 가장 큰 숫자 찾기

- ✔ input: 숫자 10개가 들어있는 배열 A
- ✔ output: 가장 큰 숫자 max

max = A[0]	쓰기 1
for i = 1 to 9	쓰기 9
if (A[i] > max)	읽기 9, 비교 9
max = A[i]	쓰기 9
return max	쓰기 1

읽기/쓰기 cost가 M, 비교 cost가 N일 때 M * 29 + N * 9

시간 복잡도 표현

- ✔ 최악의 경우 (worst case analysis)
  요구되는 시간의 상한선(upper bound)을 통해 분석
- ✔ 최선의 경우 (best case analysis)
  요구되는 시간의 하한선(lower bound)을 통해 분석
- ✔ 평균의 경우 (average case analysis)
  많은 확인 과정이 필요하여 잘 쓰이지 않음

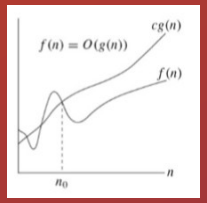
시간 복잡도 점근적 표현

- ✔ 알고리즘은 복잡도가 매우 높을 때 문제
  ⇒ 충분히 낮은 복잡도를 목표로 한다
- ✔ 보통 input size가 작을 땐 문제가 발생하지 않는다
  ⇒ input size가 커질수록 복잡도가 어떻게 변해가는지
  확장성 중요
- ✔ input size의 증가에 따른 복잡도 증가 점근적 표현

시간 복잡도 점근적 표현

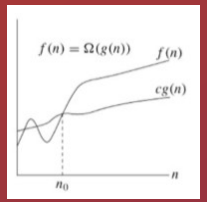
✔ O() 빅오 표기법

알고리즘의 점근적 상한선을 나타내는 방법



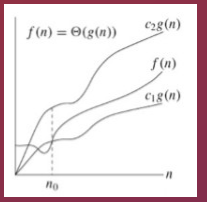
✔ Ω() 빅오메가 표기법

알고리즘의 점근적 하한선을 나타내는 방법



✔ Θ() 빅세타 표기법

O()와 Ω()이 일치하는 경우, 점근적 평균을 나타내는 방법



시간 복잡도 점근적 표현

최고차항에서 계수를 제외한 값만 표기

- ✔ O(1) | 상수 시간 걸리는 알고리즘
- ✔ O(log n) | 로그 시간 걸리는 알고리즘
- ✔ O(n) | 선형 시간 걸리는 알고리즘
- ✔ O(n log n) | 선형 로그 시간 걸리는 알고리즘
- ✔ $O(n^2)$ | 제곱 시간 걸리는 알고리즘
- ✔ $O(n^3)$ | 세제곱 시간 걸리는 알고리즘
- ✔ $O(n^k)$ | k제곱 시간 걸리는 알고리즘
- ✔ $O(2^n)$ | 지수 시간 걸리는 알고리즘

효율적인 알고리즘이 필요한 이유

✔ 효율성에 따른 소요시간 비교

$O(n^2)$	n=1000	n=1백만	n=10억
PC	< 1초	2시간	300년
슈퍼컴퓨터	< 1초	1초	1주일

O(n log n)	n=1000	n=1백만	n=10억
PC	< 1초	< 1초	5분
슈퍼컴퓨터	< 1초	< 1초	< 1초

다음 시간에는

다양한 알고리즘 (1) Greedy Algorithm

에 대해 배워보도록 하겠습니다★