

3절 알고리즘 분석

- 설계한 알고리즘의 효율성 분석
- 알고리즘 분석에 사용하는 용어와 표준 분석방법 학습

시간복잡도 분석

- 알고리즘 효율성 분석 기법
- 기준: 입력크기에 대해 특정 단위연산이 수행되는 횟수

입력크기 : 입력값의 크기

- 예제
 - 리스트의 길이
 - 행렬의 행과 열의 수
 - 나무(트리)의 마디와 이음선의 수
 - 그래프의 정점과 간선의 수
- 주의: 입력과 입력크기는 일반적으로 다름.
 - 피보나찌 함수 `fib`에 사용되는 입력값 n 의 크기는 n 을 이진법으로 표기했을 때의 길이인 $(\lfloor \lg n \rfloor + 1)$ 이다.
 - 예제: $n = 13$ 의 입력크기는 $\lfloor \lg 13 \rfloor + 1 = 4$.

단위연산: 명령문 또는 명령문 덩어리(군)

- 단위연산의 실행 횟수가 알고리즘의 실행 시간 결정
- 예제
 - 비교문(comparison)
 - 지정문(assignment)
 - 반복문
 - 모든 기계적 명령문 각각의 실행
 - 예제: PythonTutor의 Step 계산
- 순차검색과 이분검색 알고리즘의 단위 연산: while 반복문 전체
- 피보나찌 함수의 단위 연산: 함수 본체 전체

주의사항

- 단위연산을 지정하는 일반적인 규칙 없음.
- 경우에 따라 두 개의 다른 단위연산을 고려해야 할 수도 있음.
 - 예제: 키를 비교하여 정렬하는 경우
 - 비교와 지정이 서로 다른 비율로 발생하기에 서로 독립적인 단위연산으로 다
룹
- 단위연산의 실행횟수가 입력크기뿐만 아니라 입력값에도 의존할 수 있음.

알고리즘의 시간복잡도

- 입력값의 입력크기 n 에 대해 지정된 단위연산이 수행되는 횟수 $f(n)$ 을 계산하는 함수 f 로 표현
- 시간복잡도 함수: 시간복잡도를 표현하는 함수

시간복잡도 종류

- 단위연산 실행횟수가 입력값에 상관없이 입력크기에만 의존하는 경우
 - 일정 시간복잡도: $T(n)$
- 단위연산 실행횟수가 입력값과 입력크기 모두에 의존하는 경우
 - 최악 시간복잡도: $W(n)$
 - 평균 시간복잡도: $A(n)$
 - 최선 시간복잡도: $B(n)$

일정 시간복잡도

- 일정 시간복잡도 $T(n)$
 - 입력값에 상관없이 입력크기 n 에만 의존하는 단위연산 실행횟수
- 예제
 - 리스트의 원소 모두 더하기
 - 교환정렬
 - 행렬곱셈(2장)

최악 시간복잡도

- 최악 시간복잡도 $W(n)$: 입력크기 n 에 대한 단위연산의 최대 실행횟수
- 예제
 - 핵발전소 시스템의 경우처럼 나쁜 사례에 대한 최악의 반응시간이 중요한 경우 활용

평균 시간복잡도

- 평균 시간복잡도 $A(n)$: 입력크기 n 에 대한 단위연산의 실행횟수 기대치(평균)
- 평균 단위연산 실행횟수가 중요한 경우 활용
- 각 입력값에 대해 확률 할당 가능
- 최악의 경우 분석보다 계산이 보다 복잡함

최선 시간복잡도

- 최선 시간복잡도 $B(n)$: 입력크기 n 에 대한 단위연산의 최소 실행횟수
- 잘 사용되지 않음.

시간복잡도 특성

- $T(n)$ 이 존재하는 경우:

$$T(n) = W(n) = A(n) = B(n)$$

- 일반적으로:

$$B(n) \leq A(n) \leq W(n)$$

일정 시간복잡도를 구할 수 없는 경우

- 최선의 경우 보다 최악 또는 평균의 경우 분석을 일반적으로 진행
- 평균 시간복잡도 분석
 - 다른 입력을 여러 번 사용할 때 평균적으로 걸리는 시간 알려줌.
 - 예를 들어, 속도가 느린 정렬 알고리즘이라도 평균적으로 시간이 좋게 나오는 경우 사용 가능.
- 최악 시간복잡도 분석
 - 핵발전소 감시시스템 경우처럼 단 한 번의 사고가 치명적인 경우 활용.

공간(메모리)복잡도

- 알고리즘의 메모리 사용 효율성 분석
- 책에서는 시간복잡도에 집중.
- 필요한 경우 공간복잡도 분석 활용.

예제: 일정 시간복잡도 분석

알고리즘: 리스트 항목더하기

- 문제: 크기가 n 인 리스트 S 의 모든 항목을 더하라.
- 입력: 리스트 S
- 출력: 리스트 S 에 있는 항목의 합

In [1]: *# 리스트의 항목 모두 더하기*

```
def sum(S):  
    result = 0  
  
    for i in range(len(S)):  
        result = result + S[i]  
    return result
```

In [2]: seq = list(range(11))

```
sum(seq)
```

Out[2]: 55

리스트 항목더하기 알고리즘의 $T(n)$ 구하기: 덧셈 기준

- 단위연산: 덧셈
- 입력크기: 리스트의 크기 n

- 모든 경우 분석:
 - 리스트의 내용에 상관없이 for-반복문 n 번 실행.
 - 반복마다 덧셈 1회 실행.
 - 따라서 $T(n) = n$.

알고리즘: 교환정렬

- 문제: 리스트의 항목을 비내림차순(오름차순)으로 정렬하기
- 입력: 리스트 S
- 출력: 비내림차순으로 정렬된 리스트

In [3]: *# 교환정렬*

```
def exchangesort(S):  
    for i in range(len(S)):  
        for j in range(i+1, len(S)):  
            if (S[j] < S[i]):  
                S[i], S[j] = S[j], S[i]
```

In [4]: `seq = [1, 4, 5, 2, 7, 4]`
`exchangesort(seq)`
`print(seq)`

`[1, 2, 4, 4, 5, 7]`

교환정렬 알고리즘의 $T(n)$ 구하기 : 조건문 기준

- 단위연산: 조건문 ($s[j]$ 와 $s[i]$ 의 비교)
- 입력크기: 리스트의 길이 n

교환정렬 알고리즘 일정 시간복잡도 분석

- j-반복문이 실행할 때마다 조건문 한 번씩 실행
- 조건문의 총 실행횟수
 - $i = 1$ 인 경우: $(n - 1)$ 번
 - $i = 2$ 인 경우: $(n - 2)$ 번
 - ...
 - $i = (n - 1)$ 인 경우: 1 번
- 그러므로 다음 성립:

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

- 확인

```
In [5]: # 교환정렬

def exchangesort_1(S):
    count = 0
    for i in range(len(S)):
        for j in range(i+1, len(S)):
            count += 1
            if (S[j] < S[i]):
                S[i], S[j] = S[j], S[i]
    return count
```

```
In [6]: seq = [1, 4, 5, 2, 7, 4]
print(exchangesort_1(seq))
```

15

- 실제로

$$15 = \frac{6 \cdot 5}{2}$$

예제: 최악 시간복잡도 분석

교환정렬 알고리즘의 $W(n)$ 구하기 : 교환 기준

- 단위연산: 교환하는 연산 ($s[i]$ 와 $s[j]$ 의 교환)
- 입력크기: 정렬할 항목의 수 n

- 최악의 경우 분석:
 - 조건문의 결과에 따라서 교환 연산의 실행여부 결정
 - 최악의 경우
 - 조건문이 항상 참(true)인 경우
 - 즉, 입력 배열이 거꾸로 정렬되어 있는 경우
 - 이때, 조건문 실행 횟수와 동일하게 실행됨. 즉, 일정 시간복잡도와 동일.

$$W(n) = \frac{(n-1)n}{2}$$

순차검색 알고리즘의 $W(n)$ 구하기: 항목 비교 연산 기준

- 단위연산: 리스트 s 의 항목과 값 x 와의 비교연산
 - `S[location] != x`
- 입력크기: 리스트 크기 n

- 최악의 경우 분석:

- x 가 리스트의 마지막 항목이거나, 리스트에 포함되지 않은 경우, 단위연산이 n 번 수행된다. 즉,

$$W(n) = n$$

- 주의: 입력(s 와 x)에 따라서 검색횟수가 달라지므로, 일정 시간복잡도 분석 불가능.

예제: 평균 시간복잡도 분석

순차검색 알고리즘의 $A(n)$ 구하기: 항목 비교 연산 기준

- 단위연산: 리스트 S 의 항목과 값 x 와의 비교연산
 - $S[\text{location}] \neq x$
- 입력크기: 리스트 크기 n

- 경우 1

- 가정

- x 가 리스트 s 안에 있음
 - 리스트의 항목이 모두 다름.
 - x 가 리스트의 특정 위치에 있을 확률 동일, 즉 $1/n$. 단, n 은 리스트 s 의 길이.

- x 가 리스트의 k 번째 있다면, s 를 찾기 위해서 수행하는 단위연산의 횟수는 k .

$$\begin{aligned}
 A(n) &= \sum_{k=1}^n \left(k \times \frac{1}{n} \right) \\
 &= \frac{1}{n} \times \sum_{k=1}^n k \\
 &= \frac{1}{n} \times \frac{n(n+1)}{2} \\
 &= \frac{n+1}{2}
 \end{aligned}$$

- 경우 2

- 가정

- x 가 리스트 s 안에 없을 수도 있음.
 - x 가 리스트 s 안에 있을 확률: p

- x 가 배열에 없을 확률: $1 - p$

- x 가 리스트의 k 번째 항목일 확률: p/n

$$\begin{aligned}
 A(n) &= \sum_{k=1}^n \left(k \times \frac{p}{n} \right) + n(1-p) \\
 &= \frac{p}{n} \times \frac{n(n+1)}{2} + n(1-p) \\
 &= n \left(1 - \frac{p}{2} \right) + \frac{p}{2}
 \end{aligned}$$

- 예를 들어 $p = 1$ 이면:

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

- 예를 들어 $p = 1/2$ 이면:

$$A(n) = \frac{3n + 1}{4}$$

예제: 최선 시간복잡도 분석

교환정렬 알고리즘의 $B(n)$ 구하기 : 교환 기준

- 단위연산: 교환하는 연산 ($s[i]$ 와 $s[j]$ 의 교환)
- 입력크기: 정렬할 항목의 수 n

- 최선의 경우 분석:
 - 조건문의 결과에 따라서 교환 연산의 실행여부 결정
 - 최선의 경우
 - 조건문이 항상 거짓(false)이 되는 경우
 - 즉, 입력 배열이 이미 오름차순(비내림차순)으로 정렬되어 있는 경우
 - 이때, 교환이 전혀 발생하지 않음.
 - 따라서 $B(n) = 0$.

- 확인

```
In [7]: # 교환정렬

def exchangesort_2(S):
    count = 0
    for i in range(len(S)):
        for j in range(i+1, len(S)):
            if (S[j] < S[i]):
                count += 1
                S[i], S[j] = S[j], S[i]
    return count
```

```
In [8]: seq = [1, 2, 4, 4, 5, 7]
print(exchangesort_2(seq))
```

0

순차검색 알고리즘의 $B(n)$ 구하기: 항목 비교 연산 기준

- 단위연산: 리스트 s 의 항목과 값 x 와의 비교연산
 - $S[\text{location}] \neq x$
- 입력크기: 리스트 크기 n

- 최선의 경우 분석:
 - x 가 $s[0]$ 일 때, 입력의 크기에 상관없이 단위연산이 한 번 수행
 - 따라서 $B(n) = 1$.

복잡도 함수 예제

- $f(n) = 1$
- $f(n) = \lg n$
- $f(n) = n$
- $f(n) = 1000n$

- $f(n) = n^2$
- $f(n) = \frac{n(n-1)}{2}$
- $f(n) = 3n^2 + 4n$

복잡도 함수와 실행시간

예제

- 아래 두 알고리즘 중에서 어떤 알고리즘 선택?
 - 알고리즘 A의 시간복잡도: $1000n$
 - 알고리즘 B의 시간복잡도: n^2
- n^2 이 $1000n$ 보다 복잡도가 커보임. 하지만...

- 정답: n 의 크기에 따라 달라짐.
 - $n \leq 1,000$: 알고리즘 B 선택
 - $n > 1,000$: 알고리즘 A 선택

- 이유:

$$n^2 > 1000n \iff n > 1000$$