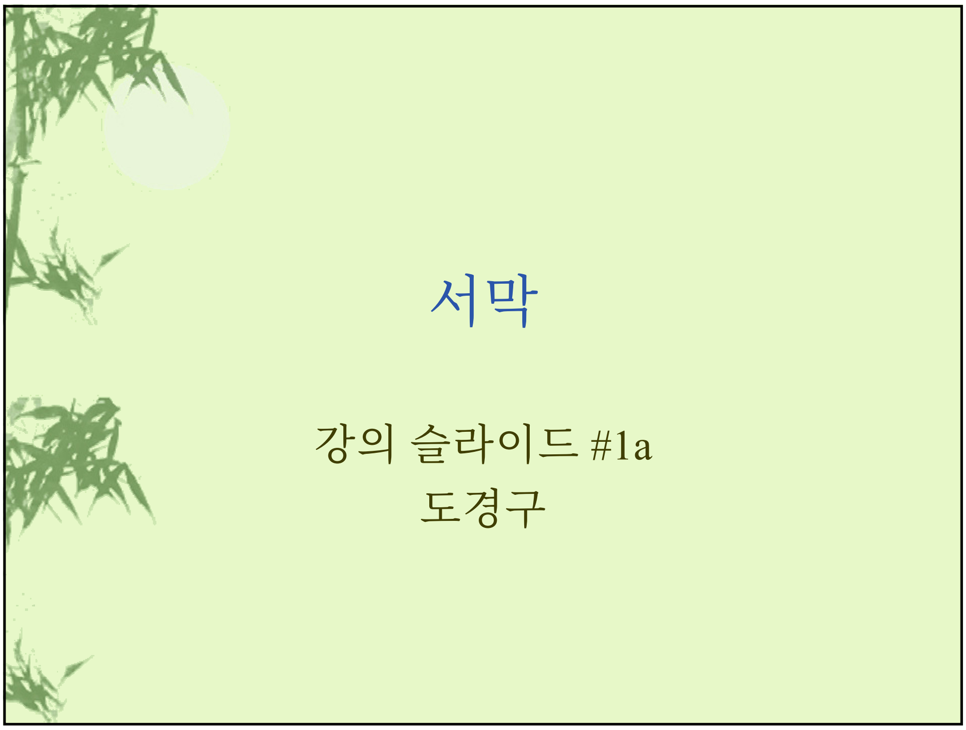
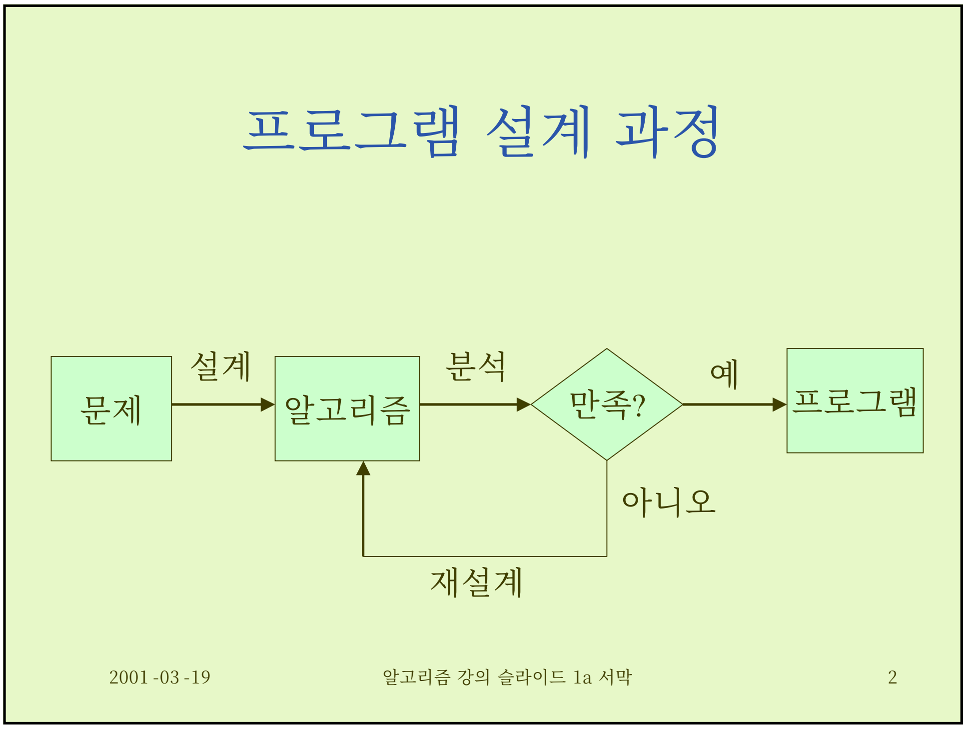

서막
강의 슬라이드 #1a
도경구

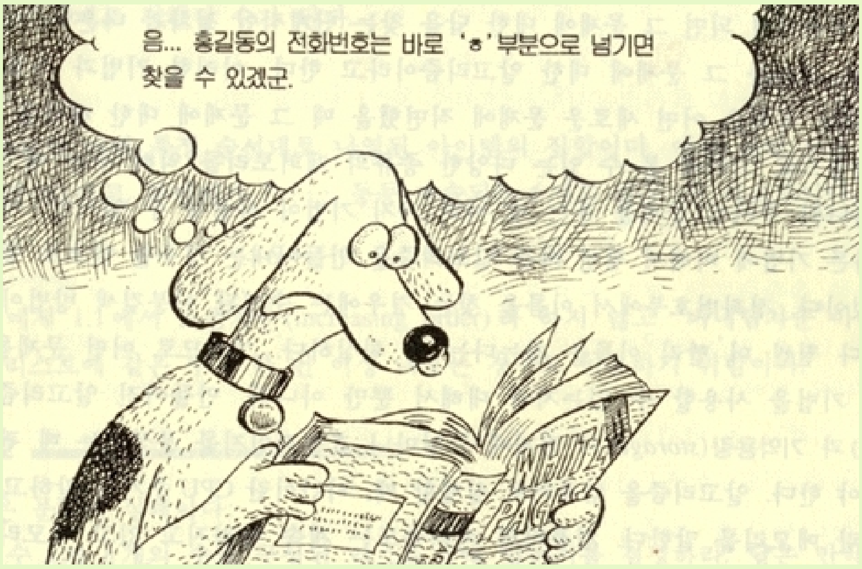

프로그램 설계 과정
문제
설계
알고리즘
분석
만족?
예
프로그램
아니오
재설계
2001 -03 -19
알고리즘 강의 슬라이드 1a 서막
2

이 과목의 학습목표

- Design(설계)
  - ✓ 알고리즘을 설계하는 기법을 배운다.
- Analysis(분석)
  - ✓ 알고리즘을 분석하여 시간/공간 복잡도를 구하는 방법을 배운다.
- Computational Complexity(계산복잡도)
  - ✓ 문제를 분석하여 계산적 복잡도를 구하는 방법을 배운다.

정의: 알고리즘(Algorithm)

- 문제에 대한 답을 찾기 위하여 계산하는 절차
- 단계별로 주의 깊게 설계된 계산과정
- 입력을 받아서 출력으로 전환시켜주는 일련의 계산절차

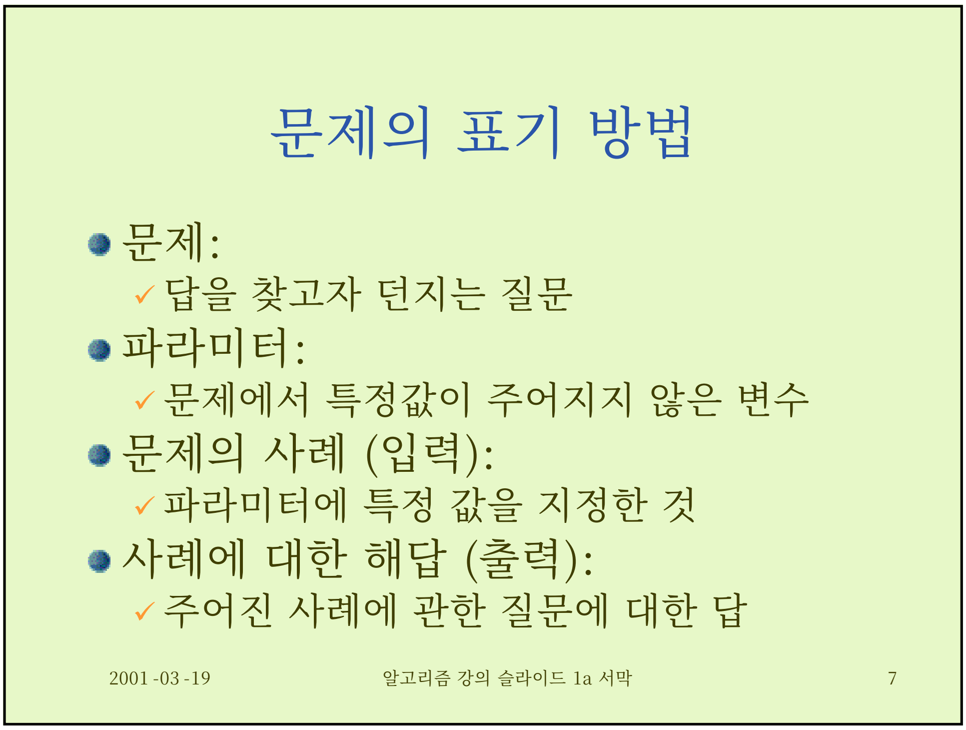


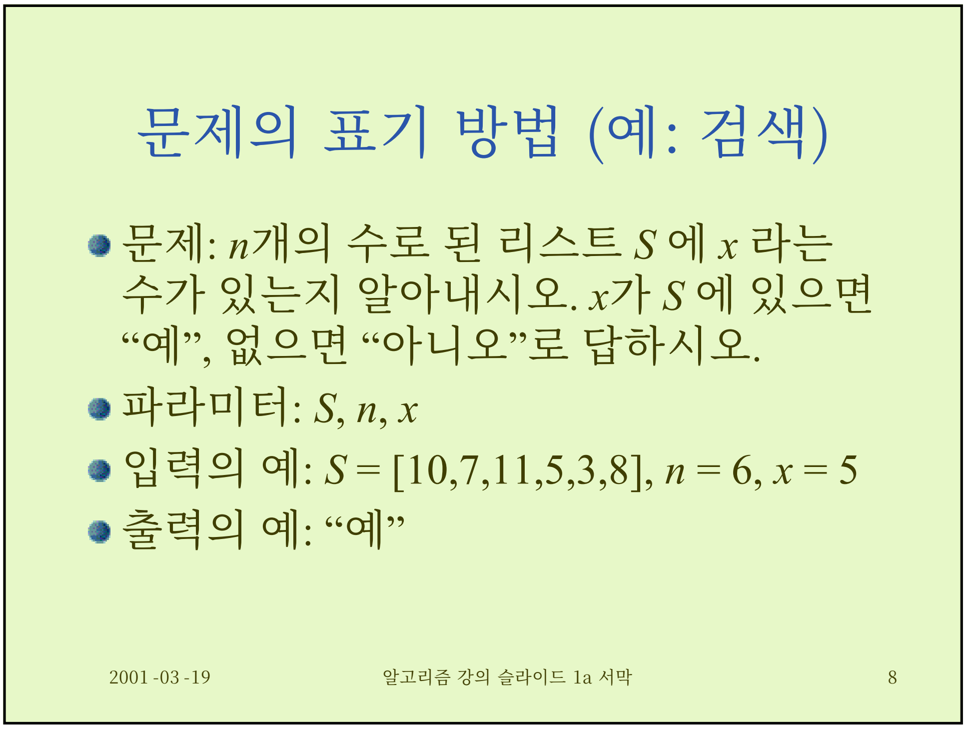
2001 -03 -19 알고리즘 강의 슬라이드 1a 서막 5

알고리즘의 예

- 문제: 전화번호부에서 "홍길동"의 전화번호 찾기
- 알고리즘
  - ✓ 순차검색: 첫 쪽부터 홍길동이라는 이름이 나올 때까지 순서대로 찾는다.
  - ✓ 수정된 이분검색: 전환번호부는 "가나다"순으로 되어있으므로 먼저 "ㅎ"이 있을 만한 곳으로 넘겨 본 후 앞뒤로 뒤적여가며 찾는다.
- 분석: 어떤 알고리즘이 더 좋은가?

2001 -03 -19 알고리즘 강의 슬라이드 1a 서막 6


문제의 표기 방법
문제:
답을 찾고자 던지는 질문
파라미터:
문제에서 특정값이 주어지지 않은 변수
문제의 사례 (입력):
파라미터에 특정 값을 지정한 것
사례에 대한 해답 (출력):
주어진 사례에 관한 질문에 대한 답
2001 -03 -19
알고리즘 강의 슬라이드 1a 서막
7


문제의 표기 방법 (예: 검색)
문제: n개의 수로 된 리스트 S 에 x 라는 수가 있는지 알아내시오. x가 S 에 있으면 "예", 없으면 "아니오"로 답하시오.
파라미터: S, n, x
입력의 예: S = [10,7,11,5,3,8], n = 6, x = 5
출력의 예: "예"
2001 -03 -19
알고리즘 강의 슬라이드 1a 서막
8

알고리즘의 표기

- 자연어: 한글 또는 영어
- 프로그래밍언어: C, C++, Java, ML 등
- 의사코드(Pseudo-code)
  - ✓ 직접 실행할 수 있는 프로그래밍언어는 아니지만, 거의 실제 프로그램에 가깝게 계산과정을 표현할 수 있는 언어
- 알고리즘은 보통 의사코드로 표현한다.
- 이 강의에서는 C++에 가까운 의사코드를 사용한다.

2001 -03 -19 알고리즘 강의 슬라이드 1a 서막 9

C++와 의사코드의 차이점(1)

- 배열 인덱스의 범위에 제한 없음
  - ✓ C++는 반드시 0부터 시작
  - ✓ 의사코드는 임의의 값 사용 가능
- 프로시저의 파라미터에 2차원 배열 크기의 가변성 허용
  - ✓ 예: `void pname(A[][]) { … }`
- 지역배열에 변수 인덱스 허용
  - ✓ 예: `keytype S[low..high];`

2001 -03 -19 알고리즘 강의 슬라이드 1a 서막 10

C++와 의사코드의 차이점(2)

- 수학적 표현식 허용
  - `low <= x && x <= high` ⇒
    `low ≤ x ≤ high`
  - `temp = x; x = y; y = temp` ⇒
    `exchange x and y`
- C++에 없는 타입 사용 가능
  - `index`: 첨자로 사용되는 정수 변수
  - `number`: 정수(int) 또는 실수(float) 모두 사용가능
  - `bool`: "true"나 "false" 값을 가질 수 있는 변수

C++와 의사코드의 차이점(3)

- 제어 구조
  - `repeat (n times) { … }`
- 프로시저와 함수
  - 프로시저: `void pname(…) {…}`
  - 함수: `returntype fname (…) {… return x;}`
- 참조파라미터(reference parameter)를 사용하여 프로시저의 결과값 전달
  - 배열: 참조 파라미터로 전달
  - 기타: 데이터타입 이름 뒤에 &를 붙임
  - const 배열: 전달되는 배열의 값이 불변

순차검색 알고리즘 (Sequential Search)

- 문제: 크기가 n인 배열 S에 x가 있는가?
- 입력(파라미터): (1) 양수 n, (2) 배열 $S[1..n]$, (3) 키 x
- 출력: x가 S의 어디에 있는지의 위치. 만약 없으면 0.
- 알고리즘(자연어):
  - ✓ x와 같은 아이템을 찾을 때까지 S에 있는 모든 아이템을 차례로 검사한다.
  - ✓ 만일 x와 같은 아이템을 찾으면 S에서 위치를 내주고, S를 모두 검사하고도 찾지 못하면 0을 내준다.

2001-03-19 알고리즘 강의 슬라이드 1a 서막 13

- 순차검색 알고리즘 (의사코드)

```
void seqsearch(int n,                  // 입력(1)
               const keytype S[],      //     (2)
               keytype x,              //     (3)
               index& location) {      // 출력
  location = 1;
  while (location <= n && S[location] != x)
    location++;
  if (location > n)
    location = 0;
}
```

✓ while-루프: 아직 검사할 항목이 있고, x를 찾지 못했나?
✓ if-문: 모두 검사하였으나, x를 찾지 못했나?

2001-03-19 알고리즘 강의 슬라이드 1a 서막 14

사색

- 순차검색 알고리즘으로 키를 찾기 위해서 S에 있는 항목을 몇 개나 검색해야 하는가?
  - 키와 같은 항목의 위치에 따라 다름
  - 최악의 경우: n
- 좀 더 빨리 찾을 수는 없는가?
  - S에 있는 항목에 대한 정보가 없는 한 더 빨리 찾을 수 없다.

이분검색 알고리즘 (Binary Search)

- 문제: 크기가 n인 정렬된 배열 S에 x가 있는가?
- 입력: (1) 양수 n, (2) 배열 $S[1..n]$, (3) 키 x
- 출력: x가 S의 어디에 있는지의 위치. 만약 없으면, 0.

이분검색 알고리즘

```
void binsearch(int n,                    // 입력(1)
               const keytype S[],        //     (2)
               keytype x,                //     (3)
               index& location) {        // 출력
  index low, high, mid;
  low = 1; high = n;
  location = 0;
  while (low <= high && location == 0) {
    mid = (low + high) / 2;    // 정수나눗셈
    if (x == S[mid]) location = mid;
    else if (x < S[mid]) high = mid - 1;
    else low = mid + 1;
  }
}
```

✓ while-루프: 아직 검사할 항목이 있고, x를 찾지 못했나?

2001 -03 -19 알고리즘 강의 슬라이드 1a 서막 17

사색

- 이분검색 알고리즘으로 키를 찾기 위해서 S에 있는 항목을 몇 개나 검색해야 하는가?
  - ✓ while 문을 수행할 때마다 검색 대상의 총 크기가 반 씩 감소하기 때문에 최악의 경우라도 $\lg n + 1$개만 검사하면 된다.

2001 -03 -19 알고리즘 강의 슬라이드 1a 서막 18

비교: 순차검색 vs. 이분검색

배열의 크기	순차검색	이분검색
n	n	$\lg n + 1$
128	128	8
1,024	1,024	11
1,048,576	1,048,576	21
4,294,967,296	4,294,967,296	33

* 최악의 경우

n번째 피보나찌 수 구하기

<u>피보나찌(Fibonacci) 수열의 정의</u>

$$f_0 = 0$$
$$f_1 = 1$$
$$f_n = f_{n-1} + f_{n-2} \qquad \text{for } n \geq 2$$

예: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, …

피보나찌 수 구하기 알고리즘 (재귀적 방법)

- 문제: *n*번째 피보나찌 수를 구하라.
- 입력: 양수 *n*
- 출력: *n* 번째 피보나찌 수
- 알고리즘:

```
int fib (int n) {
  if (n <= 1)
    return n;
  else
    return fib(n-1) + fib(n-2);
}
```

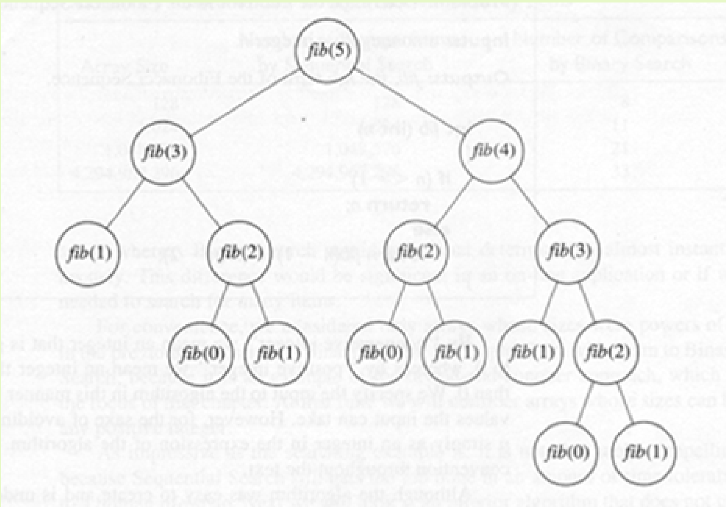
2001-03-19 알고리즘 강의 슬라이드 1a 서막 21

사색

- 피보나찌 수 구하기 재귀 알고리즘은 수행속도가 매우 느리다.
  - ✓ 이유: 같은 피보나찌 수를 중복 계산
  - ✓ 예: `fib(5)`계산에 `fib(2)` 3번 중복계산

2001-03-19 알고리즘 강의 슬라이드 1a 서막 22

fib(5)의 재귀 트리



fib(n)의 함수 호출 횟수 계산

$T(n) = fib(n)$을 계산하기 위하여 *fib* 함수를 호출하는 횟수
즉, 재귀 트리 상의 마디의 개수

$T(0) = 1$

$T(1) = 1$

$T(n) = T(n-1) + T(n-2) + 1$ for $n \geq 2$

$> 2 \times T(n-2)$ 왜냐하면 $T(n-1) > T(n-2)$

$> 2^2 \times T(n-4)$

$> 2^3 \times T(n-6)$

$\ldots$

$> 2^{n/2} \times T(0)$

$= 2^{n/2}$

계산한 호출 횟수의 검증

정리: 재귀적 알고리즘으로 구성한 재귀 트리의 마디의 수를 T(n)이라고 하면, $n \geq 2$인 모든 n에 대하여 $T(n) > 2^{n/2}$ 이다.

증명: (n에 대한 수학적 귀납법으로 증명)

귀납출발점: $T(2) = T(1) + T(0) + 1 = 3 > 2 = 2^{2/2}$

$T(3) = T(2) + T(1) + 1 = 5 > 2.83 \approx 2^{3/2}$

귀납가정: $2 \leq m <$ n인 모든 m 에 대해서 $T(m) > 2^{m/2}$ 이라 가정

귀납절차: $T(n) > 2^{n/2}$임을 보이면 된다.

$$
\begin{aligned}
T(n) &= T(n-1) + T(n-2) + 1 \\
&> 2^{(n-1)/2} + 2^{(n-2)/2} + 1 \quad \text{[귀납가정에 의하여]} \\
&> 2^{(n-2)/2} + 2^{(n-2)/2} \\
&= 2 \times 2^{(n/2)-1} \\
&= 2^{n/2}
\end{aligned}
$$

피보나찌 수 구하기 알고리즘 (반복적 방법)

```
int fib2 (int n) {
  index i;
  int f[0..n];
  f[0] = 0;
  if (n > 0) {
    f[1] = 1;
    for (i = 2; i <= n; i++)
      f[i] = f[i-1] + f[i-2];
  }
  return f[n];
}
```

사색

- 반복 알고리즘은 수행속도가 훨씬 더 빠르다.
  - 이유: 중복 계산이 없음
- 계산하는 항의 총 개수
  - $T(n) = n + 1$
  - 즉, $f[0]$부터 $f[n]$까지 한번씩 만 계산

두 피보나찌 알고리즘의 비교

n	n+1	$2^{n/2}$	반복	재귀(하한)
40	41	1,048,576	41*ns*	1048μs
60	61	1.1×10^{9}	61*ns*	1*s*
80	81	1.1×10^{12}	81*ns*	18*min*
100	101	1.1×10^{15}	101*ns*	13*days*
120	121	1.2×10^{18}	121*ns*	36*years*
160	161	1.2×10^{24}	161*ns*	3.8×10^{7}*years*
200	201	1.3×10^{30}	201*ns*	4×10^{13}*years*

1 ns = 10^{-9} second 1 µs = 10^{-6} second

알고리즘의 분석(Analysis)

- 시간복잡도(Time Complexity) 분석
  - 입력크기에 따라서 단위연산이 몇 번 수행되는지 결정하는 절차
- 표현 척도
  - 단위연산(basic operation)
    - 비교(comparison), 지정(assignment)
  - 입력크기(input size)
    - 배열의 크기, 리스트의 길이, 행렬에서 행과 열의 크기, 트리에서 마디와 이음선의 수

2001 -03 -19 알고리즘 강의 슬라이드 1a 서막 29

분석 방법의 종류 (上)

- 모든 경우 분석(Every-case analysis)
  - 입력크기에만 종속
  - 입력 값과는 무관하게 결과 값은 항상 일정
- 최악의 경우 분석(Worst-case analysis)
  - 입력크기와 입력 값 모두에 종속
  - 단위연산이 수행되는 횟수가 최대인 경우 선택

2001 -03 -19 알고리즘 강의 슬라이드 1a 서막 30

분석 방법의 종류 (下)

- 평균의 경우 분석(Average-case analysis)
  - 입력크기와 입력 값 모두에 종속
  - 모든 입력에 대해서 단위연산이 수행되는 기대치(평균)
  - 각 입력에 대해서 확률 할당 가능
  - 일반적으로 최악의 경우보다 계산이 복잡
- 최선의 경우 분석(Best-case analysis)
  - 입력크기와 입력 값 모두에 종속
  - 단위연산이 수행되는 횟수가 최소인 경우 선택

알고리즘: 배열 덧셈

- 문제: 크기가 n인 배열 S의 모든 수를 더하라
- 입력: 양수 n, 배열 $S[1..n]$
- 출력: 배열 S에 있는 모든 수의 합
- 알고리즘:

```
number sum (int n, const number S[]) {
  index i;
  number result;

  result = 0;
  for (i = 1; i <= n; i++)
    result = result + S[i];
  return result;
}
```

알고리즘: 배열 덧셈 시간복잡도 분석

- 단위연산: 덧셈
- 입력크기: 배열의 크기 n
- 모든 경우 분석:
  - ✓ 배열 내용에 상관없이 for-루프가 n번 반복된다.
  - ✓ 각 루프마다 덧셈이 1회 수행된다.
  - ✓ 따라서, n에 대해서 덧셈이 수행되는 총 횟수는 $T(n) = n$ 이다.

- 단위연산: 지정문 (for-루프의 첨자 지정문 포함)
- 입력크기: 배열의 크기 n
- 모든 경우 분석:
  - ✓ 배열 내용에 상관없이 for-루프가 n번 반복된다.
  - ✓ 따라서, 지정문이 $T(n) = n + n + 1$번 수행된다.

알고리즘: 교환정렬

- 문제: 비내림차순으로 n개의 키를 정렬
- 입력: 양수 n, 배열 $S[1..n]$
- 출력: 비내림차순으로 정렬된 배열
- 알고리즘:

```
void exchangesort (int n, keytype S[]) {
  index i, j;

  for (i = 1; i <= n-1; i++)
    for (j = i+1; j <= n; j++)
      if (S[j] < S[i])
        exchange S[i] and S[j];
}
```

알고리즘: 교환정렬 시간복잡도 분석 I

- 단위연산: 조건문 (S[j]와 S[i]의 비교)
- 입력크기: 정렬할 항목의 수 n
- 모든 경우 분석:
  - j-루프가 수행될 때마다 조건문 1번씩 수행
  - 조건문의 총 수행횟수
    - $i=1$: j-루프 $n-1$ 번 수행
    - $i=2$: j-루프 $n-2$ 번 수행
    - $i=3$: j-루프 $n-3$ 번 수행
    - $i=n-1$: j-루프 1 번 수행
    - 따라서 $T(n)=(n-1)+(n-2)+\cdots+1=\frac{(n-1)n}{2}$

2001 -03 -19 알고리즘 강의 슬라이드 1a 서막 35

알고리즘: 교환정렬 시간복잡도 분석 II

- 단위연산: 교환하는 연산
  (exchange S[j] and S[i])
- 입력크기: 정렬할 항목의 수 n
- 최악의 경우 분석:
  - 조건문의 결과에 따라서 교환 연산의 수행여부가 결정된다.
  - 최악의 경우 = 조건문이 항상 참(true)이 되는 경우
    = 입력 배열이 꺼꾸로 정렬되어 있는 경우

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

2001 -03 -19 알고리즘 강의 슬라이드 1a 서막 36

알고리즘: 순차검색 시간복잡도 분석 (최악)

- 단위연산: 배열의 아이템과 키 x와 비교 연산
  `(S[location] != x)`
- 입력크기: 배열 안에 있는 아이템의 수 n
- 최악의 경우 분석:
  - ✓ x가 배열의 마지막 아이템이거나, x가 배열에 없는 경우 단위연산이 n번 수행된다.
  - ✓ 따라서, $W(n) = n$

¶ 순차검색 알고리즘의 경우 입력배열의 값에 따라서 검색하는 횟수가 달라지므로, 모든 경우 분석은 불가능하다.

2001 -03 -19 알고리즘 강의 슬라이드 1a 서막 37

알고리즘: 순차검색 시간복잡도 분석 (평균)

- 단위연산: 배열의 아이템과 키 x와 비교 연산
  `(S[location] != x)`
- 입력크기: 배열 안에 있는 아이템의 수 n
- 평균의 경우 분석:
  - ✓ 배열의 아이템이 모두 다르다고 가정한다.
  - ✓ 경우 1: x가 배열 S안에 있는 경우만 고려
    - ✓ $1 \le k \le n$에 대해서 x가 배열의 k번째 있을 확률 $= 1/n$
    - ✓ x가 배열의 k번째 있다면, 를 찾기 위해서 수행하는 단위연산의 횟수 $= k$
    - ✓ 따라서, $A(n) = \sum_{k=1}^{n} k \times \frac{1}{n} = \frac{1}{n} \times \sum_{k=1}^{n} k = \frac{1}{n} \times \frac{n(n+1)}{2} = \frac{n+1}{2}$

2001 -03 -19 알고리즘 강의 슬라이드 1a 서막 38

(다음 슬라이드에서 계속)

알고리즘: 순차검색 시간복잡도 분석 (평균) [계속]

- 경우2: x가 배열 S안에 없는 경우도 고려
  - x가 배열 S안에 있을 확률을 p라고 하면,
    - x가 배열의 k번째 있을 확률 $= p/n$
    - x가 배열에 없을 확률 $= 1-p$
  - 따라서,

$$A(n) = \sum_{k=1}^{n} (k \times \frac{p}{n}) + n(1-p)$$
$$= \frac{p}{n} \times \frac{n(n+1)}{2} + n(1-p)$$
$$= n(1-\frac{p}{2}) + \frac{p}{2}$$

  - $p = 1 \Rightarrow A(n) = (n+1)/2$
    $p = 1/2 \Rightarrow A(n) = 3n/4 + 1/4$

알고리즘: 순차검색 시간복잡도 분석 (최선)

- 단위연산: 배열의 아이템과 키 x와 비교 연산 (`S[location] != x`)
- 입력크기: 배열 안에 있는 아이템의 수 n
- 최선의 경우 분석:
  - x가 $S[1]$일 때, 입력의 크기에 상관없이 단위연산이 1번만 수행된다.
  - 따라서, $B(n) = 1$

사색

- 최악, 평균, 최선의 경우 분석 방법 중에서 어떤 분석이 가장 정확한가?
- 최악, 평균, 최선의 경우 분석 방법 중에서 어떤 분석을 사용할 것인가?

정확도 분석

- 알고리즘이 의도한 대로 수행되는지를 증명하는 절차
- 정확한 알고리즘이란?
  - ✓ 어떠한 입력에 대해서도 답을 출력하면서 멈추는 알고리즘
- 정확하지 않은 알고리즘이란?
  - ✓ 어떤 입력에 대해서 멈추지 않거나, 또는
  - ✓ 틀린 답을 출력하면서 멈추는 알고리즘