창의적 소프트웨어 설계

12주차 실습 – Big-O notation & Data structure

노인우, inwoo13@hanyang.ac.kr
한중수, soohan@hanyang.ac.kr

Overview

목표

- Big-O notation
- (Time complexity in sorting)
- Practice – Finding combination
- Appendix

Big-O notation

- 함수의 입력값 (정의역의 원소)이 커짐에 따라 출력값 (원소의 함수값)이 얼마나 빨리 커지는가?
- 알고리즘을 평가 시 '시간과 공간의 사용량'이 관점
- 특히 '시간'에 신경을 쓰며 따로 관계를 말하지 않는 이상 '입력값의 크기와 소요 시간의 관계'

Big-O notation

- 프로그램을 실행시 정확한 단계(step) 수를 결정하는 작업은 어렵다.
- 5n + 7 이라는 것은 5n 정도
- O(5n + 7) = O(5n), O(n^2 + 2n) = O(n^2)
  (여기서 = 기호는 '같다(equals)'가 아니라 '이다(is)'로 해석)
- "Big-O 표기는 시간 복잡도 중 해당 차수의 시간복잡도를 가지거나, 그보다 더 낮은 차수의 시간복잡도를 가진다" 의미

  cf. Big-θ (Big-theta) notation ...

Big-O notation

- 위로 갈수록 알고리즘이 매우 빠름
- 아래로 갈수록 n의 값이 커질수록 급격하게 수행 시간 증가

1	1	1	1	1	1	1	1	1	1
lg n	0	1	1.58	2	3	4	5	6	9.97
n	**1**	**2**	**3**	**4**	**8**	**16**	**32**	**64**	**1000**
n lg n	0	2	4.75	8	24	64	120	384	9966
n^2	1	4	9	16	64	256	1024	4096	1000000
n^3	1	8	27	64	512	4096	32768	262144	1000000000
2^n	2	4	8	16	256	65536	4294967296	약 1844경	약 1.07×10^{301}
n!	1	2	6	24	40320	20922789888000	약 2.63×10^{35}	약 1.27×10^{89}	약 4.02×10^{2567}

- 지수 급 이상의 시간 복잡도를 가지면 거의 써먹을 수 없다.
  암호 찾기의 경우 (영문소문(26)+영문대문(26)+숫자(10)+공백)^63

Time complexity in sorting

- 자료구조 별 시간복잡도 (참고자료)

SEARCHING

Algorithm	Data Structure	Time Complexity	Space Complexity
Linear Search	Array	O(n)	O(1)
Binary Search	Sorted Array	O(log (n))	O(1)
DFS	Graph with E edges and V vertices	O(|E|+|V|)	O(|V|)
BFS		O(|E|+|V|)	O(|V|)

SORTING

Algorithm	Data Structure	Time Complexity	Space Complexity
Bubble Sort	Array	O (n^2)	O (1)
Heap Sort	Array	O (n log (n))	O (1)
Merge Sort	Array	O (n log (n))	O (n)
Quick Sort	Array	O (n log (n))	O (n)

Practice – Finding combination

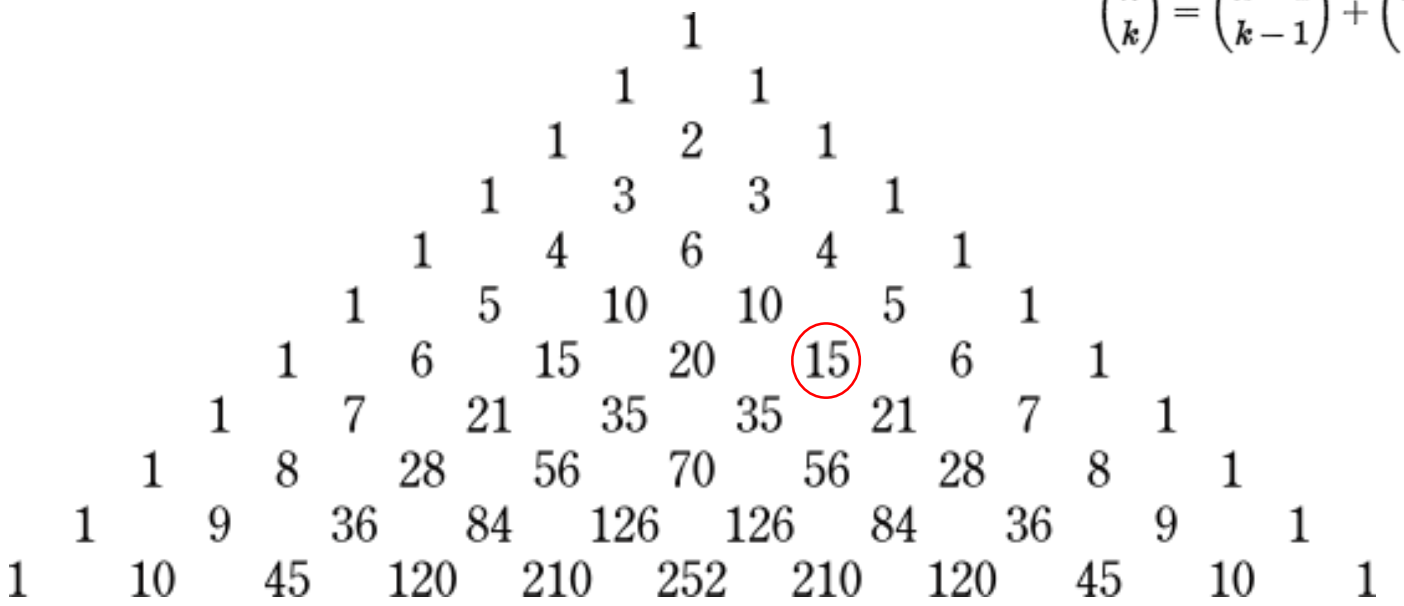
- 어떻게 해서든 지수 형태의 알고리즘 코드를 개선하여 n^x 다항 형태로 만든다면 엄청난 성능 향상을 기대할 수 있다.
- 실습 – 지수 형태의 시간복잡도를 가진 nCr 구하는 코드를 n^3 시간복잡도를 가지는 코드로 전환한다.
- 전환 후, time 함수를 써서 시간을 비교해본다.

Practice – Finding combination

- 인자로 n, r을 입력 받아 nCr을 구하는 프로그램을 작성하시오.

- 방법: 파스칼 삼각형 (Pascal's triangle) 이용한 조합 구하기

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$$



6C4 = 5C3 + 5C4

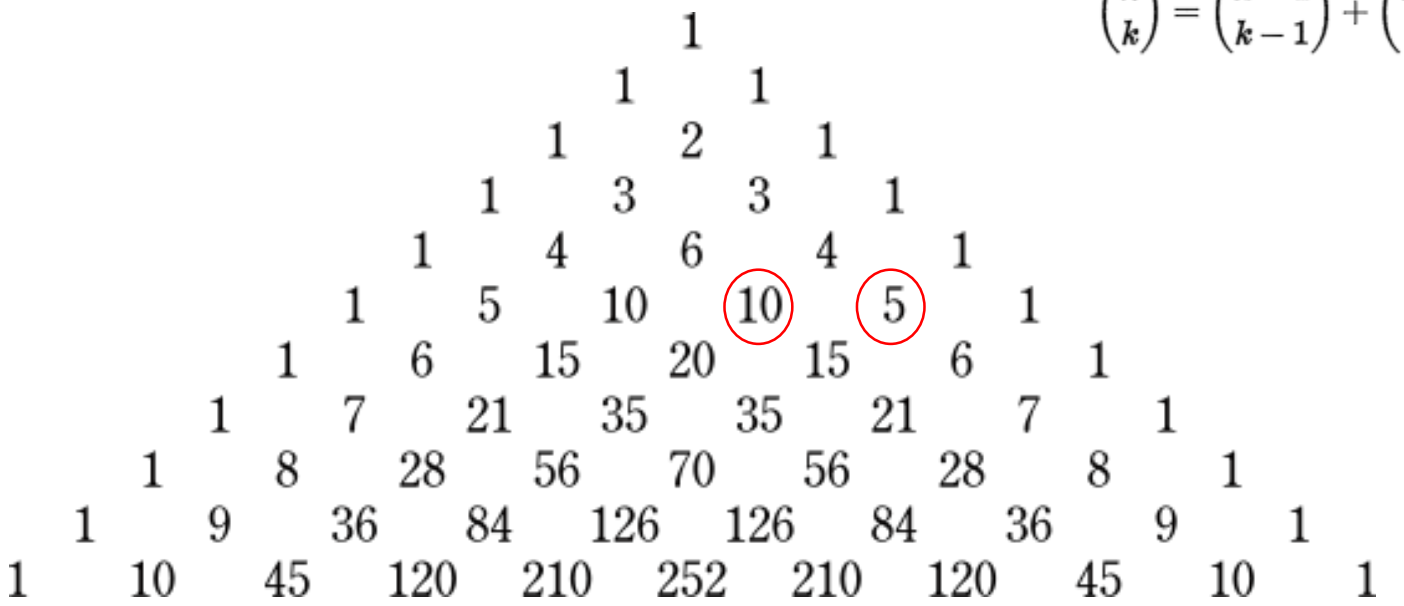
15 = 10 + 5

Practice – Finding combination

- 인자로 n, r을 입력 받아 nCr을 구하는 프로그램을 작성하시오.

- 방법: 파스칼 삼각형 (Pascal's triangle) 이용한 조합 구하기

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$$



5C4 = 4C3 + 4C4

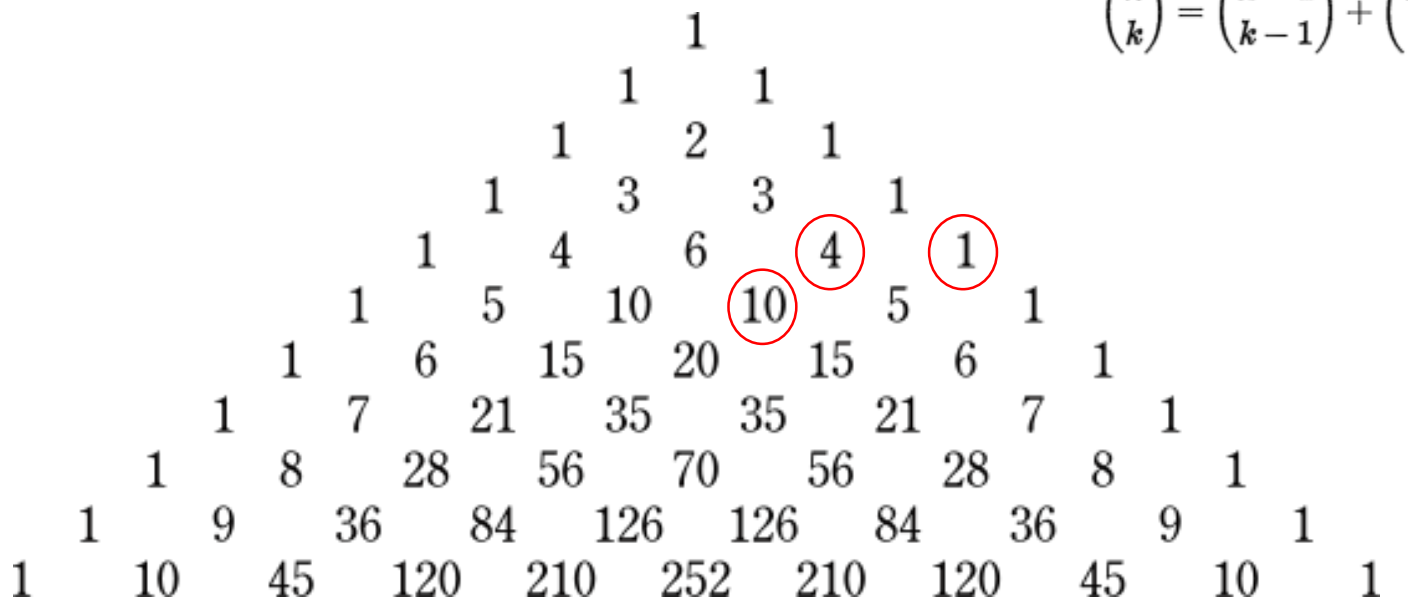
5 = 4 + 1

Practice – Finding combination

- 인자로 n, r을 입력 받아 nCr을 구하는 프로그램을 작성하시오.

- 방법: 파스칼 삼각형 (Pascal's triangle) 이용한 조합 구하기

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$$

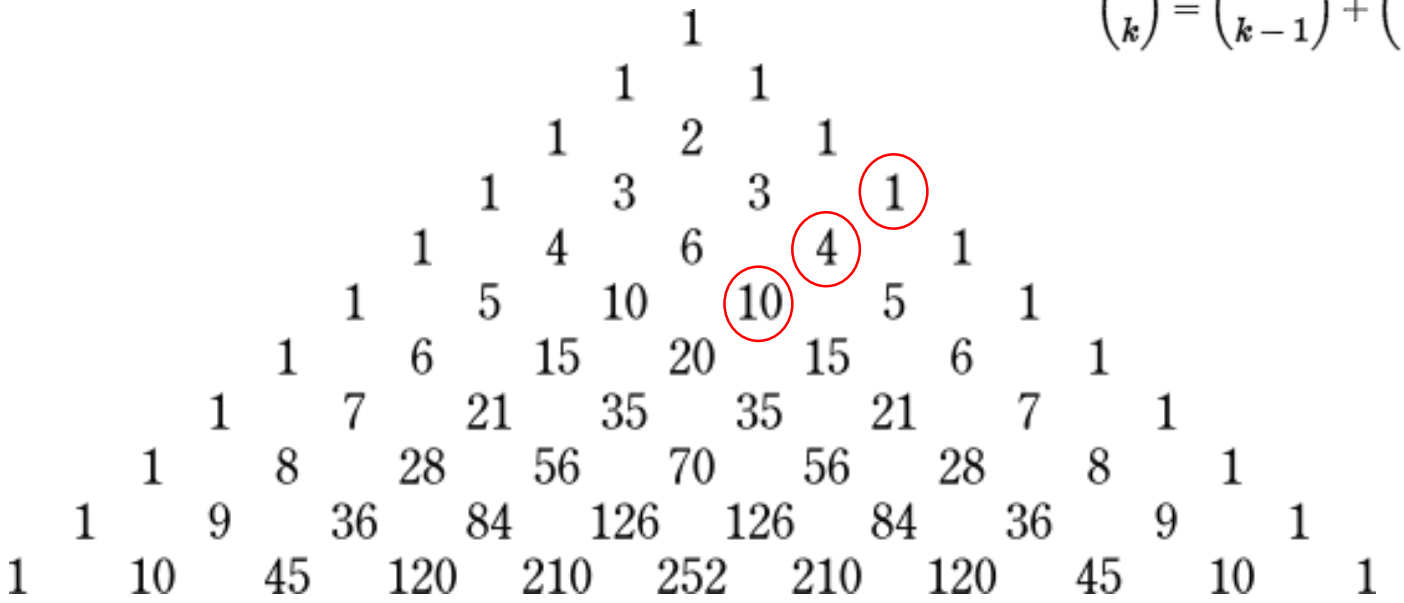


Practice – Finding combination

- 인자로 n, r을 입력 받아 nCr을 구하는 프로그램을 작성하시오.

- 방법: 파스칼 삼각형 (Pascal's triangle) 이용한 조합 구하기

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$$



6C4 = 5C3 + 4C3 + 3C3

15 = 10 + 5

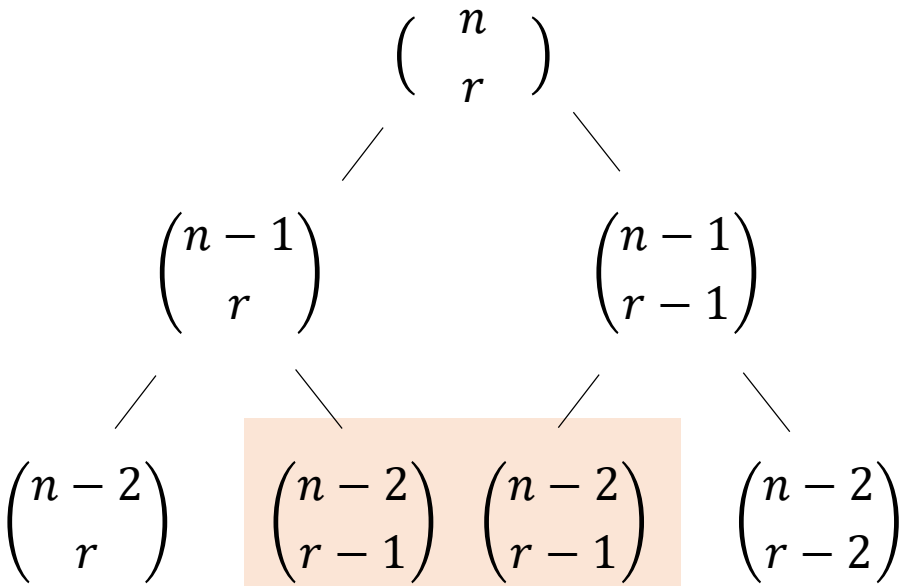
Practice – Finding combination

```
#include <iostream>
#include <cstdlib>
int combination(int n,int r) {
    if(n==r) return 1;
    if(r==1) return n;
    return combination(n-1,r) + combination(n-1,r-1);
}
int main(int argc, char *argv[]) {
   int m, n;
   long long int r_a;
   m = atoi(argv[1]);
   n = atoi(argv[2]);

   r_a = combination(m,n);
  std::cout << r_a << std::endl;
  return 0;
}
```

Practice – Finding combination

- 본 코드의 시간복잡도는 2^n
- 비효율적인 중복 계산이 일어남
- 재귀적인 구조에서 흔히 발생
  (피보나치 수열도 마찬가지)



Practice – Finding combination

```
#include <iostream>
#include <cstdlib>
long long int in[41][41];
int main(int argc, char *argv[]) {
   int m,n,i,j,k;
   int r_a;
   m = atoi(argv[1]);
   n = atoi(argv[2]);
   for(i=0;i<=40;i++)
      in[0][i]=1;
   for(i=0;i<=40;i++)
      for(j = i+1; j<=40; j++)
         for(k=0;k<j;k++)
            ?  +=  ?
   std::cout << in[n][m] << std::endl;
}
```

- 본 코드의 시간복잡도는 n^3
- 메모리를 사용해 중복계산 피함 (4*41*41 = 6724byte ≒ 7kb 사용)
- 탐색 시 인덱스 접근으로 O(1) 시간복잡도

Practice – Finding combination

◆ 수행시간 비교

	$ time ./big1 10 3	$ time ./big2 10 3
10C3	120 real 0m0.001s user 0m0.000s sys 0m0.000s	120 real 0m0.001s user 0m0.000s sys 0m0.000s
29C13	67863915 real 0m0.627s user 0m0.396s sys 0m0.000s	67863915 real 0m0.001s user 0m0.000s sys 0m0.000s
33C15	1037158320 real 0m7.735s user 0m6.132s sys 0m0.000s	1037158320 real 0m0.001s user 0m0.000s sys 0m0.000s
37C18	492762716 real 2m13.881s user 1m50.832s sys 0m0.000s	17672631900 real 0m0.001s user 0m0.000s sys 0m0.000s

※ 실행중단: (윈도우) Ctrl + C 혹은 Ctrl + Z / (Mac) Command + C 혹은 Command + Z

참고자료

- Time complexity
  https://rob-bell.net/2009/06/a-beginners-guide-to-big-o-notation/
- Big-O & Theta natation
  https://www.khanacademy.org/computing/computer-science/algorithms/asymptotic-notation/a/big-big-theta-notation
- 파스칼 삼각형
  https://en.wikipedia.org/wiki/Pascal%27s_triangle
- 동적계획법
  https://en.wikipedia.org/wiki/Dynamic_programming


Q&A

Appendix #1. 동적 계획법

- Time complexity
  https://rob-bell.net/2009/06/a-beginners-guide-to-big-o-notation/
- Big-O & Theta natation
  https://www.khanacademy.org/computing/computer-science/algorithms/asymptotic-notation/a/big-big-theta-notation
- 파스칼 삼각형
  https://en.wikipedia.org/wiki/Pascal%27s_triangle
- 동적계획법
  https://en.wikipedia.org/wiki/Dynamic_programming

Appendix #1. Dynamic Programming (동적 계획법)

- 문제를 여러 하위 문제(subproblem)으로 나누어 푼 후, 그것을 결합하여 최종적인 목적에 도달하는 것
- 같은 하위 문제가 나왔을 경우, 간단하게 해결 가능
- 계산 횟수를 줄일 때 유용,
  특히 하위 문제의 수가 기하급수적으로 증가할 때
- 문제에 대한 최적해가 부분문제에 대해서도 역시 최적해

Cf. 그리디 알고리즘 (http://ko.Wikipedia.org/wiki/탐욕_알고리즘)