Performance Analysis

Prof. Ki-Hoon Lee
Dept. of Computer Engineering
Kwangwoon University

Performance Analysis

- A priori estimates of performance
- Space complexity: the amount of memory that a program needs to run to completion
- Time complexity: the amount of computer time that a program needs to run to completion

Space Complexity

- The space needed by a program is seen to be the sum of the following components:
  - A fixed part that is independent of the characteristics (e.g., number, size) of the inputs and outputs
    - Example: space for the code, constants, etc.
  - A variable part whose size is dependent on the particular problem instance being solved
    - Example: space for variables depending on instance characteristics, recursion stack space

- $S(P) = c + S_p(\text{instance characteristics})$

Space Complexity (cont.)

- When analyzing the space complexity of a program, we shall concentrate solely on estimating S_p(instance characteristics)
- Instance characteristics: quantities related to the number and magnitude of the inputs to and outputs from the program

Space Complexity (cont.)

```
float Sum(float *a, const int n)
{
   float s = 0;
   for(int i = 0 ; i < n ; i++)
      s += a[i];
   return s;
}
```

- The program instances are characterized by n, the number of elements to be summed
- The space needed by the function is independent of n and $S_{Sum}(n) = 0$

Space Complexity (cont.)

```
float Rsum(float *a, const int n)
{
  if(n <= 0) return 0;
  else return(Rsum(a, n-1) + a[n-1]);
}
```

- Each call to *Rsum* requires at least 4 words (n, a, the returned value, and the return address)
- The depth of recursion: $n + 1$
- The recursion stack space: $4(n + 1)$

Time Complexity

- The time, $T(P)$, taken by a program P is the sum of the compile time and the run (or execution) time

- We shall concern ourselves with just the run time t_p of a program

Time Complexity (cont.)

- Program step: a segment of a program that has an execution time that is independent of the instance characteristics
- The number of steps
  - Expressions and assignment statements: 1
  - Iteration statements (e.g., for, while)
    - We shall consider the step counts only for the control part of these statements
    - Example: while(<the control part>) { … }
  - …

Time Complexity (cont.)

```
line float Sum(float * a, const int n)
1    {
2       float s = 0;
3       for(int i = 0 ; i < n ; i++)
4          s+= a[i];
5       return s;
6    }
```

행 번호	s/e	빈도	단계 수
1	0	1	0
2	1	1	1
3	1	$n+1$	$n+1$
4	1	n	n
5	1	1	1
6	0	1	0
	총 단계 수		$2n+3$

- s/e: the number of steps per execution of a statement
- The frequency of line 3 is $n + 1$ not n because i has to be incremented to n

Time Complexity (cont.)

- The best-case step count
  - The minimum number of steps that can be executed for the given parameters
- The worst-case step count: maximum
- The average-case step count: average
- Example: find an element in an array of size n
  - best: 1
  - worst: n
  - average: $n/2$

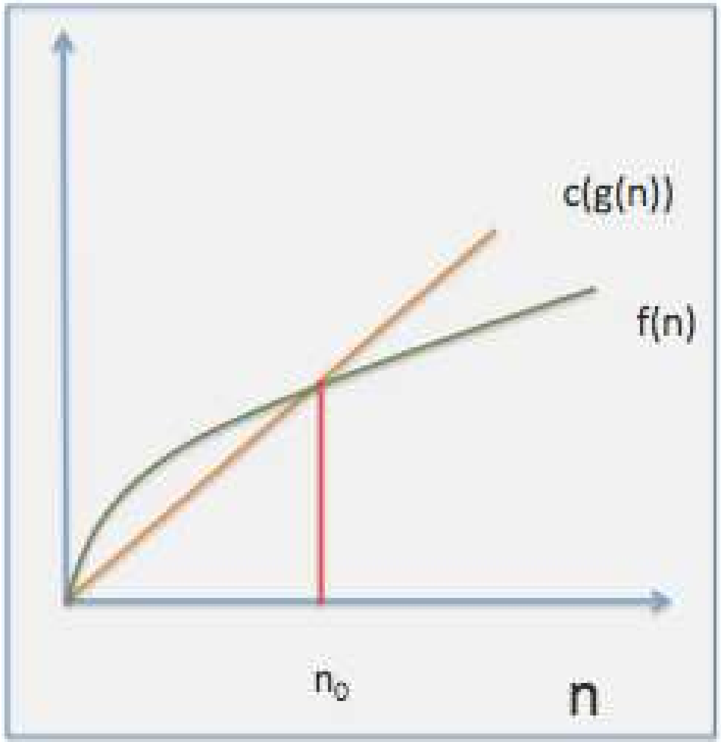
Asymptotic Notation

- Determining the exact step count is difficult and not very worthwhile

- We introduce some terminology that will enable us to make meaningful (but inexact) statements about the time and space complexities of a program

Big "oh"

- Definition: $f(n) = O(g(n))$ iff there exist positive constants c and n_0 such that $f(n) \le cg(n)$ for all n, $n \ge n_0$.
  - read as "*f* of *n* is big oh of *g* of *n*"
  - iff: if and only if



Big "oh" (cont.)

- $3n + 2 \quad = O(n)$ $\quad (c=4, n_0=2)$
- $3n + 3 \quad = O(n)$ $\quad (c=4, n_0=3)$
- $100n + 6 = O(n)$ $\quad (c=101, n_0=10)$
- $10n^2 + 4n + 2 = O(n^2)$ $\quad (c=11, n_0=5)$
- $1000n^2 + 100n - 6 = O(n^2)$ $\quad (c=1001, n_0=100)$
- $6*2^n + n^2 = O(2^n)$ $\quad (c=7, n_0=4)$
- $3n + 3 = O(n^2)$ $\quad (c=3, n_0=2)$
- $10n^2 + 4n + 2 = O(n^4)$ $\quad (c = 10, n_0=2)$
- $3n + 2 \neq O(1)$
- $10n^2 + 4n +2 \neq O(n)$

Big "oh" (cont.)

- O(1)
- O(*log* n)
- O(n)
- O(n *log* n)
- O(n^2)
- O(n^3)
- O(2^n)
- O(n!)

$\log n$	n	$n \log n$	n^2	n^3	2^n
0	1	0	1	1	2
1	2	2	4	8	4
2	4	8	16	64	16
3	8	24	64	512	256
4	16	64	256	4096	65,536
5	32	160	1024	32,768	4,294,967,296