Izmir Institute of Technology

CENG 115
Discrete Structures

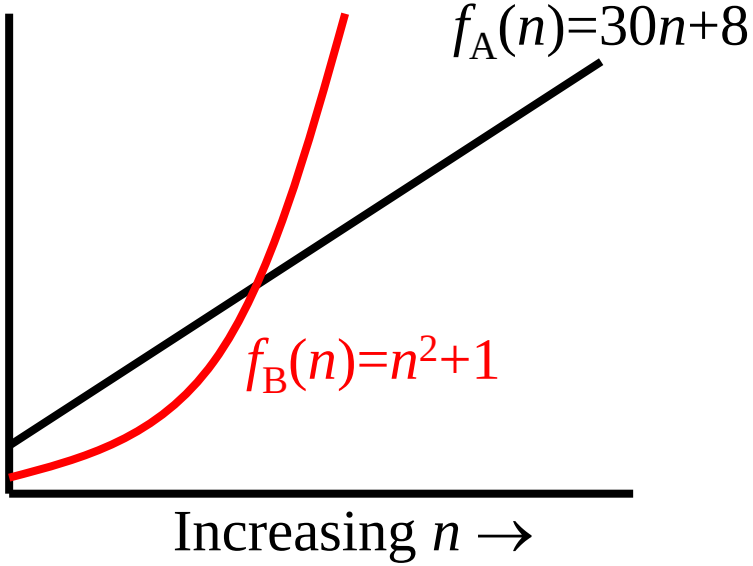
Slides are based on the Text
***Discrete Mathematics & Its Applications* (6th Edition)**
by Kenneth H. Rosen

Complexity of Algorithms

Discrete Mathematics, Rosen, 6th ed., § 3.3

Last lecture's review

- Higher order functions are faster growing.
- $f(n)=30n+8$ is $O(n)$.
- $f(n)=3n^2+500$ is $O(n^2)$.
- $f(n)= n\log n + n^2 = O(n^2)$.
- $O(n) + O(1) = O(n)$.
- $O(n^2) + O(n)= O(n^2)$.
- $O(n^2) \cdot O(n)= O(n^3)$.



Algorithmic Complexity

- We are talking about the *computational complexity* of algorithms. How difficult to perform computation? How can the efficiency of an algorithm be analyzed?
- Analysis of the time (# of steps) required to solve a problem is *time complexity*.
- The computer memory required to solve a problem is *space complexity*.

Time Complexity

- *Time complexity* is described in terms of the number of operations for a certain size of input. (*E.g.* searching a long list takes more time than searching a short one.)
- Therefore, complexity is usually expressed as a *function* of input length.

Example 1: *max* algorithm

- Problem: Find the *order of growth* (*O*) of the *max* algorithm for a list of *n* elements.
- Solution is reached through determining the number of operations (comparisons, assignments etc.)
- We can assume that each line of code takes some constant time every time it is executed.

Complexity analysis of *max*

procedure *max*($a_1, a_2, \ldots, a_n$: integers)

$v := a_1$ t_1

for $i := 2$ **to** n t_2

  if $a_i > v$ **then** $v := a_i$ t_3

} Standard times for execution of each line.

What's the function for the *exact* total time?

$$f(n) = t_1 + \sum_{i=2}^{n} (t_2 + t_3)$$

Complexity analysis, *cont.*

- Let us also assume that the time spent for executing all lines of code are the same, that is $t_1=t_2=t_3=t$.
- Then,

$$f(n) = t_1 + \sum_{i=2}^{n} (t_2 + t_3) = t + \sum_{i=2}^{n} (2t)$$

$$= t + 2t(n-1) = t(2n-1)$$

Complexity analysis, *cont.*

- What is the simplest form of the order of growth of $f(n)$?

$$f(n) = t(2n-1)$$

$$\text{is } O(1)\cdot(O(n)-O(1))$$

$$\text{is } O(1)\cdot O(n)$$

$$\text{is } O(n)$$

Worst Case Time Complexity

- *Worst-case* performance of an algorithm occurs when the largest number of operation needed to perform the task for a certain size of input.
- E.g. In the linear search algorithm, if the element you are searching for is at the end (or not in the list), you go through the entire list to discover that.

Example 2: Linear Search

procedure *linear search* (x: integer, a_1, a_2, …, a_n: distinct integers)

$i := 1$	t_1
while $(i \leq n \wedge x \neq a_i)$	t_2
$\quad i := i + 1$	t_3
if $i \leq n$ **then** *location* $:= i$	t_4
else *location* $:= 0$	t_5

What is the *worst-case* time comlexity?

Linear search analysis

- Worst case time complexity exact order:

$$f(n) = t_1 + \left(\sum_{i=1}^{n} (t_2 + t_3) \right) + t_4 + t_5 \text{ is } O(?)$$

$$f(n) \text{ is } O(1) + O(n) + O(1) + O(1) = O(n)$$

- What about the best case? That is the complexity for best possible input.

$$f(n) = t_1 + t_2 + t_4 \text{ is } O(1)$$

Example 3: Binary Search

procedure *binary search* (x:integer, $a_1, a_2, \ldots, a_n$: ordered integers)

$i := 1$
$j := n$
} $O(1)$

while $i<j$ **begin**

$m := \lfloor (i+j)/2 \rfloor$

if $x>a_m$ **then** $i := m+1$ **else** $j := m$
} $O(1)$

end

if $x = a_i$ **then** *location* $:= i$

else *location* $:= 0$
} $O(1)$

Key question:

How many loop iterations?

Binary search analysis

- Suppose $n=2^k$.
- Original list from $i=1$ to $j=n$ contains n elements.
- Each iteration: Size $j-i+1$ of range is cut in half.
- Loop terminates when size of range is $1=2^0$ ($i=j$).
- Therefore, number of iterations is $= k = \log_2 n$.
- Complexity ≡ # of iterations: $O(\log_2 n) = O(\log n)$.
- Even for $n \neq 2^k$ (not an integral power of 2), time complexity is still $O(\log_2 n) = O(\log n)$.

Complexity of Selection Sort

```
procedure selection sort(a1,a2,...,an)
for i:=1 to n-1
begin
   m:=i
   for j:=i+1 to n
      if aj<am then
         m:=j
   interchange am and ai
end {a1,...,an are sorted}
```

Worst-case: $\Theta(n^2)$

Best-case: $\Theta(n^2)$

← Even if the list is already sorted, **for** loop and interchange command are executed.

Complexity of Insertion Sort

procedure insertion sort($a_1,a_2,...,a_n$: real numbers with $n \geq 2$)
for j:=2 to n
begin
  i:= j-1
  while $a_j < a_i \wedge i > 0$
    i:= i-1
  m:= a_j
  for k:=0 to j-i-2
    $a_{j-k} := a_{j-k-1}$
  $a_{i+1} := m$
end {$a_1,a_2,...,a_n$ are sorted}

Worst-case: $\Theta(n^2)$
Best-case: $\Theta(n)$

Best case:
If the list is already sorted,
only one comparison in **while**,
for loop is never executed.

Terminology of the Complexities

- $\Theta(1)$ Constant
- $\Theta(\log_c n)$ Logarithmic (same order $\forall c$)
- $\Theta(\log^c n)$ Polylogarithmic

(c is a constant.)

- $\Theta(n)$ Linear
- $\Theta(n \log n)$ $n \log n$
- $\Theta(n^c)$ Polynomial
- $\Theta(c^n)$, $c>1$ Exponential
- $\Theta(n!)$ Factorial

Computer Time Examples

#opers	$n=10$	$n=10^6$
$\log n$	$3 \cdot 10^{-9}$ s	$2 \cdot 10^{-8}$ s
n	10^{-8} s	10^{-3} s
$n \log n$	$3 \cdot 10^{-8}$ s	$2 \cdot 10^{-2}$ s
n^2	10^{-7} s	17 min
2^n	10^{-6} s	$>10^{300,000}$ years
$n!$	$3 \cdot 10^{-3}$ s	****

Assume:

10^{-9} second per operation.