CST207
DESIGN AND ANALYSIS OF ALGORITHMS

Lecture 2: Theoretical Analysis

Lecturer: Dr. Yang Lu

Email: luyang@xmu.edu.my

Office: A1-432

Office hour: TBD

Sequential Search Versus Binary Search

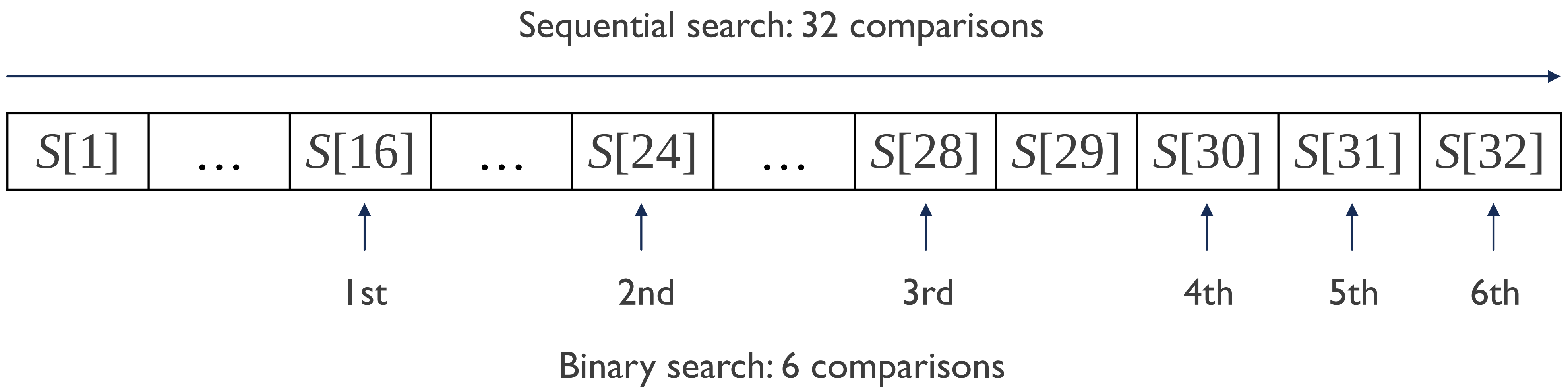
```
void seqsearch (int n,
                const keytype S[ ],
                keytype x,
                index& location)
{
    location = 1;
    while (location <= n && S[location] != x)
        location ++;
    if (location > n)
        location = 0;
}
```

```
void binsearch(int n,
               const keytype S[ ],
               keytype x,
               index& location)
{
    index low, high, mid;

    low = 1; high = n;
    location = 0;
    while (low <= high && location == 0){
        min = ⌊(low + high) / 2⌋;
        if (x == S[mid])
            location = mid;
        else if (x < S[mid])
            high = mid - 1;
        else
            low = mid + 1
    }
}
```

Sequential Search Versus Binary Search

- Assume S is a sorted array with 32 elements, and $x > S[32]$.



Sequential Search Versus Binary Search

- For an array with size 32, sequential search needs n comparisons but binary search only needs $\lg n + 1$ comparisons ($6 = \lg 32 + 1$).

Array Size	Number of Comparisons by Sequential Search	Number of Comparisons by Binary Search
128	128	8
1,024	1,024	11
1,048,576	1,048,576	21
4,294,967,296	4,294,967,296	33

The number of comparisons done by sequential search and binary search when x is larger than all the array items

Complexity Analysis

- In general, a time complexity analysis of an algorithm is the determination of how many times the basic operation is done for each value of the input size n.
- $T(n)$ is called **every-case time complexity**. It is defined as the number of times the algorithm does the basic operation for an instance of size n.

Every-Case Time Complexity

Example 1

when i=n, j=n+1>n, comparison does not execute

- For a given n, there are always n-1 passes through the for-i loop.
- For each for-i loop, there are n-1, n-2,…,1 passes though the for-j loop.
- There are always $T(n)$ comparisons for exchange sort.

```
void exchangesort (int n, keytype S[])
{
    index i, j;
    for (i=1; i<=n; i++)
        for (j=i+1; j<=n; j++)
            if (S[j] < S[i])
                exchange S[i] and S[j];
}
```

$$T(n) = (n\overbrace{-\,1) + (n-2) + (n-3) + \cdots + 1}^{\text{for-}i\text{ loop}} = \frac{(n-1)n}{2}$$

first for-j loop

Worst-Case and Best-Case Time Complexity

- For some algorithms, each run has different running time. Therefore, $T(n)$ does not exist.
- In this case, we use $W(n)$, **worst-case time complexity**, or $B(n)$, **best-case time complexity**, to measure the maximum or minimum number of times of basic operations.
- For sequential search, the complexity depends on both x and n.
  - The worst case is when x is the last element or x is not in the array.
  - The best case is when x is the first element.

$$W(n) = n,$$
$$B(n) = 1.$$

Average-Case Time Complexity

- When $T(n)$ does not exist, we may be interested in $A(n)$, **average-case time complexity**.
  - Not every time we have that good or bad luck, right?
- For sequential search:
  - The probability that x is in the kth slot is $1/n$.
  - The number of times to reach the kth slot is k.

$$A(n) = \sum_{k=1}^{n}\left(k \times \frac{1}{n}\right) = \frac{1}{n} \times \sum_{k=1}^{n} k = \frac{1}{n} \times \frac{n(n+1)}{2} = \frac{n+1}{2}.$$

How to Compare?

- Now we have two algorithms for the same problem:
  - Algorithm A needs two loops and only one basic operation in the loop: $T(n) = n^2$.
  - Algorithm B needs one loop and 1000 basic operation in the loop: $T(n) = 1000n$.
- Which one is more efficient?
  - When $n < 1000$, we choose Algorithm A.
  - What if we have no idea how large n will be?

Theoretical Analysis

- In the theoretical analysis of an algorithm, we are interested in the eventual behavior.
  - We compare algorithms for sufficiently large n.
- In this case, any algorithm with $T(n) = an$ will be eventually more efficient than any algorithm with $T(n) = bn^2$, no matter how large is a or how small is b.
- How to formally compare algorithms in the sense of "eventual"?

Asymptotic Notations

- Intuitively, just look at the dominant term.

$$g(n) = \cancel{0.1}n^3 + \cancel{10n^2 + 5n + 25}$$

  - Drop lower-order terms ($10n^2 + 5\text{n} + 25$).
  - Ignore constant coefficient (0.1).
- But we can't say that $g(n)$ equals to n^3.
  - It grows like n^3. But it doesn't equal to n^3.
- Use Θ (called "big theta") as the **order** of a function.
  - We can say that $g(n)$ is order of n^3.

$$g(n) \in \Theta(n^3)$$

Logarithm Review

Definition

$\log_b a$ is the unique number c s.t. $b^c = a$.

- Notations:
  - $\lg n = \log_2 n$ (binary logarithm)
  - $\ln n = \log_e n$ (natural logarithm)
  - $\lg^k n = (\lg n)^k$ (exponentiation)
  - $\lg \lg n = \lg(\lg n)$ (composition)
- Derivative:
  - $\frac{d(\log_a x)}{dx} = \frac{1}{x \ln a}$

- Useful identities for all real $a > 0, b > 0, c > 0$, and n, and where logarithm bases are not 1 :
  - $\log_c(ab) = \log_c a + \log_c b$
  - $\log_b a^n = n\log_b a$
  - $\log_b \left(\frac{1}{a}\right) = -\log_b a$
  - $\log_b a = (\log_a b)^{-1}$
  - $a^{\log_b c} = c^{\log_b a}$
  - $\log_b a = \frac{\log_c a}{\log_c b}$
  - $a = b^{\log_b a}$

Big O

Definition

For a given complexity function $f(n)$, $O(f(n))$ is the set of complexity functions $g(n)$ for which there exists some positive real constant c and some nonnegative integer N such that for all $n \geq N$,

$$g(n) \leq cf(n).$$

- $O(f(n))$ is a set of functions in terms of $f(n)$ that satisfy the definition.
- If $g(n) \in O(f(n))$, we say that $g(n)$ is "big O" of $f(n)$.
- No matter how large $g(n)$ is, it will eventually be smaller than $cf(n)$ for some c and some N.
- Big O puts an asymptotic upper bound on a function.

Display of Growth of Functions

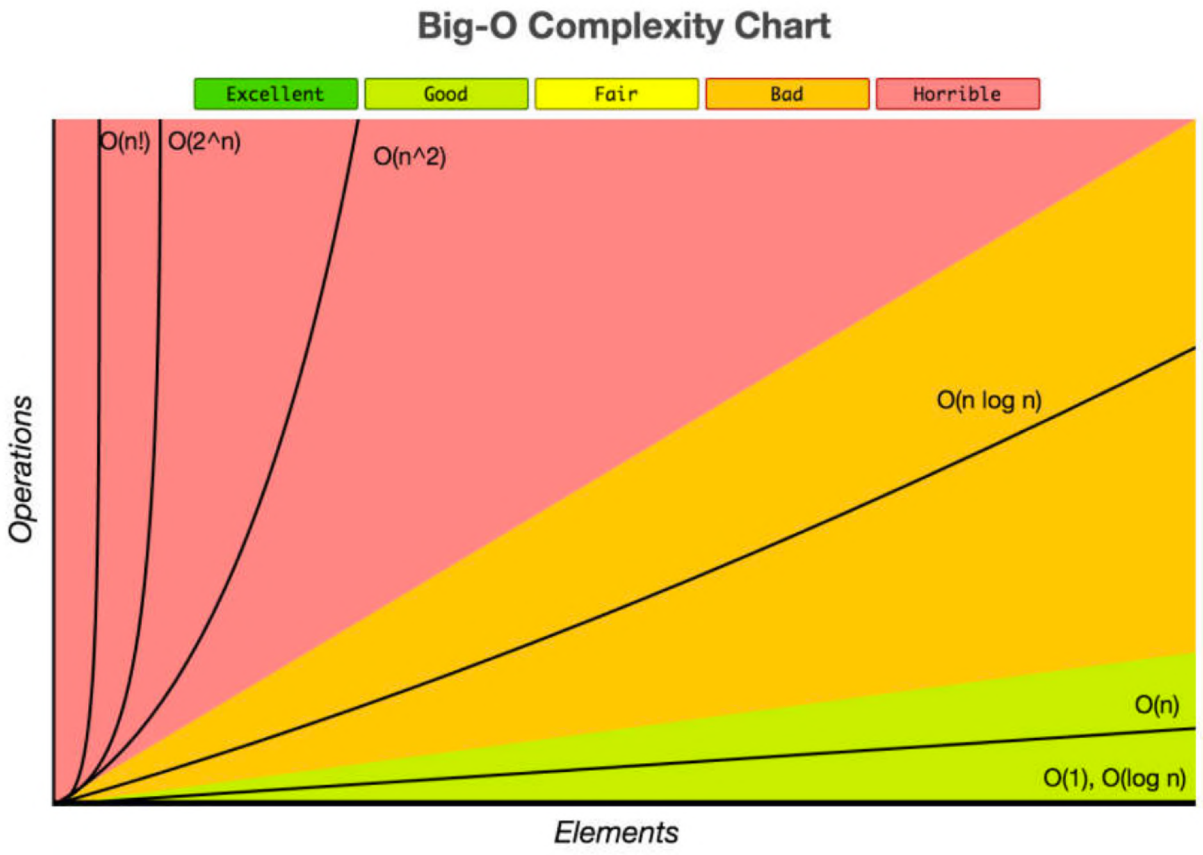


Image source: http://bigocheatsheet.com/img/big-o-complexity-chart.png

Big O

Example 2

We show that $n^2 + 10n \in O(n^2)$. Because, for $n \geq 1$,

$$n^2 + 10n \leq n^2 + 10n^2 = 11n^2,$$

we can take $c = 11$ and $N = 1$ to obtain our result.

- To show a function is in big O of another function, the key is to find a specific value of c and N that make the inequality hold.
- More examples of functions in $O(n^2)$:
  - n^2, $n^2 + n$, $n^2 + 1000n$, $1000n^2 + 1000n$, n, $n/1000$, $n^{1.99999}$, $n^2/\lg\lg\lg n$.

Big O

Example 3

Is $2^{2n} \in O(2^n)$?

Assume there exist constants $c > 0$ and $N \geq 0$, such that

$$2^{2n} \leq c2^n,$$

for all $n \geq N$. Then

$$2^{2n} = 2^n 2^n \leq c2^n,$$
$$2^n \leq c.$$

But we can't find any constant c is greater than 2^n for all $n \geq N$. So the assumption leads to a contradiction.

Then we can certify that $2^{2n} \notin O(2^n)$.

Big Ω

Definition

For a given complexity function $f(n)$, $\Omega(f(n))$ is the set of complexity functions $g(n)$ for which there exists some positive real constant c and some nonnegative integer N such that, for all $n \geq N$,

$$g(n) \geq cf(n).$$

- $\Omega\big(f(n)\big)$ is the opposite of $O\big(f(n)\big)$.
- If $g(n) \in \Omega\big(f(n)\big)$, we say that $g(n)$ is "big Ω" of $f(n)$.
- Big Ω puts an asymptotic lower bound on a function.

Formal Definition of Big Θ

Definition

For a given complexity function $f(n)$,

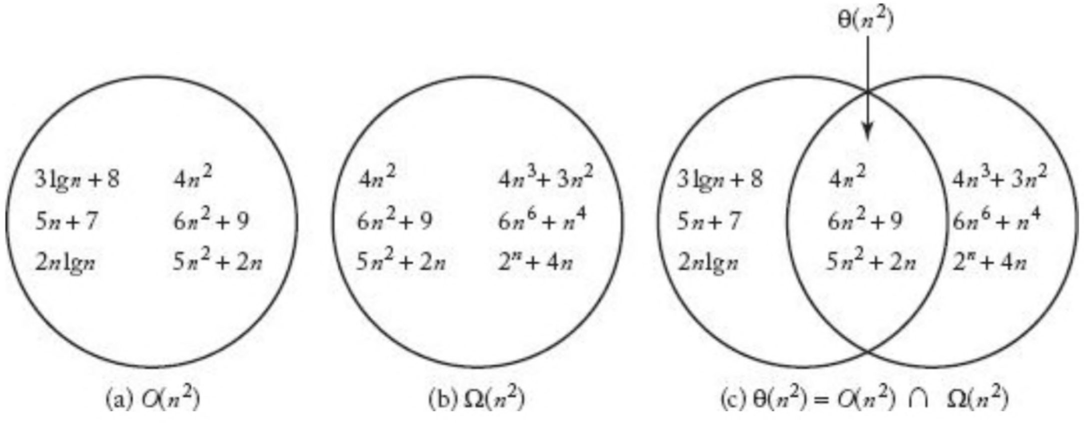
$$\Theta(f(n)) = O(f(n)) \cap \Omega(f(n)).$$

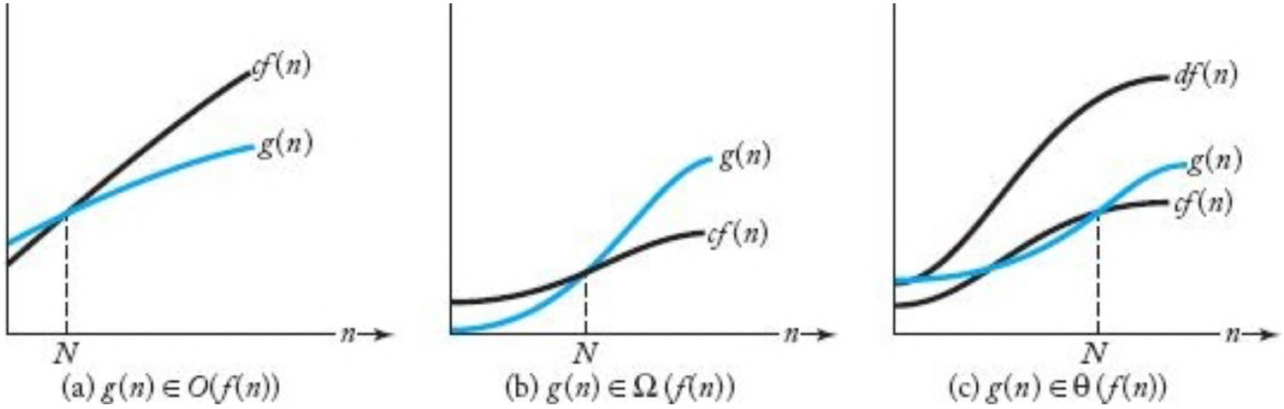
This means that $\Theta(f(n))$ is the set of complexity functions $g(n)$ for which there exists some positive real constants c and d and some nonnegative integer N such that, for all $n \geq N$,

$$cf(n) \leq g(n) \leq df(n).$$

- If $g(n) \in \Theta(f(n))$, we say that $g(n)$ is "big Θ" or simply order of $f(n)$.

Relation between Big O, Big Ω and Big Θ





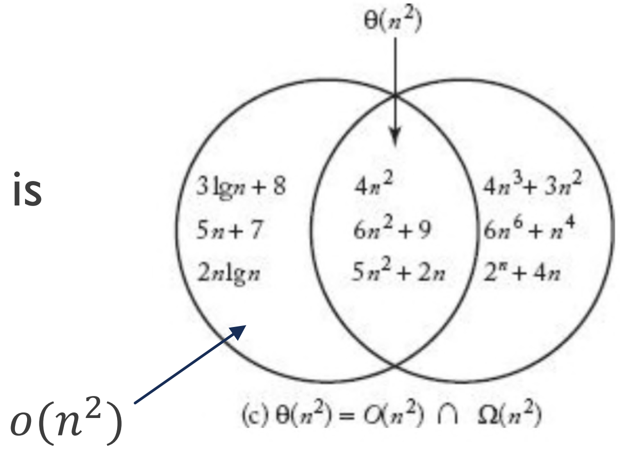
Small o

Definition

For a given complexity function $f(n)$, $o(f(n))$ is the set of all complexity functions $g(n)$ satisfying the following: For every positive real constant c there exists a nonnegative integer N such that, for all $n \geq N$,

$$g(n) \leq cf(n).$$

- If $g(n) \in o\big(f(n)\big)$, we say that $g(n)$ is "small o" of $f(n)$.
- Recall that big O requires "some c" but small o requires "every c". Small o is more strict.
- If $g(n) \in o\big(f(n)\big)$, $g(n) \in O\big(f(n)\big) - \Omega\big(f(n)\big)$.



(c) $\Theta(n^2) = O(n^2) \cap \Omega(n^2)$

Small o

Example 4

Show that $n \in o(n^2)$.

We need to find an N for every c such that, for $n \geq N$,

$$n \leq cn^2.$$

If we divide both sides of this inequality by cn, we get

$$\frac{1}{c} \leq n.$$

Therefore, for every c, it suffices to choose any $N \geq \frac{1}{c}$.

Small o

Example 5

Show that $n \notin o(5n)$.

We will use proof by contradiction to show this. We select a value of c which makes the inequality unsatisfied.

$$n \leq \frac{1}{6} 5n = \frac{5}{6} n.$$

Let $c = 1/6$. If $n \in o(5n)$, then there must exist some N such that, for $n \geq N$,

This contradiction proves that $n \notin o(5n)$.

Small ω

Definition

For a given complexity function $f(n)$, $\omega(f(n))$ is the set of all complexity functions $g(n)$ satisfying the following: For every positive real constant c there exists a nonnegative integer N such that, for all $n \geq N$,

$$g(n) \geq cf(n).$$

- If $g(n) \in \omega\big(f(n)\big)$, we say that $g(n)$ is "small ω" of $f(n)$.
- Now we have O, o, Θ, Ω, and ω. Intuitively, they just like "$\leq$", "$<$", "$=$", "$\geq$", and "$>$" for complexity functions.

Properties of Orders

- Transitivity
  - If $g(n) \in \Theta(f(n))$ and $f(n) \in \Theta(h(n))$ then $g(n) \in \Theta(h(n))$.
  - Same for O, o, Ω, and ω.
- Additivity
  - If $g(n) \in \Theta(h(n))$ and $f(n) \in \Theta(h(n))$ then $g(n) + f(n) \in \Theta(h(n))$.
  - Same for O, o, Ω, and ω.
- Reflexivity
  - If $g(n) \in \Theta(g(n))$.
  - Same for O and Ω.
- Symmetry
  - $g(n) \in \Theta(f(n))$ if and only if $f(n) \in \Theta(g(n))$.
- Transpose Symmetry
  - $g(n) \in O(f(n))$ if and only if $f(n) \in \Omega(g(n))$.
  - $g(n) \in o(f(n))$ if and only if $f(n) \in \omega(g(n))$.

Properties of Orders

- $g(n) \in O(f(n))$ and $g(n) \in \Omega(f(n))$ if and only if $g(n) \in \Theta(f(n))$.
- Consider the following ordering of complexity categories:

$$\Theta(\lg n) \quad \Theta(n) \quad \Theta(n \lg n) \quad \Theta(\mathrm{n}^2) \quad \Theta(n^j) \quad \Theta(n^k) \quad \Theta(a^n) \quad \Theta(b^n) \quad \Theta(n!)$$

where $k > j > 2$ and $b > a > 1$. If $g(n)$ is to the left of $f(n)$, then

$$g(n) \in o(f(n))$$

Notice: Big Θ is a set of functions. We can't say $\Theta(\lg n) < \Theta(n)$.

Properties of Orders

Example 6

Given $g(n) = \frac{1}{2}n(n-1)$, prove that $g(n) \in \Theta(n^2)$

Proof:

By the property, we first show that $g(n) \in O(n^2)$:

$$\frac{1}{2}n(n-1) = \frac{1}{2}n^2 - \frac{1}{2}n \le \frac{1}{2}n^2 \text{ (for } c = \frac{1}{2} \text{ and } N = 0\text{).}$$

Then we show that $g(n) \in \Omega(n^2)$:

$$\frac{1}{2}n(n-1) = \frac{1}{2}n^2 - \frac{1}{2}n \ge \frac{1}{2}n^2 - \frac{1}{2}n\frac{1}{2}n = \frac{1}{4}n^2 \text{ (for } c = \frac{1}{4} \text{ and } N = 2\text{).}$$

Thus $g(n) \in \Theta(n^2)$.

Properties of Orders

Example 7

Given $g(n) = (n+a)^b$, prove that $g(n) \in \Theta(n^b)$, for any real constants a and b, where $b > 0$.

Proof:

By the property, we first show that $g(n) \in O(n^b)$:

$$(n+a)^b \leq (n+|a|)^b \leq (2n)^b = 2^b n^b \text{ (for } c = 2^b, N = |a|).$$

Then we show that $g(n) \in \Omega(n^b)$:

$$(n+a)^b \geq (n-|a|)^b \geq \left(n - \frac{n}{2}\right)^b = \left(\frac{n}{2}\right)^b = \left(\frac{1}{2}\right)^b n^b \text{ (for } c = \left(\frac{1}{2}\right)^b, N = 2|a|).$$

Thus $g(n) \in \Theta(n^b)$.

Using a Limit to Determine Order

$$\lim_{n\to\infty} \frac{g(n)}{f(n)} = \begin{cases} c & \text{implies } g(n) \in \Theta(f(n)) \quad \text{if } c > 0 \\ 0 & \text{implies } g(n) \in o(f(n)) \\ \infty & \text{implies } g(n) \in \omega(f(n)) \end{cases}$$

Using a Limit to Determine Order

Example 8

Compare the orders of growth of $\frac{1}{2}n(n-1)$ and n^2.

$$\lim_{n->\infty}\frac{\frac{1}{2}n(n-1)}{n^2}=\frac{1}{2}\lim_{n->\infty}\frac{n^2-n}{n^2}=\frac{1}{2}\lim_{n->\infty}(1-\frac{1}{n})=\frac{1}{2},$$

Thus, $\frac{1}{2}n(n-1)=\Theta(n^2)$.

Using a Limit to Determine Order

Example 9

For $b > a > 0$,

$$a^n \in o(b^n)$$

because

$$\lim_{n->\infty} \frac{a^n}{b^n} = \lim_{n->\infty} \left(\frac{a}{b}\right)^n = 0.$$

The limit is 0 because $0 < \frac{a}{b} < 1$.

Using a Limit to Determine Order

Theorem

L'Hôpital's Rule If $f(x)$ and $g(x)$ are both differentiable with derivatives $f'(x)$ and $g'(x)$, respectively, and if

$$\lim_{x->\infty} f(x) = \lim_{x->\infty} g(x) = \infty,$$

then

$$\lim_{x->\infty} \frac{f(x)}{g(x)} = \lim_{x->\infty} \frac{f'(x)}{g'(x)},$$

whenever the limit on the right exists.

Using a Limit to Determine Order

Example 10

$$\lg n \in o(n)$$

$$\frac{d(\log_a x)}{dx} = \frac{1}{x \ln a}$$

because

$$\lim_{x->\infty} \frac{\lg x}{x} = \lim_{x->\infty} \frac{d(\lg x)/dx}{dx/dx} = \lim_{x->\infty} \frac{1/(x \ln 2)}{1} = 0.$$

Exercises

- Show the correctness of the following statements.
  - $\lg n \in O(n)$
  - $n \in O(n \lg n)$
  - $n \lg n \in O(n^2)$
  - $2^n \in \Omega(5^{\ln n})$
  - $\lg^3 n \in o(n^{0.5})$

Thank you!

- Any question?
- Don’t hesitate to send email to me for asking questions and discussion. ☺

Acknowledgement: Thankfully acknowledge slide contents shared by Prof. Xuemin Hong and Prof. Yiu-ming Cheung