Design and Analysis of Algorithms

Lecture 1

Etibar Vazirov
French - Azerbaijani University
Mathematics & CS Department
CS instructor
Contact mail:
vazirov@unistra.fr
etibar.vazirov@ufaz.az

What will you learn from this course?

The **objective** of the course is to learn practices for efficient problem solving in computing.

- Problem Solving
  - How to write a step by step procedure (an algorithm) to solve a given problem
  - What are the various paradigms of problem solving in computing
  - When to choose which alternate strategy of problem solving
- Efficiency
  - How to evaluate the worst case behavior of an algorithm
  - What are the various mathematical tools of algorithm analysis
  - How to decide which algorithm is better
- Dealing with hard problems
  - How to figure out if a given problem is easy or hard
  - What to do if we are given a hard problem

Course Overview

Asymptotic Notations

- Big O, Big Omega and Big Theta
- Problems on Big O
- Algorithmic Complexity with Asymptotic Notations

Course Overview

Recursion

- Linear Search, Greatest Common Divisor
- Factorial, Tail Recursion
- Recurrence Relations, Substitution Method
- Towers of Hanoi

Course Overview

Divide and Conquer

- Binary Search
- Master Method
- Tiling a Defective Chessboard
- Merge Sort
- Quick Sort

Course Overview

Dynamic Programing

- Fibonacci Numbers
- Rod Cutting
- Matrix Chain Multiplication
- Longest Common Subsequence

Course Overview

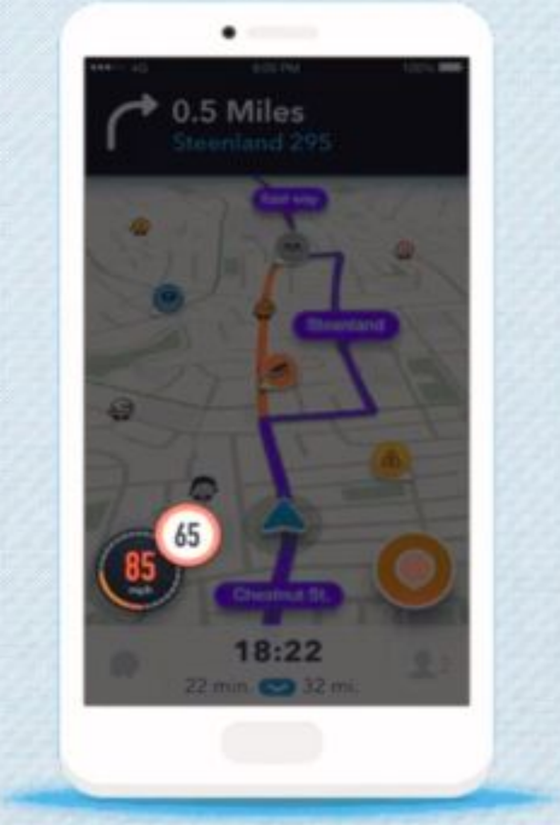
Greedy Algorithms

- Knapsack Problem
- Minimum Spanning Tree: Kruskal's Algorithm
- Disjoint Sets
- Job Sequencing with Deadlines
- Heap
- Heap Sort
- Priority Queue
- Minimum Spanning Tree: Prim's Algorithm
- Huffman's Codes

Course Overview

Shortest Path Algorithms

- Dijkstra's Algorithm
- Bellman Ford Algorithm
- Topological Sort
- Shortest Path by Topological Sort
- Floyd Warshall Algorithm



Course Overview

String Matching

- Brute Force Matcher
- String Matching with Finite Automaton
- Pattern Pre-Processing
- The Knuth Morris Pratt Algorithm

Course Overview

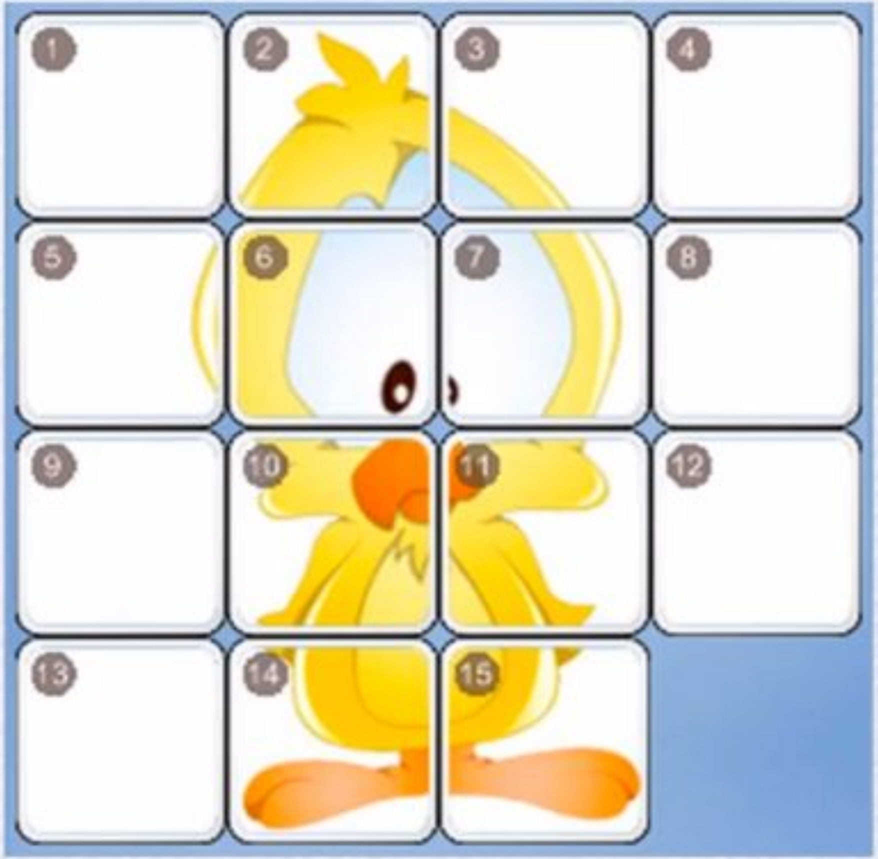
Backtracking

- Rat in Maze
- n-Queens Algorithm
- Graph Coloring
- Hamiltonian Cycles
- Subset Sum

Course Overview

Branch & Bound

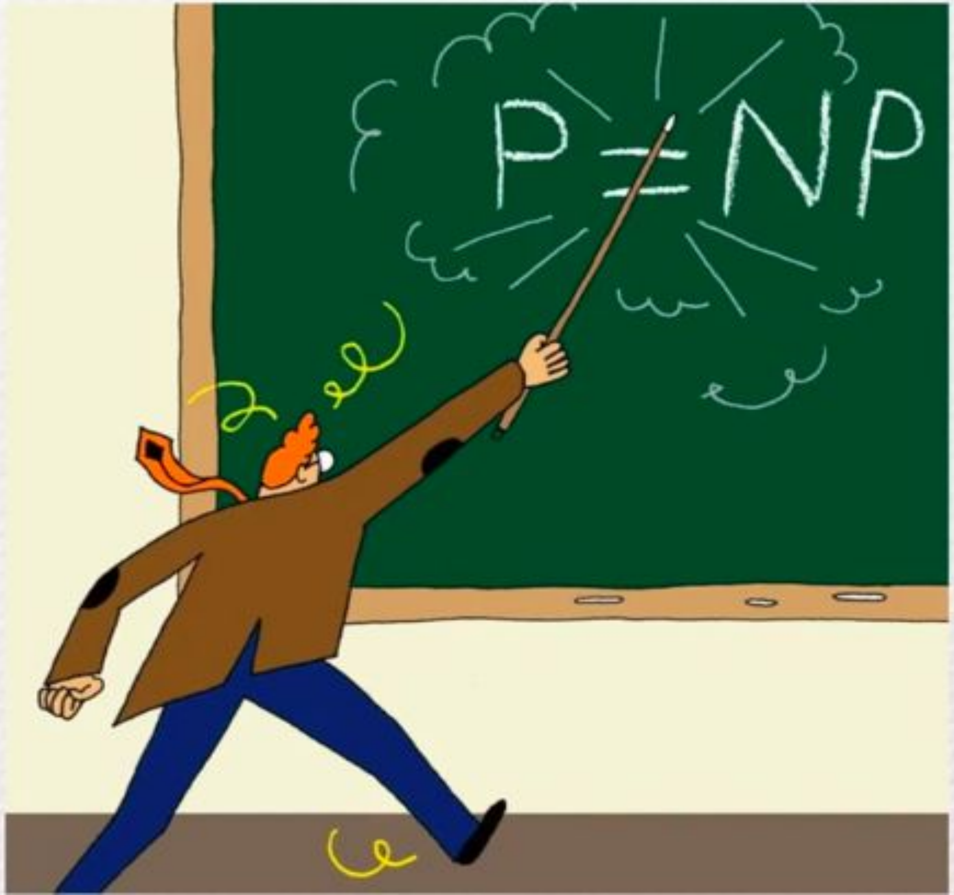
- Introduction to Branch and Bound
- 0/1 Knapsack Problem
- The 15 Puzzle Problem
- Solvability of 15 Puzzles



Course Overview

NP Completeness

Approximation Algorithms



If debugging is the process of removing bugs then programming must be the process of putting them in

- Edsger Dijkstra.

Thank You!!!

Introduction to Algorithms

Why should we know about algorithms?

Algorithms are everywhere!!!

Today various algorithms shape how we

find our way

connect with people

eat what we love

The list goes on...

buy what we need

get married

What is an Algorithm?

An **algorithm** is a

finite sequence

of

unambiguous instructions

for

solving a problem

i.e.

to obtain the required output

for a legitimate input

in a finite amount of time.

Algorithm vs. Program

An **algorithm** takes the input to a problem (function) and transforms it to the output.

— A mapping of input to output.

A computer **program** is an instance, or concrete representation, of an algorithm in some programming language.

One problem can have many algorithms.

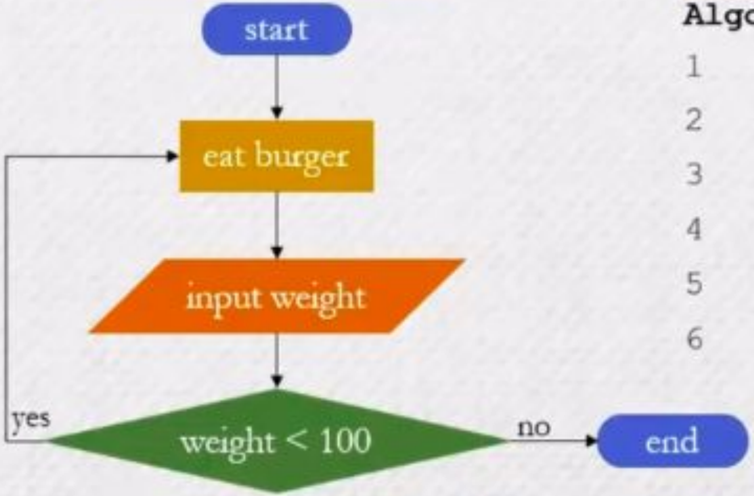
An Algorithm may be written in many ways

Jimmy is wondering whether to eat another burger.
Hare is an algorithm expressed in three different ways
Which helps him to make his decision.

in English

Step1: start
Step2: eat the next burger
Step3: check weight
Step4: if weight is less than 100 Kg
then go to Step2
Step5: end

using Flow Charts



by Pseudo Code

```
Algorithm eatBurger
    while ( NOT burgers.empty() )
        food = burgers.next()
        eat(food)
        input weight
        if (weight ≥ 100)
            break
```

Overview of the Course

Tools for Analysis

- Asymptotic Notations
- Recurrence Relations

Tools for Design

- Divide and Conquer
- Dynamic Programming
- Greedy Algorithms
- Backtracking
- Branch and Bound

Understanding the limitations

- Theory of NP Completeness

If debugging is the process of removing bugs then programming must be the process of putting them in

- Edsger Dijkstra.

Thank You!!!

Asymptotic Notations 1

Efficiency of an Algorithm

How to Measure Efficiency?

Empirical comparison (run programs)

- Empirical comparison is difficult to do **"fairly"** and is time consuming.

Asymptotic Efficiency

- Asymptotic Notations tell us about relative growth of functions and are used to measure the efficiency of an algorithm.

- Parameterize the running time by the size of the input.

  - $T(n)$ = time required by an algorithm on any input of size n.

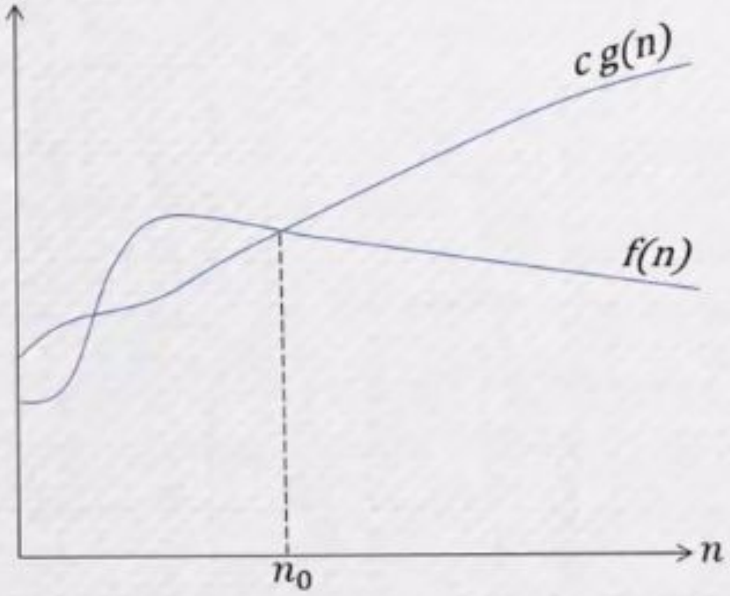
- The key measures of computational complexity are:

  - Big O [O] notation,
  - Big Theta [Θ] notation and
  - Big Omega [Ω] notation.

Big O

Let $f(n)$ and $g(n)$ be two functions of n, and c and n_0 be positive constants. Then

$$\boxed{f(n) \text{ is } O\big(g(n)\big) \text{ iff } \exists\ c, n_0 > 0 \mid f(n) \le cg(n), \forall n \ge n_0}$$



e.g. ❶ **$f(n) = 5n + 3$**

Then, $f(n) \le 5n + n,\ \forall\ n \ge 3$

i.e. $f(n) \le 6n, \forall\ n \ge 3$

$\therefore$ **f(n) is O(n)** [here $c = 6$, $n_0 = 3$]

e.g. ❷ **$f(n) = 3n^2 + 2n$**

Then, $f(n) \le 3n^2 + n^2,\ \forall\ n \ge 2$

i.e. $f(n) \le 4n^2, \forall\ n \ge 2$

$\therefore$ **f(n) is O(n^2)** [here $c = 4$, $n_0 = 2$]

Implication of Big O Notation

$f(n)$ is $O(g(n))$ means that rate of growth of $f(n)$ is slower than or equal to $g(n)$

$f(n) = 5n + 3$
means $f(n)$ is $O(n)$

Big O of a function does not depend on the constant terms

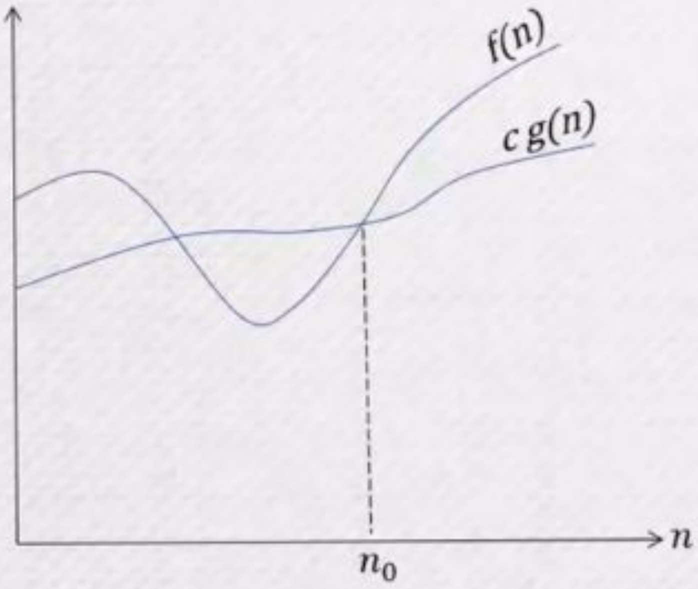
$f(n) = 3n^2 + 2n$
means $f(n)$ is $O(n^2)$

Big O of a function with multiple terms depends on the fastest growing term

Big Ω

Let $f(n)$ and $g(n)$ be two functions of n, and c and n_0 be positive constants. Then

$$\boxed{f(n) \text{ is } \Omega\big(g(n)\big) \text{ iff } \exists\ c, n_0 > 0 \mid f(n) \geq cg(n), \forall n \geq n_0}$$



e.g. ❶ $f(n) = 5n + 3$

Then, $f(n) \geq 5n,\ \forall\ n \geq 1$

$\therefore$ **$f(n)$ is $\Omega(n)$** [here $c = 5$, $n_0 = 1$]

e.g. ❷ $f(n) = 3n^2 + 2n$

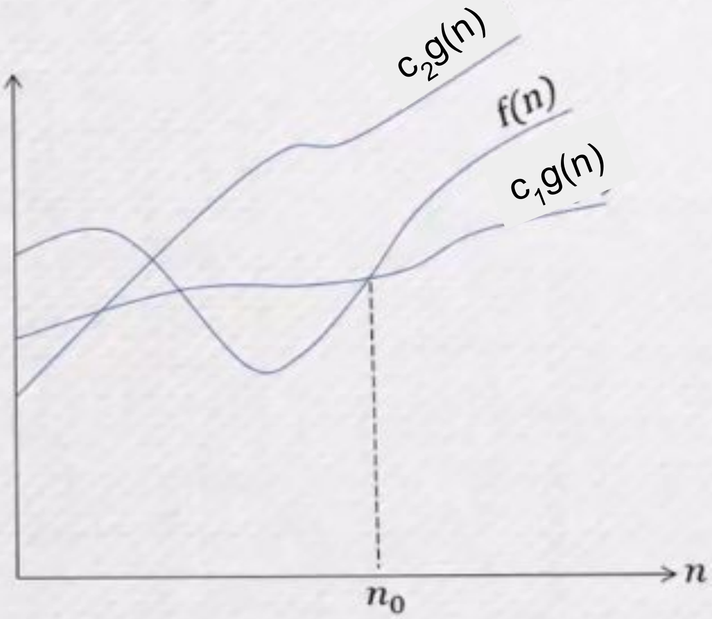
Then, $f(n) \geq 3n^2,\ \forall\ n \geq 1$

$\therefore$ **$f(n)$ is $\Omega(n^2)$** [here $c = 3$, $n_0 = 1$]

Big θ

Let $f(n)$ and $g(n)$ be two functions of n, and c_1, c_2 and n_0 be positive constants. Then

$$\boxed{f(n) \text{ is } \theta(g(n)) \text{ iff } \exists\, c_1, c_2, n_0 > 0 \mid c_1 g(n) \leq f(n) \leq c_2 g(n), \forall n \geq n_0}$$



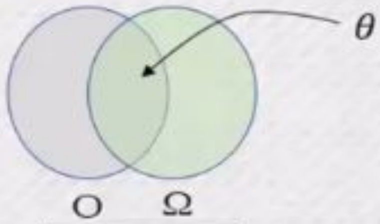
e.g. ❶ $f(n) = 5n + 3$

Then, $f(n) \geq 5n, \ \forall\, n \geq 1$

Again, $f(n) \leq 5n + n, \ \forall\, n \geq 3$

i.e. $f(n) \leq 6n, \forall\, n \geq 3$

$\therefore$ **$f(n)$ is $\theta(n)$** [here $c_1 = 5, c_2 = 6, n_0 = 3$]



$$\boxed{\theta = O \cap \Omega}$$

Asymptotically Tight vs. Loose

O and Ω give us asymptotically tight bounds.

*small o and ω give us **asymptotically loose** bounds.*

Let $f(n)$ and $g(n)$ be two functions of n, and c and n_0 be positive constants. Then

$f(n)$ *is* $o(g(n))$ iff $\exists$ c, $n_0 > 0 \mid \mathrm{f(n)} < cg(n), \forall n \geq n_0$
$f(n)$ *is* $\omega(g(n))$ iff $\exists$ c, $n_0 > 0 \mid f(n) > cg(n), \forall n \geq n_0$

If debugging is the process of removing bugs then programming must be the process of putting them in

- Edsger Dijkstra.

Thank You!!!

Asymptotic Notations 2

Some interesting facts about Big O

1 Any constant is O(1)

Let $f(n) = c$, where c is a constant
Let $d \geq c$ be another constant
Then $f(n) \leq d.1 \quad \forall\, n \geq 1$
$\therefore f(n)$ is $O(1)$

2 Any function is O of itself

Let $f(n)$ be a function of n
Let $c \geq 1$ be a constant
Then $f(n) \leq c.f(n) \quad \forall\, n \geq 1$
$\therefore f(n)$ is $O(f(n))$

3 O(1) is same as O(2)

Let $f(n)$ be $O(1)$
Then $f(n) \leq c.1 \quad \forall\, n \geq 1$
Where c is a constant
Let $d = c/2$ be another constant
So $f(n) \leq c \quad \forall\, n \geq 1$
$\Rightarrow f(n) \leq c/2 * 2 \quad \forall\, n \geq 1$
$\Rightarrow f(n) \leq d * 2 \quad \forall\, n \geq 1$

Hence
$f(n)$ is $O(2)$

4 Writing $f(n) = O(g(n))$ is NOT correct

Since $n \leq 1.n^2 \quad \forall\, n \geq 1$
$\Rightarrow n$ is $O(n^2) \quad \ldots$ (i)
By **2** n^2 is $O(n^2) \quad \ldots$ (ii)

If we write $n = O(n^2)$ and $n^2 = O(n^2)$
Then that will imply $n = n^2$
Which is mathematically incorrect

Some interesting facts about Big O

5 <u>Big O is Transitive</u>

i.e. if $f_1(n)$ is $O(f_2(n))$ and $f_2(n)$ is $O(f_3(n)$ then $f_1(n)$ is $O(f_3(n))$

$f_1(n)\ is\ O(f_2(n))$

$\Rightarrow \frac{f_1(n)}{f_2(n)} \leq c \qquad \forall\, n \geq n_1$

$f_2(n)\ is\ O(f_3(n))$

$\Rightarrow \frac{f_2(n)}{f_3(n)} \leq d \qquad \forall\, n \geq n_2$

So,

$\frac{f_1(n)}{f_3(n)} = \frac{f_1(n)}{f_2(n)} * \frac{f_2(n)}{f_3(n)} \leq c * d \qquad \forall\, n \geq \max(n_1, n_2)$

$\Rightarrow f_1(n) \leq c * d * f_3(n) \qquad \forall\, n \geq \max(n_1, n_2)$

$\therefore$ $f_1(n)$ is $O(f_3(n))$

Some interesting facts about Big O

6 Big O is Multiplicative

i.e. if $f_1(n)$ is $O(g(n))$ and $f_2(n)$ is $O(h(n))$ then $f_1(n) * f_2(n)$ is $O(g(n) * h(n))$

Some interesting facts about Big O

❻ Big O is Multiplicative

i.e. if $f_1(n)$ is $O(g(n))$ and $f_2(n)$ is $O(h(n))$ then $f_1(n) * f_2(n)$ is $O(g(n) * h(n))$

$$f_1(n) \text{ is } O(g(n))$$
$$\Rightarrow \frac{f_1(n)}{g(n)} \le c \qquad \forall n \ge n_1$$

$$f_2(n) \text{ is } O(h(n))$$
$$\Rightarrow \frac{f_2(n)}{h(n)} \le d \qquad \forall n \ge n_2$$

$$\text{So, } \frac{f_1(n)}{g(n)} * \frac{f_2(n)}{h(n)} \le c * d \qquad \forall n \ge \max(n_1, n_2)$$
$$\Rightarrow f_1(n) * f_2(n) \le c * d * (g(n) * h(n)) \qquad \forall n \ge \max(n_1, n_2)$$

$\therefore$ $f_1(n) * f_2(n)$ is $O(g(n) * h(n))$

Some interesting facts about Big O

❼ All logs grow at the same rate

i.e. $\log_a n$ is $O(\log_b n)$ and $\log_b n$ is $O(\log_a n)$

$\log_a n = \log_a b * \log_b n$
$\Rightarrow \log_a n = c * \log_b n$
Where $c = \log_a b$ is a constant
Let $c \le d$ be another constant
Then $\log_a n \le d * \log_b n \quad \forall n \ge 1$
$\therefore \log_a n$ is $O(\log_b n)$

Similarly it can be shown that $\log_b n$ is $O(\log_a n)$

Some interesting facts about Big O

8 If f(n) is a polynomial, then f(n) is O of the highest power of n in f(n)

Let $f(n) = a_m n^m + a_{m-1} n^{m-1} + \cdots + a_1 n + a_0 \, , m \geq 1$

Some interesting facts about Big O

8 If f(n) is a polynomial, then f(n) is O of the highest power of n in f(n)

Let $f(n) = a_m n^m + a_{m-1} n^{m-1} + \cdots + a_1 n + a_0 \,, m \geq 1$

Then $f(n) \leq a_m n^m + a_{m-1} n^m + \cdots + a_1 n^m + a_0 n^m, \; \forall\, \mathrm{n} \geq 1$

$\Rightarrow f(n) \leq \underline{(a_m + a_{m-1} + \cdots + a_1 + a_0)} n^m, \; \forall\, \mathrm{n} \geq 1$

$\Rightarrow f(n) \leq c * n^m, \; \forall\, \mathrm{n} \geq 1$

Where $c = (a_m + a_{m-1} + \cdots + a_1 + a_0)$ *is a constant*

$\therefore f(n)$ is $O(n^m)$

9 1 is $O(\log_2 n)$ is $O(n)$ is $O(n \log_2 n)$ is $O(n^2)$ is $O(2^n)$

Some interesting facts about Big O

8 If f(n) is a polynomial, then f(n) is O of the highest power of n in f(n)

Let $f(n) = a_m n^m + a_{m-1} n^{m-1} + \cdots + a_1 n + a_0\,, m \geq 1$

Then $f(n) \leq a_m n^m + a_{m-1} n^m + \cdots + a_1 n^m + a_0 n^m,\ \forall\, \mathrm{n} \geq 1$

$\Rightarrow f(n) \leq \underline{(a_m + a_{m-1} + \cdots + a_1 + a_0)} n^m,\ \forall\, \mathrm{n} \geq 1$

$\Rightarrow f(n) \leq c * n^m,\ \forall\, \mathrm{n} \geq 1$

Where $c = (a_m + a_{m-1} + \cdots + a_1 + a_0)$ *is a constant*

$\therefore f(n)$ is $O(n^m)$

9 1 is $O(\log_2 n)$ is $O(n)$ is $O(n \log_2 n)$ is $O(n^2)$ is $O(2^n)$

$1 \leq 2 \leq 4 \leq 8 \leq 16 \leq 16$

$\leq \log_2 n \leq n \leq n \log_2 n \leq n^2 \leq 2^n \qquad \forall\, n \geq 4$

is $O(\log_2 n)$ is $O(n)$ is $O(n \log_2 n)$ is $O(n^2)$ is $O(2^n)$

Some interesting facts about Big O

10. $n!$ is $O(n^n)$

Some interesting facts about Big O

⑩ $n!\ is\ O(n^n)$

$$n! = n * (n-1) * (n-2) * \cdots * 2 * 1$$

$$\Rightarrow n! \le n * n * n * \cdots * n * n, \qquad \forall\, \mathrm{n} \ge 1$$

$$\Rightarrow n! \le n^n, \qquad \forall\, \mathrm{n} \ge 1$$

$$\therefore n!\ \text{is}\ O(n^n)$$

⑪ $\log_2 n!$ is $O(n \log_2 n)$

Some interesting facts about Big O

10 $n!\ is\ O(n^n)$

$$n! = n * (n-1) * (n-2) * \cdots * 2 * 1$$

$$\Rightarrow n! \le n * n * n * \cdots * n * n, \qquad \forall\, n \ge 1$$

$$\Rightarrow n! \le n^n, \qquad \forall\, n \ge 1$$

$$\therefore n! \text{ is } O(n^n)$$

11 $\log_2 n! \text{ is } O(n \log_2 n)$

$$By\ \textbf{10}\ n! \le n^n \qquad \forall\, n \ge 1$$

$$\text{so } \log_2 n! \le n \log_2 n \qquad \forall\, n \ge 1$$

$$\therefore \log_2 n! \text{ is } O(n \log_2 n)$$

If debugging is the process of removing bugs then programming must be the process of putting them in

- Edsger Dijkstra.

Thank You!!!

Asymptotic Notations 3

Algorithmic Complexity with Asymptotic Notation

Analyzing an Algorithm Asymptotically

- The running time of an algorithm T(n) on input of size n, is the number of times the instructions in the algorithm are executed.

```
Algorithm findMean
1   input n
2   sum = 0
3   i = 0
4   while (i < n)
5       input number
6       sum = sum + number
7       i = i + 1
8   mean = sum / n
```

Statement	*Number of times executed*
1-3	1
4	n+1
5 -7	n
8	1

- The computing time for this algorithm in terms on input size n is: $T(n) = 4n + 5$.
- Hence the Computational Complexity is $O(n)$ and $\Omega(n)$, i.e. $\Theta(n)$

Asymptotic Upper and Lower Bounds

- Big O [$f(n) \leq cg(n)$] defines an *asymptotic upper* bound i.e. the *worst case* growth of our algorithm

- Big Ω [$f(n) \geq cg(n)$] defines an *asymptotic lower* bound i.e. the *best case* growth of our algorithm

- Big Θ [$c_1 g(n) \leq f(n) \leq c_2 g(n)$] defines both an *asymptotic lower* and an *asymptotic upper* bound i.e. the *best and worst case* growth of our algorithm

Examples

1 Sequence of statements

```
b = 1          // constant cost
b = b + 3      // constant cost
a = b          // constant cost
```

O(1)
Ω(1)
Θ(1)

2 A single loop

```
for (j = 1 to n)
{
      print "j = ", j
}
```

O(n)
Ω(n)
Θ(n)

Examples

3 Two Nested Loops

```
sum = 0
for (i = 1 to n)
    for (j = 1 to n)
        sum++;
```

Examples

③ Two Nested Loops

```
sum = 0
for (i = 1 to n)
    for (j = 1 to n)
        sum++;
```

$O(n^2)$
$\Omega(n^2)$
$\Theta(n^2)$

Examples

3 Two Nested Loops

```
sum = 0
for (i = 1 to n)
    for (j = 1 to n)
        sum++;
```

$O(n^2)$
$\Omega(n^2)$
$\Theta(n^2)$

4 Three Nested Loops

```
for (i = 1 to n)
    for (j = 1 to n)
        for (k = 1 to n)
            print "Algorithms rock!!!"
```

Examples

3 Two Nested Loops

```
sum = 0
for (i = 1 to n)
    for (j = 1 to n)
        sum++;
```

$O(n^2)$
$\Omega(n^2)$
$\Theta(n^2)$

4 Three Nested Loops

```
for (i = 1 to n)
    for (j = 1 to n)
        for (k = 1 to n)
            print "Algorithms rock!!!"
```

$O(n^3)$
$\Omega(n^3)$
$\Theta(n^3)$

Examples

5 Loops in Sequence

```
for (i = 1 to n)
        print "i = ", i
for (j = 1 to n)
         print "j = ", j
```

Examples

5 Loops in Sequence

```
n { for (i = 1 to n)
          print "i = ", i
n { for (j = 1 to n)
           print "j = ", j
```

O(n)
Ω(n)
Θ(n)

6 Nested and Single Loop in Sequence

```
for (i = 1 to n)
      for (j = 1 to n)
            print "Working together is cool"
for (k = 1 to n)
      print "I prefer to go solo"
```

Examples

5 Loops in Sequence

```
for (i = 1 to n)
        print "i = ", i
for (j = 1 to n)
        print "j = ", j
```

$O(n)$
$\Omega(n)$
$\Theta(n)$

6 Nested and Single Loop in Sequence

```
for (i = 1 to n)
        for (j = 1 to n)
                print "Working together is cool"
for (k = 1 to n)
        print "I prefer to go solo"
```

$O(n^2)$
$\Omega(n^2)$
$\Theta(n^2)$

Examples

7 Another single loop

```
for (j = 1 to n * n)
{
        print "j = ", j
}
```

Examples

7 Another single loop

```
for (j = 1 to n * n)
{
        print "j = ", j
}
```

$O(n^2)$
$\Omega(n^2)$
$\Theta(n^2)$

8 One more loop

```
n = 100000
for (j = 1 to n)
{
        print "looks are deceptive"
}
```

Examples

7 Another single loop

```
for (j = 1 to n * n)
{
        print "j = ", j
}
```

$O(n^2)$
$\Omega(n^2)$
$\Theta(n^2)$

8 One more loop

```
n = 100000
for (j = 1 to n)
{
        print "looks are deceptive"
}
```

$O(1)$
$\Omega(1)$
$\Theta(1)$

Examples

9 Loop with break

```
for (j = 1 to n)
{
        if (sum > 0)
                break
}
```

Examples

9 Loop with break

```
for (j = 1 to n)
{
        if (sum > 0)
                break
}
```

O(n)

Examples

9 Loop with break

```
for (j = 1 to n)
{
        if (sum > 0)
                break
}
```

O(n)
Ω(1)

If debugging is the process of removing bugs then programming must be the process of putting them in

- Edsger Dijkstra.

Thank You!!!

Next week...

Recursion