Introduction:

Course Logistics and Complexity Review

Fatemeh Panahi
Department of Computer Science
University of Toronto
CSC263-Fall 2017
Lecture 1

Today

- About the Course
  - Why CSC263?
  - What CSC263 is about?
  - How to do well in CSC263?
- Logistics
- Review
  - Asymptotic notations (Lower bound, Upper bound, tight bound)
  - Algorithm complexity
    - Worst case, Best case, Average case

Why CSC263?

- ~~To graduate.~~
- To be a good programmer.
- To land a job.
- To develop excellent software in your start-up.

Why CSC263?

- The areas that data structures are applied extensively:

Compiler Design	Operating System	Database Management System
Graphics	Simulation	Artificial Intelligence

Background (Required)

- Theory of computation
  - Inductions
  - Recursive functions, Master Theorem
  - Asymptotic notations.

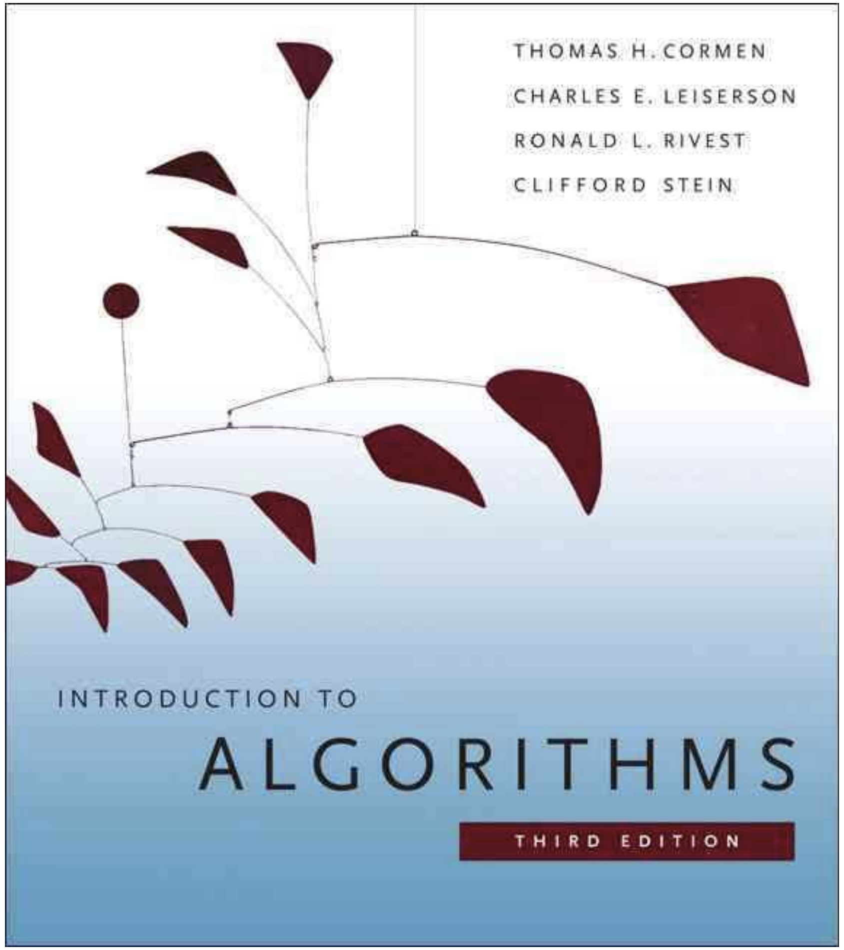
- Probability theory
  - Probabilities and counting
  - Random variables
  - Distribution
  - Expected value

Logistics -Textbook

- Introduction to Algorithms, Third Edition by Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein.

- **Recommended book:**

  Data Structures and Algorithms in C++ by Michael T. Goodrich, Roberto Tamassia, David M. Mount



Logistics – course page

- Course webpage: http://www.cs.toronto.edu/~fpanahi/2017-fall-csc263.html
- Course forum: https://piazza.com/utoronto.ca/summer2017/csc263/home
  All course materials and announcements will be posted in Piazza. We will be monitoring the course forum regularly to answer your questions.
- All assignment submissions will be done electronically, using the Markus system. Submission instruction will be provided in Piazza.

Logistics – Sections

- This course is offered in two sections.

Section	LEC0101, LEC2003	LEC0201, LEC2000, LEC2201
Lectures	Wed 12:00-14:00 (BA 1190)	Wed 15:00-17:00 (EM 001)
Tutorials	Fri 10:00-11:00 (BA 1190)	Fri 13:00-14:00 (EM 001)
Office Hours: By appointment	Mon 09:00 - 10:30 (BA 3219)	Mon 10:30 - 12:00 (BA 3219)

- The first tutorial will be on 15 Sep. Friday.
- For TAs' contact info, see Piazza.

Logistics - Course Mark Composition

- Assignments: 40%
  - 4 Assignments 10% each.
  - **Assignment 0:** nothing but making a group of two people in Markus. Deadline: Sep 20
  - **Assignment 1:** The handout will be posted on Sep 16 in Piazza.
- Midterm Test: 20%, Oct 25, 12:00 – 13:00 (There is a small possibility of change)
  If you cannot make it send an email to the instructor along with your document of proof before Sep 18.
- Final Exam: 40%, time TBA in Exam Period
- Participation: 2% bonus point

Participation

- We encourage you to participate in the lectures, tutorials and also office hours when you have questions.
  - Actively engage in lectures and tutorials.
  - Answer the questions in Piazza.
  - Win the class contests.

Small Quizzes

- There will be small quizzes at the beginning of some lectures to review the materials.
- You can win little prizes!

How to do well in CSC263?

1- Be interested! ☺

2- Solve as many exercises from the text-book as you can.

3- Give us feedback for any improvement.

Quick Survey

- Have you taken CSC263 or CSC265 before?

- Are you familiar with
  - Running time complexity of algorithms
    - Worst case, Average case, Best case
  - Asymptotic notations?
    - Big O, Big Ω, Θ

- Probability theory
  - Sample space
  - Expected value

Let’s get started!

READING Assignments: Chapters 2, 3; Sections 4.5, 5.1, 5.2.

What CSC263 is about?

- Abstract Data Type (ADT): Set of objects together with set of operations on these objects.
    - Example: Stack

      Objects: lists (or sequences)

      Operations: PUSH(S, v), POP(S), ISEMPTY(S)

- ADT's important for specification.
- provide modularity and reuse since **usage is independent of implementation**

What CSC263 is about?

- Data Structure: implementation of an ADT, a way to represent objects, and algorithms for every operation.

  Example: stack implementations.

  a) Linked list (keep pointer to head).
     ISEMPTY: test head == None.
     PUSH: insert at front of list.
     POP: remove front of list (if not empty)

  b) Array with counter (size of stack).
     ISEMPTY: test counter == 0.
     PUSH: insert at index counter, increment counter.
     POP: (if not empty) decrement counter, remove from index counter.

What CSC263 is about?

- Analysis:
  - Correctness of algorithms
  - Run time complexity analysis

Algorithm Analysis (Review)

Complexity: amount of resources required by an algorithm, measured as a function of input size.

Resource: running time or memory (space), usually.

Why analyse complexity?

For comparison, e.g., choose between different implementations.

Input size

Input size is problem-dependent. Examples:

- For numbers, "size" = number of bits.
- For lists, "size" = number of elements.
- For graphs, "size" = number of vertices.

- measure must be roughly proportional to true bit-size (no. of bits required to fully encode input).
- In practice, allow ourselves to use $size([a_1, a_2, \dots, a_n]) = n$, when it is really $size(a_1) + \dots + size(a_n)$ proportional to n only if each a_i has constant size.

Algorithm Analysis (Review)

For a problem P with input size n there are two algorithms.

Algorithm A

Time complexity : $T_A(n)$

Algorithm B

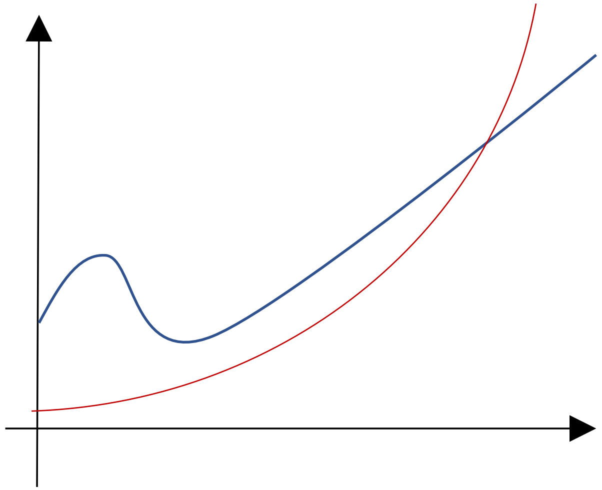
Time complexity : $T_B(n)$

Which algorithm is better?

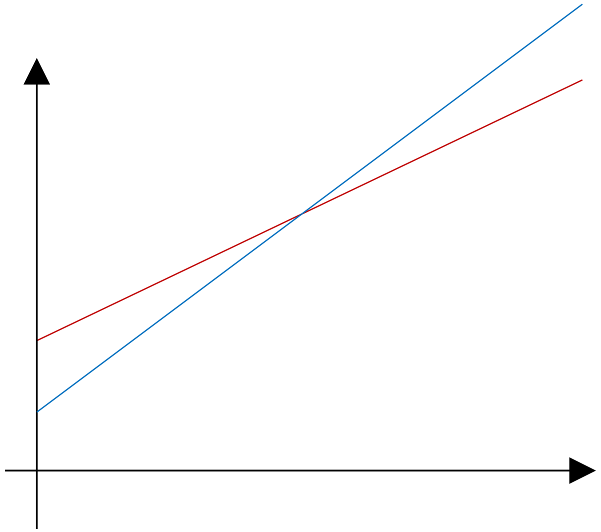
How do you compare two functions?

Algorithm Analysis (Review)

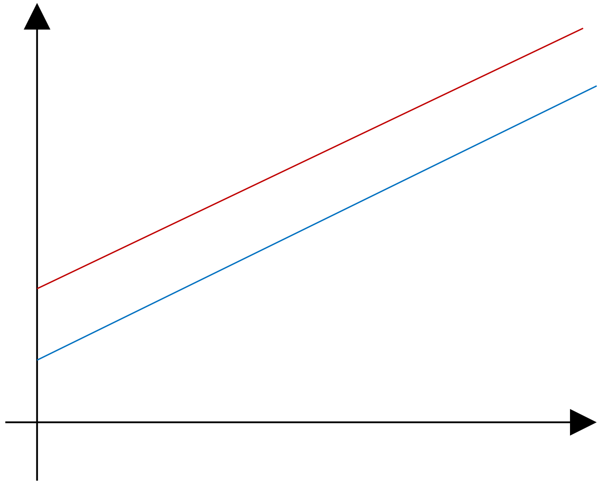
How do you compare two functions?



A



B



C

Algorithm Analysis (Review)

How do you compare two numbers?

$$2 < 3$$
$$40 < 55$$

Upper bound and lower bound for variables:

$$15 < x < 45$$
$$20 \leq x \leq 30$$

$$22 \leq x \leq 22$$

x is 22

Algorithm Analysis (Review)

How would you compare two functions?

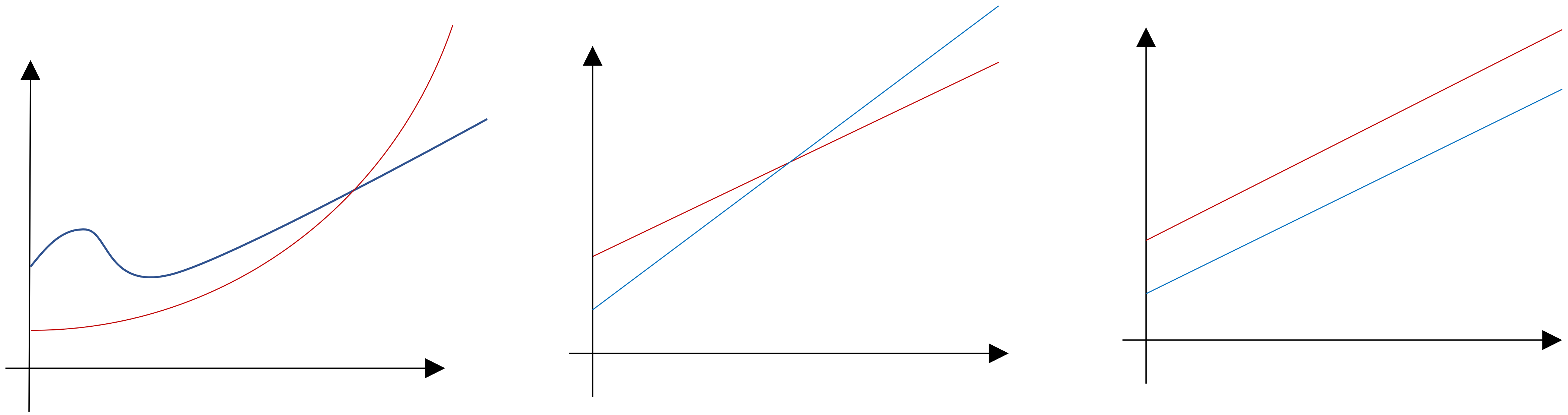
We want to define something similar for functions $<_f, >_f, =_f, \leq_f, \geq_f$

$$100\,x <_f 10x^2$$

Upper bound and lower bound for functions:

$$x <_f f(x) <_f x^3$$
$$\log x \leq_f f(x) \leq_f x^2$$
$$x \log x \leq_f f(x) \leq_f x \log x$$
$$f(x) = x \log x$$

Algorithm Analysis (Review)



We only care about

1. Large values of x
2. Growth of functions

Asymptotic notations

$f(n) \leq_f g(n)$ or $f(n) \in O(g(n))$
$\rightarrow \exists n_0, c_0 > 0$ such that for $n > n_0$ $\quad 0 \leq f(n) \leq c_0\, g(n)$

$f(n) <_f g(n)$ or $f(n) \in o(g(n))$
$\rightarrow \forall c_0 > 0\ \exists n_0 > 0$ such that for $n > n_0$ $\quad 0 \leq f(n) < c_0\, g(n)$

$f(n) \geq_f g(n)$ or $f(n) \in \Omega(g(n))$
$\rightarrow \exists n_0, c_0 > 0$ such that for $n > n_0$ $\quad f(n) \geq c_0\, g(n) \geq 0$

$f(n) >_f g(n)$ or $f(n) \in \omega(g(n))$
$\rightarrow \forall c_0 > 0\ \exists n_0 > 0$ such that for $n > n_0$ $\quad f(n) > c_0\, g(n) \geq 0$

$f(n) =_f g(n)$ or $f(n) \in \Theta(g(n))$
$\rightarrow \exists n_0, c_1, c_2 > 0$ such that for $n > n_0$ $\quad 0 \leq c_1 g(n) \leq f(n) \leq c_2\, g(n)$

Asymptotic notations

Examples:

$$10\,n^2 + 3 \in O(n^2) \in O(n^3) \in O(n^4)$$

$$10\,n^2 + 3 \in o(n^3) \in o(n^4)$$

$$10\,n^2 + 3 \in \Omega(n^2) \in \Omega(n) \in \Omega(1)$$

$$10\,n^2 + 3 \in \omega(n) \in \omega(1)$$

$$10\,n^2 + 3 \in \Theta(n^2)$$

$$40\,n^2 + 10\,n\log 5n + 100\sqrt{n}\log n + 5n + 200 \in \Theta(\,n^2)$$

Asymptotic notations

Order of growth of some common functions

$$O(1) < O(\log^* n) < O(\log n) < O(\sqrt{n}) < O(n) < O(n \log n) < O(n^2) < O(n^3) < O(2^n)$$

$\log^* n = \min\{ i \geq 0 : log^{(i)} n \leq 1\}$

The iterated logarithm is a very slowly growing function:

$\log^* 2 = 1$
$\log^* 4 = 2$
$\log^* 16 = 3$
$\log^* 65536 = 4$
$\log^* (2^{65536}) = 5$

Asymptotic notations (Upper bound and Lower bound)

Let $T(n) = \boxed{\pi\, n \lg n} + 2^{12}\, n - \sqrt{n} \lg n$

What is the best lower bound and upper bound to $T(n)$ that you find in the options listed here.

$\Omega(1)$	$O(1)$
$\Omega(\lg n)$	$O(\lg n)$
$\Omega(\lg^2 n)$	$O(\lg^2 n)$
$\Omega(\sqrt{n} \lg n)$	$O(\sqrt{n} \lg n)$
$\boxed{\Omega(n)}$	$O(n)$
$\Omega(n\sqrt{n})$	$\boxed{O(n\sqrt{n})}$
$\Omega(n^2)$	$O(n^2)$

Coding examples

Example 1:

for $(\, i = 0\,;\ \boldsymbol{i} < \boldsymbol{n}\,;\ \boldsymbol{i} + +\,)$

m += i;

Example 2:

for $(\, i = 0\,;\ i < n\,;\ i + +\,)$

for$(\, j = 0\,;\ j < n\,;\ j + +\,)$

sum[i] += entry[i][j];

Coding examples

Example 3:

for ($i = 0$; $i < n$; $i++$)

for($j = 0$; $j < i$; $j++$)

m += j;

Example 4:

i = 1;

while ($i < n$) {

tot += i;

i = i * 2;

}

Coding examples

Example 5:

```
i = n;
while (i > 0) {
        tot += i;
        i = i / 2;
}
```

Example 6:

```
for ( i = 0 ; i < n ; i + + )
  for( j = 0 ; j < n ; j + + )
    for( k = 0 ; k < n ; k + + )
      sum[i][j] += entry[i][j][k];
```

Coding examples

Example 7:

for ($i = 0$; $i < n$; $i++$)

for(j=0 ; j< $\sqrt{n}$; j++)

m += j;

Example 8:

for ($i = 0$; $i < n$; $i++$)

for($j = 0$; $j <$ $\sqrt{995}$; $j++$)

m += j;

Coding examples

Example 9:

```
for ( i = 0 ; i < n ; i + + )
{
    m += j;
    m += j;
    m += j;
    ...
    m += j;   // 31 times
}
```

Complexity Review - Example

Example: Input A, A.length = n

```
i = NIL
for j = 0 to n − 1 do
    if A[j] = v then
        i = j
        return i
return i
```

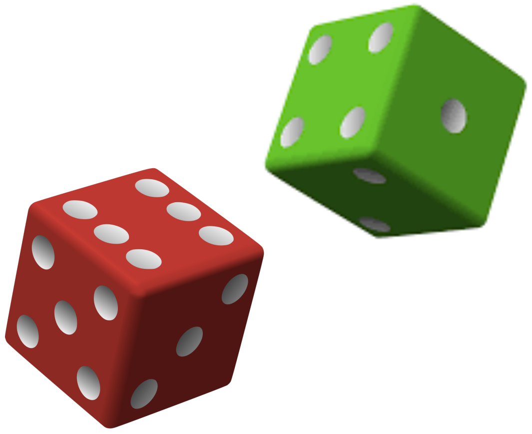
Running time, Space complexity

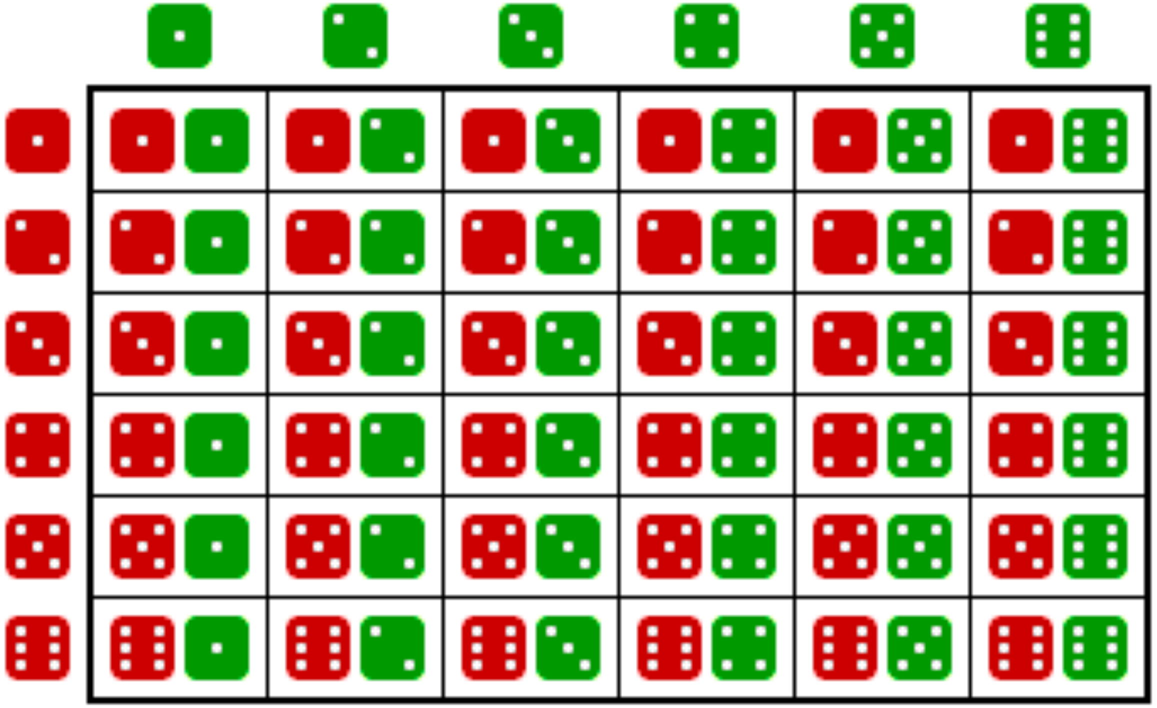
- Best Case: Minimum complexity over the sample space
- Average Case: Expected value over the sample space by considering the probability distribution over inputs
- Worst case: Maximum complexity over the sample space

- What was sample space?!

Probability Review

- Outcome: a possible result of an experiment (Algorithm). (Rolling 2 dies and getting 1,1)
- Event: a set of one or more outcomes. (Rolling 2 dies and getting only even numbers)
- Sample Space S is a set of elementary events. (Sample space for rolling 2 dies)





Complexity Review – Sample Space

- Any problem has some input.
- Sample space for a problem is the set all possible inputs that effects on the behavior of the algorithm.

Complexity Review

Example: Input A, A.length = n

```
i = NIL
for j = 0 to n − 1 do
        if A[j] = v then
                i = j
                return i
return i
```

- Sample space: infinitely many inputs!
- behavior of the algorithm determined by only one factor : position of v inside A.

$v\ occurs\ at\ position\ \ 0\ \ in\ A,$
$v\ occurs\ at\ position\ \ 1\ \ in\ A$
$\ldots$
$v\ occurs\ at\ position\ n-1\ in\ A,$
$v\ does\ not\ occur\ in\ A.$

So pick "representative" inputs (one for each execution time),
$\boldsymbol{S_n = \{ (A, v) : A = [0, 1, 2, \ldots, n-1]}$
$\boldsymbol{and\ v = 0, 1, 2, \ldots, n \}}.$

Running time

- For algorithm A, $t(x)$ represents number of "steps" executed by A on input x with size n.
- Best-case running time $T(n) = \min\{t(x) : x \text{ is an input of size } n\}$
- Worst–case running time $T(n) = \max\{t(x) : x \text{ is an input of size } n\}$

(Mostly useless!)

Example: Input A

```
i = NIL
for j = 0 to n − 1 do
        if A[j] = v then
                i = j
                return i
return i
```

Best-Case:

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

Worst Case:

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

Average-Case running time

- Average Case: Expected value over the sample space by considering the probability distribution over inputs.

  t_n be a random variable that denotes the number of comparisons executed (Line #3)

$$E(t_n) = \sum_{(A,v) \in S_n} t_n(A,v) \times P[(A,v)]$$

$\Pr[(A,n)] = p,$ v not in A as special

$\Pr[(A,v)] = (1-p)/n$ other cases equally likely

Average-Case running time

Let $t_n(A, v)$ be the number of execution of line 3 for input A, v.

$$t_n(A, v) = \begin{cases} v + 1 & if\ 0 \le v \le\ n - 1 \\ n & if\ v\ = n \end{cases}$$

$$E(t_n) = \sum_{(A,v)\in S_n}^{n} t_n(A, v) \times P[(A, v)] = n \times p + \sum_{v=0}^{n-1} (v + 1) \times (1 - p)/n$$

$$= np + \frac{1-p}{n} \sum_{v=1}^{n} v = np + (1 - p)(n + 1)/2 = (n + 1 + np - p)/2$$

Average case: $O(n), \Omega(n), \Theta(n)$

Questions?