COMP 8547
Advanced Computing Concepts

Course instructor: Dr. Olena Syrotkina

Winter 2024: 01 – Algorithm Analysis

TEAM

Instructor:
Dr. Olena Syrotkina
Email: olena.syrotkina@uwindsor.ca
Contact time: Tue 10:00 – 11:00; Wed /Thu 12:00 – 14:00;
Office Location: 300 *Ouellette Ave.*, *Room 4004*

Graduate Assistants

Prithvi Muthineni
Email: muthine@uwindsor.ca

Abhishek Mahajan
Email: mahaja52@uwindsor.ca

Hai Ly
Email: lyhai@uwindsor.ca

Rajath Bharadwaj
Email: bharadw3@uwindsor.ca

COURSE INTRODUCTION

This course covers advanced topics in principles and applications of algorithm design and analysis, programming techniques, advanced data structures, languages, compilers and translators, regular expressions, grammars, computing and intractability. Cases studies and applications in current programming languages are explored in class and labs. This course is restricted to students in the Master of Applied Computing program.

Assessments

- ✓ *Labs & Assignments*
- ✓ *Participation in seminars/workshops*
- ✓ *Random lecture quizzes*
- ✓ *Mid-term Exam*
- ✓ *Final project*

COURSE SCHEDULE

Weeks	Topics	Dates	Deadlines
1	Algorithm Analysis	Sec 1: Jan 8, 2024; Sec 3: Jan 10, 2024 Sec 2: Jan 9, 2024; Sec 4: Jan 11, 2024	Lab 1 (Introductory lab); **P1: Forming a group for the final project** (Sec 1: Jan 15; Sec 2: Jan 16; Sec 3: Jan 17; Sec 4: Jan 18)
2	Linear Data Structures	Sec 1: Jan 15, 2024; Sec 2: Jan 16, 2024; Sec 3: Jan 17, 2024; Sec 4: Jan 18, 2024	Lab 2; **P2: Choosing the variant of your final project** (Sec 1: Jan 22; Sec 2: Jan 23; Sec 3: Jan 24; Sec 4: Jan 25)
3	Search Trees	Sec 1: Jan 22, 2024; Sec 2: Jan 23, 2024; Sec 3: Jan 24, 2024; Sec 4: Jan 25, 2024	Lab 3;
4	Sorting	Sec 1: Jan 29, 2024; Sec 2: Jan 30, 2024; Sec 3: Jan 31, 2024; Sec 4: Feb 1, 2024	Lab 4; Assignment 1;
5	Advanced Design and Analysis	Sec 1: Feb 5, 2024; Sec 2: Feb 6, 2024; Sec 3: Feb 5, 2024; Sec 4: Feb 1, 2024	Lab 5; **P3: Milestone Report I**

Next page

COURSE SCHEDULE

Weeks	Topics	Dates	Deadlines
6	Graph Algorithms	Sec 1: Feb 12, 2024; Sec 2: Feb 13, 2024; Sec 3: Feb 14, 2024; Sec 4: Feb 15, 2024	Lab 6; Assignment 2;
	Reading Week	**Feb 17-25, 2024**	
7	Memory Management	Sec 1: Feb 26, 2024; Sec 2: Feb 27, 2024; Sec 3: Feb 28, 2024; Sec 4: Feb 29, 2024	Lab 7;
8	Text Processing	Sec 1: Mar 4, 2024; Sec 2: Mar 5, 2024; Sec 3: Mar 6, 2024; Sec 4: Mar 7, 2024	Lab 8; **P4: Milestone Report II**
9	Languages	Sec 1: Mar 11, 2024; Sec 2: Mar 12, 2024; Sec 3: Mar 13, 2024; Sec 4: Mar 14, 2024	Lab 9; Assignment 3;
10	Intractability and Computing	Sec 1: Mar 18, 2024; Sec 2: Mar 19, 2024; Sec 3: Mar 20, 2024; Sec 4: Mar 21, 2024	Lab 10; Assignment 4;
11	Final Project Defence (in-person at 300 Ouellette Ave.)	Sec 1: Mar 25, 2024; Sec 2: Mar 26, 2024; Sec 3: Mar 27, 2024; Sec 4: Mar 28, 2024	**P5: Final project report, presentation, project source code** (bonus + 1 pts for those groups who will defend their projects during Week 11)
12	Final Project Defence (in-person at 300 Ouellette Ave.)	Sec 1: Apr 1, 2024; Sec 2: Apr 2, 2024; Sec 3: Apr 3, 2024; Sec 4: Apr 4, 2024	**P5: Final project report, presentation, project source code (no bonus)**

COURSE EVALUATION

Labs & Assignments	25%
Participation in seminars/workshops	5%
Final project	40%
Mid-term Exam	25%
Random lecture quizzes	5%

COURSE POLICY

- **Academic Honesty**:
  - ✓ Refer to UWindsor Policy Page.
  - ✓ If you're not sure, ask the professor.
  - ✓ Plagiarism will not be tolerated.

All discussion and announcements will be on Brightspace course page.

- **Missed Assessment Make-up:**

No make-ups will be considered for

- ✓ missing a mid-exam;
- ✓ missing labs;
- ✓ missing a random lecture quiz;
- ✓ missing a final project presentation;

- **Late Assignment:**
  - ✓ Any submission after the deadline will receive a penalty of 10% for the first 24 hrs, and so on, for up to three days. After three days, the mark will be zero.

ASSESSMENTS

➢ **Labs:** *to understand the topics discussed in the lectures and to learn more about advanced algorithms and data structures.*

1. Labs should be completed during the class. Students are required to show a GA/TA their completed lab after the class, and the GA/TA will mark it as 'done,' 'not done,' or 'partially done.'

2. If a student misses the class, they will lose points for the lab, **even if they submit it on Brightspace after the lab has concluded.**

3. Each lab must be submitted through Brightspace in **both *.java and *.txt formats (+ screenshots)** and is subject to a plagiarism check.

4. <u>0.5 points </u>will be awarded for each lab if two conditions are met:
   A) The student attended the lab and showed the result to the GA;
   B) The plagiarism check originality score does not **exceed 50%**.

5. **No points will be awarded for labs submitted via email, Teams, or other platforms, for sending zip archives, or for failing to submit your code in *.txt files or screenshots.**

ASSESSMENTS

- ## Participation in seminars/workshops:

To receive your participation marks you are required to attend a total of 10 Seminars/Workshops (CS Workshops, Colloquiums, and Thesis defence or proposal) during the Winter 2024 term. You will be required to register for the event, sign in and complete the QR code after the event. The attendance will be tracked by the Admin staff and will be provided to the instructors at the end of the term to calculate your participation marks.

These events are regularly announced, and you can find the announcements at the following link, as well:
https://www.uwindsor.ca/science/computerscience/event-calendar/month
The participation is saved by registering to the attendance list (or scanning a QR code) available at each event.

ASSESSMENTS

- **Random lecture quizzes (RLQ):**

These are quizzes regarding lectures to test the topics discussed in the lectures. RLQ will be conducted randomly at any time (before or after the class) and will contain multiple-choice questions.

There are no make-ups for missed RLQ.

ASSESSMENTS

- **Final project - 40 points**

Milestone reports I and II - 0 points*

*There are no points for milestone reports. However, for not submitting their milestone reports, students will be faced a penalty of 10% (for each report) deducted from their final project total mark.

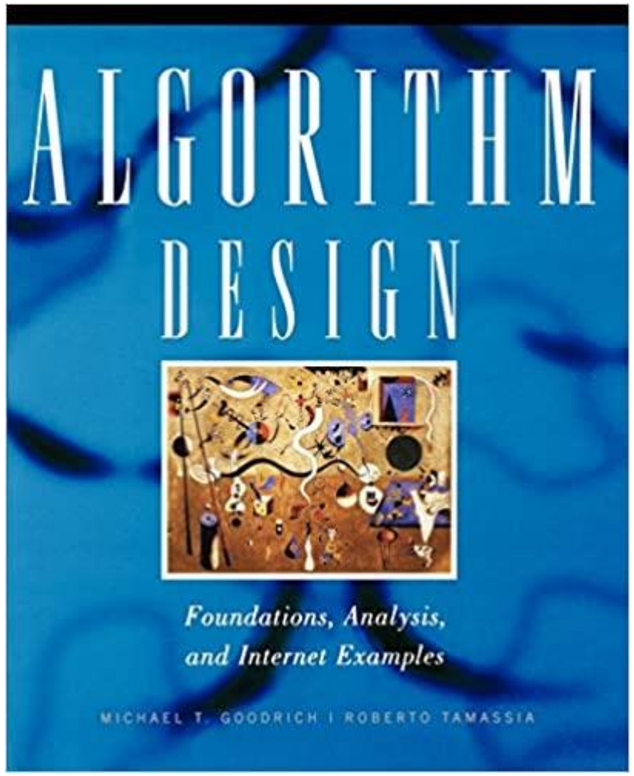
Final project report - 5 points

Presentation and Q&A (during your class in the 11th and 12th week) - 35 points

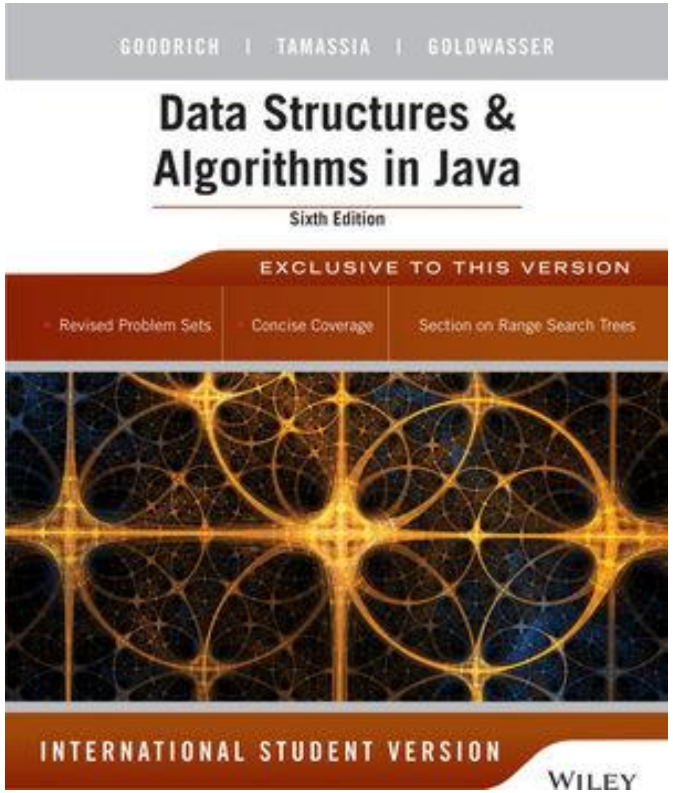
STUDENT RESPONSIBILITIES

- ➢ Be polite in all dealings with the professor, the GA/TAs, and the other students.
- ➢ Come to the class **on time** and ready to participate in the learning process.
- ➢ If you miss any announcement, it is your responsibility to catch up on instruction you have missed.
- ➢ Make sure that you do not plagiarize in any assignment.
- ➢ Make sure to submit all assignments (including the project) on time.

TEXTBOOKS


ALGORITHM DESIGN
Foundations, Analysis, and Internet Examples
MICHAEL T. GOODRICH | ROBERTO TAMASSIA

Algorithm Design and Applications by M. Goodrich and R. Tamassia, Wiley, 2015.


GOODRICH | TAMASSIA | GOLDWASSER
Data Structures & Algorithms in Java
Sixth Edition
EXCLUSIVE TO THIS VERSION
Revised Problem Sets
Concise Coverage
Section on Range Search Trees
INTERNATIONAL STUDENT VERSION
WILEY

Data Structures and Algorithms in Java, 6th Edition, by M. Goodrich and R. Tamassia, Wiley, 2014

Chapter 1 – Algorithm Analysis

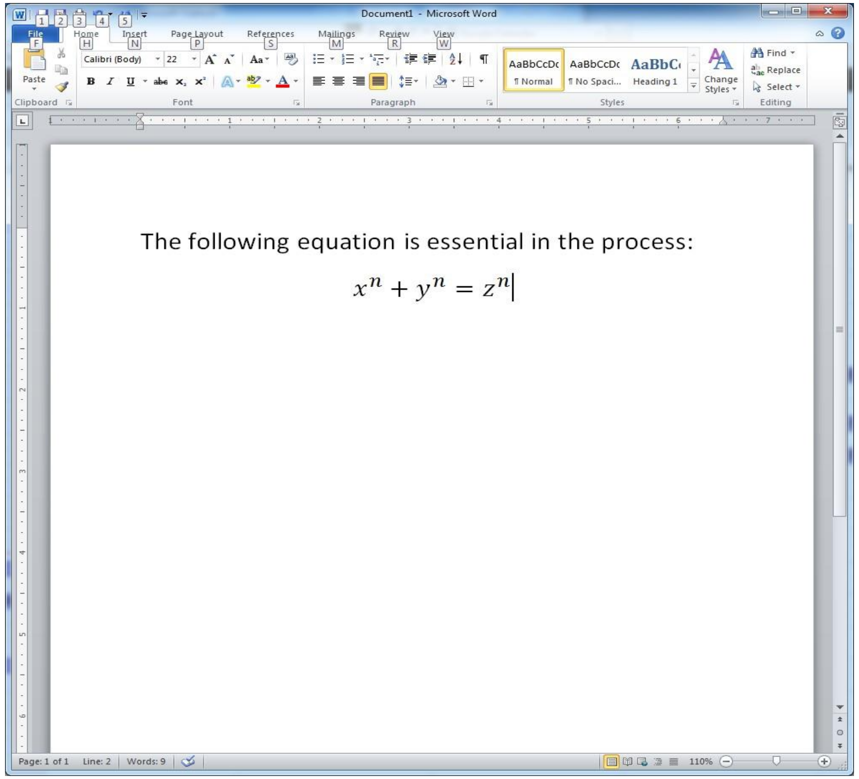
Contents

- Introduction to computing
- Algorithms
- Experimental analysis
- Pseudocode
- Random access machine
- Functions in algorithm analysis
- Asymptotic analysis
- Asymptotic notation
  - Big-Oh, Big-Omega, Big-Theta, Little-Oh, Little-Omega
- Examples
  - Search
  - Prefix averages
  - Maximum contiguous subsequence sum
- Java and Eclipse

Computing: What is computer science?

My computer crashed ☹
How do I fix it?

How do I insert an equation
in MS Word? ☺



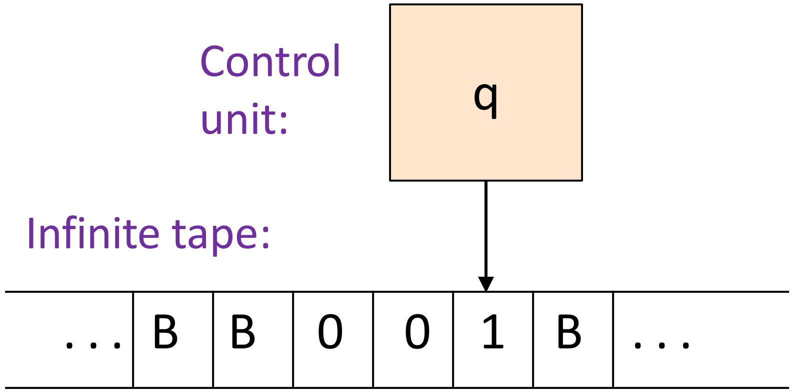
The following equation is essential in the process:

$$x^n + y^n = z^n$$

None of these!

Abstract vs. Real Computer

Abstract computer:
Turing machine (TM)



Control unit:

q

Infinite tape:

. . . | B | B | 0 | 0 | 1 | B | . . .

Real computers:

NASA’s supercomputer

My little smartphone

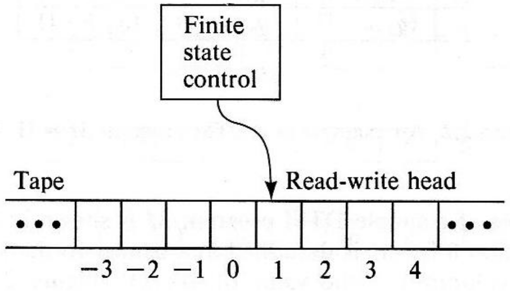
- TMs and Computers are structurally different but equivalent:
  - They do exactly the same!
- A TM solves/decides problems/languages:
  - Recursive languages
- A problem/language is recursive if:
  - There is an algorithm that solves/decides it

My own view of computing

Turing Machine (TM) – A. Turing, 1936

Abstract computing machine used for:

- Real computations
- Math definitions
- Solve any computing (decidable) problem



Language

- A set of strings from an alphabet
- TM decides/solves any recursive language (= algorithm)

L = {100, 1000, 1100, 10000, 10100, … }

Context-free Grammars – N. Chomsky, 1956

- Context-free languages (CFL) defined by context-free grammars
- CFL are (a subset of) recursive languages
- TM can then parse strings of CFLs

Σ = {0,1}, $N = S = \{P\}$

Productions:

$$P \to \epsilon \mid P \to 0 \mid P \to 1$$

$$P \to 0P0 \mid P \to 1P1$$

Program

- A computer program can be seen as a string of a language (mainly CFLs)
- First programming language: Fortran - 1955

```
public class Maximum {
    public static void main(String[] args) {
      int currentMax = …;
      …
    }
}
```

Algorithm

- Written in pseudocode
- Translated to any programming language
- Key: complexity analysis - D. Knuth, 1962

```
Algorithm $arrayMax(A, n)$
    $currentMax \leftarrow A[0]$
    for $i \leftarrow 1$ to $n - 1$ do
        if $A[i] > currentMax$ then
            $currentMax \leftarrow A[i]$
    return $currentMax$
```

Algorithm

- Definition: An algorithm is a sequence of steps used to solve a problem in a finite amount of time

Input
Output

- Properties
  - Correctness: must provide the correct output for *every* input
  - Performance: Measured in terms of the resources used (time and space)
  - End: must finish in a *finite* amount of time

Input size

- Performance of an algorithm measured in terms of the input size
- Examples:
  - Number of elements in a list or array A: *n*
  - Number of cells in an *m*x*n* matrix: *m*, *n*
  - Number of bits in an integer: *n*
  - Number of nodes in a tree: *n*
  - Number of vertices and edges in a graph: $|V|$, $|E|$

A

21	**22**	**23**	**25**	**24**	**23**	**22**
0	1	2	3	4	5	6

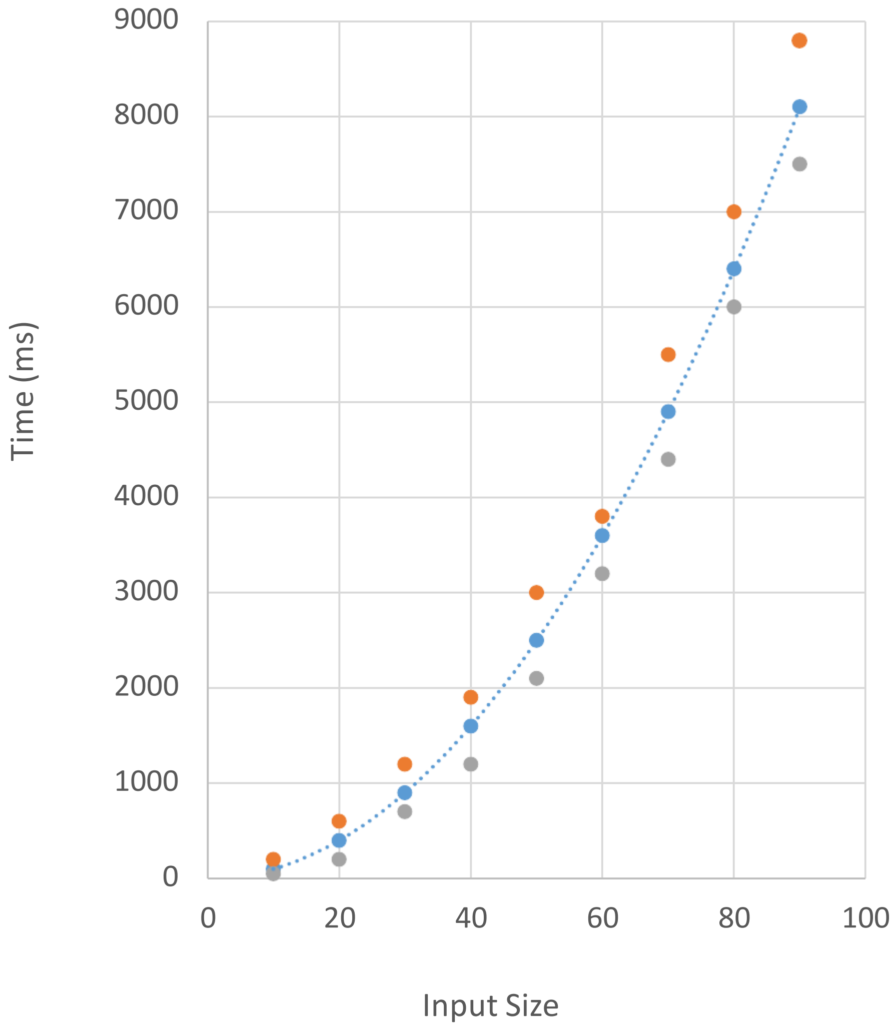
Experimental vs Theoretical analysis

Experimental analysis:

- Write a program that implements the algorithm
- Run the program with inputs of varying size and composition
- Keep track of the CPU time used by the program on each input size
- Plot the results on a two-dimensional plot

Limitations:

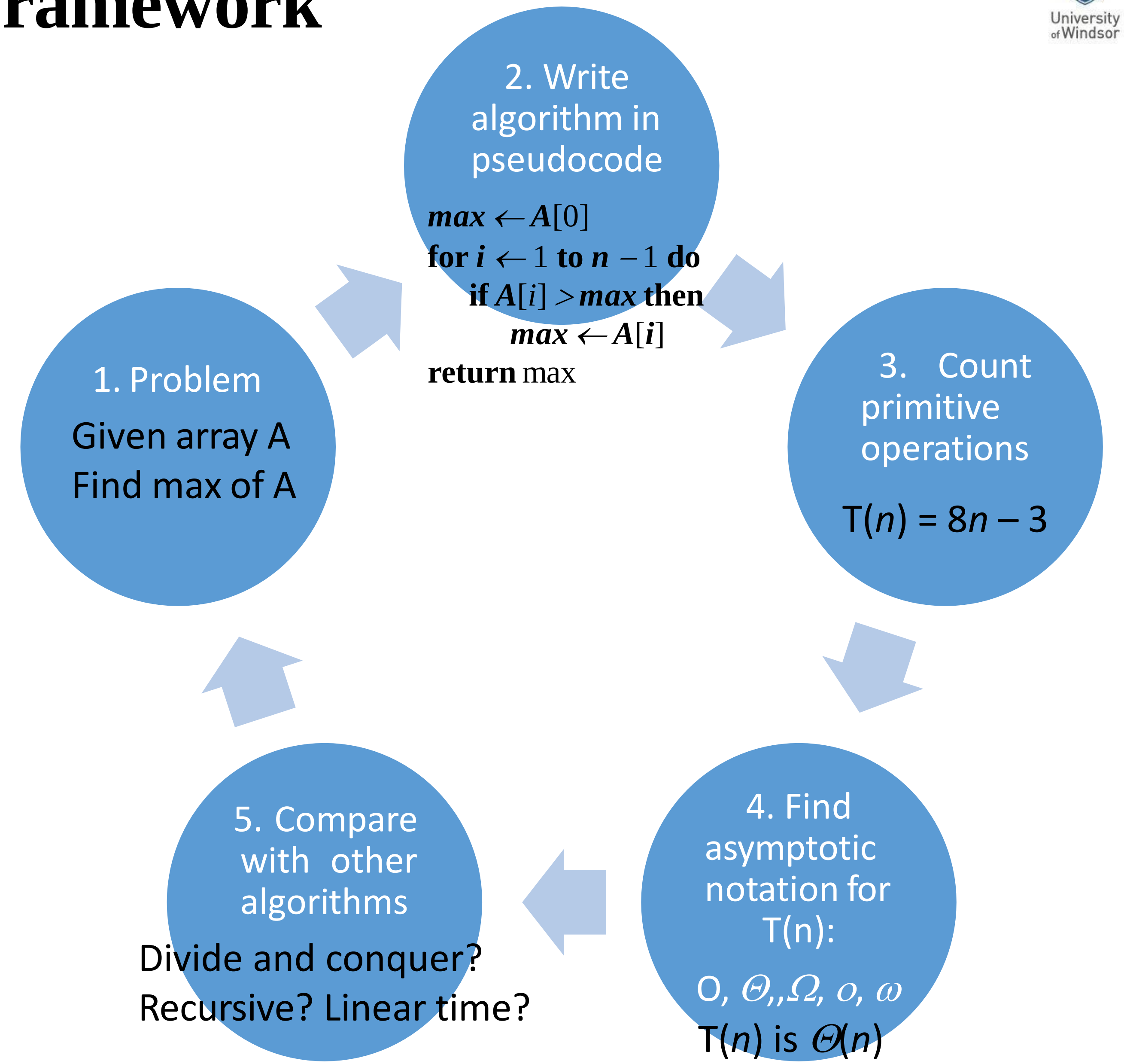
- Depends on hardware and programming language
- Need to implement the algorithm and debug the programs



Theoretical analysis – main framework

Advantages:

- Uses a high-level description of the algorithm instead of an implementation
- Characterizes running time as a function of the input size, *n*.
- Takes into account *all* possible inputs
- Allows us to evaluate the speed of an algorithm independently of the hardware/software environment



1. Problem

Given array A

Find max of A

2. Write algorithm in pseudocode

```
$max \leftarrow A[0]$
for $i \leftarrow 1$ to $n - 1$ do
    if $A[i] > max$ then
        $max \leftarrow A[i]$
return max
```

3. Count primitive operations

T(*n*) = 8*n* – 3

4. Find asymptotic notation for T(n):

O, Θ,,Ω, o, ω

T(*n*) is Θ(*n*)

5. Compare with other algorithms

Divide and conquer?

Recursive? Linear time?

Pseudocode

- Control flow
  - **if** … **then** … [**else** …]
  - **while** … **do** …
  - **repeat** … **until** …
  - **for** … **do** …
  - Indentation replaces braces
- Method declaration

```
Algorithm method (arg [, arg…])
    Input …
    Output …
    return …
```

- Method call

  var.method (arg [, arg…])

- Return value

  return *expression*

- Expressions

  ← Assignment (like = in Java)

  = Equality testing (like == in Java)

  n^2 Superscripts and other mathematical formatting allowed

Example: find max element of an array

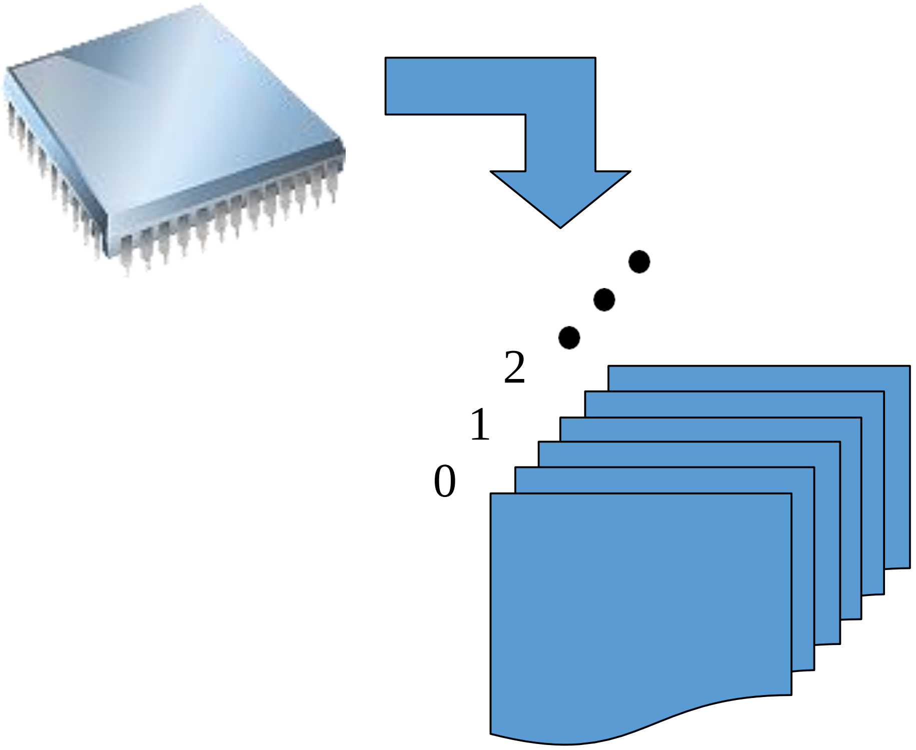
```
Algorithm arrayMax(A, n)
    Input array A of n integers
    Output maximum element of A

    currentMax ← A[0]
    for i ← 1 to n − 1 do
        if A[i] > currentMax then
            currentMax ← A[i]
    return currentMax
```

Pseudocode provides a high-level description of an algorithm and avoids to show details that are unnecessary for the analysis.

Random Access Machine (RAM)

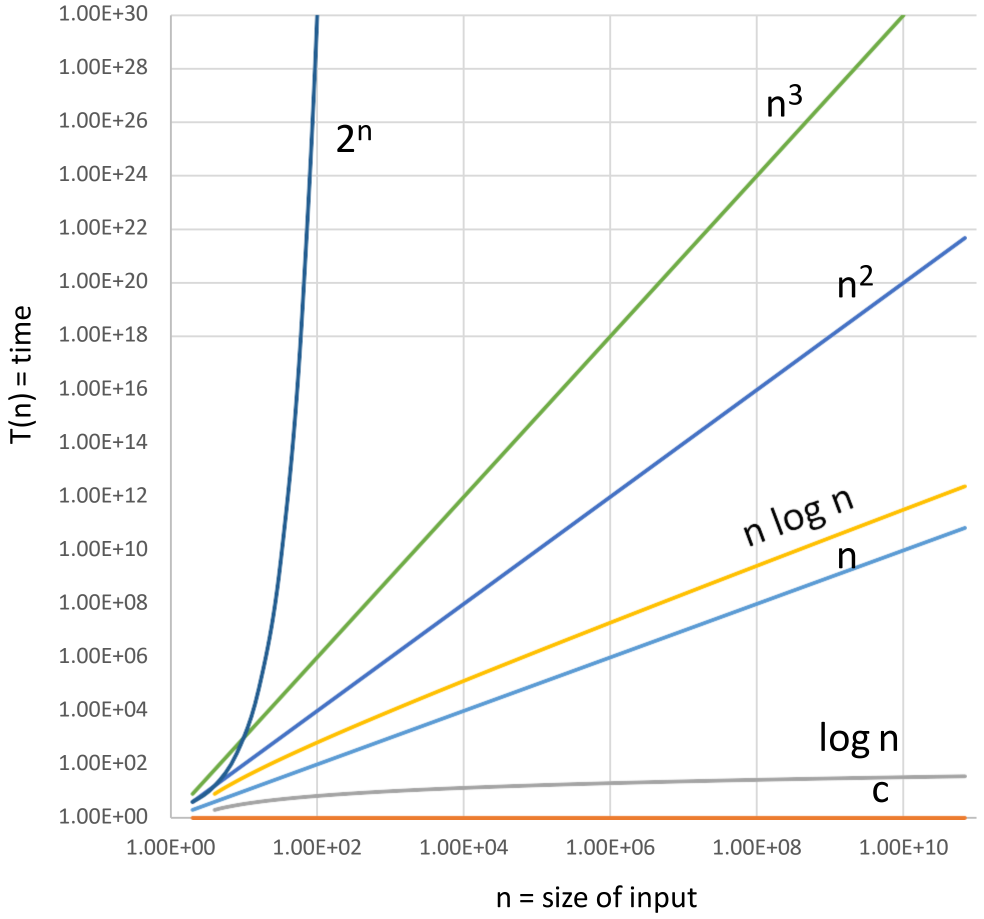
- It has a **CPU** – equivalent to that of a conventional computer
- A potentially unbounded bank of **memory** cells
- Each memory cell:
  - can hold an arbitrary number or character
  - is referenced by a number or index
  - Can be accessed in unit time



Most important functions used in Algorithm Analysis

- The following functions often appear in algorithm analysis:
  - Constant $\approx \mathbf{1}$ (or c)
  - Logarithmic $\approx \log \boldsymbol{n}$
  - Linear $\approx \boldsymbol{n}$
  - N-Log-N $\approx \boldsymbol{n} \log \boldsymbol{n}$
  - Quadratic $\approx \boldsymbol{n}^2$
  - Cubic $\approx \boldsymbol{n}^3$
  - Exponential $\approx \mathbf{2}^n$

- In a log-log chart, the slope of the line corresponds to the growth rate of the function (except exponential)



Primitive operations

- Basic computations performed by an algorithm
- Identifiable in pseudocode
- Largely independent of the programming language
- Exact definition not important (we will see why later)
- Take a constant amount of time in the RAM model (one unit of time or constant time)

Examples:

- Evaluating an expression

e.g. $a - 5 + c\sqrt{b}$

- Assigning a value to a variable

e.g. $a \leftarrow 23$

- Indexing into an array

e.g. $A[i]$

- Calling a method

e.g. *v.method()*

- Returning from a method

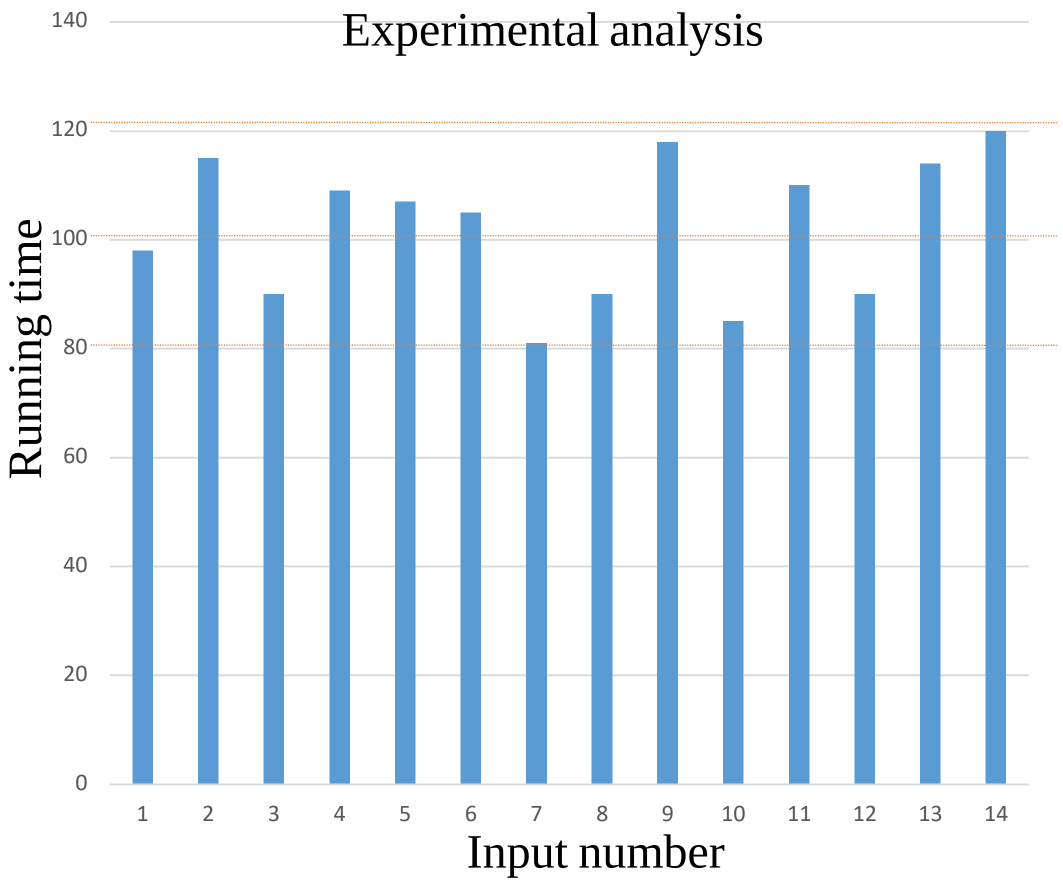
e.g. *return a*

Counting primitive operations:

```
Algorithm arrayMax(A, n)                         # operations
    currentMax ← A[0]                                  2
    for i ← 1 to n − 1 do                              2n
        if A[i] > currentMax then                  2(n − 1)
            currentMax ← A[i]                      2(n − 1)
        { increment counter i }                    2(n − 1)
    return currentMax                                  1
                                                  ________

                                       Total: T(n) = 8n − 3
```

Case analysis



Three cases

- Worst case:
  - among all possible inputs, the one which takes the largest amount of time.
- Best case:
  - The input for which the algorithm runs the fastest
- Average case:
  - The average is over all possible inputs
  - Can be considered as the expected value of T(n), which is a random variable

Example: Best vs Worst Case Scenario

Asymptotic notation

Name	Notation /use	Informal name	Bound	Notes
Big-Oh	$O(n)$	order of	Upper bound – tight	The most commonly used notation for assessing the complexity of an algorithm
Big-Theta	$\Theta(n)$		Upper and lower bound – tight	The most accurate asymptotic notation
Big-Omega	$\Omega(n)$		Lower bound – tight	Mostly used for determining lower bounds on problems rather than algorithms (e.g., sorting)
Little-Oh	$o(n)$		Upper bound – loose	Used when it is difficult to obtain a tight upper bound
Little-Omega	$\omega(n)$		Lower bound – loose	Used when it is difficult to obtain a tight lower bound

Asymptotic notation

Big - Oh Notation (O)

- Big - Oh notation is used to define the upper boundary of an algorithm in terms of Time Complexity.
- Worst Case.

Big - Omega Notation (Ω)

- Big - Omega notation is used to define the lower boundary of an algorithm in terms of Time Complexity.
- Best Case

Big - Theta Notation (Θ)

- Big - Theta notation is used to define the average boundary of an algorithm in terms of Time Complexity.
- Average Case

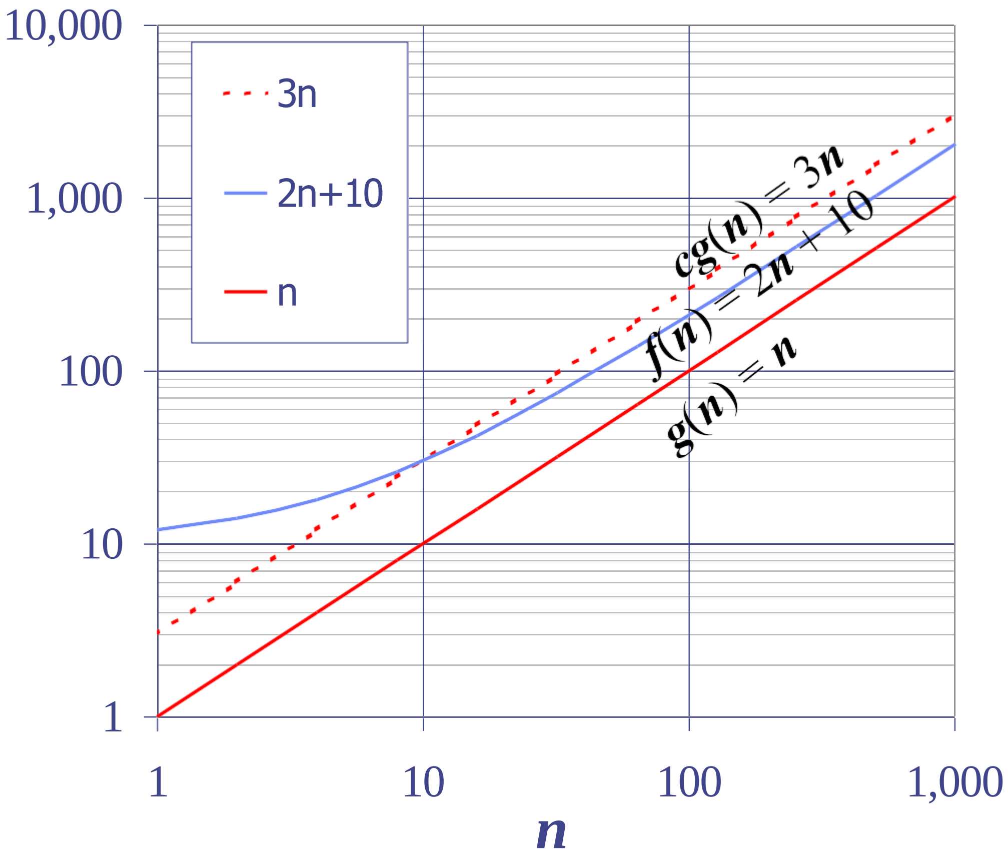
Big-Oh notation

- Given functions $f(n)$ and $g(n)$, we say that $f(n)$ is $O(g(n))$ if there are positive constants c and n_0 such that

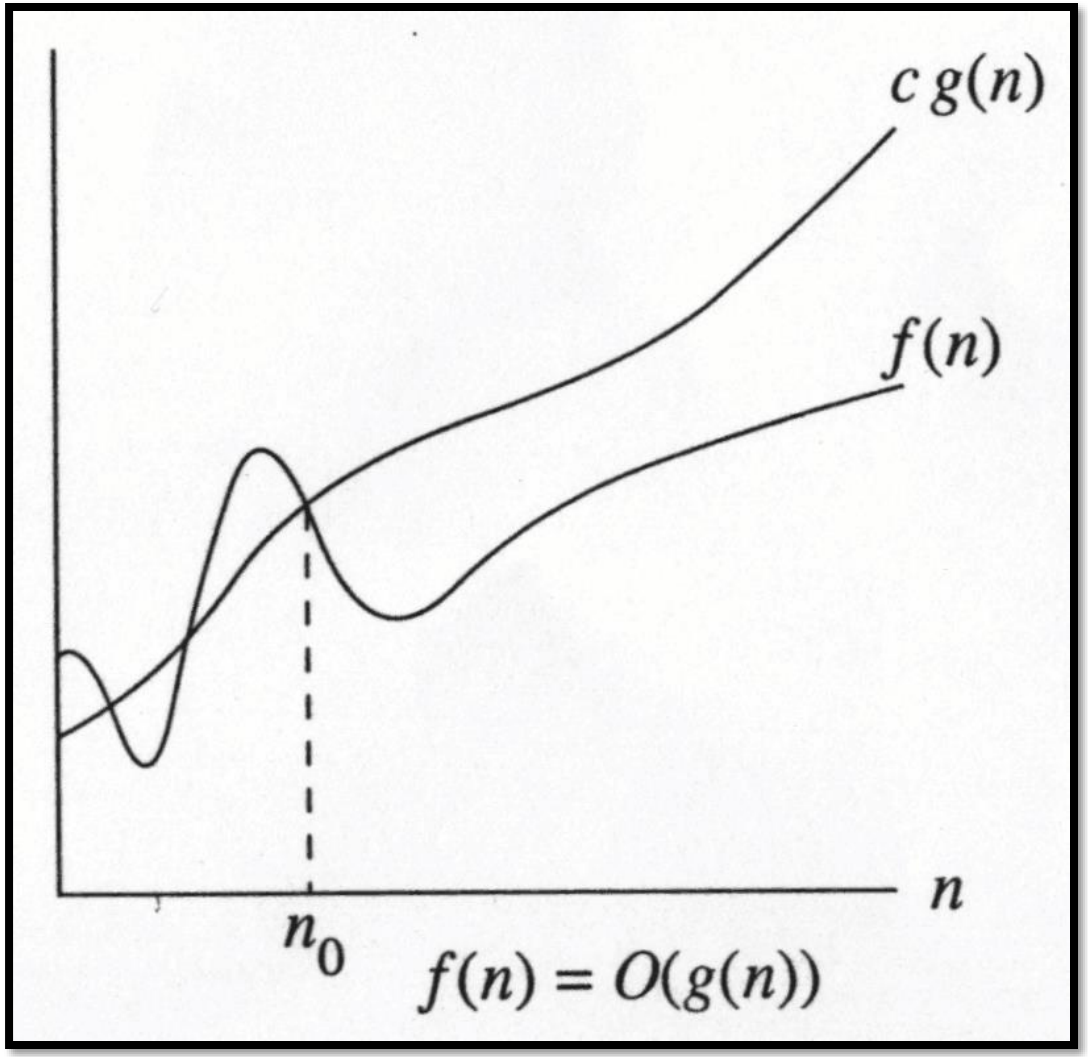
  $f(n) \leq cg(n)$ for $n \geq n_0$

- Example: $2n + 10$ is $O(n)$
  - $2n + 10 \leq cn$
  - $(c - 2)\, n \geq 10$
  - $n \geq 10/(c - 2)$
  - Pick $c = 3$ and $n_0 = 10$
  - It follows that

  $2n + 10 \leq 3n$ for $n \geq 10$



Asymptotic notation: Big-Oh



The idea is to establish a relative order among functions for large

n ∃ c , n_0 > 0 such that

$$f(n) \leq c\ g(n)$$

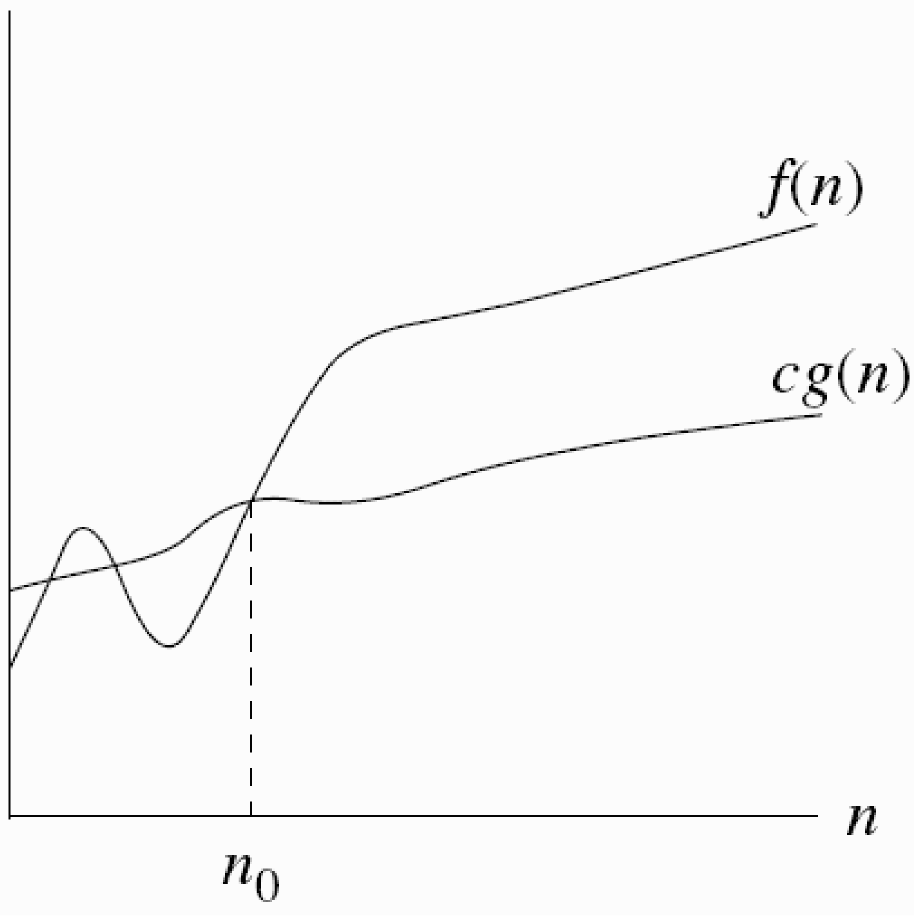
when $n \geq n_0$

f(n) grows no faster than g(n) for “large” n

Big-Oh rules - properties

- The statement "***f*(*n*) is *O*(*g*(*n*))**" means that the growth rate of ***f*(*n*)** is no more than the growth rate of ***g*(*n*)**
- ***O*(*g*(*n*))** is a *set* or *class* of functions: it contains all the functions that have the *same growth rate*
- If ***f*(*n*)** is a polynomial of degree ***d***, then ***f*(*n*)** is $O(n^d)$
  - If $d = 0$, then ***f*(*n*)** is ***O*(1)**
  - Example: $n^2 + 3n - 1$ is $O(n^2)$
- We always use the *simplest* expression of the class/set
  - E.g., we state 2n + 3 is O(n) instead of O(4n) or O(3n+1)
- We always use the *smallest* possible class/set of functions
  - E.g., we state 2n is O(n) instead of $O(n^2)$ or $O(n^3)$
- Linearity of asymptotic notation
  - O(f(n)) + O(g(n)) = O(f(n) + g(n)) = O(max{f(n),g(n)}), where "max" is wrt "growth rate"
  - Example: $O(n) + O(n^2) = O(n + n^2) = O(n^2)$

Asymptotic notation: Big-Ω (Big-Omega)



f(n) = O(g(n))

If there are positive constants c and n_0

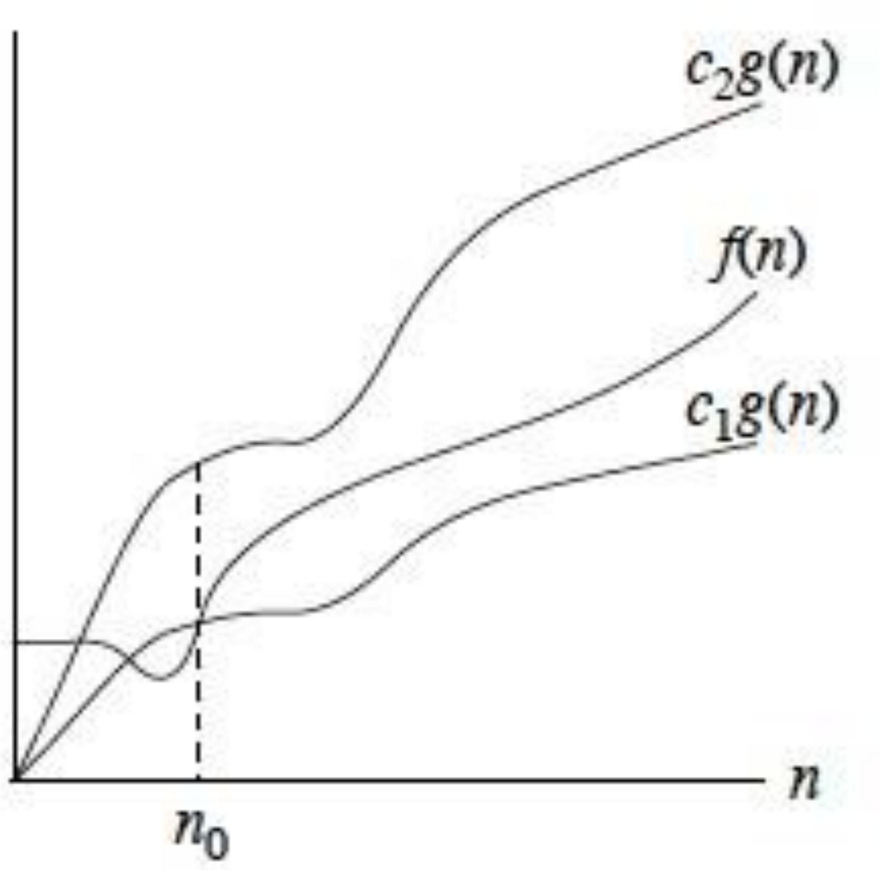
Such that

f(n) > c x g(n)

for all n >= n_0

g(n) is an asymptotic Lower bound for f(n)

Asymptotic notation: Big-θ (Big-Theta)



$f(n) = O(g(n))$

If there are positive constants c1,c2 and n_0

Such that

c1g(n) < f(n) < c2 x g(n)

for all n >= n_0

g(n) is an asymptotic Tight boundary for f(n)

Big-Omega and Big-Theta notations

- **big-Omega**
  - f(n) is Ω(g(n)) if there is a constant c > 0 and an integer constant $n_0 \geq 1$ such that f(n) $\geq$ c g(n) for n $\geq n_0$

  Example: $3n^3 - 2n + 1$ is $\Omega(n^3)$
- **big-Theta**
  - f(n) is Θ(g(n)) if there are constants c' > 0 and c'' > 0 and an integer constant $n_0 \geq 1$ such that c' g(n) $\leq$ f(n) $\leq$ c'' g(n) for n $\geq n_0$
  - Example: 5n log n – 2n is Θ(n log n)

- Important axiom:
  - f(n) is O(g(n)) and Ω(g(n)) $\Leftrightarrow$ f(n) is Θ(g(n)
  - Example: $5n^2$ is $O(n^2)$ and $\Omega(n^2) \Leftrightarrow 5n^2$ is $\Theta(n^2)$

Asymptotic notation – graphical comparison

Big-Oh

- f(n) is O(g(n)) if f(n) is asymptotically **less than or equal** to g(n)

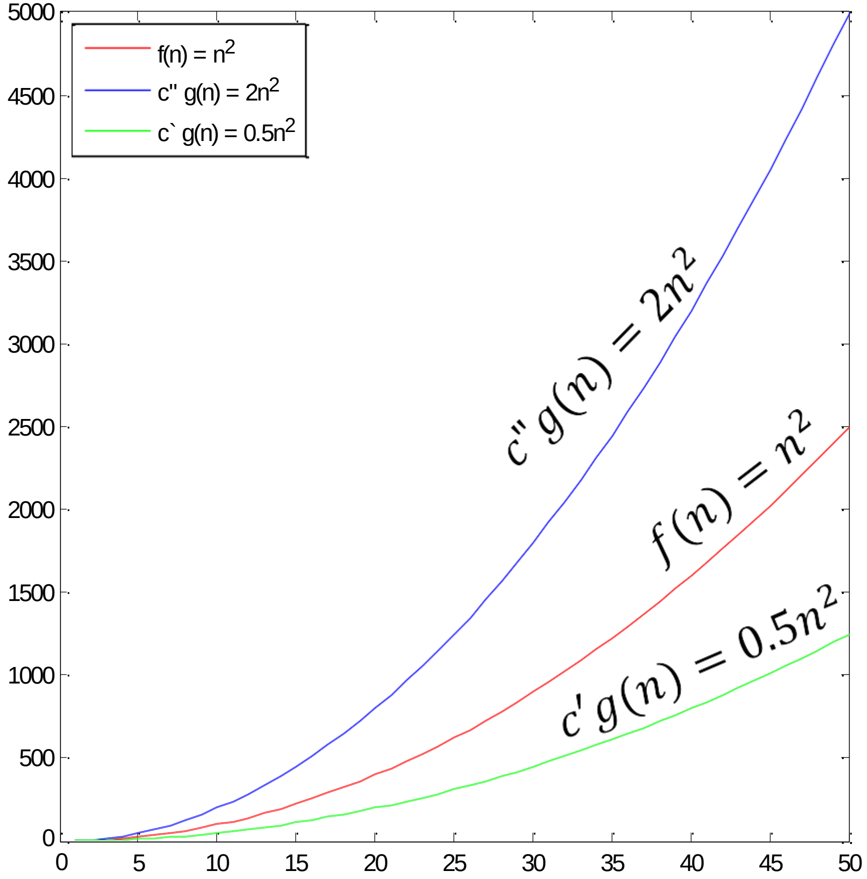
Big-Omega

- f(n) is Ω(g(n)) if f(n) is asymptotically **greater than or equal** to g(n)

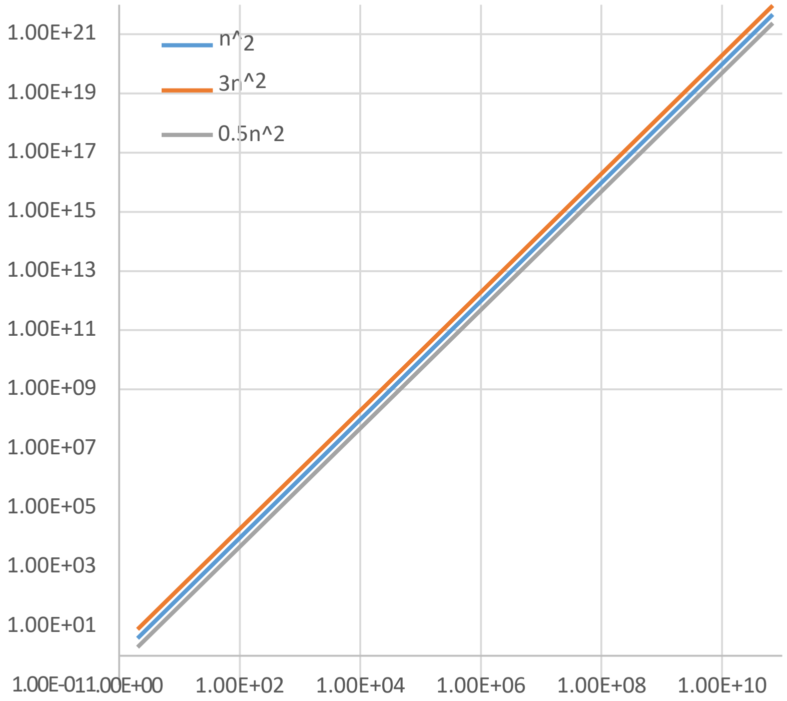
Big-Theta

- f(n) is Θ(g(n)) if f(n) is asymptotically **equal** to g(n)

Normal scale:



Log-log scale:



Little-Oh and Little-Omega notations

- **Little-Oh**
  - f(n) is o(g(n)) if *for any* constant c > 0, there is a constant $n_0 > 0$ such that f(n) < c g(n) for $n \geq n_0$

  Example: $3n^2 - 2n + 1$ is o(n^3), while $3n^2 - 2n + 1$ **is not** o(n^2)

- **Little-Omega**
  - f(n) is ω(g(n)) if for any constant c > 0, there is a constant $n_0 > 0$ such that f(n) > c g(n) for $n \geq n_0$
  - Example: $3n^2 - 2n + 1$ is ω(n), while $3n^2 - 2n + 1$ **is not** ω(n^2)

Important axiom:

f(n) is o(g(n)) $\Leftrightarrow$ g(n) is ω(f(n))

- Comparison with O and Ω
  - For O and Ω, the inequality holds if **there exists** a constant c > 0
  - For o and ω , the inequality holds **for all** constants c > 0

Case study 1: Search in a Map (sorted list)

- Problem: Given a sorted array S of integers (a map), find a key k in that map.
- One of the most important problems in computer science
- Solution 1: Linear search
  - Scan the elements in the list one by one
  - Until the key k is found
- Example:

S[*i*]	**8**	**12**	**19**	**22**	**23**	**34**	**41**	**48**
i	0	1	2	3	4	5	6	7

- Linear search runs in *linear* time.

Algorithm linearSearch(S, *k*, *n*):

Input: Sorted array S of size *n*, and key *k*

Output: Null or the element found

```
$i \leftarrow 0$
while $i < n$ and $S[i] != k$
    $i \leftarrow i + 1$
if $i = n$ then
    return null
else
    $e \leftarrow S[i]$
    return $e$
```

Worst-case running time: T(n) = 3n + 4 ➔ T(n) is O(n)

Case study 1: Search in a Map (sorted list)

- Problem: Given a sorted array of integers (a map), find a key k in that map.
- Solution 2: Binary search
- Binary search runs in *logarithmic* time
- Same problem:
  - Two algorithms run in different times

S[*i*]	8	12	19	22	23	34	41	48
i	0	1	2	3	4	5	6	7

```
Algorithm binarySearch(S, k, low, high):
Input: A key k
Output: Null or the element found
if low > high then
        return null
else
  mid $\leftarrow \lfloor (low + high) / 2 \rfloor$
  e $\leftarrow$ S[mid]
  if k = e.getKey() then
        return e
  else if k < e.getKey() then
           return binarySearch(S, k, low, mid-1)
     else
           return binarySearch(S, k, mid+1, high)
```

Worst-case running time: $T(n) = T(n/2) + 1$ ➔ $T(n)$ is $O(\log n)$

Case study 2: Prefix averages

- The ***i***-th prefix average of an array ***S*** is the average of the first $(\boldsymbol{i}+1)$ elements of ***S*:**

$$\boldsymbol{A}[\boldsymbol{i}] = (\boldsymbol{S}[0] + \boldsymbol{S}[1] + \ldots + \boldsymbol{S}[\boldsymbol{i}])/(\boldsymbol{i}+1)$$

- Problem: Compute the array ***A*** of prefix averages of another array ***S***
- Has applications in financial analysis
- Solution 1: A quadratic-time algorithm: quadPrefixAve
- Example:

S	**21**	**23**	**25**	**31**	**20**	**18**	**16**
	0	1	2	3	4	5	6

A	**21**	**22**	**23**	**25**	**24**	**23**	**22**
	0	1	2	3	4	5	6

```
Algorithm quadPrefixAve(S, n)
    Input: array S of n integers
    Output: array A of prefix averages of S
                                                    #operations
    A ← new array of n integers                     n
    for i ← 0 to n - 1 do                           n
        s ← S[0]                                    n – 1
        for j ← 1 to i do                           1 + 2 + …+ (n - 1) ⎫  n(n – 1)
            s ← s + S[j]                            1 + 2 + …+ (n - 1) ⎭  ────────
                                                                              2
        A[i] ← s / (i + 1)                          n - 1
    return A                                        1
```

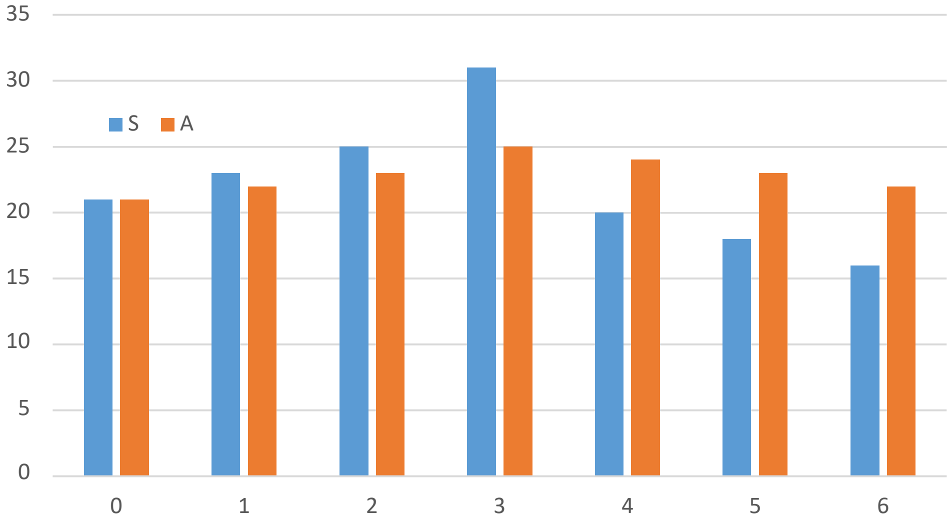
$T_2(n) = 2n + 2(n-1) + 2n(n-1)/2 + 1$ is $O(n^2)$

Case study 2: Prefix averages

- Solution 2: A linear-time algorithm: linearPrefixAve
- For each element being scanned, keep the running sum

S	21	23	25	31	20	18	16
	0	1	2	3	4	5	6

A	21	22	23	25	24	23	22
	0	1	2	3	4	5	6



```
Algorithm linearPrefixAve(S, n)
    Input: array S of n integers
    Output: array A of prefix averages of S
                                                    #operations
    A ← new array of n integers                          n
    s ← 0                                                1
    for i ← 0 to n - 1 do                                n
        s ← s + S[i]                                   n – 1
        A[i] ← s / (i + 1)                             n – 1
    return A                                             1
```

$T_2(n) = 4n$ is $O(n)$

Case study 3: Maximum contiguous subsequence sum (MCSS)

- Problem:
  - Given: a sequence of integers (possibly negative) $A = A_1, A_2, \dots, A_n$
  - Find: the maximum value of $\sigma_{k=}^{j}\ A_k$
  - If all integers are negative the MCSS is 0
- Example:
  - For A = -3, 10, -2, 11, -5, -2, 3 the MCSS is 19
  - For A = -7, -10, -1, -3 the MCSS is 0
  - For A = 12, -5, -6, -4, 3 the MCSS is 12
- Various algorithms solve the *same* problem
  - Cubic time
  - Quadratic time
  - Divide and conquer
  - Linear time

MCSS: Cubic vs quadratic time algorithms

```
Algorithm cubicMCSS(A,n)
Input: A sequence of integers A of length n
Output: The value of the MCSS
maxS ← 0
for i ← 0 to n – 1 do
    for j ← i to n – 1 do
        curS ← 0
        for k ← i to j do
            curS ← curS + A[k]
        if curS > maxS
            maxS ← curS
return maxS
```

$$\sum_{i=0}^{n-1} \sum_{j=i}^{n-1} \sum_{k=i}^{j} 1 = \frac{n^3+3n^2+2n}{6}$$

$T(\mathrm{n}) = \frac{n^3+3n^2+2n}{6} + c$ is $O(n^3)$

```
Algorithm quadraticMCSS(A,n)
Input: A sequence of integers A of length n
Output: The value of the MCSS
maxS ← 0
for i ← 0 to n – 1 do
    for j ← i to n – 1 do
        curS ← curS + A[j]
        if curS > maxS
            maxS ← curS
return maxS
```

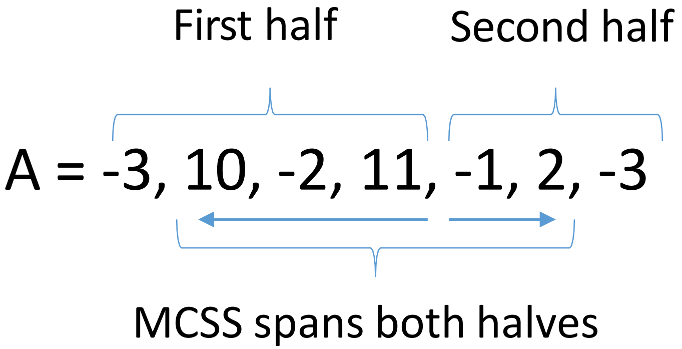
$$\propto \frac{n(n-1)}{2}$$

The double sum will give $O(n^2)$

Example: for A = -3, 10, -2, 11, -5, -2, 3 the MCSS is 19

MCSS: Divide and conquer

- Main features:
  - Rather lengthy
  - Split the sequence into two



- Algorithm:
  - Divide subsequence into two halves
  - Find max left border sum (left arrow)
  - Find max right border sum (right arrow)
  - Return the sum of both maximums as the max sum
  - Do this recursively for each half
  - Complexity given by $T(n) = 2T(n/2) + n$, where $T(1) = 1$
  - Runs in $O(n \log n)$

Linear time algorithm

- Tricky parts of this algorithm are:
  - No MCSS will **start** or **end** with a negative number
  - We only find the value of the MCSS
  - But if we need the actual subsequence, we'll need to resort on at least divide and conquer

Example: for A = -3, 10, -2, 11, -5, -2, 3 the MCSS is 19

```
Algorithm linearMCSS(A,n)
Input: A sequence of integers A of length n
Output: The value of the MCSS
$maxS \leftarrow 0$; $curS \leftarrow 0$
for $j \leftarrow 0$ to $n - 1$ do
    $curS \leftarrow curS + A[j]$
    if $curS > maxS$
        $maxS \leftarrow curS$
    else
        if $curS < 0$
            $curS \leftarrow 0$
return $maxS$
```

The single for loop gives $O(n)$

Example: Best vs worst case

Loops:

Worst Case: take maximum

Best Case: take minimum

	worst	best
$i \leftarrow 0$	1	1
while $i < n$ **and** $A[i]$!= 7	n	1
$\quad i \leftarrow i + 1$	n	0
	O(n)	**O(1)**

Worst-case input:

3	1	4	2	3	2	1	8	
0	1	2	3	4	5	6	7	**n**

Best-case input:

7	1	5	4	8	2	1	9	
0	1	2	3	4	5	6	7	**n**

Our programming language: Java

According to Sun’s Java developers:

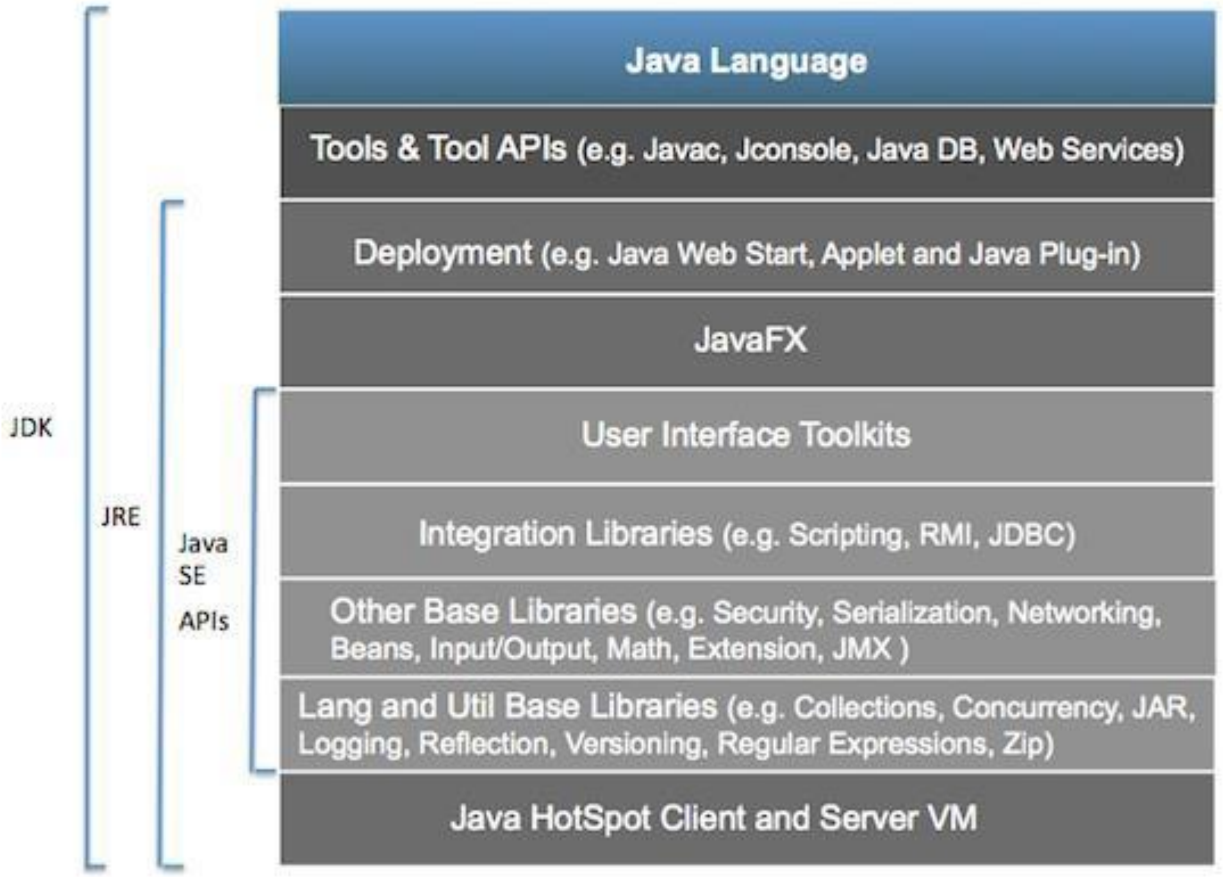
“Java is a simple, object-oriented, distributed, secure, architecture, robust, multi threaded and dynamic language. The program can be written once, and run anywhere”

•It runs on the Java Virtual Machine (JVM)

Main features of Java 8

- Object is the main data type, and can be as general as we wanted:
  - public Object read()
  - Public void write(Object x)
- Classes and Interfaces can be generic too:
  - public class MyClass<AnyType> {
        public AnyType read() {…}
        ….
- Static methods can be generic too
  - Public static <AnyType> boolean find(AnyType [] a, AnyType x) { … }
- Generic classes/objects have some restrictions (primitive types, and others)

Java SE Conceptual Diagram



Java Language

Tools & Tool APIs (e.g. Javac, Jconsole, Java DB, Web Services)

Deployment (e.g. Java Web Start, Applet and Java Plug-in)

JavaFX

User Interface Toolkits

Integration Libraries (e.g. Scripting, RMI, JDBC)

Other Base Libraries (e.g. Security, Serialization, Networking, Beans, Input/Output, Math, Extension, JMX)

Lang and Util Base Libraries (e.g. Collections, Concurrency, JAR, Logging, Reflection, Versioning, Regular Expressions, Zip)

Java HotSpot Client and Server VM

Java – some facts*

- 10 Million Java Developers Worldwide
- #1 Choice for Developers
- #1 development platform in the cloud
- 5 million students study Java
- 2nd most important programming language for IEEE Spectrum

5 million
students study java

15 billion
devices run Java

10 million
Java developers worldwide

#1 platform
for development in the cloud

*Source: www.java.com – June 2017

Java – more facts

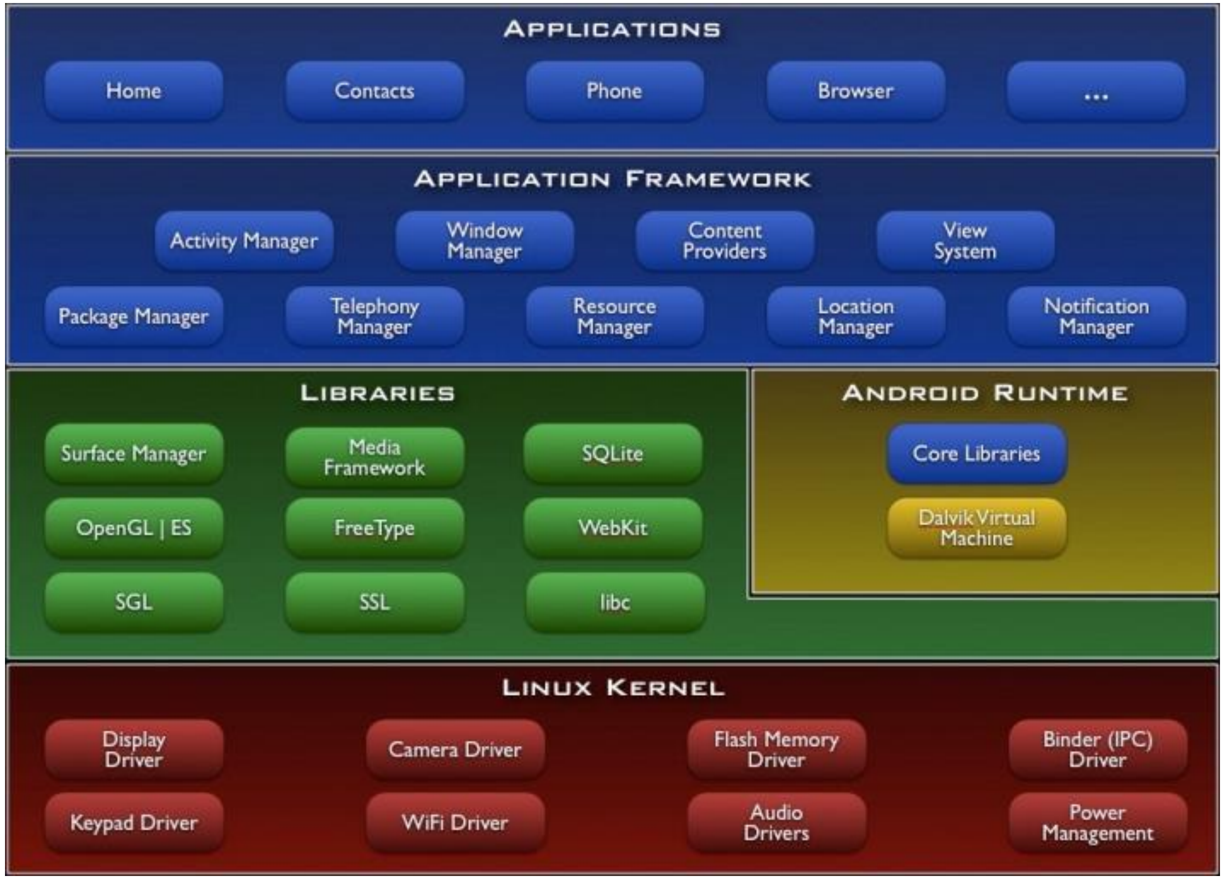
Java installation window

Java Setup - Progress

Status: Installing Java

Java
ORACLE
ATMs, Smartcards, POS Terminals, Blu-ray Players, PCs
Set Top Boxes, Routers, Servers, Switches
Lottery Systems, Access Control Systems, Building Controls
3 Billion
Devices Run Java
Java
#1 Development Platform
ORACLE

Android apps are written in Java



APPLICATIONS

Home | Contacts | Phone | Browser | ...

APPLICATION FRAMEWORK

Activity Manager | Window Manager | Content Providers | View System

Package Manager | Telephony Manager | Resource Manager | Location Manager | Notification Manager

LIBRARIES

Surface Manager | Media Framework | SQLite

OpenGL | ES | FreeType | WebKit

SGL | SSL | libc

ANDROID RUNTIME

Core Libraries

Dalvik Virtual Machine

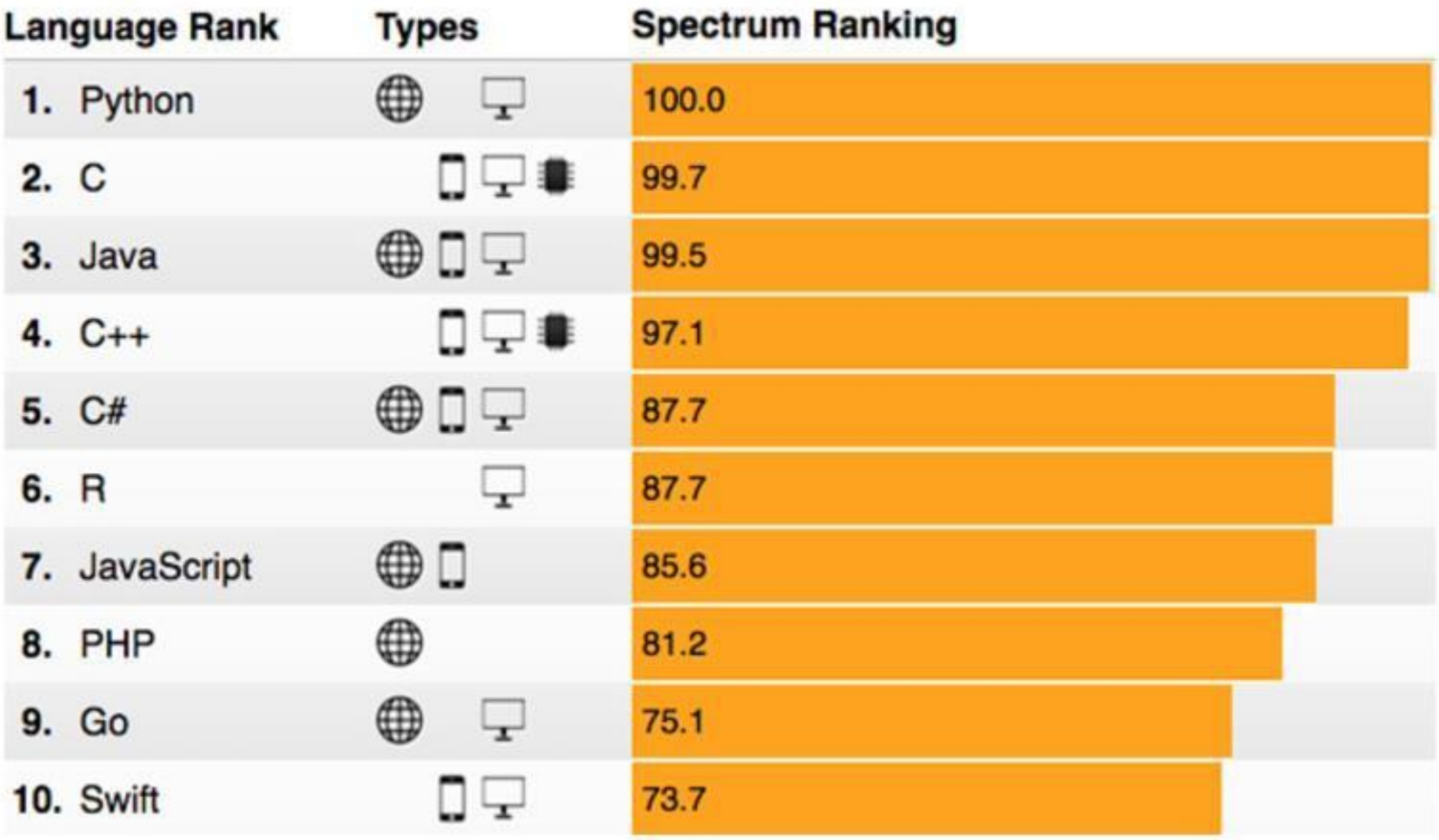
LINUX KERNEL

Display Driver | Camera Driver | Flash Memory Driver | Binder (IPC) Driver

Keypad Driver | WiFi Driver | Audio Drivers | Power Management

Java – even more…

- Java ranked the among the top programming languages of 2017 by IEEE Spectrum



Language Rank	Types	Spectrum Ranking
1. Python	🌐 🖥	100.0
2. C	📱 🖥 ▪	99.7
3. Java	🌐 📱 🖥	99.5
4. C++	📱 🖥 ▪	97.1
5. C#	🌐 📱 🖥	87.7
6. R	🖥	87.7
7. JavaScript	🌐 📱	85.6
8. PHP	🌐	81.2
9. Go	🌐 🖥	75.1
10. Swift	📱 🖥	73.7

http://spectrum.ieee.org/static/interactive-the-top-programming-languages-2017

Java SE and EE 8

- Our implementations will use Java Standard Edition (Java SE 8)
  - http://www.oracle.com/technetwork/java/javase/overview/index.html
- A more advanced edition is the Java Enterprise Edition (Java EE 8)
  - http://www.oracle.com/technetwork/java/javaee/overview/index.html
- Java EE is developed using the Java Community Process
- Includes contributions from experts (industry, commercial, organizations, etc.)
- Releases and new features are aligned with contributors
- Main features:
  - a rich platform, widely used, scalable, low risk, etc.
  - enhances HTML5 support and increase developer productivity
  - Java EE 8 developers have support for latest Web applications and frameworks

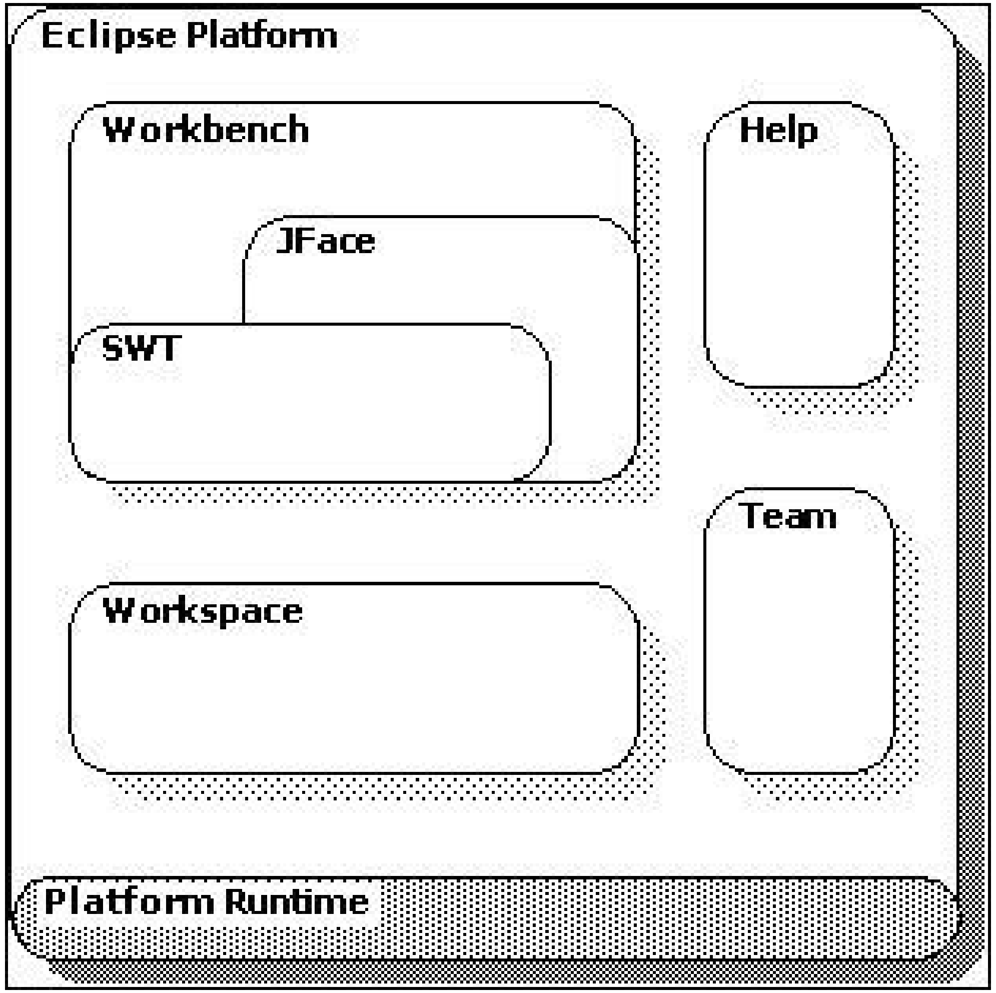
*Image from www.oracle.com

Eclipse

- Main features
  - IDE and Tools
    - Desktop IDEs: Supports Java Integrated Development Environment (IDE), PHP, Java SE/EE, C/C++
    - Web IDEs: Software placed in the cloud, and accessed from anywhere (desktop, laptop or tablet)
  - Community of Projects
    - Can participate and contribute to other projects
  - Collaborative Working Groups
    - It’s an open industry collaboration used to develop new industry platforms
    - Current groups include automotive, location tech, science, long-term support, Internet of Things (IoT), PolarSys (embedded systems)

Eclipse – main platform

- Eclipse platform is structured as a set of components implemented through plug-ins
- Components are built on top of a small runtime engine
- The *Workbench* is the desktop development environment
- It’s an integration tool for creation, management, and navigation of workspace resources



Review and Further Reading

- Math and proofs:
  - Arithmetic and geometric series
  - Summations
  - Logarithms and properties
  - Proofs and justifications
  - Basic probability
  - Section 1.2 of [3] and Sections 1.1, 1.2, 1.3, 1.4 and 1.6 of [1]
  - Appendices A and C of [5].
- Algorithm analysis:
  - Ch. 4 of [2], Ch. 2 of [4], Ch. 3 of [5]
- Java
  - Main sites by Oracle: [7],[8],[9]
- Eclipse
  - Main site: [6]. Basic tutorial: Eclipse > Workbench User Guide > Getting started

References

1. Algorithm Design and Applications by M. Goodrich and R. Tamassia, Wiley, 2015.
2. Data Structures and Algorithms in Java, 6th Edition, by M. Goodrich and R. Tamassia, Wiley, 2014. *(On reserve in the Leddy Library)*
3. Data Structures and Algorithm Analysis in Java, 3rd Edition, by M. Weiss, Addison-Wesley, 2012.
4. Algorithm Design by J. Kleinberg and E. Tardos, Addison-Wesley, 2006.
5. Introduction to Algorithms, 2nd Edition, by T. Cormen et al., McGraw-Hill, 2001.
6. The Eclipse Foundation, http://www.eclipse.org/
7. Java by Oracle: http://www.oracle.com/technetwork/java/index.html
8. Java Standard Edition (Java SE 8) http://www.oracle.com/technetwork/java/javaee/overview/index.html
9. Java Enterprise Edition (Java EE 8) http://www.oracle.com/technetwork/java/javaee/overview/index.html

Lab – Practice

1. Use the class MaxSumTest.java provided in the source code.
   a) Create a new Java project in Eclipse (called "Algorithm Analysis")
   b) Create a new package called "analysis"
   c) Create a new class called MaxSumTest and enter the provided source code.
   d) Run the four algorithms with the example provided in the source code.
   e) Create random sequences of n integers between –n/2 and +3n/2
   f) Run the program of 1.e for n = 2^i , where i = 3, 4, …, 16, and record the CPU time for each algorithm and each value of i
   g) Create a table with the times obtained and discuss these. Compare the CPU times with the complexity (running time) of the algorithms
   h) Plot the CPU times obtained for each algorithm and compare the plots with those of the most important functions used in algorithm analysis (slide 11)

2. *Repeat all items of #1 for the Prefix Averages problem.
   a) In this case, you will have to implement the two algorithms (as they are not provided).

Exercises

1. Sort the functions $12n^2$, $3n$, $0.5 \log n$, $n \log n$, $2n^3$ in increasing order of growth rate.
2. Algorithm A uses $20 n \log n$ operations, while algorithm B uses $2n^2$ operations. Assume all operations take the same time. What is the value of n0 for which A will run faster? Which algorithm will you use if your inputs are of size 10,000?
3. *Prove or disprove that (a) $f(n) = 2n^2 + 3$ is $O(n)$, (b) $f(n)$ is $O(n^{10})$, (c) $f(n)$ is $\Omega(1)$, (d) $f(n)$ is $\omega(1)$, (e) $f(n)$ is $\omega(n^{0.5})$, (f) $f(n)$ $\Theta(n^2)$, (e) $f(n)$ is $o(n^{9/4})$.
4. Consider A = 3, 4, -7, 3, 6, -3, 2, 8, -1. Find the MCCS using the four algorithms discussed in class. *How many operations will the linear-time algorithm use?
5. Find the worst-case running time in O-notation for algorithms linearMCCS and quadraticMCCS. Show all steps used in the calculations. What about Ω and Θ? Also, ω and o?
6. *Implement linearMCCS and divide-and-conquer-MCCS. Run them on several random sequences of size 1,000. Which one is faster? What are the inputs for which the algorithms run faster/slower? Take the average CPU times and compare the with the running times of the algorithms.
7. Are these statements true? (a) If $f(n)$ is $O(g(n))$, then $f(n)$ is $\Omega(g(n))$; (b) if $f(n)$ is $O(g(n))$, then $g(n)$ is $\Omega(f(n))$; (c) if $f(n)$ is $O(g(n))$ and $f(n)$ is $\Omega(g(n))$, then $f(n)$ is $\Theta(g(n))$; (d) if $f(n)$ is $O(g(n))$ and $g(n)$ is $\Omega(f(n))$, then $f(n)$ is $\Theta(g(n))$; (e) if $f(n)$ is $o(g(n))$, then $f(n)$ is $O(g(n))$; (f) if $f(n)$ is $\Omega(g(n))$, then $f(n)$ is $\omega(g(n))$.
8. Give five different reasons for why we use Java in this course.
9. Describe the main features of the Eclipse IDE platform.
10. What is the advantage of using Java Enterprise Edition?
11. Consider A = 3, 4, -7, 3, 6, -3, 2, 8, -1. Can the algorithm linear MCCS be modified to run a little bit faster on this example? Yes/no? why not?
12. Give an example of an algorithm whose best and worst case running times differ in more than a constant (i.e., asymptotically different).
13. *Implement linear search and binary search on an array A. Run a variety experiments on different lists of different sizes. Which algorithm will you use and for which sizes?
14. List the main properties of the rules for O-notation. Do these apply to Ω and Θ? How?
15. What is the difference between problem and algorithm? Explain how a problem can be solved using different algorithms and how they may have different complexities. Give an example.
16. Give examples of functions for the different types of asymptotic notation.