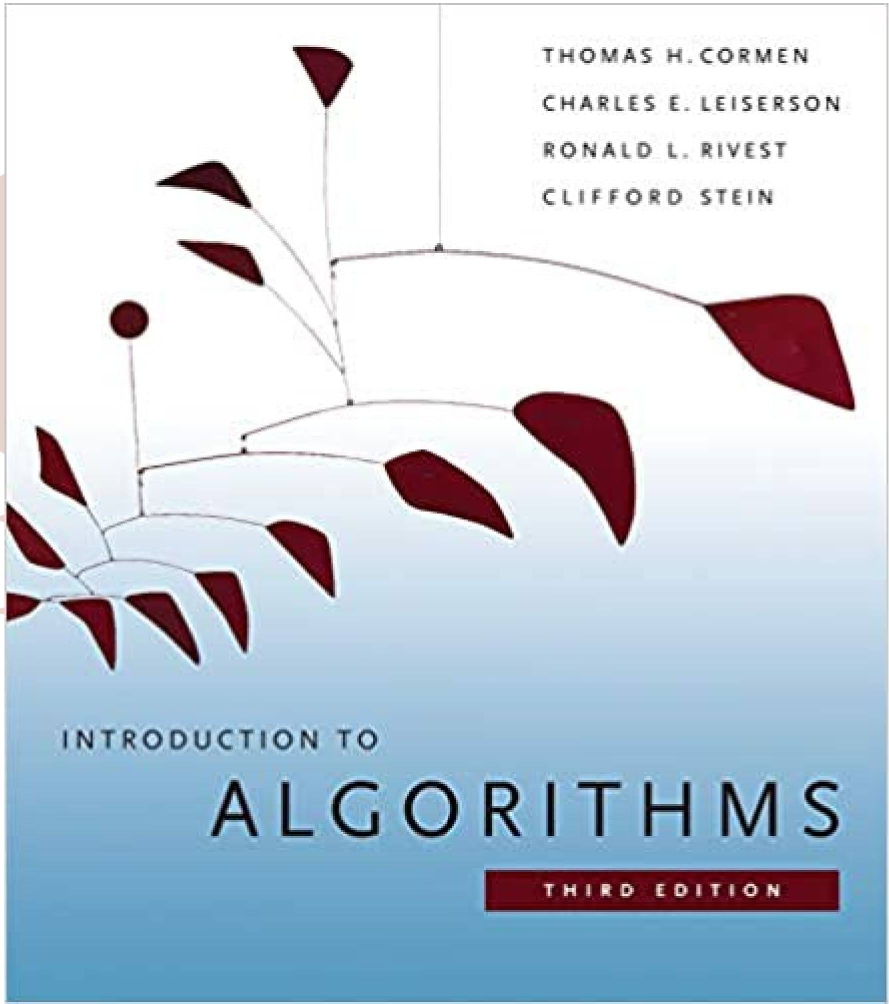
CS 07540 Advanced Design and Analysis of Algorithms

Week 1

- Syllabus and Course Outline
    - Cormen *et al*. Introduction to Algorithms, 3rd edition, MIT Press (2009)
    - Homework Assignment IDEs
- Why ADAA?
    - Golay Complementary Sequences
- Asymptotic Notation
- Homework Assignment on Syllabus



Syllabus

Topics Outline (tentative)

- Data Structures
  - Heaps
  - Binary Search Trees
  - 2-3 trees
  - AVL trees, Red-black trees
  - Binomial Heaps – Binomial Trees
  - Amortized Analysis (potential and accounting methods)
  - Fibonacci Heaps
  - Scapegoat Trees
  - Sequencing of Union Find operations
  - Union/Find tree implementation
  - Union/Find average case analysis

- Searching and Sorting
    - Binary Search
    - Radix Sort
    - Bucket Sort
    - Counting Sort
    - Bubble Sort
    - Merge Sort
    - QuickSort
    - HeapSort
    - Randomized QuickSort
    - Median Finding (Selection) – information theoretic lower bound

- Parallel Algorithms
    - PRAM, CREW, CRCW, EREW
    - Prefix Sums
    - Accelerated Cascading
- Graph Theory
    - Minimal Spanning Tree
        - Kruskal’s Algorithm, Prim’s Algorithm
    - Shortest Path
        - Dijkstra's algorithm
        - Floyd-Warshall algorithm

- NP Completeness
  - Definitions
  - NP Completeness and NP Hardness
  - Classic Problems
  - Reductions
- Other
  - Run-time analysis
  - Transitive Closure of Boolean Matrix
    - Strassen algorithm
  - Interpolations
  - Fourier Transforms
  - Integer Multiplication
    - Schönhage–Strassen algorithm
  - Polynomial Multiplication

Academic Integrity Policy

I) Purpose

The purpose of the academic integrity policy is to provide students, faculty, and staff with guidelines about what behaviors violate academic integrity expectations, and the process for addressing academic integrity problems.

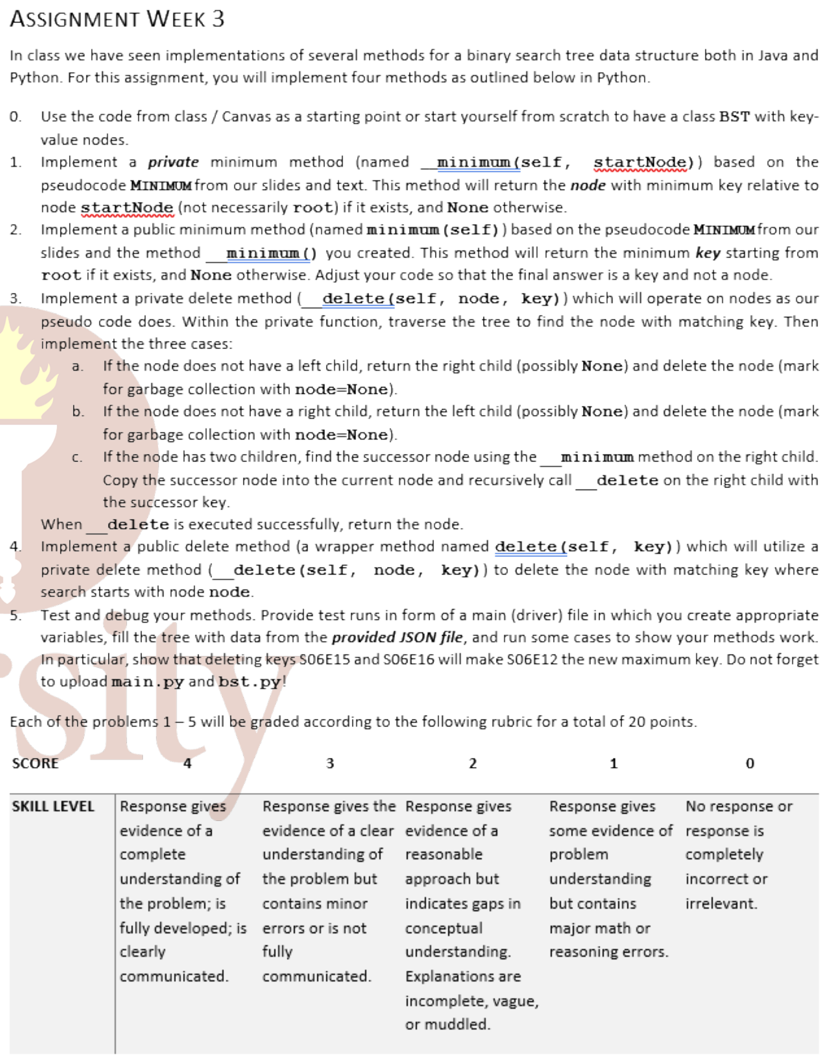
V) Policy

4) Violations of academic integrity are classified into four Levels based on the seriousness of the behaviors and the possible sanctions imposed.

c) Level 3 offenses are even more serious in nature and involve dishonesty on a more significant portion of course work, such as significant portions of a major paper, hourly or final examination. If a student had previously been found responsible either of one or more violations at Level 2 or higher, or of two Level 1 violations, an additional violation at any level will automatically become at least a Level 3 violation. A sanction for a level 3 violation will not exceed failure for the course and Academic Integrity Probation. **Example: Copying from or giving assistance to others on an hourly or final examination, plagiarizing major portions of an assignment, using forbidden material on an hourly or final examination, presenting the work of another as one's own, or altering a graded examination for the purposes of re-grading.**

Programming Homework Assignments

We will have several programming homework assignments throughout the semester. Assignments will spell out what functionality you are to implement and which programming language to use: Python or Java. In addition, all submissions **must** work without adjustments in PyCharm and IntelliJ Idea, respectively. Both are provided free of charge as community editions by then JetBrains. So, **for homework, class files and driver/main must work in PyCharm or IntelliJ IDE.**



ASSIGNMENT WEEK 3

In class we have seen implementations of several methods for a binary search tree data structure both in Java and Python. For this assignment, you will implement four methods as outlined below in Python.

0. Use the code from class / Canvas as a starting point or start yourself from scratch to have a class `BST` with key-value nodes.
1. Implement a ***private*** minimum method (named `__minimum(self, startNode)`) based on the pseudocode MINIMUM from our slides and text. This method will return the ***node*** with minimum key relative to node `startNode` (not necessarily `root`) if it exists, and `None` otherwise.
2. Implement a public minimum method (named `minimum(self)`) based on the pseudocode MINIMUM from our slides and the method `__minimum()` you created. This method will return the minimum ***key*** starting from `root` if it exists, and `None` otherwise. Adjust your code so that the final answer is a key and not a node.
3. Implement a private delete method (`__delete(self, node, key)`) which will operate on nodes as our pseudo code does. Within the private function, traverse the tree to find the node with matching key. Then implement the three cases:
   a. If the node does not have a left child, return the right child (possibly `None`) and delete the node (mark for garbage collection with `node=None`).
   b. If the node does not have a right child, return the left child (possibly `None`) and delete the node (mark for garbage collection with `node=None`).
   c. If the node has two children, find the successor node using the `__minimum` method on the right child. Copy the successor node into the current node and recursively call `__delete` on the right child with the successor key.

   When `__delete` is executed successfully, return the node.
4. Implement a public delete method (a wrapper method named `delete(self, key)`) which will utilize a private delete method (`__delete(self, node, key)`) to delete the node with matching key where search starts with node `node`.
5. Test and debug your methods. Provide test runs in form of a main (driver) file in which you create appropriate variables, fill the tree with data from the ***provided JSON file***, and run some cases to show your methods work. In particular, show that deleting keys S06E15 and S06E16 will make S06E12 the new maximum key. Do not forget to upload `main.py` and `bst.py`!

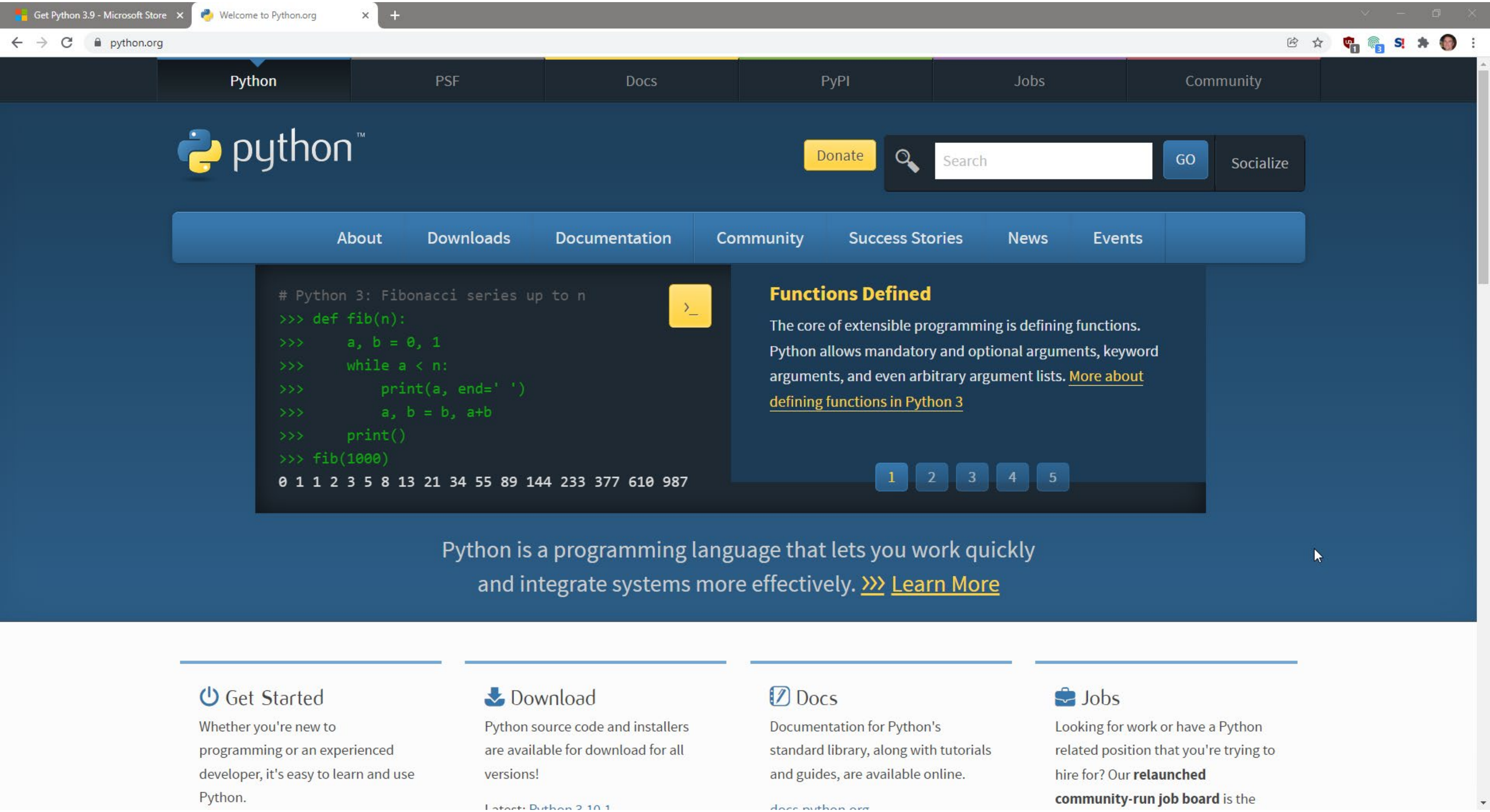
Each of the problems 1 – 5 will be graded according to the following rubric for a total of 20 points.

SCORE	4	3	2	1	0
SKILL LEVEL	Response gives evidence of a complete understanding of the problem; is fully developed; is clearly communicated.	Response gives the evidence of a clear understanding of the problem but contains minor errors or is not fully communicated.	Response gives evidence of a reasonable approach but indicates gaps in conceptual understanding. Explanations are incomplete, vague, or muddled.	Response gives some evidence of problem understanding but contains major math or reasoning errors.	No response or response is completely incorrect or irrelevant.

Python

Python is maintained by a large community of volunteers. As free open-source software, changing and redistributing Python is possible. There is a standard repository, and it is a good idea to stick to a standard. The following steps show how this is done on a Windows machine using Python's web site. It will guess your operating system from your browser information. Web browsers are quite chatty.

- How to use Python on Windows 🕮
- How to use Python on macOS 🕮
- How to use Python through Rowan University's Virtual Desktop 🕮



```
# Python 3: Fibonacci series up to n
>>> def fib(n):
>>>     a, b = 0, 1
>>>     while a < n:
>>>         print(a, end=' ')
>>>         a, b = b, a+b
>>>     print()
>>> fib(1000)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
```

Functions Defined

The core of extensible programming is defining functions. Python allows mandatory and optional arguments, keyword arguments, and even arbitrary argument lists. More about defining functions in Python 3

Python is a programming language that lets you work quickly and integrate systems more effectively. >>> Learn More

Get Started

Whether you're new to programming or an experienced developer, it's easy to learn and use Python.

Download

Python source code and installers are available for download for all versions!

Latest: Python 3.10.1

Docs

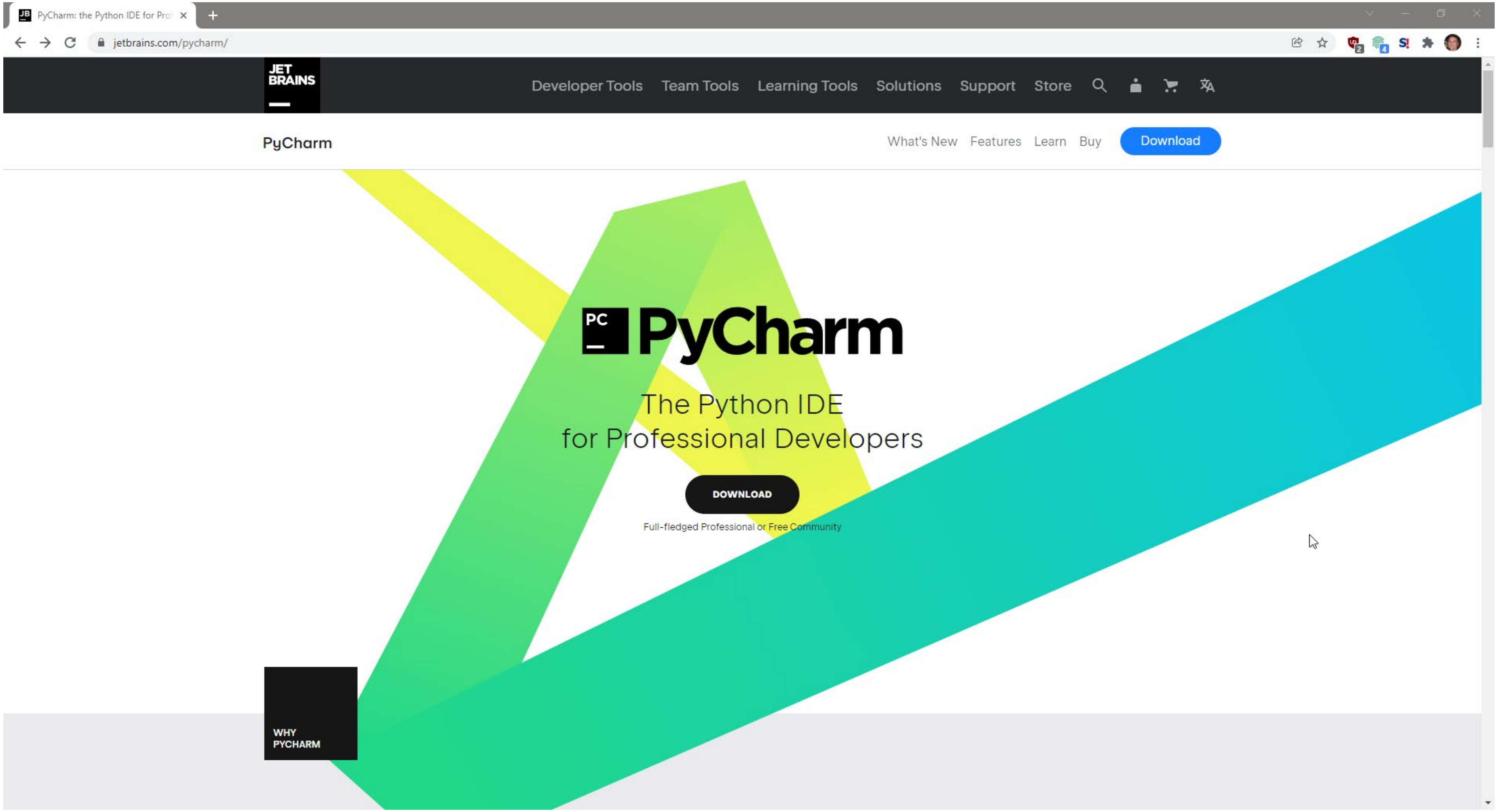
Documentation for Python's standard library, along with tutorials and guides, are available online.

docs.python.org

Jobs

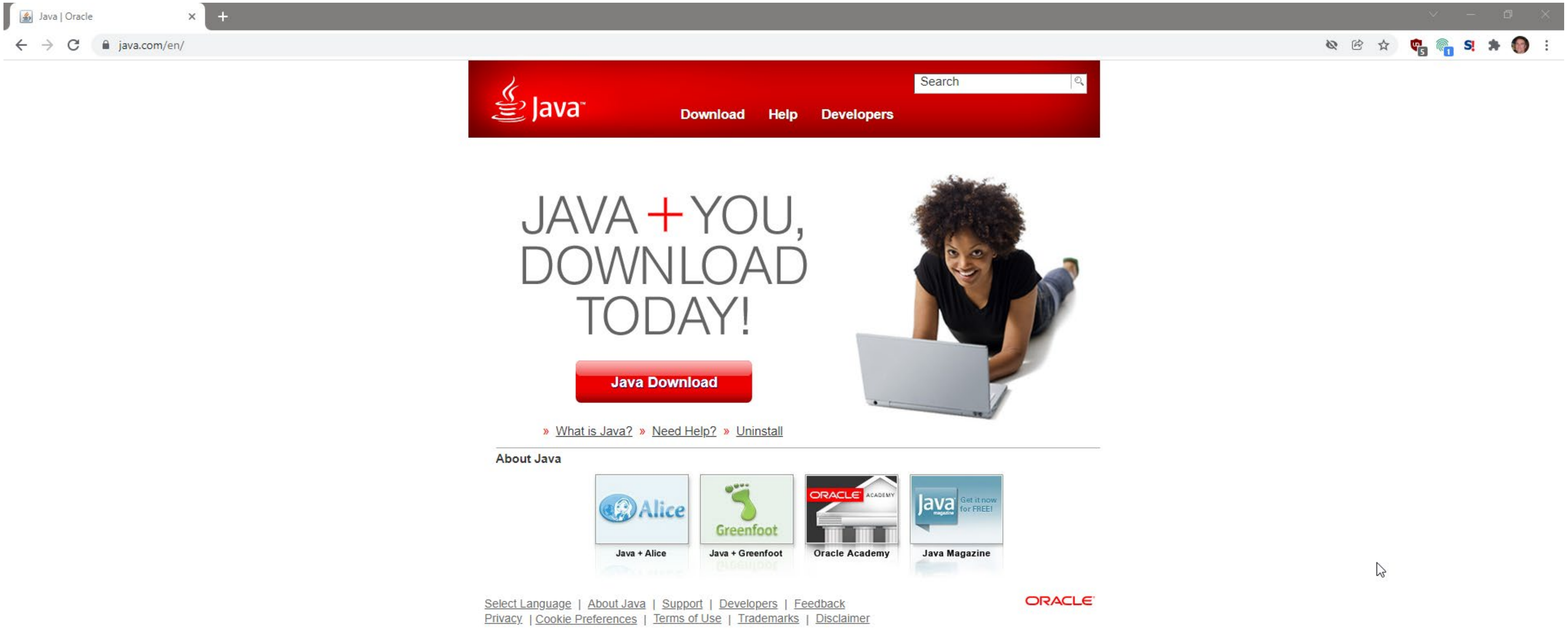
Looking for work or have a Python related position that you're trying to hire for? Our **relaunched community-run job board** is the

PyCharm will recognize your existing Python installation and use it as its interpreter. If you want to change the appearance of PyCharm, use **`Ctrl-Alt-S`** (Settings) and go to Editor → Color Scheme → General.


PyCharm: the Python IDE for Pro
jetbrains.com/pycharm/
JET BRAINS
Developer Tools
Team Tools
Learning Tools
Solutions
Support
Store
PyCharm
What's New
Features
Learn
Buy
Download
PC
PyCharm
The Python IDE
for Professional Developers
DOWNLOAD
Full-fledged Professional or Free Community
WHY PYCHARM

Java

Java is **not** free open-source software, so our first step is to install Java from Oracle. Technically, we need the Java Runtime Environment version 8 (JRE) with the Java Virtual Machine (JVM). Make sure you do not have (too many) competing Java versions installed. We will assume a clean installation. In order to create applications, we will also need a Java Development Kit (JDK) which Oracle or IntelliJ will provide. **For Homework, submit class files and driver/main from the PyCharm or IntelliJ IDE.**


Java | Oracle

java.com/en/

Search

Download | Help | Developers

JAVA + YOU, DOWNLOAD TODAY!

Java Download

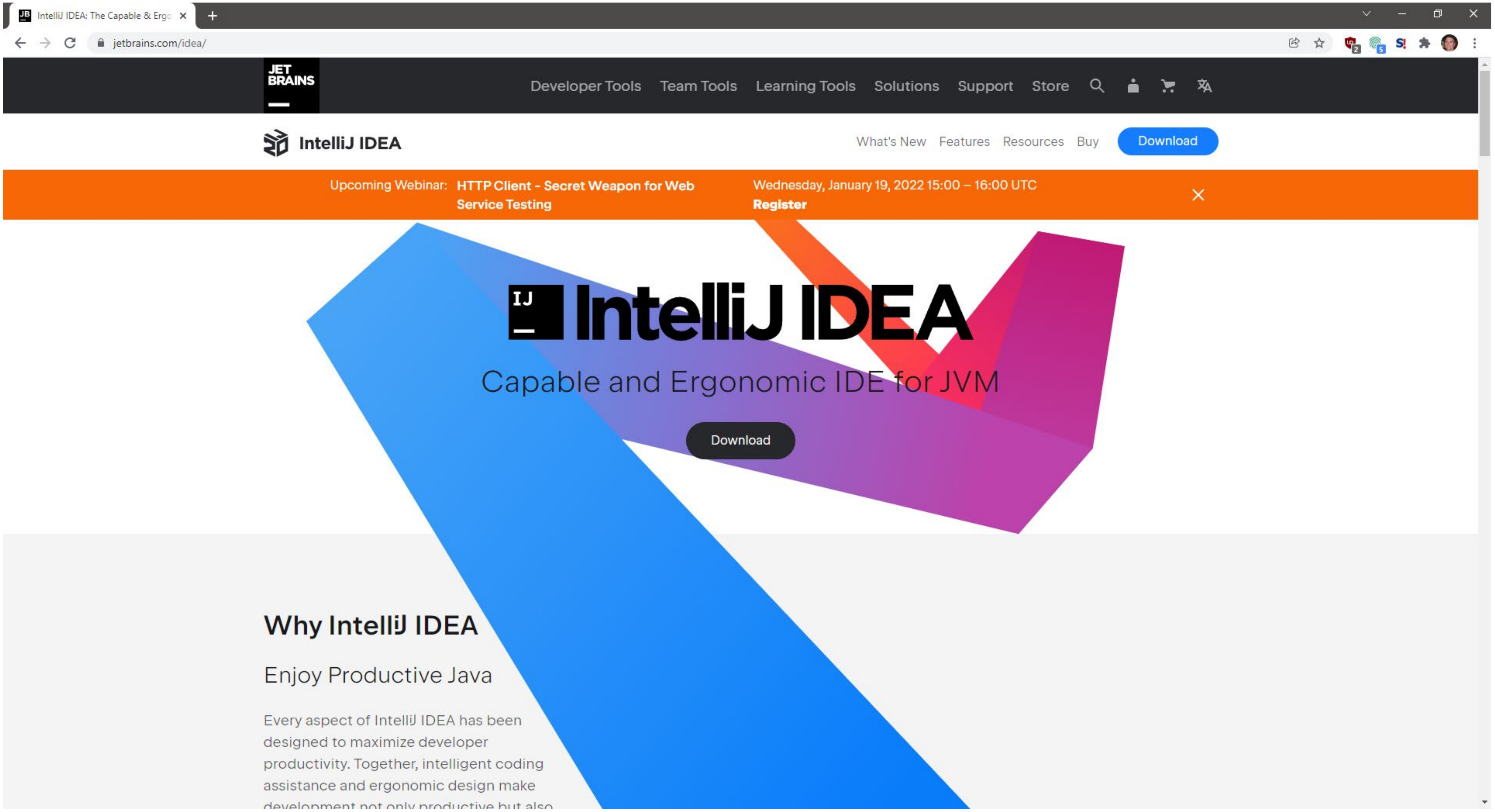
» What is Java? » Need Help? » Uninstall

About Java

Java + Alice | Java + Greenfoot | Oracle Academy | Java Magazine

Select Language | About Java | Support | Developers | Feedback
Privacy | Cookie Preferences | Terms of Use | Trademarks | Disclaimer

ORACLE



Upcoming Webinar: **HTTP Client - Secret Weapon for Web Service Testing** Wednesday, January 19, 2022 15:00 – 16:00 UTC **Register**

IntelliJ IDEA

Capable and Ergonomic IDE for JVM

Why IntelliJ IDEA

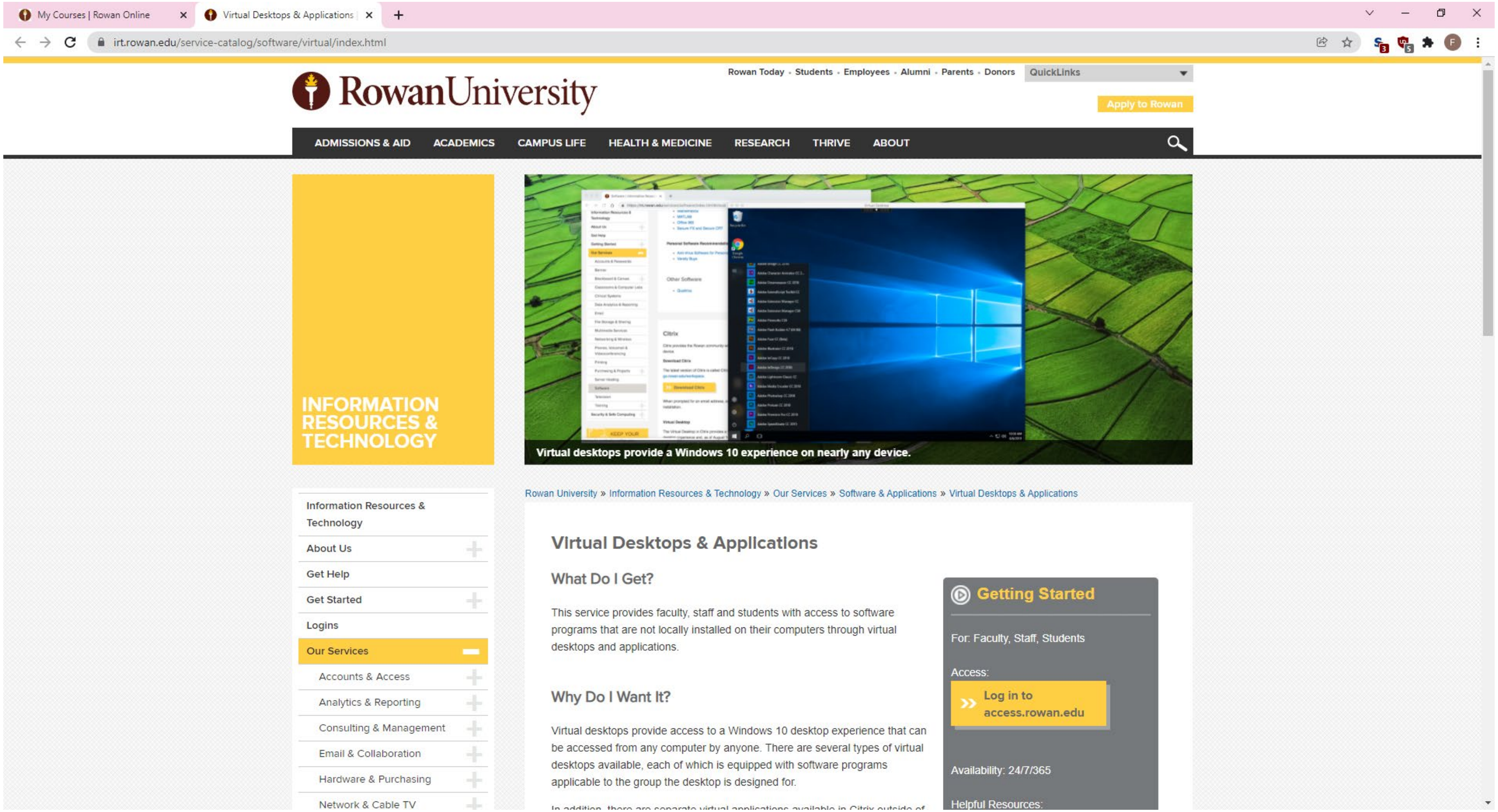
Enjoy Productive Java

Every aspect of IntelliJ IDEA has been designed to maximize developer productivity. Together, intelligent coding assistance and ergonomic design make development not only productive but also

Rowan University Virtual Desktop

Faculty, staff, and students at Rowan University have access to a virtual (windows) desktop that can be accessed from anywhere. It uses Citrix Workspace, so you need to install that on your device. There are versions for

- Windows
- Mac (macOS)
- iOS
- Linux
- Android
- Chrome
- HTML5



My Courses | Rowan Online

Virtual Desktops & Applications |

irt.rowan.edu/service-catalog/software/virtual/index.html

Rowan Today · Students · Employees · Alumni · Parents · Donors

QuickLinks

RowanUniversity

Apply to Rowan

ADMISSIONS & AID ACADEMICS CAMPUS LIFE HEALTH & MEDICINE RESEARCH THRIVE ABOUT

INFORMATION RESOURCES & TECHNOLOGY

Virtual desktops provide a Windows 10 experience on nearly any device.

Rowan University » Information Resources & Technology » Our Services » Software & Applications » Virtual Desktops & Applications

- Information Resources & Technology
- About Us
- Get Help
- Get Started
- Logins
- Our Services
  - Accounts & Access
  - Analytics & Reporting
  - Consulting & Management
  - Email & Collaboration
  - Hardware & Purchasing
  - Network & Cable TV

Virtual Desktops & Applications

What Do I Get?

This service provides faculty, staff and students with access to software programs that are not locally installed on their computers through virtual desktops and applications.

Why Do I Want It?

Virtual desktops provide access to a Windows 10 desktop experience that can be accessed from any computer by anyone. There are several types of virtual desktops available, each of which is equipped with software programs applicable to the group the desktop is designed for.

Getting Started

For: Faculty, Staff, Students

Access:

Log in to access.rowan.edu

Availability: 24/7/365

Helpful Resources:

You should have a working environment for our course now.

Degreeworks

Track your progress with Degreeworks.

Why ADAA?

This course is a graduate level treatment for solving various problems efficiently. Proper algorithm and resource analysis can make the difference between a program completing its task in minutes or months. Eventually you are expected to look at a problem and

- identify proper algorithms to solve the problem at hand
- identify potential bottlenecks
- improve performance by targeted optimization (both in runtime and resources).

> *… an algorithm is a finite sequence of mathematically rigorous instructions, typically used to solve a class of specific problems or to perform a computation. Algorithms are used as specifications for performing calculations and data processing. (📖 Wikipedia)*

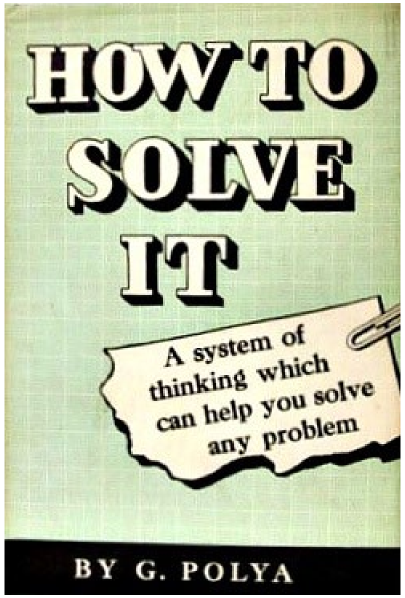
In this course, we want to analyze runtime behavior of some often used and important programs/algorithms.

> *In computer science, the analysis of algorithms is the process of finding the computational complexity of algorithms – the amount of time, storage, or other resources needed to execute them. (📖 Wikipedia)*

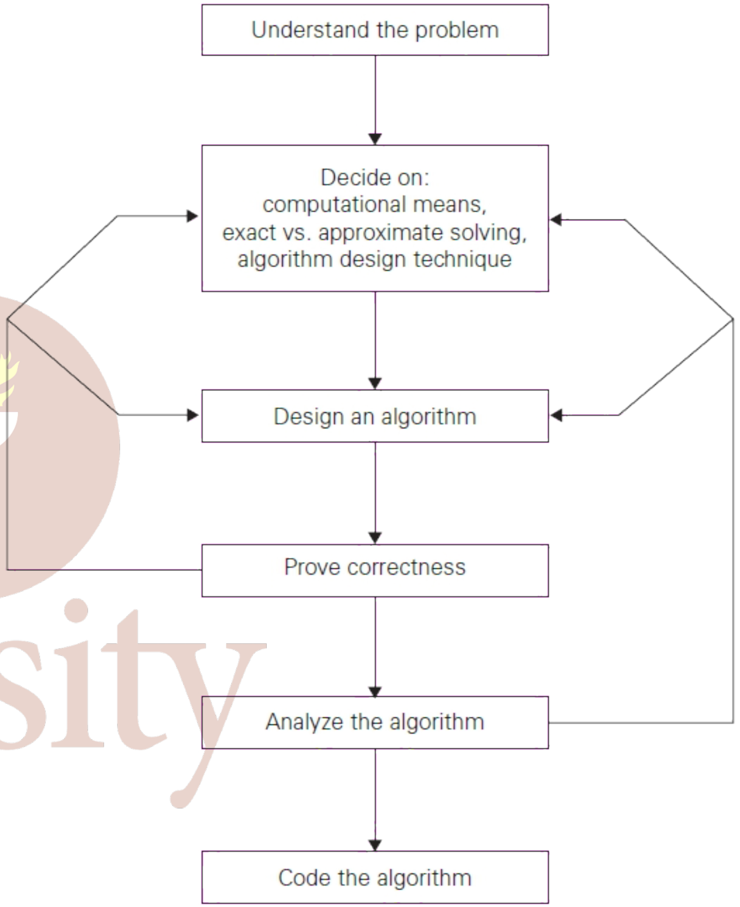
Donald Knuth's The Art of Computer Programming is one of the most important modern works in Mathematics and Computer Science. In Volume 1 Section 1.1, he writes:

> *The modern meaning for algorithm is quite similar to that of recipe, process, method, technique, procedure, routine, rigmarole, except that the word "algorithm" connotes something just a little different. Besides merely being a finite set of rules that gives a sequence of operations for solving a specific type of problem, an algorithm has five important features:*

1) *Finiteness. An algorithm must always terminate after a finite number of steps.*
2) *Definiteness. Each step of an algorithm must be precisely defined; the actions to be carried out must be rigorously and unambiguously specified for each case.*
3) *Input. An algorithm has zero or more inputs: quantities that are given to it initially before the algorithm begins, or dynamically as the algorithm runs. These inputs are taken from specified sets of objects.*
4) *Output. An algorithm has one or more outputs: quantities that have a specified relation to the inputs.*
5) *Effectiveness. An algorithm is also generally expected to be effective, in the sense that its operations must all be sufficiently basic that they can in principle be done exactly and in a finite length of time by someone using pencil and paper.*



A variation of George Pólya's "How to Solve It" (1945) (a four-step method for solving problems) gives us a roadmap for our design and analysis of algorithms. Note that designing an algorithm often comes with the necessity for designing data structures that go with it.



We will start with an actual research problem as motivation.

Enumerating Golay Complementary Sequences

Enumerate all Golay Complementary Sequences for a given length n and given number of phase shifts H.

(For practical applications, $n = 2^m$ and $H = 2^h$ are used.)

Whenever we embark on a journey to solve a problem, we should ask:

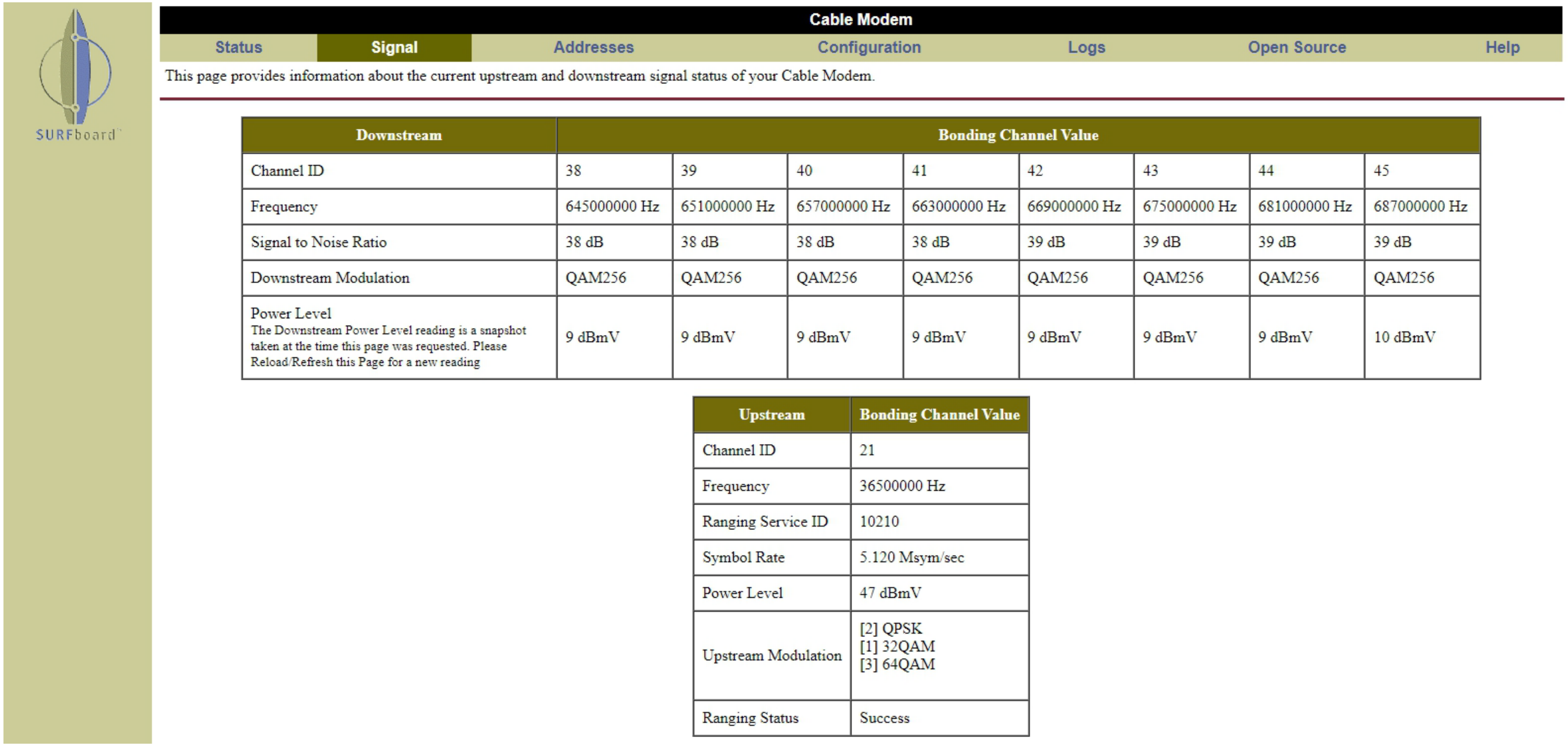
- What exactly is the object we are studying?
- What exactly are we trying to do?
- Why do we care?

(Dr. Jonathan Jedwab, Simon Fraser University)

IEEE 802.11 is part of the IEEE 802 set of local area network (LAN) technical standards, and specifies the set of media access control (MAC) and physical layer (PHY) protocols for implementing wireless local area network (WLAN) computer communication. The standard and amendments provide the basis for wireless network products using the Wi-Fi brand and are the world's most widely used wireless computer networking standards.

Why Do We Care?

IEEE 802.11 standards form a family of standards which started being published in 1997 and is evolving. Many of these standards involve a technique called OFDM (orthogonal frequency-division multiplexing). Digital data streams are encoded as multiple (usually power of 2) parallel lower rate data substreams, each of which is used to modulate (equally spaced) subcarriers using PSK (phase-shift keying). Variations of these techniques may go by the names of BPSK, QPSK, QAM64 etc. This involves many interesting problems (handshakes, synchronization, Doppler-effect, encoding, decoding, error-correction, etc.).



SURFboard

Cable Modem

Status | Signal | Addresses | Configuration | Logs | Open Source | Help

This page provides information about the current upstream and downstream signal status of your Cable Modem.

Downstream	Bonding Channel Value							
Channel ID	38	39	40	41	42	43	44	45
Frequency	645000000 Hz	651000000 Hz	657000000 Hz	663000000 Hz	669000000 Hz	675000000 Hz	681000000 Hz	687000000 Hz
Signal to Noise Ratio	38 dB	38 dB	38 dB	38 dB	39 dB	39 dB	39 dB	39 dB
Downstream Modulation	QAM256	QAM256	QAM256	QAM256	QAM256	QAM256	QAM256	QAM256
Power Level The Downstream Power Level reading is a snapshot taken at the time this page was requested. Please Reload/Refresh this Page for a new reading	9 dBmV	9 dBmV	9 dBmV	9 dBmV	9 dBmV	9 dBmV	9 dBmV	10 dBmV

Upstream	Bonding Channel Value
Channel ID	21
Frequency	36500000 Hz
Ranging Service ID	10210
Symbol Rate	5.120 Msym/sec
Power Level	47 dBmV
Upstream Modulation	[2] QPSK [1] 32QAM [3] 64QAM
Ranging Status	Success

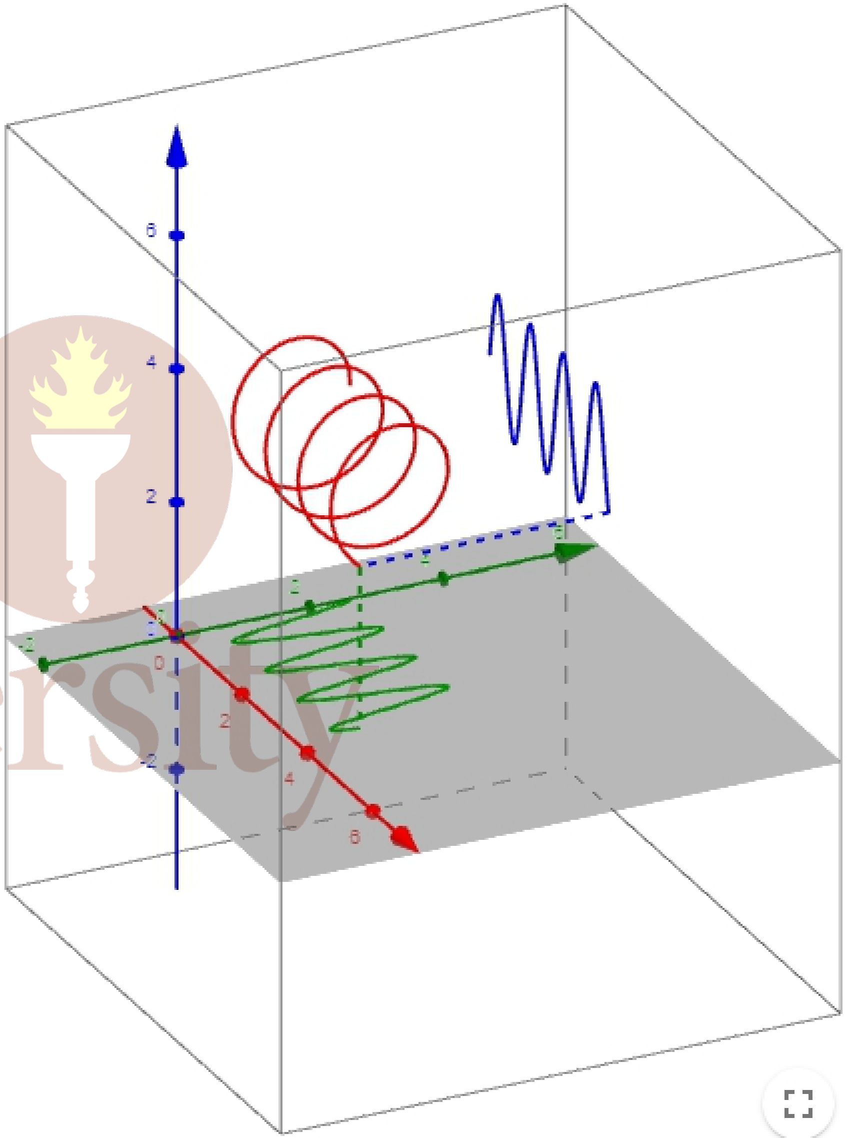
OFDM is a widely used transmission method since it has some major benefits:

- more resistant to frequency selective fading than single carrier
- interference may not affect all channels, and subchannel errors could possibly be corrected with error-correcting codes used for transmission
- efficient bandwidth use
- simple implementation using (Fast) Fourier Transforms
- low sensitivity to time synchronization error

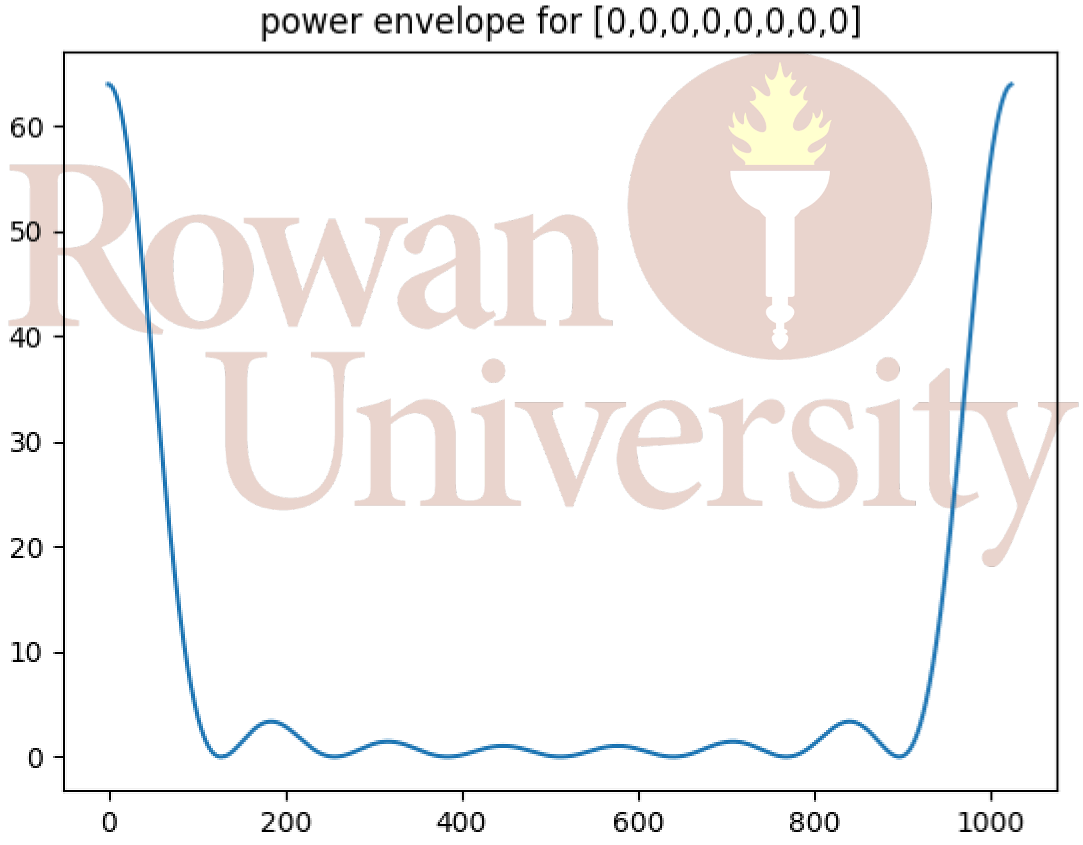
One of the main disadvantages is

- high peak to average power ratio

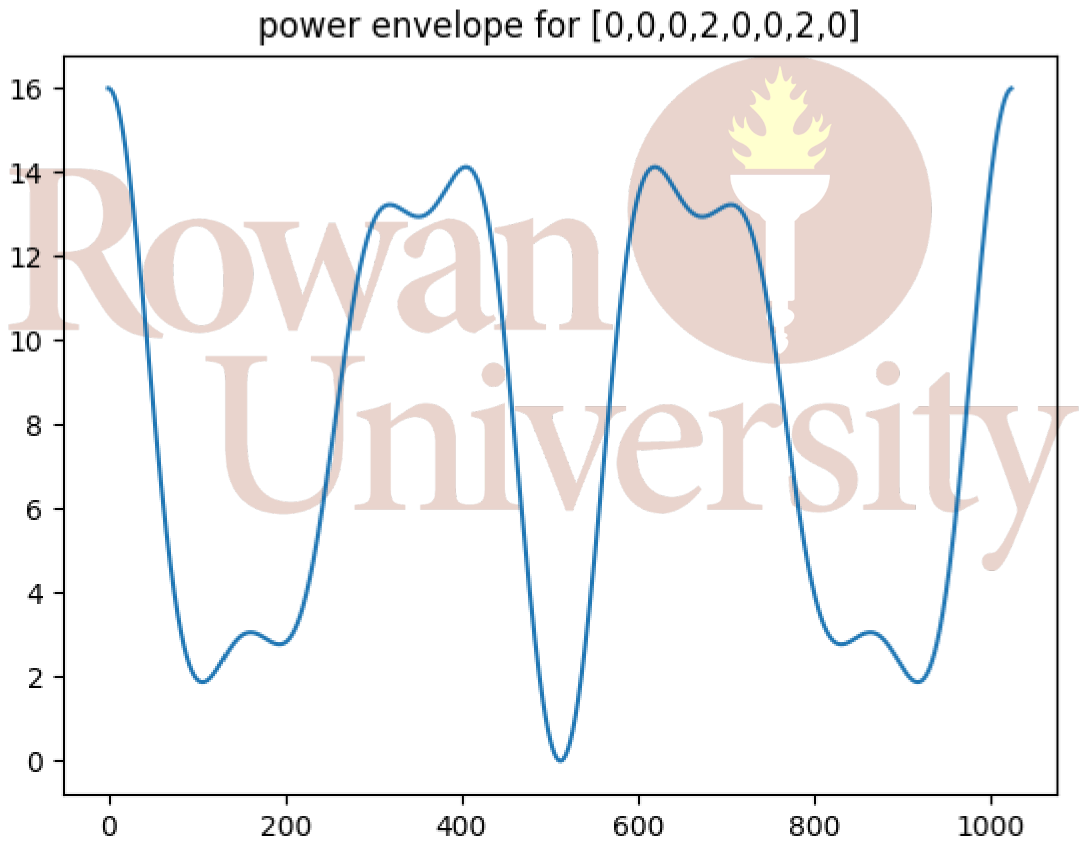
Recall that a unimodular signal is transmitted as a complex envelope $S(t) = e^{j\omega t+\varphi}$. We see the real part as cosine $\cos(\omega t + \varphi)$, but there is an imaginary part $\sin(\omega t + \varphi)$ that needs to be considered when calculating the power.



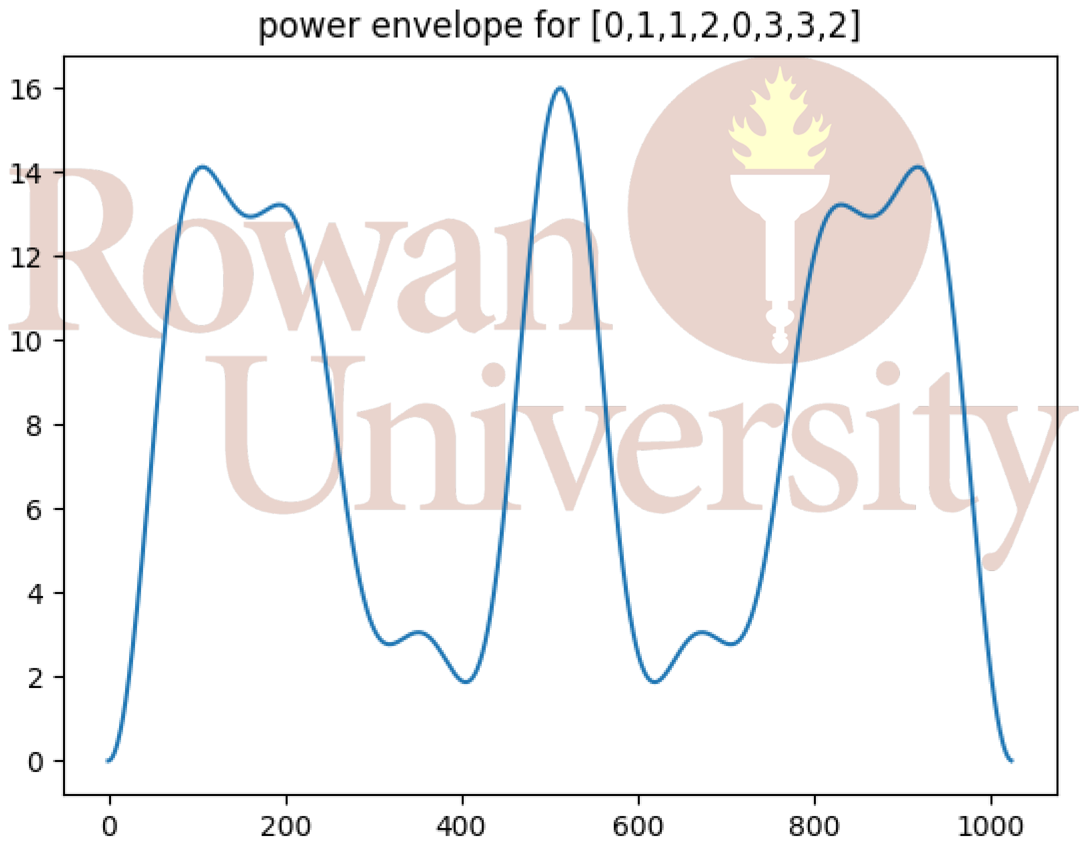
Transmitting unimodular cosine waves on 8 subchannels ($\cos(\omega_k t) + j\sin(\omega_k t)$ for $i = 0 \ldots 7$) may result in a ***power envelope*** of $8 \cdot 8$ units.



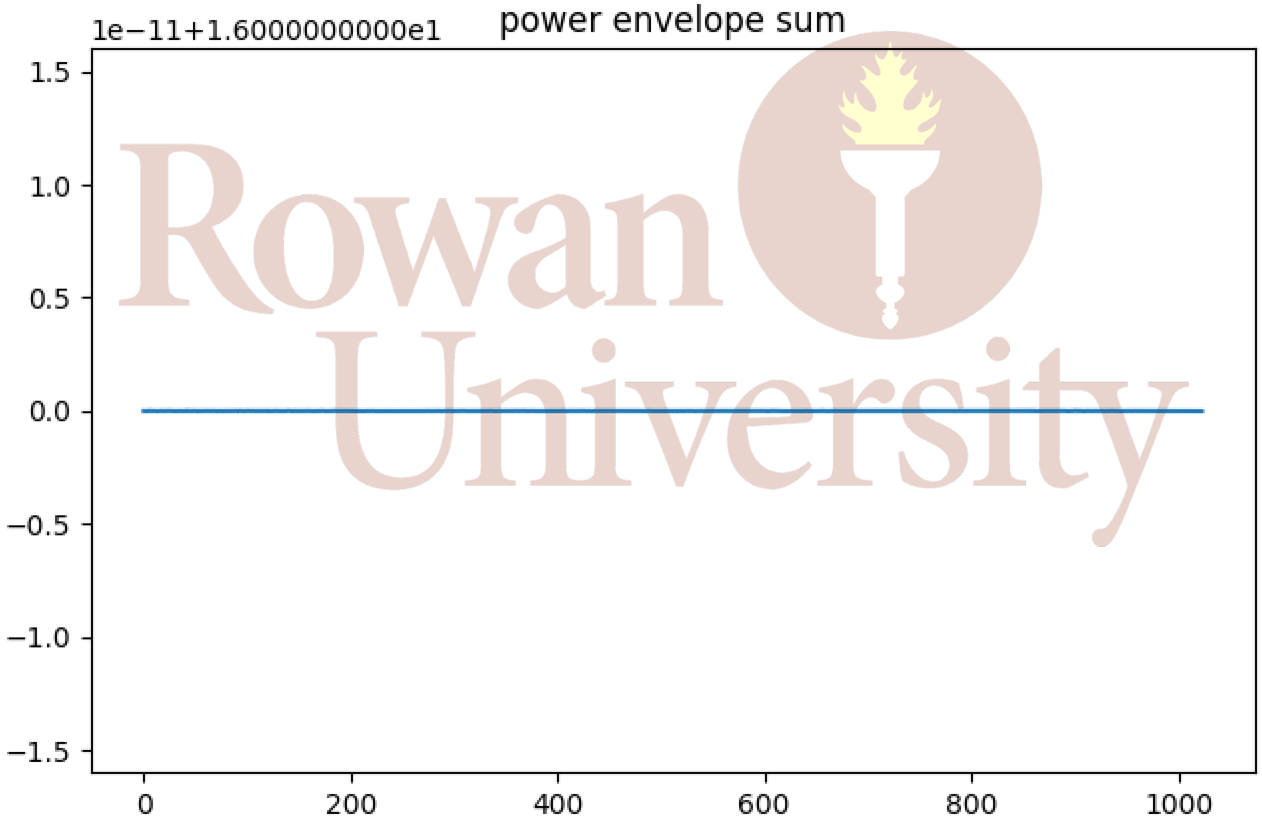
If the signals are chosen carefully, we can get away with a lower power demand of twice the channels only ($\cos(\omega_k t + \varphi_k) + j\sin(\omega_k t + \varphi_k)$):



Here is another example. Interesting power envelope compared to the previous example.

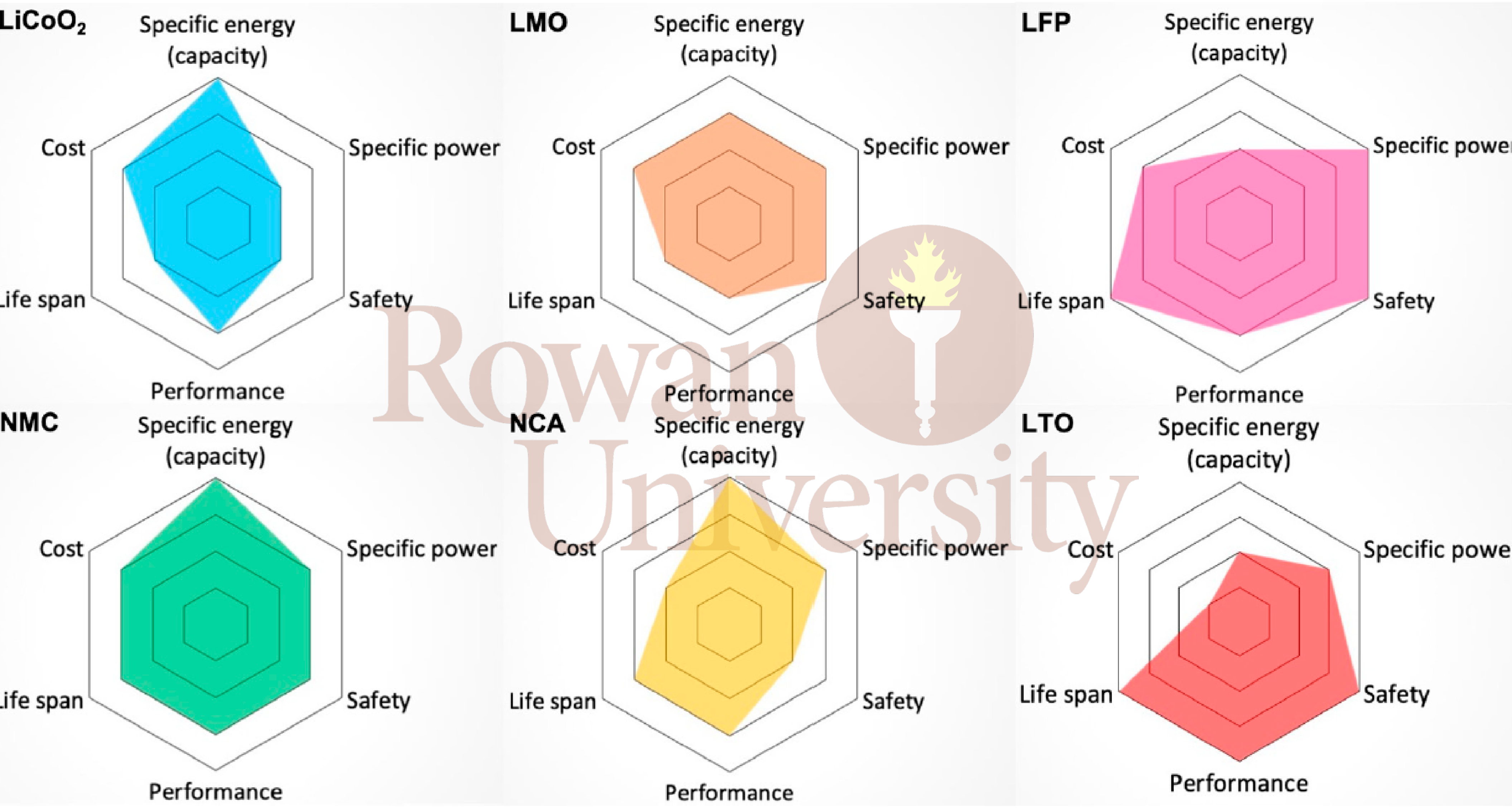


The sum of these two power envelopes is 16 at all times. That means, each individual power envelope is limited to twice the average power.



When peak power is limited compared to the average (the latter being governed by the number of channels), we can use batteries which do not need highest performance. They tend to be much cheaper then high-performance batteries. While not phone batteries, EV batteries offer an example for price-performance trade-offs:

- Cobalt Oxide (LCO)
- Lithium Manganese Oxide (LMO)
- Lithium Iron Phosphate (LFP)
- Lithium Nickel Manganese Cobalt Oxide (NMC)
- Lithium Nickel Cobalt Aluminium Oxide (NCA)
- Lithium Titanate (LTO)


$LiCoO_2$
Specific energy (capacity)
Specific power
Safety
Performance
ife span
Cost
LMO
Specific energy (capacity)
Specific power
Safety
Performance
Life span
Cost
LFP
Specific energy (capacity)
Specific powe
Safety
Performance
Life span
Cost
NMC
Specific energy (capacity)
Specific power
Safety
Performance
ife span
Cost
NCA
Specific energy (capacity)
Specific power
Safety
Performance
Life span
Cost
LTO
Specific energy (capacity)
Specific powe
Safety
Performance
Life span
Cost

The Object

Golay complementary sequences are pairs of complex-valued sequences such that their out-of-phase aperiodic autocorrelation coefficients sum to zero.

A complex value represents a magnitude and a direction.

- Magnitude is amplitude
- Direction is phase shift
- Position within the sequence is subchannel

We will focus on unimodular (same magnitude) systems. Then we can move from complex-valued sequences to unsigned integer data symbol sequences.

Using Golay Complementary Sequences in OFDM systems provides the following advantages, among others:

- Power envelope is limited to twice number of subchannels used (Budišin 1991)
- Standard Golay sequences are cosets of error-correcting Reed-Muller codes (Davis Jedwab 1999)
- Standard Golay Sequences can be constructed from a single seed pair of arrays (Jedwab Parker 2010)
- All known polyphase Golay sequences can be constructed from a small number of seed pairs of arrays (Fiedler 2013)

The quaternary (*i.e.*, base 4) phase shift sequence $A = [0,1,1,2,0,3,3,2]$ corresponds to a signal

$$\begin{aligned}
⬚ \quad & \cos\left(\omega_0 t + {}^{2\pi j}/_{4} \cdot 0\right) + j \cdot \sin\left(\omega_0 t + {}^{2\pi j}/_{4} \cdot 0\right) \\
+ \quad & \cos\left(\omega_1 t + {}^{2\pi j}/_{4} \cdot 1\right) + j \cdot \sin\left(\omega_1 t + {}^{2\pi j}/_{4} \cdot 1\right) \\
⬚ \quad & \qquad\qquad\qquad\qquad \vdots \\
+ \quad & \cos\left(\omega_6 t + {}^{2\pi j}/_{4} \cdot 3\right) + j \cdot \sin\left(\omega_6 t + {}^{2\pi j}/_{4} \cdot 3\right) \\
+ \quad & \cos\left(\omega_7 t + {}^{2\pi j}/_{4} \cdot 2\right) + j \cdot \sin\left(\omega_7 t + {}^{2\pi j}/_{4} \cdot 2\right) \\
= \quad & \sum_{k=0}^{7} e^{j\omega_k t + {}^{2\pi j}/_{4} \cdot a_k}
\end{aligned}$$

The frequencies ω_k are equally spaced starting with a base frequency $\omega = \omega_0$. The data creates phase shifts $\varphi_k = {}^{2\pi j}/_{4} \cdot a_k$. If we denote this signal as $S(t)$, then

$$
\begin{aligned}
|S(t)|^2 &= \left(e^{j\omega}\sum_k e^{j(2\pi k\Delta f+\varphi_k)}\right)\left(\overline{e^{j\omega}\sum_k e^{j(2\pi k\Delta f+\varphi_k)}}\right) \\
\square &= \sum_{k_1}\sum_{k_2} e^{j(2\pi k_1\Delta f+\varphi_{k_1})-j(2\pi k_2\Delta f+\varphi_{k_2})} \\
\square &= \sum_{u=k_1-k_2} e^{j(2\pi u\Delta f)} \sum_{k=k_1} e^{j(\varphi_k-\varphi_{k-u})}
\end{aligned}
$$

The inner sum is called autocorrelation of the signal and does not depend on time or frequencies. For a given signal it can be calculated in $\Theta(n^2)$. Each term in the outher sum does not depend on the actual signal, so we can precalculate it for any signal we may want to investigate. The outer sum itself then turns into an inner product (dot-product). If this inner product can be parallelized, we can get the power envelop in $\Theta(n^2 + m)$ where m is the number of samples.

The Exact Object

For a given length n and a given (even) modulus H, construct all unsigned integer sequences $A = [a_0, a_1, \dots, a_{n-1}]$ of length n and modulus H such that there exists a sequence $B = [b_0, b_1, \dots, b_{n-1}]$ which cancels the out-of-phase autocorrelations of A (for $u \neq 0$). If we call $\xi = e^{2\pi j/H}$ then the autocorrelation function for A is

$$C_A(u) = \sum_{k=0}^{n-1-u} \xi^{(a_k - a_{k+u})}$$

Complex multiplication can be replaced with efficient look-up tables for small H.

	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32	34	36
1	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32	34	36
2	4	16	36	64	100	144	196	256	324	400	484	576	676	784	900	1,024	1,156	1,296
3	✗	16	✗	64	✗	144	✗	256	✗	400	✗	576	✗	784	✗	1,024	✗	1,296
4	8	64	216	512	1,000	1,728	2,744	4,096	5,832	8,000	10,648	13,824	17,576	21,952	27,000	32,768	39,304	46,656
5	✗	64	✗	256	✗	576	✗	1,024	✗	1,600	✗	2,304	✗	3,136	✗	4,096	✗	5,184
6	✗	256	✗	2,048	✗	6,912	✗	16,384	✗	32,000	✗	55,296	✗	87,808	✗	131,072	✗	186,624
7	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
8	48	768	3,888	12,288	30,000	62,208	115,248	196,608	314,928	480,000	702,768	995,328	1,370,928	1,843,968	2,430,000	3,145,728	4,009,008	5,038,848
9	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
10	32	1,536	2,880	16,384	20,000	70,272	76,832	196,608	212,544	448,000	468,512	889,344	913,952	1,580,544	1,627,200	2,621,440	2,672,672	4,116,096
11	✗	64	✗	256	✗	576	✗	1,024	✗	1,600	✗	2,304	✗	3,136		✓	✗	✓
12	✗	4,608	✗	73,728	✗	373,248	✗	1179,648	✗	✓	✗	✓	✗	✓		✓	✗	✓
13	✗	64	✗	256	✗	576	✗	1,024	✗	✓	✗	✓	✗	✓		✓	✗	✓
14	✗	✗	✗	✗	✗		✗				✗		✗				✗	
15	✗	✗	✗	✗	✗		✗		✗		✗		✗				✗	
16	384	13,312	98,496	401,408	1,200,000	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
17	✗	✗	✗		✗		✗				✗		✗				✗	
18	✗	3,072	✗	✓	✗	✓	✗	✓	✗	✓	✗	✓	✗	✓		✓	✗	✓
19	✗	✗	✗		✗		✗				✗		✗				✗	
20	272	26,880	70,416	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
21	✗	✗	✗		✗		✗		✗		✗		✗				✗	
22	✗	1,024	✗	✓		✓	✗	✓		✓	✗	✓	✗	✓		✓	✗	✓
23	✗	✗	✗		✗		✗				✗		✗				✗	
24	✗	98,304	✗	✓		✓	✗	✓	✗	✓	✗	✓	✗	✓		✓	✗	✓
25	✗	✗			✗		✗				✗		✗				✗	
26	16	1,280	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
27	✗	✗	✗		✗		✗		✗		✗		✗				✗	
28	✗	✗			✗		✗				✗		✗				✗	
29	✗				✗		✗				✗		✗				✗	
30	✗	✓	✗	✓		✓		✓	✗	✓	✗	✓	✗	✓		✓	✗	✓
31	✗				✗		✗				✗		✗				✗	
32	3,840	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
33	✗		✗		✗		✗		✗		✗		✗				✗	

Table 3. Existence and number of Golay sequences

s	exactly *s* Golay sequences exist
✓	Golay sequences do exist, but their exact number is unknown
✗	no Golay pairs exist
☐	Golay sequences may or may not exist

What Techniques are Involved?

- Exhaustive Search
  - Limit search to representatives under automorphisms
- Branch-and-Bound
  - Use Fourier Transforms to calculate power envelop of even/odd index subsequences
    - Fourier Transforms use parallel dot product
  - Reject envelops of ratio larger than 4
  - Store subsequence candidates in dynamic array
  - Use Fourier Transforms to calculate power envelop of interleaved sequences
  - Reject envelops of ratio larger than 2
- Backtracking
  - Use Depth-First Search with a heap to construct a matching mate for each candidate

Manual runtime profiling (here growth for number of phases) is tedious.

Our aim will be a theoretical runtime analysis, when possible.

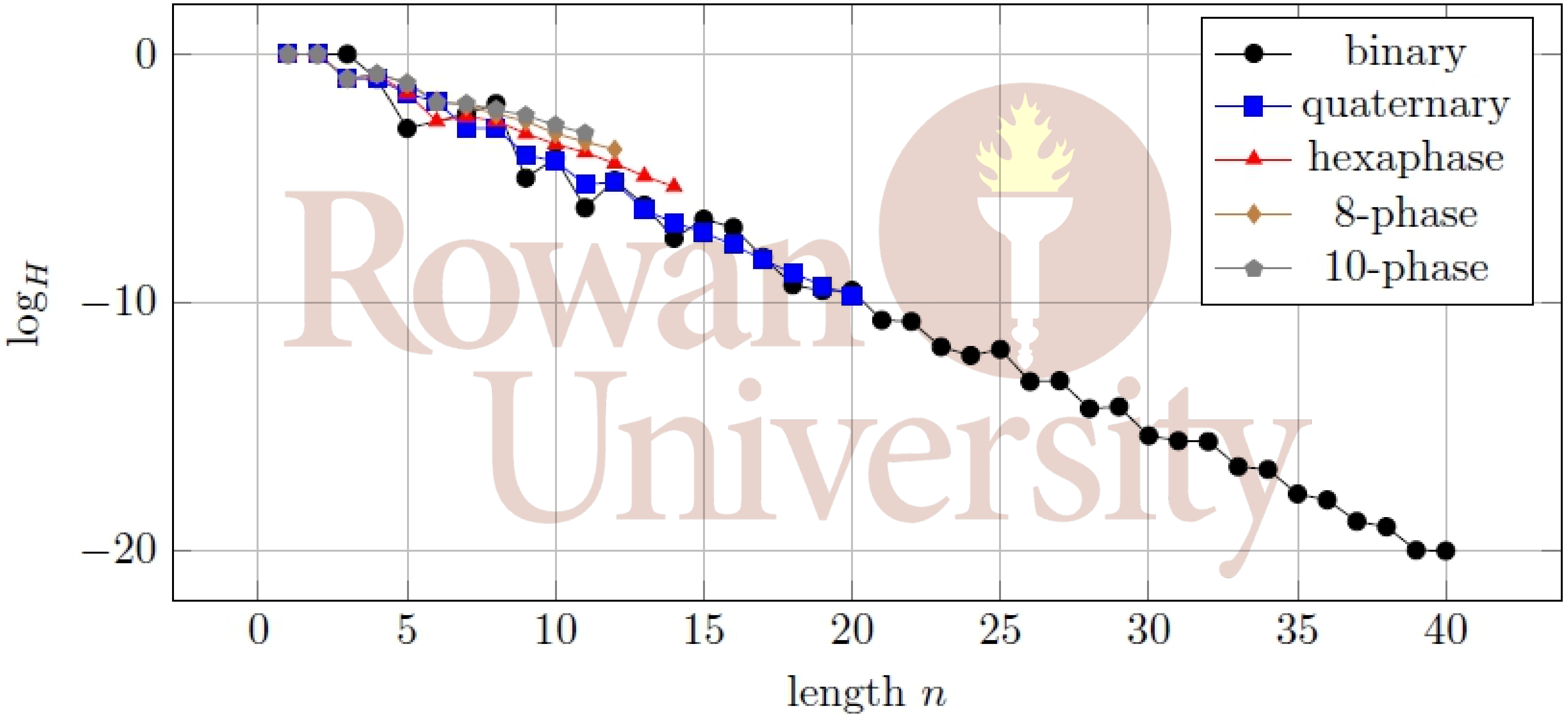


FIGURE 1. $\log_H$ fraction of H-phase sequences with property (10)

($K = \exp(2\pi\sqrt{-1}/1024)$)

Horizontal (here blue and red) lines in a logarithmic plot mean the algorithm is essentially exponential.

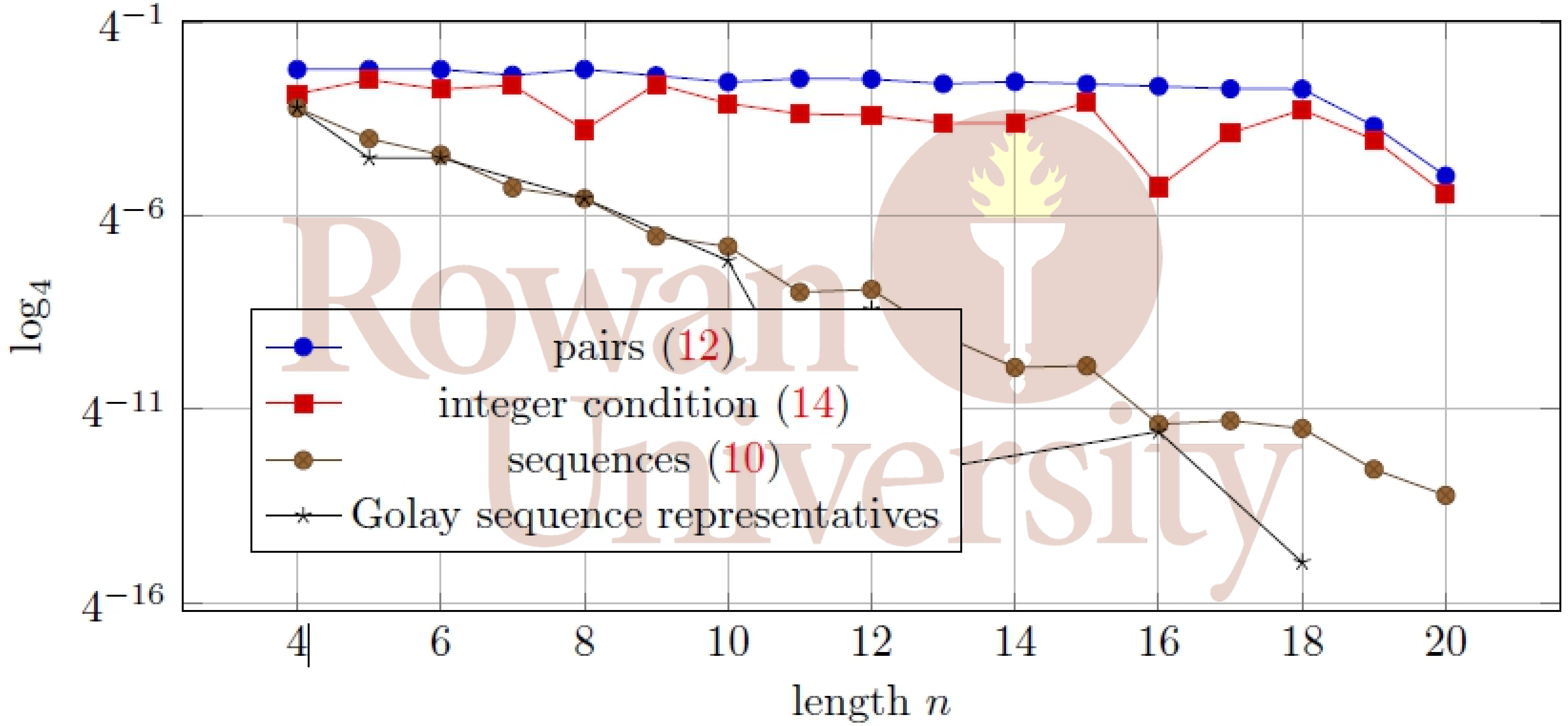


FIGURE 2. $\log_4$ fraction of quaternary sequences satisfying enumeration conditions

Runtime is usually measured not in a unit of time but a unit of number of steps/operations necessary to complete for the algorithm to come to a conclusion. We need some notation.

Asymptotic Notation

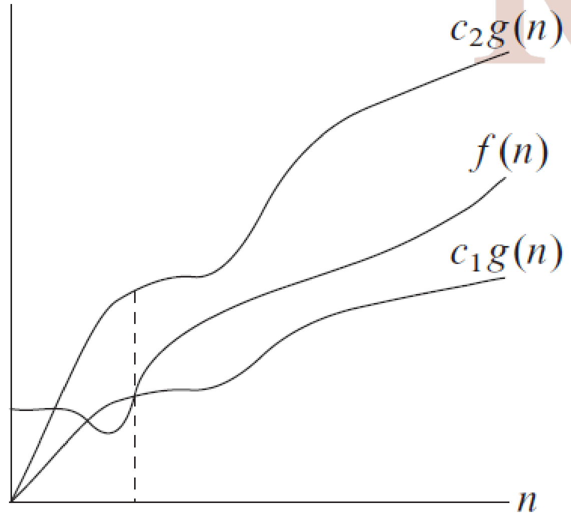
Recall that O, Θ, and Ω are mathematical notations to describe the asymptotic complexity of an algorithm (or function).

Asymptotic notation in mathematics refers to the growth of functions as the argument goes to infinity. You may have seen something like $\lim_{n\to\infty} \frac{an^2+bn+c}{n^2} = a$. That means that asymptotically, for large n, $an^2 + bn + c$ behaves like an^2. We can write $an^2 + bn + c = \Theta(n^2)$: up to a constant (in this case a), this function behaves like n^2 for **large** n. Sometimes such behavior only starts when n is past a certain value (large N_0), but asymptotically that does not make a difference – we are going to infinity with n.

***Θ*-Notation**

We say that some algebraic expression / function $f(n)$ is $\Theta(g(n))$ if $f(n)$ is essentially the same as $g(n)$ for large n, up to constant scalars. This is a ***tight bound***.

$$\Theta(g(n)) = \{f(n) \mid 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \text{ for all } n > N_0\}$$



$f(n) = \Theta(g(n))$

Think of it as an analog of the Sandwich Theorem (or Squeeze Theorem) from Calculus I. In Calculus terms it means $c_1 \leq \lim_{n\to\infty} \frac{f(n)}{g(n)} \leq c_2$, that is, for large n these two functions are roughly scalar multiples and grow at the same rate.

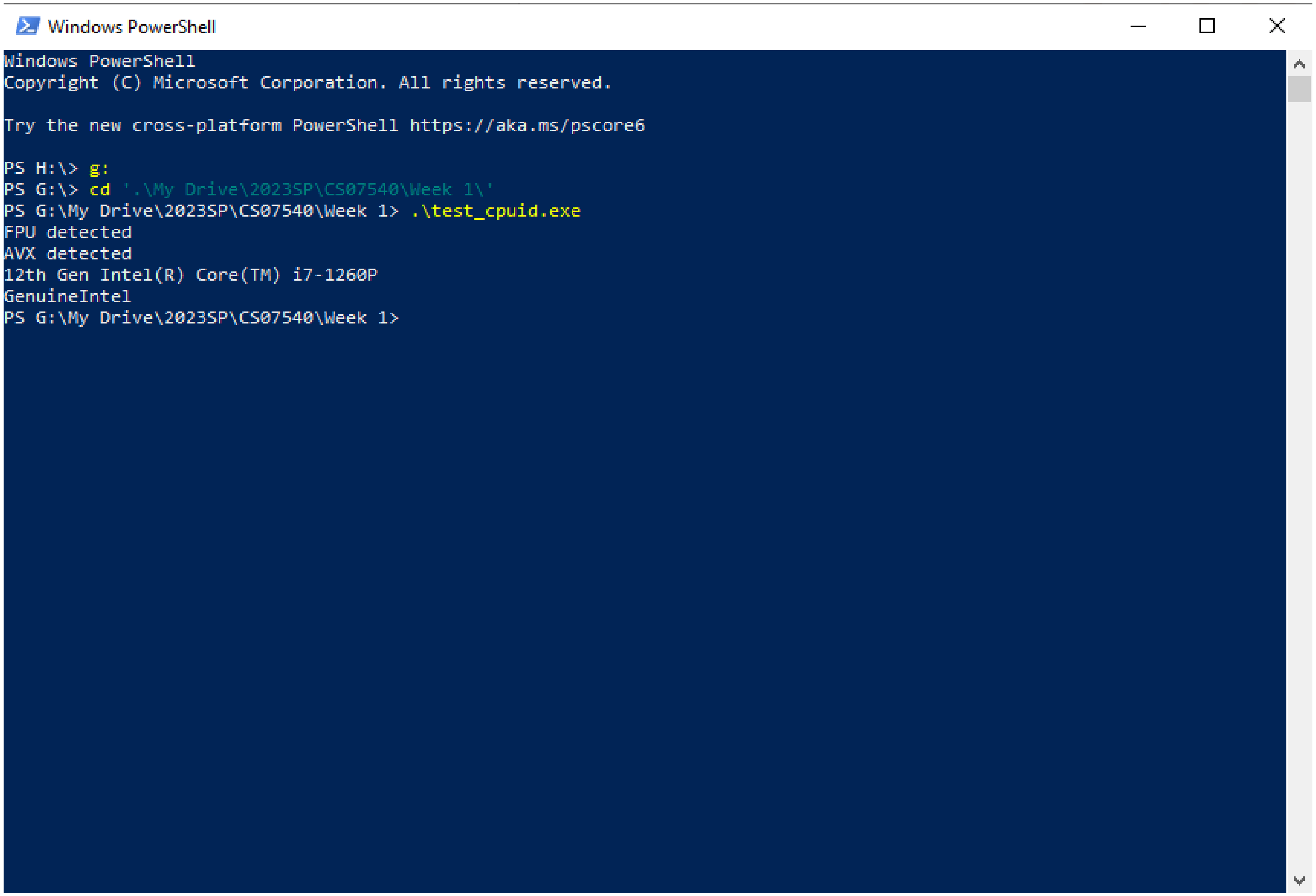
Example for Θ

One of the fundamental ***abstract data types*** is an ***array***. It holds an ordered collection of items accessible by an integer index. It is usually implemented as a contiguous range of memory containing (or pointing to) the data and hence supports offset calculation. We are going to look at a C++/ASM example implementing the CPUID call on x86 CPUs. This call returns four 32-bit registers which we write to an array of length four (for capabilities of the CPU). The two stubs shown are a (partial) class declaration and its implementation. We will see the arrays as variable names initialized with type, name, and length:

```
uint32_t cpuid_output[4]; ← Array of type uint32_t and length 4
std::string name[13];     ← Array of type string and length 13
```

```
#if GCC
#ifdef __i386__
#define ARCH_X86
#endif
#endif

class cpuid {
    public:
    cpuid();
    ~cpuid();
    void call_cpuid_asm(uint32_t EAX, uint32_t ECX, uint32_t (&EAX_EBX_ECX_EDX)[4]);
    void extract_x86_flags(uint32_t ECX, uint32_t EDX);
    void extract_x86_extended_flags(uint32_t EBX);
    std::string get_brand_string();
    std::string get_vendor_string();

    bool has_fpu() const;
    …
    bool has_avx2() const;

    private:
    std::string brand_string;
    void set_brand_string();
    std::string vendor_string;
    void set_vendor_string();

    bool m_has_fpu = false;
    …
    bool m_has_avx2 = false;
};
```

```
void cpuid::set_brand_string()
{
    #ifdef ARCH_X86
    // store EAX, EBX, ECX, and EDX
    uint32_t cpuid_output[3][4];
    call_cpuid_asm(0x80000000, 0, cpuid_output[0]);
    if(cpuid_output[0][0] >= 0x80000004) {
        // retrieve brand string in EAX, EBX, ECX, EDX
        char brandString[48] = {0, };
        call_cpuid_asm(0x80000002, 0, cpuid_output[0]);
        call_cpuid_asm(0x80000003, 0, cpuid_output[1]);
        call_cpuid_asm(0x80000004, 0, cpuid_output[2]);
        // DWORD/uint32_t to char conversion
        for(int i=0; i<3; i++)
            for(int j=0; j<4; j++)
                *((uint32_t*) brandString + 4*i+j) = cpuid_output[i][j];
        this->brand_string = std::string(brandString);
    }
    #endif
}
```


Windows PowerShell

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS H:\> g:
PS G:\> cd '.\My Drive\2023SP\CS07540\Week 1\'
PS G:\My Drive\2023SP\CS07540\Week 1> .\test_cpuid.exe
FPU detected
AVX detected
12th Gen Intel(R) Core(TM) i7-1260P
GenuineIntel
PS G:\My Drive\2023SP\CS07540\Week 1>
```

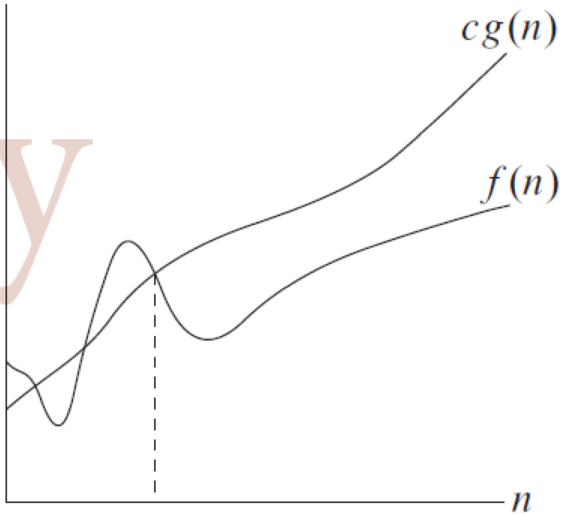
When we access an array to retrieve an element at index **`idx`**, the program uses the base address of the array (which is the address of **`Array[0]`**) and adds **`idx*size_of(data_type)`** to find the (virtual) memory address of the element at index **`idx`**. Hence it takes two operations (multiplication and addition) to retrieve an element from an array. With $c_1 = 1$ and $c_2 = 2$ we can say that retrieval in an array of length n, say $f(n)$, is bounded below and above by constants. Therefore, **retrieval by index in any array** (regardless of size n) **is** $\Theta(1)$ – constant time (if the array fits into physical main memory).

O-Notation

Θ-notation is not always attainable. We may not have a good grasp at the correct complexity. If we know a worst-case scenario, we can still create an ***asymptotic upper bound***.

$$\mathrm{O}\big(g(n)\big) = \{f(n) \mid 0 \le f(n) \le c \cdot g(n) \text{ for all } n > N_0\}$$

means that $g(n)$ is an upper bound for these functions f. It does not mean it is a good bound (such as a least upper bound) or even of the correct order (n^2 versus n^3). We strive to use O only when it is a **tight** upper bound. Note that Θ-bounds are also O-bounds.



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

Example for O

Supposed we want to find the position of a particular value in an array with **FINDINDEX** (or return **FAIL** if the value does not exist). The following Java-like pseudocode should help us determine the number of steps it would take at most to accomplish this task.

```
FINDINDEX(Array, value)
  for(int idx = 0; idx < n; i++) {
    if(Array[idx] == value)
      return idx;
    }
  return fail;
```

In best case, the first element at index 0 is the correct element. In worst case, we have to go through the whole array. Hence **FINDINDEX** for an array has an upper bound n and is $O(n)$. Is this the best upper bound?

If the array is ordered, can we do better than $O(n)$?

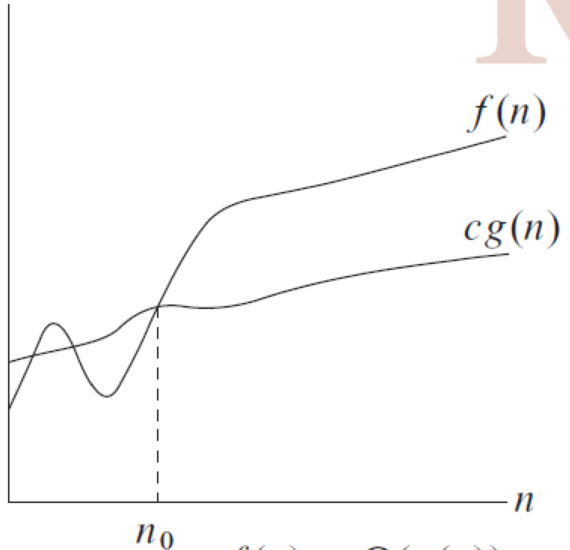
Yes, with a heap-like approach. If it is known that the array is ordered smallest to largest, then we can test the median (middle) index first. It will tell us whether the element we are looking for is in the first half or the second half of the array. That means we can ignore half of the array! Continuing recursively, we can find and element in $c \cdot \log_2(n)$ steps. **For ordered arrays, find will be an $O(\log(n))$ algorithm!** (Why do we write $O(\log(n))$ and not $O(\log_2(n))$?)

Ω-Notation

Given asymptotic upper bounds, it is natural to ask whether there are ***asymptotic lower bounds***. This is the holy grail!

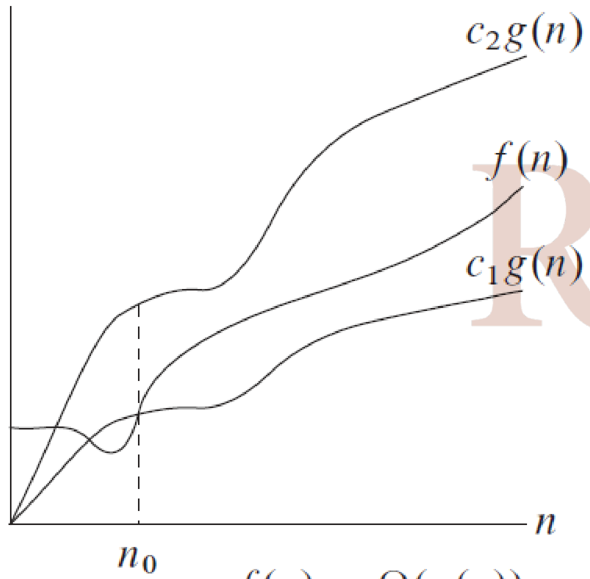
$$\mathrm{O}\big(g(n)\big) = \{f(n) \mid 0 \leq c \cdot g(n) \leq f(n) \text{ for all } n > N_0\}$$

The three notations are linked in that a function is $\Theta\big(g(n)\big)$ exactly if it is both $O\big(g(n)\big)$ and $\Omega\big(g(n)\big)$.

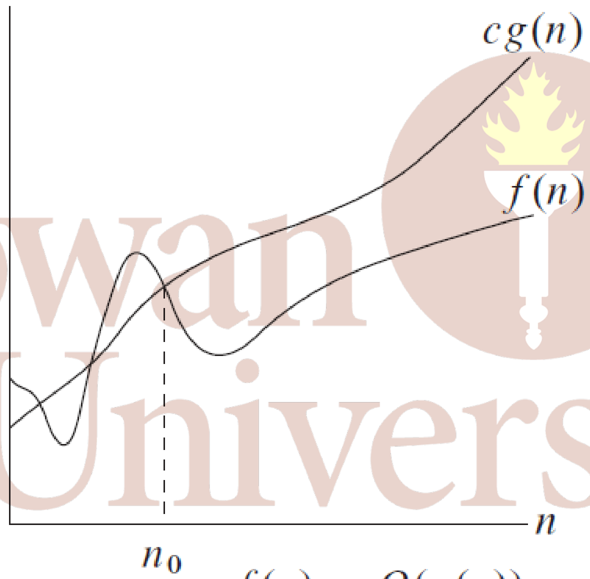


$f(n) = \Omega(g(n))$

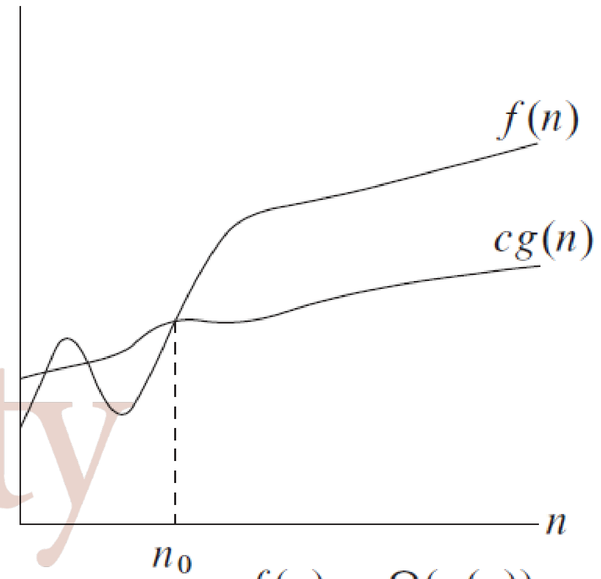
Finally, the O and Ω notations have “little” companions that indicate that the expression may not be tight.



$f(n) = \Theta(g(n))$



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



$f(n) = \Omega(g(n))$

What do these functions or expressions refer to? The notations are linked to specific algorithms! Algorithms are just a means to an end, so we need to distinguish between the problem we are trying to solve and the algorithm we are going to use. Think

- Problem: Find a minimal spanning tree
- Algorithm: Prim’s Algorithm (which is just one specific algorithm)

Let’s use an example.

- Problem: Sorting a list of values.
- Algorithm: Selection Sort

Selection Sort Example

Selection sort is an in-place comparison sorting algorithm. It has an $O(n^2)$ time complexity, which makes it inefficient on large lists, and generally performs worse than the similar insertion sort. Selection sort is noted for its simplicity and has performance advantages over more complicated algorithms in certain situations, particularly where auxiliary memory is limited. 📖

```java
import java.util.ArrayList;
import java.util.List;
import static java.util.Collections.swap;

public class SelectionSort {
    void sort(List<Integer> A) {
        for (int i = 0; i < A.size(); i++) {
            int smallestElementIndex = i;
            for (int j = i + 1; j < A.size(); j++)
                if (A.get(j) < A.get(smallestElementIndex))
                    smallestElementIndex = j;
            swap(A, smallestElementIndex, i);
        }
    }

    public static void main (String[] args){
        SelectionSort obj = new SelectionSort();
        List<Integer> A = new ArrayList<>(List.of(126,0,3, 215, 12, 22, 11));
        System.out.println("Given array");
        System.out.println(A);
        obj.sort(A);
        System.out.println("Sorted array");
        System.out.println(A);
        }
    }
```

If we start with a list of size n, we can analyze the loop structure and count the number of iterations. For $i = 0$ we get an inner loop of size n. For $i = 1$ we get $n - 1$, and so on. Since

$$\sum_{i=0}^{n-1}(n-i) = \frac{1}{2}(n-1)n$$

we get an asymptotic runtime of $\left(\frac{1}{2}n^2 - \frac{1}{2}n\right)$ steps. Note that the n^2 term dominates for large n.

Formally we say that the worst-case runtime $T_A(X)$ of algorithm A with input X ($|X| = n$) is

$$T_A(n) = \max_{|X|=n} T_A(X)$$

Given a particular problem P (such as sorting), we first need an algorithm A that solves P before we can talk about runtime. Our algorithm is selection sort. With A being selection sort, the runtime T_A is $\frac{1}{2}(n^2 - n)$ for any input X of size n. The worst-case complexity of a problem P is the worst-case running time of the fastest algorithm for solving it.

$$T_P(n) = \min_{A \text{ solves } P} T_A(n)$$

$$T_P(n) = \min_{A \text{ solves } P} \left(\max_{|X|=n} T_A(X) \right)$$

When we describe an algorithm A that solves P in $O\big(f(n)\big)$ time, we have an ***upper bound*** on the complexity of P.

$$T_P(n) \leq T_A(n) = O\big(f(n)\big)$$

We have

- P is sorting
- A is selection sort (which is a particular implementation of sorting)
- $f(n) = \frac{1}{2}n^2 - \frac{1}{2}n \sim n^2$

Hence an upper bound on the complexity of sorting n objects is n^2, written $O(n^2)$.

But how do we know whether the algorithm we used to solve our problem is any good? Good compared to what? It would be nice to have both upper and lower tight bounds to evaluate an algorithm.

- $O(\square)$ describes the upper bound of the complexity.
- $\Omega(\square)$ describes the lower bound of the complexity.
- $\Theta(\square)$ describes the exact bound of the complexity (upper and lower are the same, up to scalars and constants).
- $o(\square)$ describes the upper bound excluding the exact bound.

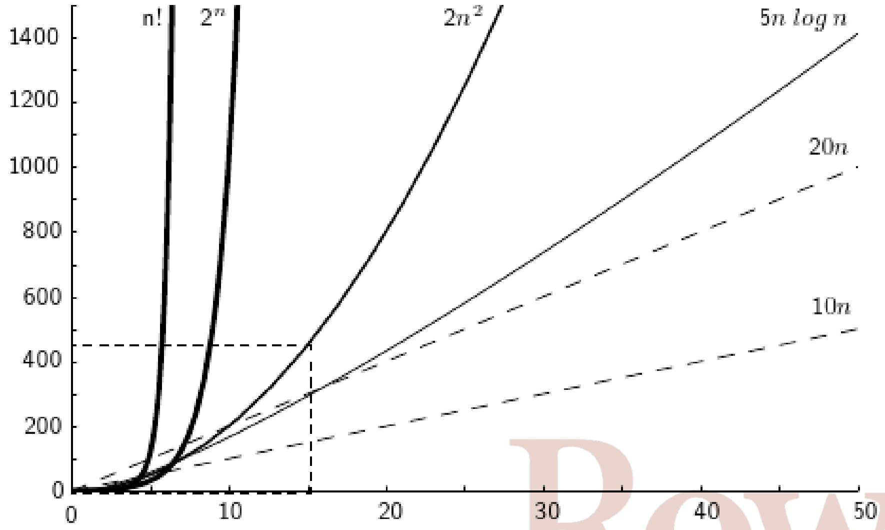
If we have $f(n) = \frac{1}{2}n^2 - \frac{1}{2}n$, then

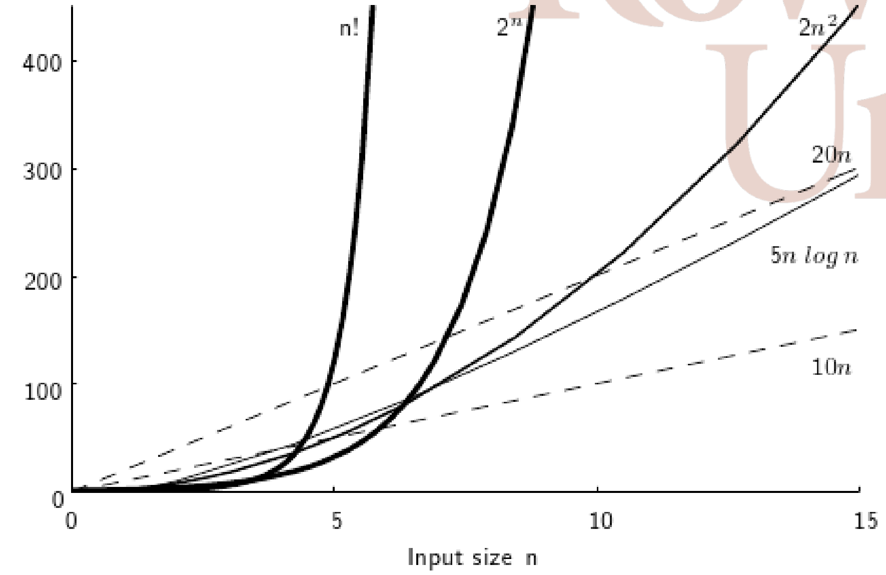
- $f(n)$ is $O(n^3)$ and also $O(n^2)$ but not $O(n)$
- $f(n)$ is $\Omega(n)$, but we can also argue for $\Omega(n^2)$ for larger n
- $f(n)$ is $\Theta(n^2)$
- $f(n)$ is $o(n^3)$

Usually, when we write $O(\square)$, we really want $\Theta(\square)$.

TABLE 2.1 Values (some approximate) of several functions important for analysis of algorithms

n	$\log_2 n$	n	$n \log_2 n$	n^2	n^3	2^n	$n!$
10	3.3	10^1	$3.3 \cdot 10^1$	10^2	10^3	10^3	$3.6 \cdot 10^6$
10^2	6.6	10^2	$6.6 \cdot 10^2$	10^4	10^6	$1.3 \cdot 10^{30}$	$9.3 \cdot 10^{157}$
10^3	10	10^3	$1.0 \cdot 10^4$	10^6	10^9		
10^4	13	10^4	$1.3 \cdot 10^5$	10^8	10^{12}		
10^5	17	10^5	$1.7 \cdot 10^6$	10^{10}	10^{15}		
10^6	20	10^6	$2.0 \cdot 10^7$	10^{12}	10^{18}		





There is a tremendous difference between the growth of 2^n and $n!$, yet both are often referred to as “exponential-growth functions” (or simply “exponential”). Strictly speaking, only the former should be referred to as such since $n! \sim \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$ (Stirling’s Formula).

Algorithms that require an exponential number of operations are practical for solving only problems of very small sizes.

Worst-Case, Best-Case, and Average-Case Efficiencies

The worst-case efficiency of an algorithm is its efficiency for the worst-case input of size n – it runs the longest among all possible inputs of that size. It is the easiest to estimate since we assume worst performance for each subproblem. **This will be our focus**.

Example: Finding the minimum in an array of size n has worst performance $O(n)$, since we cannot be sure to have found the minimum until we looked at all n elements.

Example: Finding the minimum in an ordered array of size n has exact performance $\Theta(1)$, since we the minimum is $\text{Array}[0]$. Constant time. It does not get any better than that.

The analysis of the best-case efficiency is not nearly as important as that of the worst-case efficiency. However, a best-case analysis will give us a minimum runtime to expect from our algorithm.

In a typical environment, the worst case does not happen all the time. Rather, we expect some sort of “averaging out” of worst cases and best cases. This field of complexity analysis is called Amortized Analysis. ***Amortized analysis*** is not average-case analysis – probability is not involved; an amortized analysis guarantees the average performance of each operation in the worst case for a sequence of operations.

It is the job of the algorithm **analyst** to figure out as much information about an algorithm as possible, and it is the job of the **programmer** who implements an algorithm to apply that knowledge to develop programs that solve problems effectively.

Complexity

Definition: We say that an algorithm solves a problem in polynomial time if its worst-case time efficiency belongs to $O(p(n))$ where $p(n)$ is a polynomial of the problem's input size n.

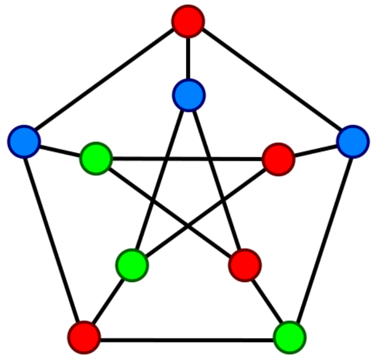
Problems that can be solved in polynomial time are called tractable, and problems that cannot be solved in polynomial time are called intractable.

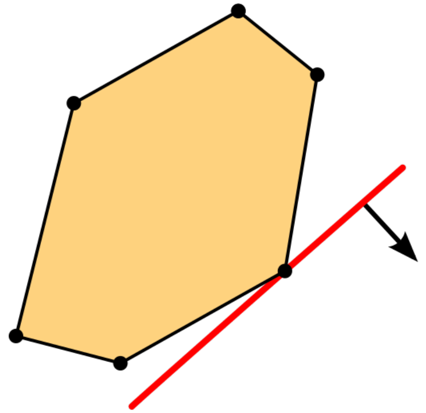
(Note that problems solvable in, say, logarithmic time are solvable in polynomial time as well.)

Definition: Class P is a class of decision problems that can be solved in polynomial time by (deterministic) algorithms. This class of problems is called polynomial.

There are many important problems for which no polynomial-time algorithm has been found, nor has the impossibility of such an algorithm been proved. Examples include

- Hamiltonian Circuits
- Traveling Salesman Problem (linked to Hamiltonian circuit)
- Knapsack Problem
- Graph-coloring Problem (chromatic number)
- Integer linear programming problem





Definition: A **nondeterministic algorithm** is a two-stage procedure that takes as its input an instance I of a decision problem and does the following.

- Nondeterministic (“guessing”) stage: An arbitrary string S is generated that can be thought of as a candidate solution to the given instance I.
- Deterministic (“verification”) stage: A deterministic algorithm takes both I and S as its input and outputs yes if S represents a solution to instance I.

(We require a nondeterministic algorithm to be capable of “guessing” a solution at least once and to be able to verify its validity.)

A nondeterministic algorithm is said to be **nondeterministic polynomial** if the time efficiency of its verification stage is polynomial.

Definition: Class NP is the class of decision problems that can be solved by nondeterministic polynomial algorithms. This class of problems is called nondeterministic polynomial.

$$P \subseteq NP$$

Most mathematicians and computer scientists conjecture

$$P \neq NP$$

Definition: A decision problem D_1 is said to be polynomially reducible to a decision problem D_2, if there exists a function t that transforms instances of D_1 to instances of D_2 such that:

1. t maps all yes instances of D_1 to yes instances of $D_2$2 and all no instances of D_1 to no instances of D_2.
2. t is computable by a polynomial time algorithm.

Definition: A decision problem D is said to be NP-complete if:

1. t belongs to class NP
2. every problem in NP is polynomially reducible to D.