**HEALTHCURE**

**MAJOR PROJECT REPORT**

*Submitted by*

**{Name} (Enrollment No)**

**Computer Science**

**{Your College Name}**

{Address of College}

**{Your College Name}**

*To Whom It May Concern*

I, **{Name}** Enrollment No. {Enrollment No}, student of **Bachelors of Technology (CS), a class of 20XX-XX, {Your College Name}, {Location of your college}** hereby declare that the Summer Internship project report entitled **“HealthCure”** is an original work and the same has not been submitted to any other Institute for the award of any other degree.

Date:

Place: New Delhi

**{Name}**

**Enrollment No:**

**Computer Science**

**{Batch}**

**ACKNOWLEDGEMENT**

First and foremost, I wish to express my profound gratitude to {your mentor} for giving me the opportunity to carry out my training.

No words can express my deep sense of gratitude to {your mentor}, without whom this training would not have turned up this way. My heartfelt thanks to him for his immense help and support, useful discussions and valuable recommendations throughout the course of my training.

I wish to thank my respected faculty and my lab mates for their support.

Last but not the least I thank the almighty for enlightening me with his blessings.

**{Name}**

**Enrollment No:**

**Computer Science**

**{Batch}**

**ABSTRACT**

Artificial intelligence (AI) is gradually changing medical practice. With recent progress in digitized data acquisition, machine learning and computing infrastructure, AI applications are expanding into areas that were previously thought to be only the province of human experts. In this Review Article, we outline recent breakthroughs in AI technologies and their biomedical applications, identify the challenges for further progress in medical AI systems, and summarize the economic, legal and social implications of AI in healthcare.

We have used Python which has an extensive and comprehensive collection of freely available packages covering a variety of topics. Scientific Python libraries such as NumPy, SciPy, and pandas provide efficient implementation of numerical operations and tasks common in science and engineering. These libraries provide a strong base from which more advanced scientific software can be built without needing to worry about low-level algorithms. Additionally, many domain specific packages exist which address the scientific needs of the meteorological community. Also we have used Convolutional Neural Networks to classify various diseases and also some state of the art Machine Learning algorithms.

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **S.No.** | **Chapter** | **Page Number** |
|  | LIST OF PICTURES | 16 |
| 1 | **INTRODUCTION** | 18-25 |
| 1.1 | Introduction to Python | 18 |
| 1.2 | History | 19 |
| 1.3 | What can Python do | 20 |
| 1.4 | Why Python | 20 |
| 1.5 | Python Language compared to other Languages | 21 |
| 1.6 | Libraries | 21 |
| 1.7 | Implementations | 22 |
| 1.8 | Python Line Structure | 22 |
| 1.9 | Indentation | 24 |
| 1.10 | Python Coding Style | 25 |
| **2** | **INTRODUCTION TO DATA STRUCTURES AND ALGORITHMS** | 26-60 |
| 2.1 | What is a Data Structure | 26 |
| 2.3 | Applications of Data Structure and Algorithms | 27 |
| 2.4 | Characteristics of a Data Structure | 27 |
| 2.5 | Need for Data Structure | 27 |
| 2.6 | Execution Time Cases | 28 |
| 2.7 | Basic Terminology | 28 |
| 2.8 | What is an Algorithm? | 29 |
| 2.9 | Characteristics of an Algorithm | 29 |
| 2.10 | How to Write an Algorithm? | 29 |
| 2.11 | Algorithm Analysis | 30 |
| 2.12 | Algorithm Complexity | 31 |
| 2.12.1 | Space Complexity | 31 |
| 2.12.2 | Time Complexity | 32 |
| 2.13 | Asymptotic Analysis | 32 |
| 2.14 | Asymptotic Notations | 32 |
| 2.15 | Common Asymptotic Notations | 34 |
| 2.16 | Greedy Algorithms | 35 |
| 2.16.1 | Counting Coins | 35 |
| 2.17 | Divide and Conquer | 36 |
| 2.17.1 | Divide/Break | 36 |
| 2.17.2 | Conquer/Solve | 36 |
| 2.17.3 | Merge/Combine | 37 |
| 2.18 | Dynamic Programming | 37 |
| 2.19 | Linked Lists | 38 |
| 2.20 | Stacks and Queues | 39 |
| 2.21 | Sorting Techniques | 40 |
| 2.22 | In-place Sorting and Not-in-place Sorting | 40 |
| 2.23 | Stable and Not Stable Sorting | 40 |
| 2.24 | Adaptive and Non-Adaptive Sorting Algorithm | 41 |
| 2.25 | Important Terms | 42 |
| 2.26 | Bubble Sort | 42 |
| 2.26.1 | How Bubble Sort Works? | 42 |
| 2.27 | Insertion Sort | 45 |
| 2.27.1 | How Insertion Sort Works? | 45 |
| 2.28 | Selection Sort | 47 |
| 2.28.1 | How Selection Sort Works? | 47 |
| 2.29 | Merge Sort | 50 |
| 2.29.1 | How Merge Sort Works? | 50 |
| 2.30 | Quick Sort | 51 |
| 2.30.1 | Quick Sort Pivot Algorithm | 51 |
| 2.30.2 | Quick Sort Algorithm | 52 |
| 2.31 | Recursion | 52 |
| 2.31.1 | Properties | 52 |
| 2.31.2 | Implementation | 52 |
| 2.31.3 | Analysis of Recursion | 53 |
| 2.31.4 | Time Complexity | 53 |
| 2.31.5 | Space Complexity | 53 |
| 2.32 | Graph | 54 |
| 2.32.1 | Graph Data Structure | 54 |
| 2.32.2 | Basic Operations | 55 |
| 2.33 | Trees | 55 |
| 2.33.1 | Important Terms | 56 |
| 2.33.2 | Binary Search Tree Representation | 56 |
| 2.33.3 | Tree Node | 56 |
| 2.33.4 | BST Basic Operations | 57 |
| 2.34 | Heap | 57 |
| 2.34.1 | Max Heap Construction Algorithm | 58 |
| 2.34.2 | Max Heap Deletion Algorithm | 58 |
| 2.35 | Spanning Tree | 59 |
| 2.35.1 | General Properties of Spanning Tree | 59 |
| 2.35.2 | Mathematical Properties of Spanning Tree | 60 |
| 2.35.3 | Application of Spanning Tree | 60 |
| 2.35.4 | Minimum Spanning Tree (MST) | 60 |
| 2.35.5 | Minimum Spanning-Tree Algorithm | 60 |
| 3 | Convolutional Neural Network | 61 |
| 3.1.1 | Introduction to CNN | 61 |
| 3.1.2 | What is a Convolutional Neural Network? | 62 |
| 3.1.3 | Layers in a Convolutional Neural Network | 62 |
| 3.1.3.1 | Convolution Layer | 63 |
| 3.1.3.2 | ReLU layer | 63 |
| 3.1.3.3 | Pooling Layer | 64 |
| 3.2 | XGBoost Algorithm | 67 |
| 3.2.1 | What is XGBoost? | 67 |
| 3.2.2 | How to build an intuition for XGBoost? | 68 |
| 3.2.3 | Why does XGBoost perform so well? | 69 |
| 3.2.4 | How XGBoost optimizes standard GBM algorithm | 69 |
| 3.2.4.1 | System Optimization: | 69 |
| 3.2.4.2 | Algorithmic Enhancements: | 70 |
| 3.3 | Bagging and Random Forest Ensemble Algorithms for Machine Learning | 70 |
| 3.3.1 | Bootstrap Method | 70 |
| 3.3.2 | Bootstrap Aggregation (Bagging) | 71 |
| 3.4 | Random Forest | 72 |
| 3.4.1 | AdaBoost (Adaptive Boosting) | 72 |
| 3.4.2 | Gradient Boosting | 73 |
| 4 | Diseases | 75 |
| 4.1 | Covid-19 | 75 |
| 4.1.1 | Symptoms | 75 |
| 4.1.2 | Causes | 77 |
| 4.1.3 | Risk factor | 77 |
| 4.1.4 | Diagnosis | 77 |
| 4.2 | Brain tumor | 78 |
| 4.2.1 | Symptoms | 78 |
| 4.2.2 | Causes | 78 |
| 4.2.3 | Risk factor | 78 |
| 4.2.4 | Diagnosis | 78 |
| 4.3 | Breast cancer | 81 |
| 4.3.1 | Symptoms | 81 |
| 4.3.2 | Causes | 81 |
| 4.3.3 | Risk factor | 82 |
| 4.3.4 | Diagnosis | 83 |
| 4.4 | Alzheimer | 84 |
| 4.4.1 | Symptoms | 84 |
| 4.4.2 | Causes | 86 |
| 4.4.3 | Risk factor | 86 |
| 4.4.4 | Diagnosis | 88 |
| 4.5 | Diabetes | 90 |
| 4.5.1 | Symptoms | 90 |
| 4.5.2 | Causes | 91 |
| 4.5.3 | Risk factor | 92 |
| 4.5.4 | Diagnosis | 93 |
| 4.6 | Pneumonia | 95 |
| 4.6.1 | Symptoms | 95 |
| 4.6.2 | Causes | 95 |
| 4.6.3 | Risk factor | 96 |
| 4.6.4 | Diagnosis | 97 |
| 4.7 | Heart disease | 98 |
| 4.7.1 | Symptoms | 98 |
| 4.7.2 | Causes | 100 |
| 4.7.3 | Risk factor | 103 |
| 4.7.4 | Diagnosis | 104 |
|  | CONCLUSION | 106 |

**LIST OF PICTURES**

|  |  |  |
| --- | --- | --- |
| **S. NO.** | **Figure Description** | **Page** |
| Fig 1.1 | Python | 18 |
| Fig 1.2 | History of Python | 19 |
| Fig 1.3 | Applications of Python | 20 |
| Fig 1.4 | Advantages Of  Python | 21 |
| Fig 1.5 | Physical Lines | 23 |
| Fig 1.6 | Commenting in Python | 23 |
| Fig 1.7 | Line Joining | 24 |
| Fig 1.8 | Multiple Statements On a Single Line | 24 |
| Fig 1.9 | Indent Error | 25 |
| Fig. 2.1 | How to write an algorithm | 30 |
| Fig. 2.2 | Big O Notation | 33 |
| Fig. 2.3 | Omega Notation | 33 |
| Fig. 2.4 | Theta Notation | 34 |
| Fig. 2.5 | Divivde and Conquer | 36 |
| Fig. 2.6 | Representation of a Linked List | 38 |
| Fig. 2.7 | Representation of a Stack | 39 |
| Fig. 2.8 | Representation of a Queue | 40 |
| Fig. 2.9 | Stable Sorting | 41 |
| Fig. 2.10 | Unstable Sorting | 41 |
| Fig 2.11 – 2.22 | Bubble Sorting | 42-44 |
| Fig 2.23- 2.35 | Insertion Sorting | 45-47 |
| Fig 2.36- 2.42 | Selection Sorting | 47-49 |
| Fig 2.43- 2.49 | Merge Sorting | 50-51 |
| Fig 2.50 | Recursion | 53 |
| Fig 2.51 | Undirected Graph | 54 |
| Fig 2.52 | Graph | 55 |
| Fig 2.53 | Tree | 55 |
| Fig 2.54 | Binary Search Tree | 56 |
| Fig 2.55 | Min Heap | 57 |
| Fig 2.56 | Max Heap | 58 |
| Fig 2.57 | Spanning Tree | 59 |
| Fig 3.1 | CNN making prediction on a bird Image | 61 |
| Fig 3.2 | CNN making prediction on Flower Images | 62 |
| Fig 3.3 | Representing 8 in pixels | 62 |
| Fig 3.4 | Showing image and filter | 63 |
| Fig 3.5 | ReLU Activation Function | 63 |
| Fig 3.6 | Max Pooling | 64 |
| Fig 3.7 | Max Pooled Image | 64 |
| Fig 3.8 | CNN Structure | 64 |
| Fig 3.9 | Flattening the Pooled Image | 65 |
| Fig 3.10 | Flattened Image fed to Fully Connected Layer | 65 |
| Fig 3.11 | Full CNN Structure | 65 |
| Fig 3.12 | Feeding Bird Image to the CNN | 66 |
| Fig 3.13 | Two Different parts of a CNN Structure | 66 |
| Fig 3.14 | Bagging and Boosting Techniques | 67 |
| Fig 3.15 | XGBoost | 69 |

**CHAPTER 1**

**INTRODUCTION TO PYTHON**

**1.1 Introduction to Python**

Python is an interpreted, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991, Python's design philosophy emphasizes code readability with its notable use of significant whitespace. Its language constructs and object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects.

Python is dynamically typed and garbage-collected. It supports multiple programming paradigms, including procedural, object-oriented, and functional programming. Python is often described as a "batteries included" language due to its comprehensive standard library.

Python was conceived in the late 1980s as a successor to the ABC language. Python 2.0, released 2000, introduced features like list comprehensions and a garbage collection system capable of collecting reference cycles. Python 3.0, released 2008, was a major revision of the language that is not completely backward-compatible, and much Python 2 code does not run unmodified on Python 3. Due to concern about the amount of code written for Python 2, support for Python 2.7 (the last release in the 2.x series) was extended to 2020. Language developer Guido van Rossum shouldered sole responsibility for the project until July 2018 but now shares his leadership as a member of a five-person steering council. Python interpreters are available for many operating systems. A global community of programmers develops and maintains CPython, an open source reference implementation. A non-profit organization, the Python Software Foundation, manages and directs resources for Python and CPython development.

It is used for:

* web development (server-side),
* software development,
* mathematics,
* System scripting.



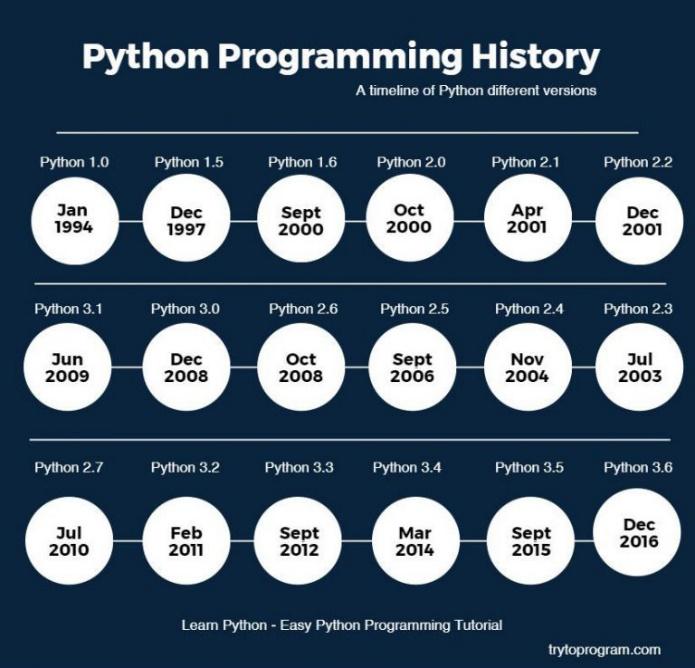
**Fig 1.1 Python**

Python was conceived in the late 1980s by Guido van Rossum at Centrum Wiskunde&Informatica (CWI) in the Netherlands as a successor to the ABC language (itself inspired by SETL), capable of exception handling and interfacing with the Amoeba operating system. Its implementation began in December 1989. Van Rossum continued as Python's lead developer until July 12, 2018, when he announced his "permanent vacation" from his responsibilities as Python's Benevolent Dictator for Life, a title the Python community bestowed upon him to reflect his long-term commitment as the project's chief decision-maker. In January, 2019, active Python core developers elected Brett Cannon, Nick Coghlan, Barry Warsaw, Carol Willing and Van Rossum to a five-member "Steering Council" to lead the project.

Python 2.0 was released on 16 October 2000 with many major new features, including a cycle-detecting garbage collector and support for Unicode.

**1.2 History**

Python was conceived in the late 1980s by Guido van Rossum at Centrum Wiskunde&Informatica (CWI) in the Netherlands as a successor to the ABC language (itself inspired by SETL), capable of exception handling and interfacing with the Amoeba operating system. Its implementation began in December 1989. Van Rossum continued as Python's lead developer until July 12, 2018, when he announced his "permanent vacation" from his responsibilities as Python's Benevolent Dictator for Life, a title the Python community bestowed upon him to reflect his long-term commitment as the project's chief decision-maker. In January, 2019, active Python core developers elected Brett Cannon, Nick Coghlan, Barry Warsaw, Carol Willing and Van Rossum to a five-member "Steering Council" to lead the project.



**Fig 1.2 History of Python**

Python 2.0 was released on 16 October 2000 with many major new features, including a cycle-detecting garbage collector and support for Unicode.

Python 3.0 was released on 3 December 2008. It was a major revision of the language that is not completely backward-compatible.Many of its major features were back ported to Python 2.6.x and 2.7.x version series. Releases of Python 3 include the 2 to 3 utility, which automates (at least partially) the translation of Python 2 code to Python 3.

Python 2.7's end-of-life date was initially set at 2015 then postponed to 2020 out of concern that a large body of existing code could not easily be forward-ported to Python 3. In January 2017, Google announced work on a Python 2.7 to Go trans compiler to improve performance under concurrent workloads.

**1.3 What can Python do?**

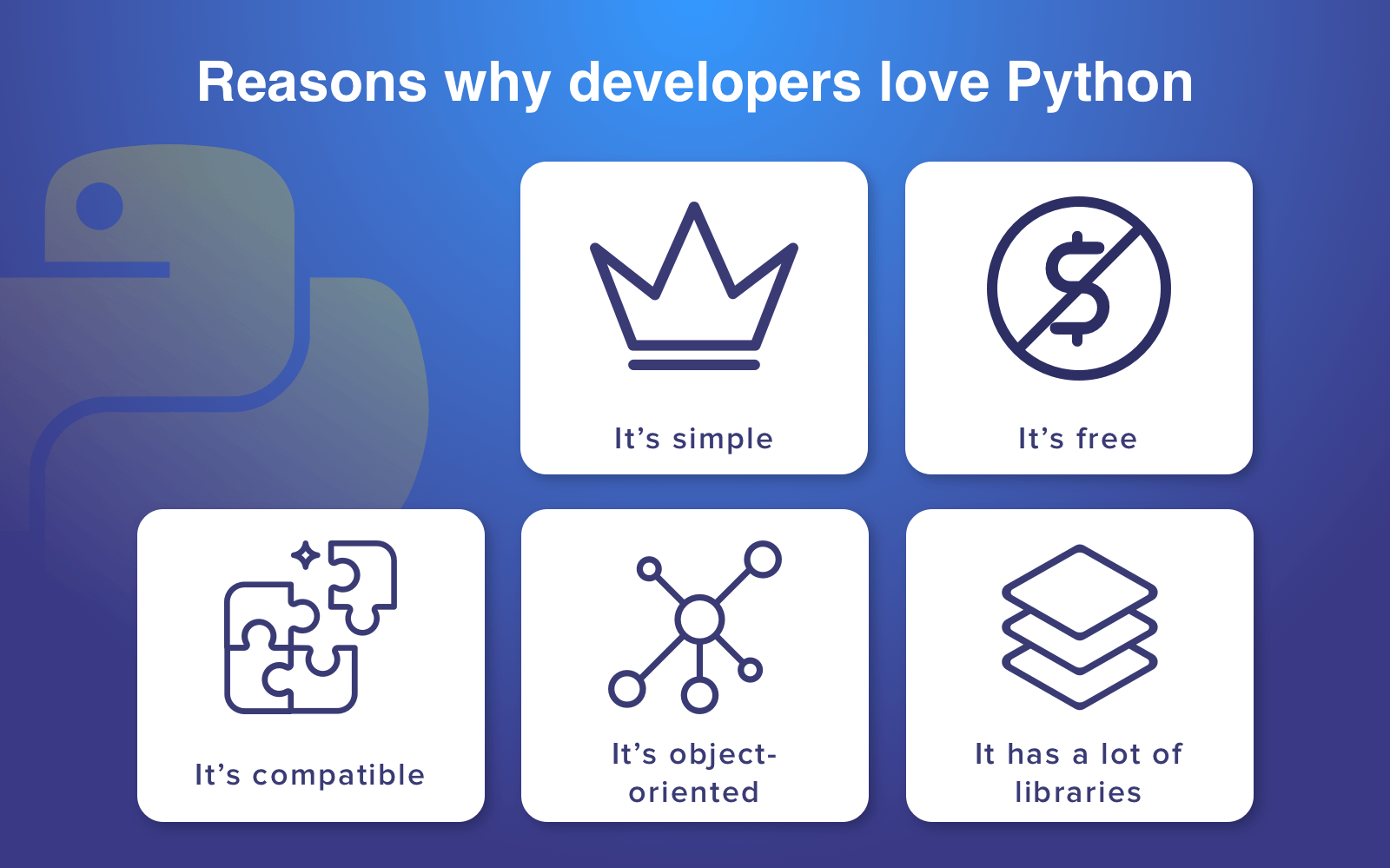
* Python can be used on a server to create web applications.
* Python can be used alongside software to create workflows.
* Python can connect to database systems. It can also read and modify files.
* Python can be used to handle big data and perform complex mathematics.
* Python can be used for rapid prototyping, or for production-ready software development.



**Fig 1.3 Applications of Python**

**1.4 Why Python?**

* Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc.).
* Python has a simple syntax similar to the English language.
* Python has syntax that allows developers to write programs with fewer lines than some other programming languages.
* Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.
* Python can be treated in a procedural way, an object-orientated way or a functional way.



**Fig 1.4 Advantages of Python**

**1.5 Python Syntax compared to other Languages**

* Python was designed for readability, and has some similarities to the English language with influence from mathematics.
* Python uses new lines to complete a command, as opposed to other programming languages which often use semicolons or parentheses.
* Python relies on indentation, using whitespace, to define scope; such as the scope of loops, functions and classes. Other programming languages often use curly-brackets for this purpose.

**1.6 Libraries**

Python's large standard library, commonly cited as one of its greatest strengths, provides tools suited to many tasks. For Internet-facing applications, many standard formats and protocols such as MIME and HTTP are supported. It includes modules for creating graphical user interfaces, connecting to relational databases, generating pseudorandom numbers, arithmetic with arbitrary precision decimals, manipulating regular expressions, and unit testing.

Some parts of the standard library are covered by specifications (for example, the Web Server Gateway Interface (WSGI) implementation wsgiref follows PEP 333), but most modules are not. They are specified by their code, internal documentation, and test suites (if supplied). However, because most of the standard library is cross-platform Python code, only a few modules need altering or rewriting for variant implementations.

As of March 2018, the Python Package Index (PyPI), the official repository for third-party Python software, contains over 130,000 packages with a wide range of functionality, including:

* Graphical user interfaces
* Web frameworks
* Multimedia
* Databases
* Networking
* Test frameworks
* Automation
* Web scraping
* Documentation
* System administration
* Scientific computing
* Text processing
* Image processing

**1.7 Implementations**

**Reference implementation**

[CPython](https://en.wikipedia.org/wiki/CPython) is the [reference implementation](https://en.wikipedia.org/wiki/Reference_implementation) of Python. It is written in [C](https://en.wikipedia.org/wiki/C_(programming_language)), meeting the [C89](https://en.wikipedia.org/wiki/C89_(C_version)) standard with several select [C99](https://en.wikipedia.org/wiki/C99) features. It compiles Python programs into an intermediate [bytecode](https://en.wikipedia.org/wiki/Bytecode) which is then executed by its [virtual machine](https://en.wikipedia.org/wiki/Virtual_machine). CPython is distributed with a large standard library written in a mixture of C and native Python. It is available for many platforms, including [Windows](https://en.wikipedia.org/wiki/Microsoft_Windows) and most modern [Unix-like](https://en.wikipedia.org/wiki/Unix-like) systems. Platform portability was one of its earliest priorities.

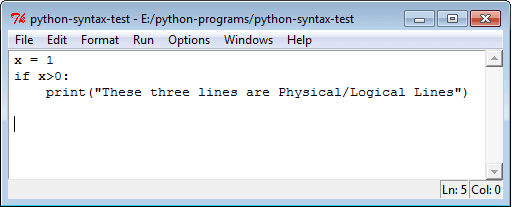
**Other implementations**

* PyPy is a fast, compliant interpreter of Python 2.7 and 3.5. Its just-in-time compiler brings a significant speed improvement over CPython.
* Stack less Python is a significant fork of CPython that implements micro threads; it does not use the C memory stack, thus allowing massively concurrent programs. PyPy also has a stackless version.
* MicroPython and CircuitPython are Python 3 variants optimised for microcontrollers. This includes Lego Mindstorms EV3.

**1.8 Python Line Structure**

A Python program is divided into a number of logical lines and every logical line is terminated by the token NEWLINE. A logical line is created from one or more physical lines. A line contains only spaces, tabs, form feeds possibly a comment, is known as a blank line, and Python interpreter ignores it.

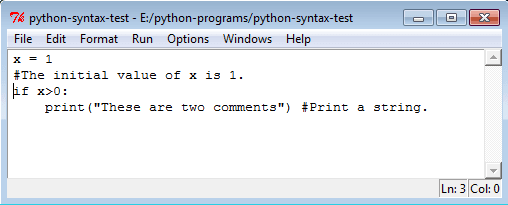
A physical line is a sequence of characters terminated by an end-of-line sequence (in windows it is called CR LF or return followed by a linefeed and in Unix, it is called LF or linefeed). See the following example.



**Fig 1.5 Physical Lines**

* **Comments in Python**

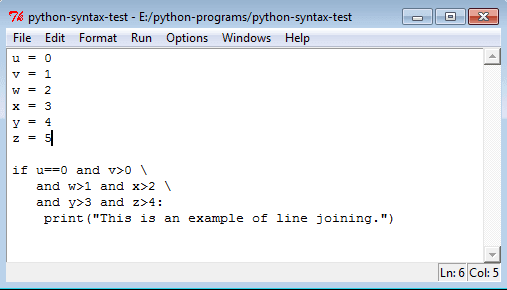
A comment begins with a hash character (#) which is not a part of the string literal and ends at the end of the physical line. All characters after the # character up to the end of the line are part of the comment and the Python interpreter ignores them. See the following example. It should be noted that Python has no multi-lines or block comments facility.



**Fig 1.6 Commenting in Python**

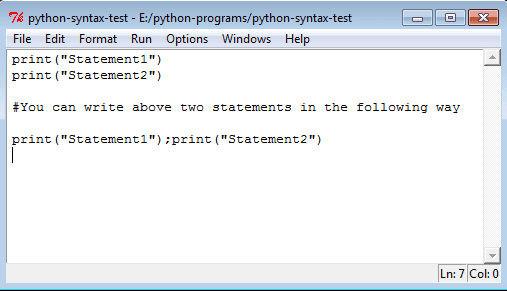
* **Joining two lines**

When you want to write a long code in a single line you can break the logical line in two or more physical lines using backslash character(\). Therefore when a physical line ends with a backslash characters (\) and not a part of a string literal or comment then it can join another physical line. See the following example.



**Fig 1.7 Line Joining**

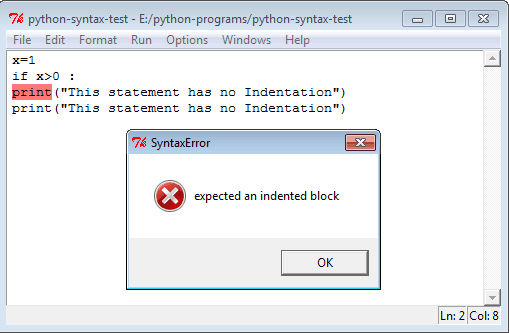
* **Multiple Statements on a Single Line**

You can write two separate statements into a single line using a semicolon (;) character between two line.

**Fig 1.8 Multiple Statements on a Single Line**

**1.9Indentation:**

Python uses whitespace (spaces and tabs) to define program blocks whereas other languages like C, C++ use braces ({}) to indicate blocks of codes for class, functions or flow control. The number of whitespaces (spaces and tabs) in the indentation is not fixed, but all statements within the block must be the indented same amount. In the following program, the block statements have no indentation.



**Fig 1.9 Indent Error**

**1.10 Python Coding Style:**

* Use 4 spaces per indentation and no tabs.
* Do not mix tabs and spaces. Tabs create confusion and it is recommended to use only spaces.
* Maximum line length: 79 characters which help users with a small display.
* Use blank lines to separate top-level function and class definitions and single blank line to separate methods definitions inside a class and larger blocks of code inside functions.
* When possible, put inline comments (should be complete sentences).
* Use spaces around expressions and statements.

**CHAPTER 2**

**INTRODUCTION TO DATA STRUCTURES AND ALGORITHMS**

**2.1 What is a Data Structure?**

Data Structures are the programmatic way of storing data so that data can be used efficiently. Almost every enterprise application uses various types of data structures in one or the other way. This tutorial will give you a great understanding on Data Structures needed to understand the complexity of enterprise level applications and need of algorithms, and data structures. Data Structure is a systematic way to organize data in order to use it efficiently. Following terms are the foundation terms of a data structure.

* **Interface** − each data structure has an interface. Interface represents the set of operations that a data structure supports. An interface only provides the list of supported operations, type of parameters they can accept and return type of these operations.
* **Implementation** − Implementation provides the internal representation of a data structure. Implementation also provides the definition of the algorithms used in the operations of the data structure.

**2.2 Why to Learn Data Structure and Algorithms?**

As applications are getting complex and data rich, there are three common problems that applications face now-a-days.

* **Data Search** − Consider an inventory of 1 million (106) items of a store. If the application is to search an item, it has to search an item in 1 million (106) items every time slowing down the search. As data grows, search will become slower.
* **Processor speed** − Processor speed although being very high, falls limited if the data grows to billion records.
* **Multiple requests** − as thousands of users can search data simultaneously on a web server, even the fast server fails while searching the data.

To solve the above-mentioned problems, data structures come to rescue. Data can be organized in a data structure in such a way that all items may not be required to be searched, and the required data can be searched almost instantly.

**2.3 Applications of Data Structure and Algorithms**

Algorithm is a step-by-step procedure, which defines a set of instructions to be executed in a certain order to get the desired output. Algorithms are generally created independent of underlying languages, i.e. an algorithm can be implemented in more than one programming language.

From the data structure point of view, following are some important categories of algorithms −

* **Search** − Algorithm to search an item in a data structure.
* **Sort** − Algorithm to sort items in a certain order.
* **Insert** − Algorithm to insert item in a data structure.
* **Update** − Algorithm to update an existing item in a data structure.
* **Delete** − Algorithm to delete an existing item from a data structure.

The following computer problems can be solved using Data Structures −

* Fibonacci number series
* Knapsack problem
* Tower of Hanoi
* All pair shortest path by Floyd-Warshall
* Shortest path by Dijkstra
* Project scheduling

## 2.4 Characteristics of a Data Structure

* **Correctness** − Data structure implementation should implement its interface correctly.
* **Time Complexity** − Running time or the execution time of operations of data structure must be as small as possible.
* **Space Complexity** − Memory usage of a data structure operation should be as little as possible.

## 2.5 Need for Data Structure

As applications are getting complex and data rich, there are three common problems that applications face now-a-days.

* **Data Search** − Consider an inventory of 1 million (106) items of a store. If the application is to search an item, it has to search an item in 1 million (106) items every time slowing down the search. As data grows, search will become slower.
* **Processor speed** − Processor speed although being very high, falls limited if the data grows to billion records.
* **Multiple requests** − as thousands of users can search data simultaneously on a web server, even the fast server fails while searching the data.

To solve the above-mentioned problems, data structures come to the rescue. Data can be organized in a data structure in such a way that all items may not be required to be searched, and the required data can be searched almost instantly.

## 2.6 Execution Time Cases

There are three cases which are usually used to compare various data structure's execution time in a relative manner.

* **Worst Case** − this is the scenario where a particular data structure operation takes maximum time it can take. If an operation's worst case time is ƒ(n) then this operation will not take more than ƒ(n) time where ƒ(n) represents the function of n.
* **Average Case** − this is the scenario depicting the average execution time of an operation of a data structure. If an operation takes ƒ(n) time in execution, then m operations will take mƒ(n) time.
* **Best Case** − this is the scenario depicting the least possible execution time of an operation of a data structure. If an operation takes ƒ(n) time in execution, then the actual operation may take time as the random number which would be maximum as ƒ(n).

## 2.7 Basic Terminology

* **Data** − Data are values or set of values.
* **Data Item** − Data item refers to single unit of values.
* **Group Items** − Data items that are divided into sub items are called as Group Items.
* **Elementary Items** − Data items that cannot be divided are called as Elementary Items.
* **Attribute and Entity** − an entity is that which contains certain attributes or properties, which may be assigned values.
* **Entity Set** − Entities of similar attributes form an entity set.
* **Field** − Field is a single elementary unit of information representing an attribute of an entity.
* **Record** − Record is a collection of field values of a given entity.
* **File** − File is a collection of records of the entities in a given entity set.

**2.8 What is an Algorithm?**

Algorithm is a step-by-step procedure, which defines a set of instructions to be executed in a certain order to get the desired output. Algorithms are generally created independent of underlying languages, i.e. an algorithm can be implemented in more than one programming language.

From the data structure point of view, following are some important categories of algorithms

* **Search** − Algorithm to search an item in a data structure.
* **Sort** − Algorithm to sort items in a certain order.
* **Insert** − Algorithm to insert item in a data structure.
* **Update** − Algorithm to update an existing item in a data structure.
* **Delete** − Algorithm to delete an existing item from a data structure.

## 2.9 Characteristics of an Algorithm

Not all procedures can be called an algorithm. An algorithm should have the following characteristics −

* **Unambiguous** − Algorithm should be clear and unambiguous. Each of its steps (or phases), and their inputs/outputs should be clear and must lead to only one meaning.
* **Input** − an algorithm should have 0 or more well-defined inputs.
* **Output** − an algorithm should have 1 or more well-defined outputs, and should match the desired output.
* **Finiteness** − Algorithms must terminate after a finite number of steps.
* **Feasibility** − should be feasible with the available resources.
* **Independent** − an algorithm should have step-by-step directions, which should be independent of any programming code.

## 2.10 How to Write an Algorithm?

There are no well-defined standards for writing algorithms. Rather, it is problem and resource dependent. Algorithms are never written to support a particular programming code.

As we know that all programming languages share basic code constructs like loops (do, for, while), flow-control (if-else), etc. These common constructs can be used to write an algorithm.

We write algorithms in a step-by-step manner, but it is not always the case. Algorithm writing is a process and is executed after the problem domain is well-defined. That is, we should know the problem domain, for which we are designing a solution.

**Example**

Let's try to learn algorithm-writing by using an example.

**Problem** − Design an algorithm to add two numbers and display the result.

**Step 1** − START

**Step 2** − declare three integers **a**, **b**&**c**

**Step 3** − define values of **a**&**b**

**Step 4** − add values of **a**&**b**

**Step 5** − store output of step 4 to **c**

**Step 6** − print **c**

**Step 7** − STOP

Algorithms tell the programmers how to code the program. Alternatively, the algorithm can be written as −

**Step 1** − START ADD

**Step 2** − get values of **a**&**b**

**Step 3** − c ← a + b

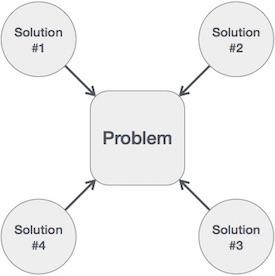
**Step 4** − display c

**Step 5** − STOP

In design and analysis of algorithms, usually the second method is used to describe an algorithm. It makes it easy for the analyst to analyze the algorithm ignoring all unwanted definitions. He can observe what operations are being used and how the process is flowing.

Writing step numbers, is optional.

We design an algorithm to get a solution of a given problem. A problem can be solved in more than one ways.



**Fig 2.1 How to write an algorithm**

Hence, many solution algorithms can be derived for a given problem. The next step is to analyze those proposed solution algorithms and implement the best suitable solution.

## 2.11 Algorithm Analysis

Efficiency of an algorithm can be analyzed at two different stages, before implementation and after implementation. They are the following −

* ***A Priori* Analysis** − this is a theoretical analysis of an algorithm. Efficiency of an algorithm is measured by assuming that all other factors, for example, processor speed, are constant and have no effect on the implementation.
* ***A Posterior* Analysis** − this is an empirical analysis of an algorithm. The selected algorithm is implemented using programming language. This is then executed on target computer machine. In this analysis, actual statistics like running time and space required, are collected.

We shall learn about *a priori* algorithm analysis. Algorithm analysis deals with the execution or running time of various operations involved. The running time of an operation can be defined as the number of computer instructions executed per operation.

## 2.12 Algorithm Complexity

Suppose **X** is an algorithm and **n** is the size of input data, the time and space used by the algorithm X are the two main factors, which decide the efficiency of X.

* **Time Factor** − Time is measured by counting the number of key operations such as comparisons in the sorting algorithm.
* **Space Factor** − Space is measured by counting the maximum memory space required by the algorithm.

The complexity of an algorithm **f(n)** gives the running time and/or the storage space required by the algorithm in terms of **n** as the size of input data.

## 2.12.1 Space Complexity

Space complexity of an algorithm represents the amount of memory space required by the algorithm in its life cycle. The space required by an algorithm is equal to the sum of the following two components −

* A fixed part that is a space required to store certain data and variables that are independent of the size of the problem. For example, simple variables and constants used, program size, etc.
* A variable part is a space required by variables, whose size depends on the size of the problem. For example, dynamic memory allocation, recursion stack space, etc.

Space complexity S(P) of any algorithm P is S(P) = C + SP(I), where C is the fixed part and S(I) is the variable part of the algorithm, which depends on instance characteristic I. Following is a simple example that tries to explain the concept −

Algorithm: SUM (A, B)

Step 1 - START

Step 2 - C ← A + B + 10

Step 3 - Stop

Here we have three variables A, B, and C and one constant. Hence S(P) = 1 + 3. Now, space depends on data types of given variables and constant types and it will be multiplied accordingly.

## 2.12.2 Time Complexity

Time complexity of an algorithm represents the amount of time required by the algorithm to run to completion. Time requirements can be defined as a numerical function T(n), where T(n) can be measured as the number of steps, provided each step consumes constant time.

For example, addition of two n-bit integers takes **n** steps. Consequently, the total computational time is T(n) = c ∗ n, where c is the time taken for the addition of two bits. Here, we observe that T(n) grows linearly as the input size increases.

**2.13 Asymptotic Analysis**

Asymptotic analysis of an algorithm refers to defining the mathematical boundation/framing of its run-time performance. Using asymptotic analysis, we can very well conclude the best case, average case, and worst case scenario of an algorithm.

Asymptotic analysis refers to computing the running time of any operation in mathematical units of computation. For example, the running time of one operation is computed as *f*(n) and may be for another operation it is computed as *g*(n2). This means the first operation running time will increase linearly with the increase in **n** and the running time of the second operation will increase exponentially when **n** increases.

Usually, the time required by an algorithm falls under three types −

* **Best Case** − Minimum time required for program execution.
* **Average Case** − Average time required for program execution.
* **Worst Case** − Maximum time required for program execution.

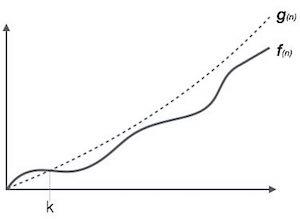
## 2.14 Asymptotic Notations

Following are the commonly used asymptotic notations to calculate the running time complexity of an algorithm.

* Ο Notation
* Ω Notation
* θ Notation

**Big Oh Notation, Ο**

The notation Ο(n) is the formal way to express the upper bound of an algorithm's running time. It measures the worst case time complexity or the longest amount of time an algorithm can possibly take to complete.



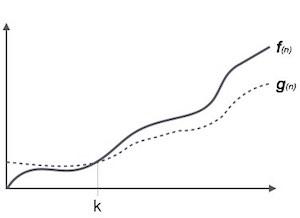
**Fig 2.2 Big O Notation**

For example, for a function ***f*(n)**

Ο(*f*(n)) = { *g*(n) : there exists c > 0 and n0 such that *f*(n) ≤ c.*g*(n) for all n > n0. }

**Omega Notation, Ω**

The notation Ω(n) is the formal way to express the lower bound of an algorithm's running time. It measures the best case time complexity or the best amount of time an algorithm can possibly take to complete.



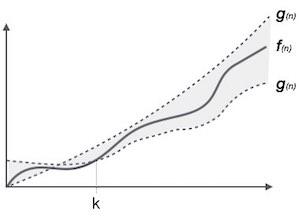
**Fig 2.3 Omega Notation**

For example, for a function ***f*(n)**

Ω(*f*(n)) ≥ { *g*(n) : there exists c > 0 and n0 such that *g*(n) ≤ c.*f*(n) for all n > n0. }

**Theta Notation, θ**

The notation θ(n) is the formal way to express both the lower bound and the upper bound of an algorithm's running time. It is represented as follows −



**Fig 2.4 Theta Notation**

θ(*f*(n)) = { *g*(n) if and only if *g*(n) = Ο(*f*(n)) and *g*(n) = Ω(*f*(n)) for all n > n0. }

## 2.15 Common Asymptotic Notations

|  |  |  |
| --- | --- | --- |
| constant | − | Ο(1) |
| logarithmic | − | Ο(log n) |
| linear | − | Ο(n) |
| n log n | − | Ο(n log n) |
| quadratic | − | Ο(n2) |
| cubic | − | Ο(n3) |
| polynomial | − | nΟ(1) |
| exponential | − | 2Ο(n) |

**2.16 Greedy Algorithms**

An algorithm is designed to achieve optimum solution for a given problem. In greedy algorithm approach, decisions are made from the given solution domain. As being greedy, the closest solution that seems to provide an optimum solution is chosen.

Greedy algorithms try to find a localized optimum solution, which may eventually lead to globally optimized solutions. However, generally greedy algorithms do not provide globally optimized solutions.

## 2.16.1 Counting Coins

This problem is to count to a desired value by choosing the least possible coins and the greedy approach forces the algorithm to pick the largest possible coin. If we are provided coins of ₹ 1, 2, 5 and 10 and we are asked to count ₹ 18 then the greedy procedure will be −

* **1** − Select one ₹ 10 coin, the remaining count is 8
* **2** − Then select one ₹ 5 coin, the remaining count is 3
* **3** − Then select one ₹ 2 coin, the remaining count is 1
* **4** − And finally, the selection of one ₹ 1 coins solves the problem

Though, it seems to be working fine, for this count we need to pick only 4 coins. But if we slightly change the problem then the same approach may not be able to produce the same optimum result.

For the currency system, where we have coins of 1, 7, 10 value, counting coins for value 18 will be absolutely optimum but for count like 15, it may use more coins than necessary. For example, the greedy approach will use 10 + 1 + 1 + 1 + 1 + 1, total 6 coins. Whereas the same problem could be solved by using only 3 coins (7 + 7 + 1)

Hence, we may conclude that the greedy approach picks an immediate optimized solution and may fail where global optimization is a major concern.

**Examples**

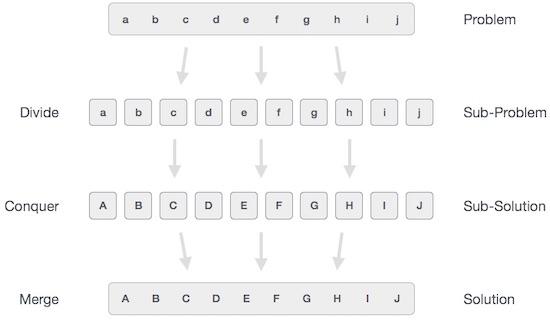
Most networking algorithms use the greedy approach. Here is a list of few of them −

* Travelling Salesman Problem
* Prim's Minimal Spanning Tree Algorithm
* Kruskal's Minimal Spanning Tree Algorithm
* Dijkstra's Minimal Spanning Tree Algorithm
* Graph - Map Coloring
* Graph - Vertex Cover
* Knapsack Problem
* Job Scheduling Problem

There are lots of similar problems that uses the greedy approach to find an optimum solution.

**2.17 Divide and Conquer**

In divide and conquer approach, the problem in hand, is divided into smaller sub-problems and then each problem is solved independently. When we keep on dividing the sub problems into even smaller sub-problems, we may eventually reach a stage where no more division is possible. Those "atomic" smallest possible sub-problem (fractions) are solved. The solution of all sub-problems is finally merged in order to obtain the solution of an original problem.



**Fig 2.5 Divivde and Conquer**

Broadly, we can understand **divide-and-conquer** approach in a three-step process.

## 2.17.1 Divide/Break

This step involves breaking the problem into smaller sub-problems. Sub-problems should represent a part of the original problem. This step generally takes a recursive approach to divide the problem until no sub-problem is further divisible. At this stage, sub-problems become atomic in nature but still represent some part of the actual problem.

## 2.17.2 Conquer/Solve

This step receives a lot of smaller sub-problems to be solved. Generally, at this level, the problems are considered 'solved' on their own.

## 2.17.3 Merge/Combine

When the smaller sub-problems are solved, this stage recursively combines them until they formulate a solution of the original problem. This algorithmic approach works recursively and conquer & merge steps works so close that they appear as one.

**Examples**

The following computer algorithms are based on **divide-and-conquer** programming approach −

* Merge Sort
* Quick Sort
* Binary Search
* Strassen's Matrix Multiplication
* Closest pair (points)

There are various ways available to solve any computer problem, but the mentioned are a good example of divide and conquer approach.

**2.18 Dynamic Programming**

Dynamic programming approach is similar to divide and conquer in breaking down the problem into smaller and yet smaller possible sub-problems. But unlike, divide and conquer, these sub-problems are not solved independently. Rather, results of these smaller sub-problems are remembered and used for similar or overlapping sub-problems.

Dynamic programming is used where we have problems, which can be divided into similar sub-problems, so that their results can be re-used. Mostly, these algorithms are used for optimization. Before solving the in-hand sub-problem, dynamic algorithm will try to examine the results of the previously solved sub-problems. The solutions of sub-problems are combined in order to achieve the best solution.

So we can say that −

* The problem should be able to be divided into smaller overlapping sub-problem.
* An optimum solution can be achieved by using an optimum solution of smaller sub-problems.
* Dynamic algorithms use Memoization.

## Comparison

In contrast to greedy algorithms, where local optimization is addressed, dynamic algorithms are motivated for an overall optimization of the problem.

In contrast to divide and conquer algorithms, where solutions are combined to achieve an overall solution, dynamic algorithms use the output of a smaller sub-problem and then try to optimize a bigger sub-problem. Dynamic algorithms use Memoization to remember the output of already solved sub-problems.

**Example**

The following computer problems can be solved using dynamic programming approach −

* Fibonacci number series
* Knapsack problem
* Tower of Hanoi
* All pair shortest path by Floyd-Warshall
* Shortest path by Dijkstra
* Project scheduling

Dynamic programming can be used in both top-down and bottom-up manner. And of course, most of the times, referring to the previous solution output is cheaper than recomputing in terms of CPU cycles.

**2.19 Linked Lists**

In computer science, a linked list is a linear collection of data elements whose order is not given by their physical placement in memory. Instead, each element points to the next. It is a data structure consisting of a collection of nodes which together represent a sequence. In its most basic form, each node contains: data, and a reference (in other words, a link) to the next node in the sequence. This structure allows for efficient insertion or removal of elements from any position in the sequence during iteration. More complex variants add additional links, allowing more efficient insertion or removal of nodes at arbitrary positions. A drawback of linked lists is that access time is linear (and difficult to pipeline). Faster access, such as random access, is not feasible. Arrays have better cache locality compared to linked lists.

Singly-linked-list.svg

**Fig 2.6 Representation of a Linked List**

A linked list whose nodes contain two fields: an integer value and a link to the next node. The last node is linked to a terminator used to signify the end of the list.

Linked lists are among the simplest and most common data structures. They can be used to implement several other common abstract data types, including lists, stacks, queues, associative arrays, and S-expressions, though it is not uncommon to implement those data structures directly without using a linked list as the basis.

The principal benefit of a linked list over a conventional array is that the list elements can be easily inserted or removed without reallocation or reorganization of the entire structure because the data items need not be stored contiguously in memory or on disk, while restructuring an array at run-time is a much more expensive operation. Linked lists allow insertion and removal of nodes at any point in the list, and allow doing so with a constant number of operations by keeping the link previous to the link being added or removed in memory during list traversal.

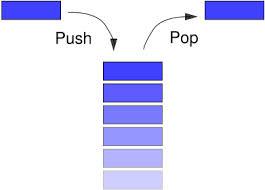
On the other hand, since simple linked lists by themselves do not allow random access to the data or any form of efficient indexing, many basic operations—such as obtaining the last node of the list, finding a node that contains a given datum, or locating the place where a new node should be inserted—may require iterating through most or all of the list elements. The advantages and disadvantages of using linked lists are given below. Linked list are dynamic, so the length of list can increase or decrease as necessary. Each node does not necessarily follow the previous one physically in the memory.

## 2.20 Stacks and Queues

### **Stack**

Stack is a container of objects that are inserted and removed according to the last-in first-out (LIFO) principle.

In the pushdown stacks only two operations are allowed: push the item into the stack, and pop the item out of the stack. A stack is a limited access data structure - elements can be added and removed from the stack only at the top. Push adds an item to the top of the stack, pop removes the item from the top. A helpful analogy is to think of a stack of books; you can remove only the top book, also you can add a new book on the top.

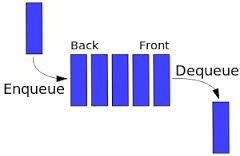


**Fig 2.7 Representation of a Stack**

### **Queue**

Queue is a container of objects (a linear collection) that are inserted and removed according to the first-in first-out (FIFO) principle.

An excellent example of a queue is a line of students in the food court of the UC. New additions to a line made to the back of the queue, while removal (or serving) happens in the front. In the queue only two operations are allowed enqueue and dequeue. Enqueue means to insert an item into the back of the queue, dequeue means removing the front item. The picture demonstrates the FIFO access. The difference between stacks and queues is in removing. In a stack we remove the item the most recently added; in a queue, we remove the item the least recently added.

**Fig 2.8 Representation of a Queue**

**2.21 Sorting Techniques**

Sorting refers to arranging data in a particular format. Sorting algorithm specifies the way to arrange data in a particular order. Most common orders are in numerical or lexicographical order.

The importance of sorting lies in the fact that data searching can be optimized to a very high level, if data is stored in a sorted manner. Sorting is also used to represent data in more readable formats. Following are some of the examples of sorting in real-life scenarios −

* **Telephone Directory** − the telephone directory stores the telephone numbers of people sorted by their names, so that the names can be searched easily.
* **Dictionary** − the dictionary stores words in an alphabetical order so that searching of any word becomes easy.

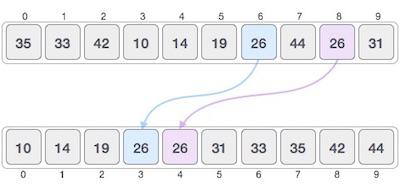
## 2.22 In-place Sorting and Not-in-place Sorting

Sorting algorithms may require some extra space for comparison and temporary storage of few data elements. These algorithms do not require any extra space and sorting is said to happen in-place, or for example, within the array itself. This is called **in-place sorting**. Bubble sort is an example of in-place sorting.

However, in some sorting algorithms, the program requires space which is more than or equal to the elements being sorted. Sorting which uses equal or more space is called **not-in-place sorting**. Merge-sort is an example of not-in-place sorting.

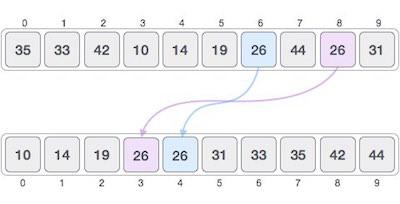
## 2.23 Stable and Not Stable Sorting

If a sorting algorithm, after sorting the contents, does not change the sequence of similar content in which they appear, it is called **stable sorting**.



**Fig 2.9 Stable Sorting**

If a sorting algorithm, after sorting the contents, changes the sequence of similar content in which they appear, it is called **unstable sorting**.



**Fig 2.10 Unstable Sorting**

Stability of an algorithm matters when we wish to maintain the sequence of original elements, like in a tuple for example.

## 2.24 Adaptive and Non-Adaptive Sorting Algorithm

A sorting algorithm is said to be adaptive, if it takes advantage of already 'sorted' elements in the list that is to be sorted. That is, while sorting if the source list has some element already sorted, adaptive algorithms will take this into account and will try not to re-order them.

A non-adaptive algorithm is one which does not take into account the elements which are already sorted. They try to force every single element to be re-ordered to confirm their sortedness.

## 2.25 Important Terms

Some terms are generally coined while discussing sorting techniques, here is a brief introduction to them −

**Increasing Order**

A sequence of values is said to be in **increasing order**, if the successive element is greater than the previous one. For example, 1, 3, 4, 6, 8, 9 are in increasing order, as every next element is greater than the previous element.

**Decreasing Order**

A sequence of values is said to be in **decreasing order**, if the successive element is less than the current one. For example, 9, 8, 6, 4, 3, 1 are in decreasing order, as every next element is less than the previous element.

**Non-Increasing Order**

A sequence of values is said to be in **non-increasing order**, if the successive element is less than or equal to its previous element in the sequence. This order occurs when the sequence contains duplicate values. For example, 9, 8, 6, 3, 3, 1 are in non-increasing order, as every next element is less than or equal to (in case of 3) but not greater than any previous element.

**Non-Decreasing Order**

A sequence of values is said to be in **non-decreasing order**, if the successive element is greater than or equal to its previous element in the sequence. This order occurs when the sequence contains duplicate values. For example, 1, 3, 3, 6, 8, 9 are in non-decreasing order, as every next element is greater than or equal to (in case of 3) but not less than the previous one.

**2.26 Bubble Sort**

Bubble sort is a simple sorting algorithm. This sorting algorithm is comparison-based algorithm in which each pair of adjacent elements is compared and the elements are swapped if they are not in order. This algorithm is not suitable for large data sets as its average and worst case complexity are of Ο(n2) where **n** is the number of items.

## 2.26.1 How Bubble Sort Works?

We take an unsorted array for our example. Bubble sort takes Ο(n2) time so we're keeping it short and precise.

Bubble Sort

**Fig 2.11 Bubble Sorting**

Bubble sort starts with very first two elements, comparing them to check which one is greater.

Bubble Sort

**Fig 2.12 Bubble Sorting**

In this case, value 33 is greater than 14, so it is already in sorted locations. Next, we compare 33 with 27.

Bubble Sort

**Fig 2.13 Bubble Sorting**

We find that 27 is smaller than 33 and these two values must be swapped.

Bubble Sort

**Fig 2.14 Bubble Sorting**

The new array should look like this −

Bubble Sort

**Fig 2.15 Bubble Sorting**

Next we compare 33 and 35. We find that both are in already sorted positions.

Bubble Sort

**Fig 2.16 Bubble Sorting**

Then we move to the next two values, 35 and 10.

Bubble Sort

**Fig 2.17 Bubble Sorting**

We know then that 10 is smaller 35. Hence they are not sorted.

Bubble Sort

**Fig 2.18 Bubble Sorting**

We swap these values. We find that we have reached the end of the array. After one iteration, the array should look like this −

Bubble Sort

**Fig 2.19 Bubble Sorting**

To be precise, we are now showing how an array should look like after each iteration. After the second iteration, it should look like this −

Bubble Sort

**Fig 2.20 Bubble Sorting**

Notice that after each iteration, at least one value moves at the end.

Bubble Sort

**Fig 2.21 Bubble Sorting**

And when there's no swap required, bubble sorts learns that an array is completely sorted.

Bubble Sort

**Fig 2.22 Bubble Sorting**

Now we should look into some practical aspects of bubble sort.

## Algorithm

beginBubbleSort(list)

for all elements of list

if list[i]> list[i+1]

swap(list[i], list[i+1])

endif

endfor

return list

endBubbleSort

**2.27 Insertion Sort**

This is an in-place comparison-based sorting algorithm. Here, a sub-list is maintained which is always sorted. For example, the lower part of an array is maintained to be sorted. An element which is to be inserted in this sorted sub-list, has to find its appropriate place and then it has to be inserted there. Hence the name, **insertion sort**.

The array is searched sequentially and unsorted items are moved and inserted into the sorted sub-list (in the same array). This algorithm is not suitable for large data sets as its average and worst case complexity are of Ο(n2), where **n** is the number of items.

## 2.27.1 How Insertion Sort Works?

We take an unsorted array for our example.

Unsorted Array

**Fig 2.23 Insertion Sorting**

Insertion sort compares the first two elements.

Insertion Sort

**Fig 2.24 Insertion Sorting**

It finds that both 14 and 33 are already in ascending order. For now, 14 is in sorted sub-list.

Insertion Sort

**Fig 2.25 Insertion Sorting**

Insertion sort moves ahead and compares 33 with 27.

Insertion Sort

**Fig 2.26 Insertion Sorting**

And finds that 33 is not in the correct position.

Insertion Sort

**Fig 2.27 Insertion Sorting**

It swaps 33 with 27. It also checks with all the elements of sorted sub-list. Here we see that the sorted sub-list has only one element 14, and 27 is greater than 14. Hence, the sorted sub-list remains sorted after swapping.

Insertion Sort

**Fig 2.28 Insertion Sorting**

By now we have 14 and 27 in the sorted sub-list. Next, it compares 33 with 10.

Insertion Sort

**Fig 2.29 Insertion Sorting**

These values are not in a sorted order.

Insertion Sort

**Fig 2.30 Insertion Sorting**

So we swap them.

Insertion Sort

**Fig 2.31 Insertion Sorting**

However, swapping makes 27 and 10 unsorted.

Insertion Sort

**Fig 2.32 Insertion Sorting**

Hence, we swap them too.

Insertion Sort

**Fig 2.33 Insertion Sorting**

Again we find 14 and 10 in an unsorted order.

Insertion Sort

**Fig 2.34 Insertion Sorting**

We swap them again. By the end of third iteration, we have a sorted sub-list of 4 items.

Insertion Sort

**Fig 2.35 Insertion Sorting**

This process goes on until all the unsorted values are covered in a sorted sub-list. Now we shall see some programming aspects of insertion sort.

**Algorithm**

Now we have a bigger picture of how this sorting technique works, so we can derive simple steps by which we can achieve insertion sort.

**Step 1** − If it is the first element, it is already sorted. return 1;

**Step 2** − Pick next element

**Step 3** − Compare with all elements in the sorted sub-list

**Step 4** − Shift all the elements in the sorted sub-list that is greater than the value to be sorted

**Step 5** − Insert the value

**Step 6** − Repeat until list is sorted

**2.28 Selection Sort**

Selection sort is a simple sorting algorithm. This sorting algorithm is an in-place comparison-based algorithm in which the list is divided into two parts, the sorted part at the left end and the unsorted part at the right end. Initially, the sorted part is empty and the unsorted part is the entire list.

The smallest element is selected from the unsorted array and swapped with the leftmost element, and that element becomes a part of the sorted array. This process continues moving unsorted array boundary by one element to the right.

This algorithm is not suitable for large data sets as its average and worst case complexities are of Ο(n2), where **n** is the number of items.

## 2.28.1 How Selection Sort Works?

Consider the following depicted array as an example.

Unsorted Array

**Fig 2.36 Selection Sorting**

For the first position in the sorted list, the whole list is scanned sequentially. The first position where 14 is stored presently, we search the whole list and find that 10 is the lowest value.

Selection Sort

**Fig 2.37 Selection Sorting**

So we replace 14 with 10. After one iteration 10, which happens to be the minimum value in the list, appears in the first position of the sorted list.

Selection Sort

**Fig 2.38 Selection Sorting**

For the second position, where 33 is residing, we start scanning the rest of the list in a linear manner.

Selection Sort

**Fig 2.39 Selection Sorting**

We find that 14 is the second lowest value in the list and it should appear at the second place. We swap these values.

Selection Sort

**Fig 2.40 Selection Sorting**

After two iterations, two least values are positioned at the beginning in a sorted manner.

Selection Sort

**Fig 2.41 Selection Sorting**

The same process is applied to the rest of the items in the array.

Following is a pictorial depiction of the entire sorting process −



**Fig 2.42 Selection Sorting**

Now, let us learn some programming aspects of selection sort.

**Algorithm**

**Step 1** − Set MIN to location 0

**Step 2** − Search the minimum element in the list

**Step 3** − Swap with value at location MIN

**Step 4** − Increment MIN to point to next element

**Step 5** − Repeat until list is sorted

**2.29 Merge Sort**

Merge sort is a sorting technique based on divide and conquer technique. With worst-case time complexity being Ο(n log n), it is one of the most respected algorithms.

Merge sort first divides the array into equal halves and then combines them in a sorted manner.

## 2.29.1 How Merge Sort Works?

To understand merge sort, we take an unsorted array as the following −

Unsorted Array

**Fig 2.43 Merge Sorting**

We know that merge sort first divides the whole array iteratively into equal halves unless the atomic values are achieved. We see here that an array of 8 items is divided into two arrays of size 4.

Merge Sort Division

**Fig 2.44 Merge Sorting**

This does not change the sequence of appearance of items in the original. Now we divide these two arrays into halves.

Merge Sort Division

**Fig 2.45 Merge Sorting**

We further divide these arrays and we achieve atomic value which can no more be divided.

Merge Sort Division

**Fig 2.46 Merge Sorting**

Now, we combine them in exactly the same manner as they were broken down. Please note the color codes given to these lists.

We first compare the element for each list and then combine them into another list in a sorted manner. We see that 14 and 33 are in sorted positions. We compare 27 and 10 and in the target list of 2 values we put 10 first, followed by 27. We change the order of 19 and 35 whereas 42 and 44 are placed sequentially.

Merge Sort Combine

**Fig 2.47 Merge Sorting**

In the next iteration of the combining phase, we compare lists of two data values, and merge them into a list of found data values placing all in a sorted order.

Merge Sort Combine

**Fig 2.48 Merge Sorting**

After the final merging, the list should look like this −

Merge Sort

**Fig 2.49 Merge Sorting**

Now we should learn some programming aspects of merge sorting.

**Algorithm**

Merge sort keeps on dividing the list into equal halves until it can no more be divided. By definition, if it is only one element in the list, it is sorted. Then, merge sort combines the smaller sorted lists keeping the new list sorted too.

**Step 1** − if it is only one element in the list it is already sorted, return.

**Step 2** − divide the list recursively into two halves until it can no more be divided.

**Step 3** − merge the smaller lists into new list in sorted order.

**2.30 Quick Sort**

Quick sort is a highly efficient sorting algorithm and is based on partitioning of array of data into smaller arrays. A large array is partitioned into two arrays one of which holds values smaller than the specified value, say pivot, based on which the partition is made and another array holds values greater than the pivot value.

Quicksort partitions an array and then calls itself recursively twice to sort the two resulting subarrays. This algorithm is quite efficient for large-sized data sets as its average and worst-case complexity are O(nLogn) and image.png(n2), respectively.

## 2.30.1 Quick Sort Pivot Algorithm

Based on our understanding of partitioning in quick sort, we will now try to write an algorithm for it, which is as follows.

**Step 1** − Choose the highest index value has pivot

**Step 2** − Take two variables to point left and right of the list excluding pivot

**Step 3** − left points to the low index

**Step 4** − right points to the high

**Step 5** − while value at left is less than pivot move right

**Step 6** − while value at right is greater than pivot move left

**Step 7** − if both step 5 and step 6 does not match swap left and right

**Step 8** − if left ≥ right, the point where they met is new pivot

## 2.30.2 Quick Sort Algorithm

Using pivot algorithm recursively, we end up with smaller possible partitions. Each partition is then processed for quick sort. We define recursive algorithm for quicksort as follows −

**Step 1** − Make the right-most index value pivot

**Step 2** − partition the array using pivot value

**Step 3** − quicksort left partition recursively

**Step 4** − quicksort right partition recursively

**2.31 Recursion**

Some computer programming languages allow a module or function to call itself. This technique is known as recursion. In recursion, a function **α** either calls itself directly or calls a function **β** that in turn calls the original function **α**. The function **α** is called recursive function.

**Example** − a function calling itself.

intfunction(intvalue){

if(value<1)

return;

function(value-1);

printf("%d ",value);

}

**2.31.1 Properties**

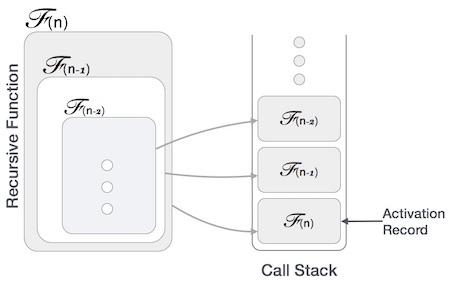
A recursive function can go infinite like a loop. To avoid infinite running of recursive function, there are two properties that a recursive function must have −

* **Base criteria** − There must be at least one base criteria or condition, such that, when this condition is met the function stops calling itself recursively.
* **Progressive approach** − the recursive calls should progress in such a way that each time a recursive call is made it comes closer to the base criteria.

**2.31.2 Implementation**

Many programming languages implement recursion by means of **stacks**. Generally, whenever a function (**caller**) calls another function (**callee**) or itself as callee, the caller function transfers execution control to the callee. This transfer process may also involve some data to be passed from the caller to the callee.

This implies, the caller function has to suspend its execution temporarily and resume later when the execution control returns from the callee function. Here, the caller function needs to start exactly from the point of execution where it puts itself on hold. It also needs the exact same data values it was working on. For this purpose, an activation record (or stack frame) is created for the caller function.



**Fig 2.50 Recursion**

This activation record keeps the information about local variables, formal parameters, return address and all information passed to the caller function.

**2.31.3 Analysis of Recursion**

One may argue why to use recursion, as the same task can be done with iteration. The first reason is, recursion makes a program more readable and because of latest enhanced CPU systems, recursion is more efficient than iterations.

**2.31.4 Time Complexity**

In case of iterations, we take number of iterations to count the time complexity. Likewise, in case of recursion, assuming everything is constant, we try to figure out the number of times a recursive call is being made. A call made to a function is Ο(1), hence the (n) number of times a recursive call is made makes the recursive function Ο(n).

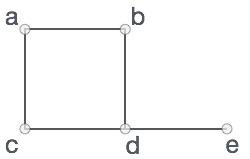
**2.31.5 Space Complexity**

Space complexity is counted as what amount of extra space is required for a module to execute. In case of iterations, the compiler hardly requires any extra space. The compiler keeps updating the values of variables used in the iterations. But in case of recursion, the system needs to store activation record each time a recursive call is made. Hence, it is considered that space complexity of recursive function may go higher than that of a function with iteration.

**2.32 Graph**

A graph is a pictorial representation of a set of objects where some pairs of objects are connected by links. The interconnected objects are represented by points termed as **vertices**, and the links that connect the vertices are called **edges**.

Formally, a graph is a pair of sets **(V, E)**, where **V** is the set of vertices and **E** is the set of edges, connecting the pairs of vertices. Take a look at the following graph −



**Fig 2.51 Undirected Graph**

In the above graph,

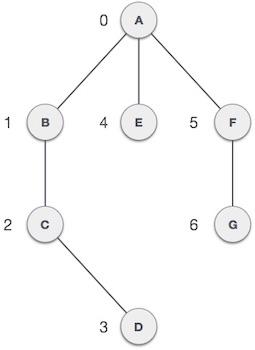
V = {a, b, c, d, e}

E = {ab, ac, bd, cd, de}

**2.32.1 Graph Data Structure**

Mathematical graphs can be represented in data structure. We can represent a graph using an array of vertices and a two-dimensional array of edges. Before we proceed further, let's familiarize ourselves with some important terms −

* **Vertex** − each node of the graph is represented as a vertex. In the following example, the labelled circle represents vertices. Thus, A to G are vertices. We can represent them using an array as shown in the following image. Here A can be identified by index 0. B can be identified using index 1 and so on.
* **Edge** − Edge represents a path between two vertices or a line between two vertices. In the following example, the lines from A to B, B to C, and so on represents edges. We can use a two-dimensional array to represent an array as shown in the following image. Here AB can be represented as 1 at row 0, column 1, BC as 1 at row 1, column 2 and so on, keeping other combinations as 0.
* **Adjacency** − Two node or vertices are adjacent if they are connected to each other through an edge. In the following example, B is adjacent to A, C is adjacent to B, and so on.
* **Path** − Path represents a sequence of edges between the two vertices. In the following example, ABCD represents a path from A to D.



**Fig 2.52 Graph**

**2.32.2 Basic Operations**

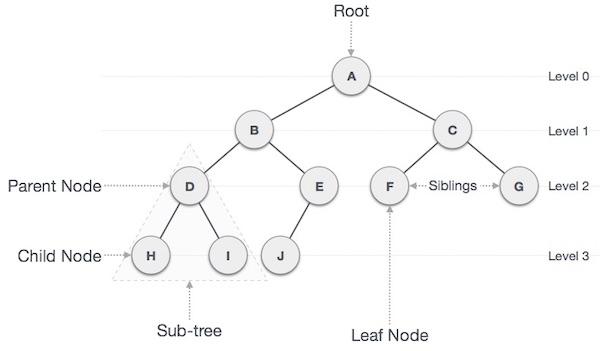
Following are basic primary operations of a Graph −

* **Add Vertex** − Adds a vertex to the graph.
* **Add Edge** − Adds an edge between the two vertices of the graph.
* **Display Vertex** − Displays a vertex of the graph.

**2.33 Trees**

Tree represents the nodes connected by edges. We will discuss binary tree or binary search tree specifically.

Binary Tree is a special data structure used for data storage purposes. A binary tree has a special condition that each node can have a maximum of two children. A binary tree has the benefits of both an ordered array and a linked list as search is as quick as in a sorted array and insertion or deletion operation are as fast as in linked list.



**Fig 2.53 Tree**

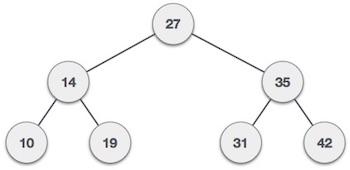
**2.33.1 Important Terms**

Following are the important terms with respect to tree.

* **Path** − Path refers to the sequence of nodes along the edges of a tree.
* **Root** − The node at the top of the tree is called root. There is only one root per tree and one path from the root node to any node.
* **Parent** − Any node except the root node has one edge upward to a node called parent.
* **Child** − The node below a given node connected by its edge downward is called its child node.
* **Leaf** − The node which does not have any child node is called the leaf node.
* **Subtree** − Subtree represents the descendants of a node.
* **Visiting** − Visiting refers to checking the value of a node when control is on the node.
* **Traversing** − Traversing means passing through nodes in a specific order.
* **Levels** − Level of a node represents the generation of a node. If the root node is at level 0, then its next child node is at level 1, its grandchild is at level 2, and so on.
* **Keys** − Key represents a value of a node based on which a search operation is to be carried out for a node.

**2.33.2 Binary Search Tree Representation**

Binary Search tree exhibits a special behavior. A node's left child must have a value less than its parent's value and the node's right child must have a value greater than its parent value.



**Fig 2.54 Binary Search Tree**

We're going to implement tree using node object and connecting them through references.

**2.33.3 Tree Node**

The code to write a tree node would be similar to what is given below. It has a data part and references to its left and right child nodes.

struct node {

int data;

struct node \*leftChild;

struct node \*rightChild;

};

In a tree, all nodes share common construct.

**2.33.4 BST Basic Operations**

The basic operations that can be performed on a binary search tree data structure, are the following −

* **Insert** − Inserts an element in a tree/create a tree.
* **Search** − Searches an element in a tree.
* **Preorder Traversal** − Traverses a tree in a pre-order manner.
* **Inorder Traversal** − Traverses a tree in an in-order manner.
* **Postorder Traversal** − Traverses a tree in a post-order manner.

**2.34 Heap**

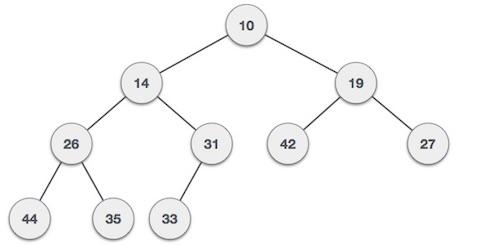
Heap is a special case of balanced binary tree data structure where the root-node key is compared with its children and arranged accordingly. If **α** has child node **β** then −

**key(α) ≥ key(β)**

As the value of parent is greater than that of child, this property generates **Max Heap**. Based on this criteria, a heap can be of two types −

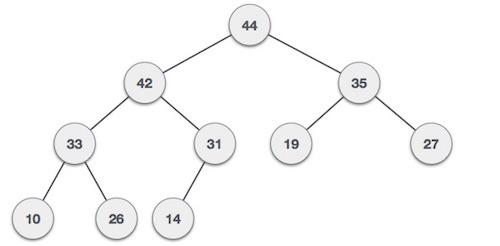
For Input → 35 33 42 10 14 19 27 44 26 31

**Min-Heap** − Where the value of the root node is less than or equal to either of its children.



**Fig 2.55 Min Heap**

**Max-Heap** − Where the value of the root node is greater than or equal to either of its children.



**Fig 2.56 Max Heap**

Both trees are constructed using the same input and order of arrival.

## 2.34.1 Max Heap Construction Algorithm

We shall use the same example to demonstrate how a Max Heap is created. The procedure to create Min Heap is similar but we go for min values instead of max values.

We are going to derive an algorithm for max heap by inserting one element at a time. At any point of time, heap must maintain its property. While insertion, we also assume that we are inserting a node in an already heapified tree.

**Step 1** − Create a new node at the end of heap.

**Step 2** − Assign new value to the node.

**Step 3** − Compare the value of this child node with its parent.

**Step 4** − If value of parent is less than child, then swap them.

**Step 5** − Repeat step 3 & 4 until Heap property holds.

**Note** − In Min Heap construction algorithm, we expect the value of the parent node to be less than that of the child node.

## 2.34.2 Max Heap Deletion Algorithm

Let us derive an algorithm to delete from max heap. Deletion in Max (or Min) Heap always happens at the root to remove the Maximum (or minimum) value.

**Step 1** − Remove root node.

**Step 2** − Move the last element of last level to root.

**Step 3** − Compare the value of this child node with its parent.

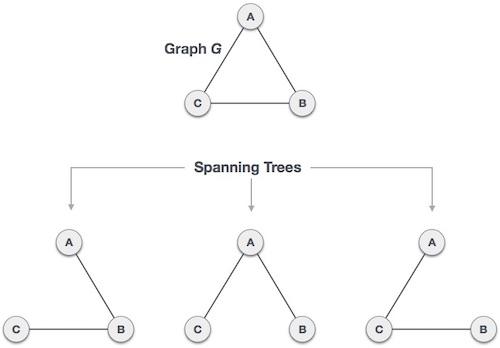
**Step 4** − If value of parent is less than child, then swap them.

**Step 5** − Repeat step 3& 4 until Heap property holds.

**2.35 Spanning Tree**

A spanning tree is a subset of Graph G, which has all the vertices covered with minimum possible number of edges. Hence, a spanning tree does not have cycles and it cannot be disconnected..

By this definition, we can draw a conclusion that every connected and undirected Graph G has at least one spanning tree. A disconnected graph does not have any spanning tree, as it cannot be spanned to all its vertices.



**Fig 2.57 Spanning Tree**

We found three spanning trees off one complete graph. A complete undirected graph can have maximum **nn-2** number of spanning trees, where **n** is the number of nodes. In the above addressed example, **n is 3,** hence **33−2 = 3** spanning trees are possible.

## 2.35.1 General Properties of Spanning Tree

We now understand that one graph can have more than one spanning tree. Following are a few properties of the spanning tree connected to graph G −

* A connected graph G can have more than one spanning tree.
* All possible spanning trees of graph G, have the same number of edges and vertices.
* The spanning tree does not have any cycle (loops).
* Removing one edge from the spanning tree will make the graph disconnected, i.e. the spanning tree is **minimally connected**.
* Adding one edge to the spanning tree will create a circuit or loop, i.e. the spanning tree is **maximally acyclic**.

## 2.35.2 Mathematical Properties of Spanning Tree

* Spanning tree has **n-1** edges, where **n** is the number of nodes (vertices).
* From a complete graph, by removing maximum **e - n + 1** edges, we can construct a spanning tree.
* A complete graph can have maximum **nn-2** number of spanning trees.

Thus, we can conclude that spanning trees are a subset of connected Graph G and disconnected graphs do not have spanning tree.

## 2.35.3 Application of Spanning Tree

Spanning tree is basically used to find a minimum path to connect all nodes in a graph. Common application of spanning trees are −

* **Civil Network Planning**
* **Computer Network Routing Protocol**
* **Cluster Analysis**

Let us understand this through a small example. Consider, city network as a huge graph and now plans to deploy telephone lines in such a way that in minimum lines we can connect to all city nodes. This is where the spanning tree comes into picture.

## 2.35.4 Minimum Spanning Tree (MST)

In a weighted graph, a minimum spanning tree is a spanning tree that has minimum weight than all other spanning trees of the same graph. In real-world situations, this weight can be measured as distance, congestion, traffic load or any arbitrary value denoted to the edges.

## 2.35.5 Minimum Spanning-Tree Algorithm

We shall learn about two most important spanning tree algorithms here −

* [Kruskal's Algorithm](https://www.tutorialspoint.com/data_structures_algorithms/kruskals_spanning_tree_algorithm.htm)
* [Prim's Algorithm](https://www.tutorialspoint.com/data_structures_algorithms/prims_spanning_tree_algorithm.htm)

**CHAPTER 3**

**CONVOLUTIONAL NEURAL NETWORKS**

**3.1 Convolutional Neural Network**

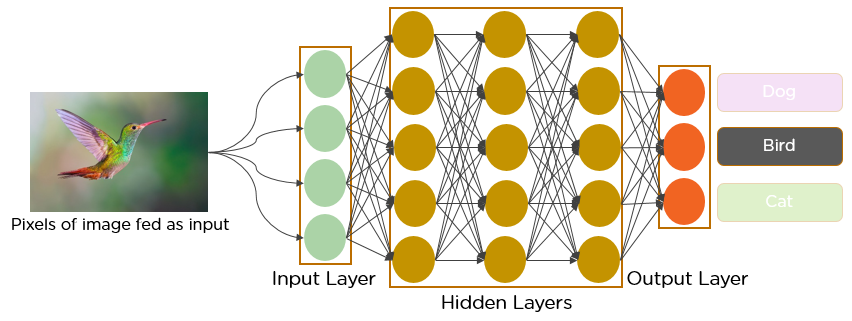
Artificial Intelligence has come a long way and has been seamlessly bridging the gap between the potential of humans and machines. And data enthusiasts all around the globe work on numerous aspects of [AI](https://www.simplilearn.com/tutorials/artificial-intelligence-tutorial/what-is-artificial-intelligence) and turn visions into reality - and one such amazing area is the domain of Computer Vision. This field aims to enable and configure machines to view the world as humans do, and use the knowledge for several tasks and processes (such as Image Recognition, Image Analysis and Classification, and so on). And the advancements in Computer Vision with Deep Learning have been a considerable success, particularly with the Convolutional Neural Network algorithm.

## 3.1.1 Introduction to CNN

Yann LeCun, director of Facebook’s Al Research Group, is the pioneer of convolutional neural networks. He built the first convolutional neural network called LeNet in 1988. LeNet was used for character recognition tasks like reading zip codes and digits.

Have you ever wondered how facial recognition works on social media, or how object detection helps in building self-driving cars, or how disease detection is done using visual imagery in healthcare? It’s all possible thanks to convolutional neural networks (CNN). Here’s an example of convolutional neural networks that illustrates how they work:

Imagine there’s an image of a bird, and you want to identify whether it’s really a bird or some other object. The first thing you do is feed the pixels of the image in the form of arrays to the input layer of the neural network (multi-layer networks used to classify things). The hidden layers carry out feature extraction by performing different calculations and manipulations. There are multiple hidden layers like the convolution layer, the ReLU layer, and pooling layer, that perform feature extraction from the image. Finally, there’s a fully connected layer that identifies the object in the image.

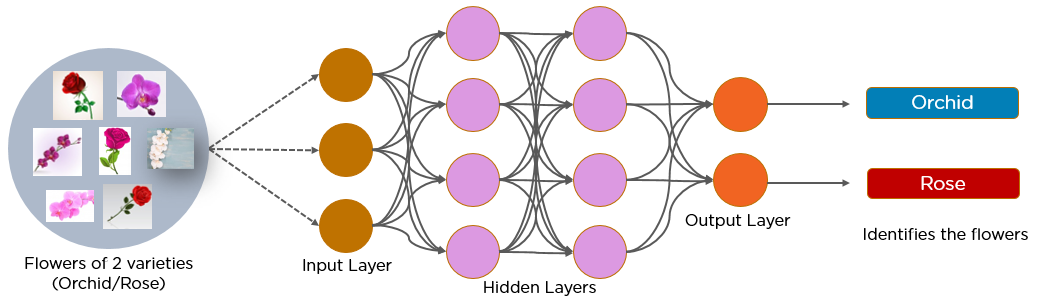


**Fig 3.1 CNN making prediction on a bird Image**

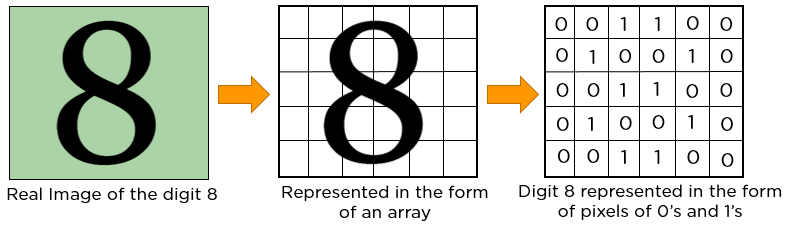
## 3.1.2 What is Convolutional Neural Network?

A convolutional neural network is a feed-forward neural network that is generally used to analyze visual images by processing data with grid-like topology. It’s also known as a ConvNet. A convolutional neural network is used to detect and classify objects in an image.

Below is a neural network that identifies two types of flowers: Orchid and Rose.



**Fig 3.2 CNN making prediction on Flower Images**

In CNN, every image is represented in the form of an array of pixel values.

**Fig 3.3 Representing 8 in pixels**

## 3.1.3 Layers in a Convolutional Neural Network

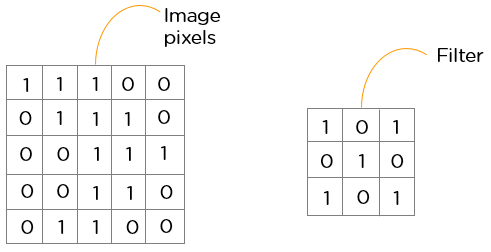
A convolution neural network has multiple hidden layers that help in extracting information from an image. The four important layers in CNN are:

1. Convolution layer
2. ReLU layer
3. Pooling layer
4. Fully connected layer

### **3.1.3.1 Convolution Layer**

This is the first step in the process of extracting valuable features from an image. A convolution layer has several filters that perform the convolution operation. Every image is considered as a matrix of pixel values.

Consider the following 5x5 image whose pixel values are either 0 or 1. There’s also a filter matrix with a dimension of 3x3. Slide the filter matrix over the image and compute the dot product to get the convolved feature matrix.

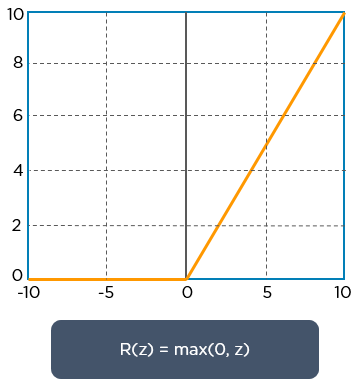


**Fig 3.4 Showing image and filter**

### **3.1.3.2 ReLU layer**

ReLU stands for the rectified linear unit. Once the feature maps are extracted, the next step is to move them to a ReLU layer.

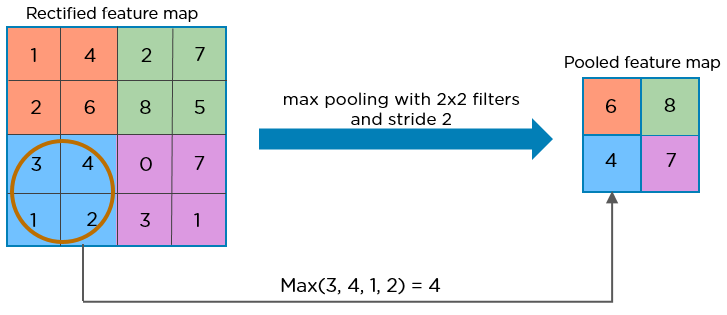
ReLU performs an element-wise operation and sets all the negative pixels to 0. It introduces non-linearity to the network, and the generated output is a **rectified feature map**. Below is the graph of a ReLU function:



**Fig 3.5 ReLU Activation Function**

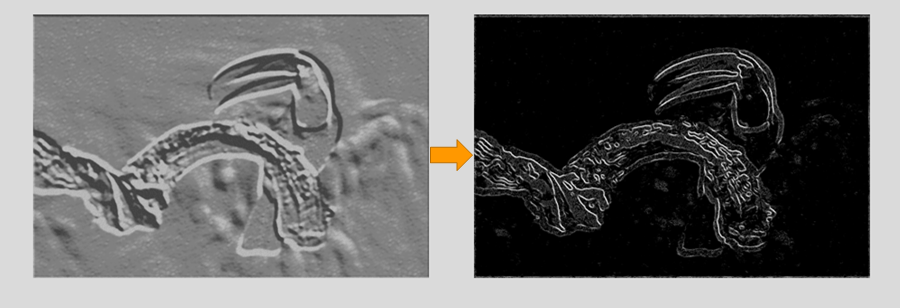
### **3.1.3.3 Pooling Layer**

Pooling is a down-sampling operation that reduces the dimensionality of the feature map. The rectified feature map now goes through a pooling layer to generate a pooled feature map.



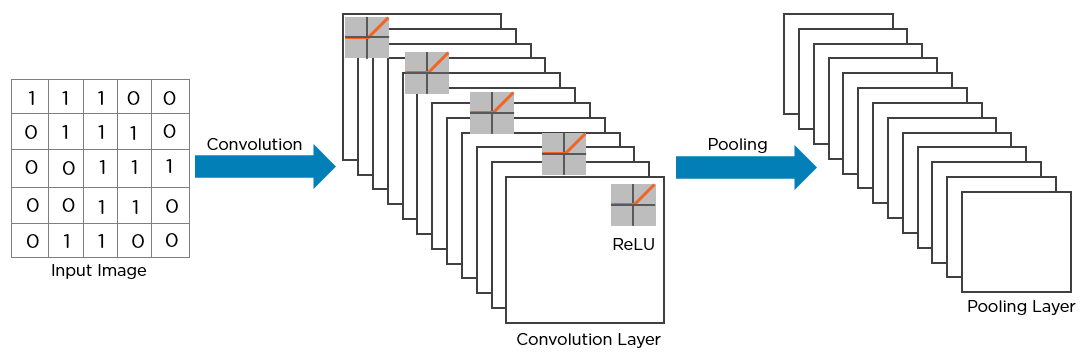
**Fig 3.6 Max Pooling**

The pooling layer uses various filters to identify different parts of the image like edges, corners, body, feathers, eyes, and beak.



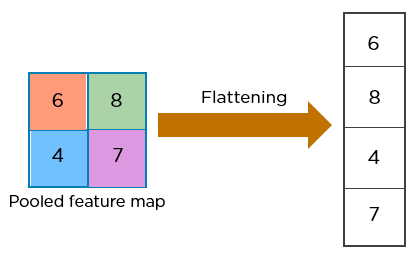
**Fig 3.7 Max Pooled Image**

Here’s how the structure of the convolution neural network looks so far:



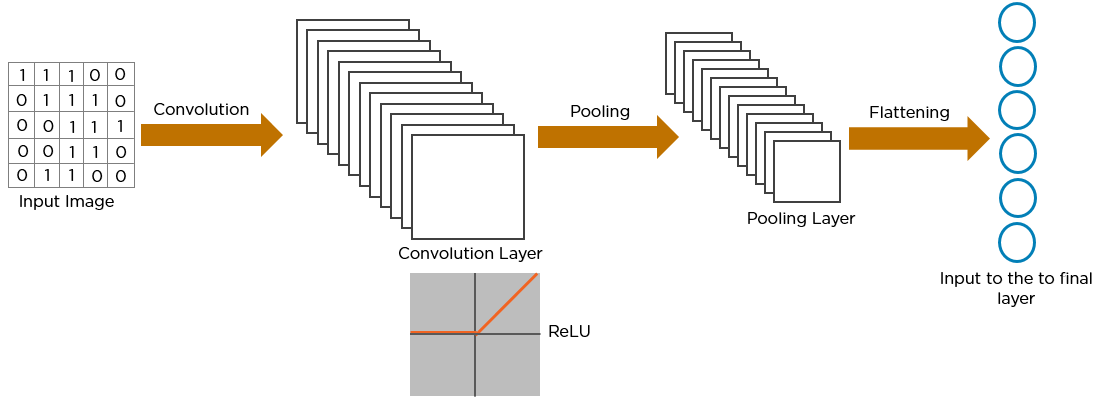
**Fig 3.8 CNN Structure**

The next step in the process is called **flattening**. Flattening is used to convert all the resultant 2-Dimensional arrays from pooled feature maps into a single long continuous linear vector.

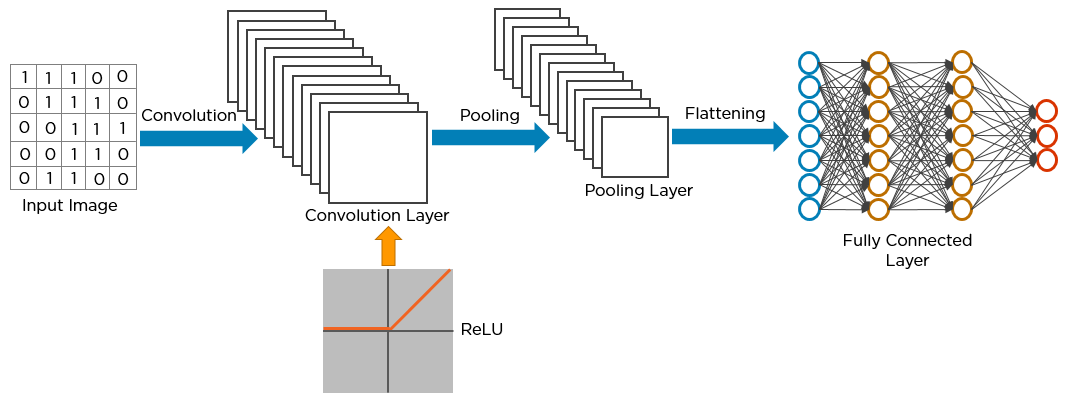


**Fig 3.9 Flattening the Pooled Image**

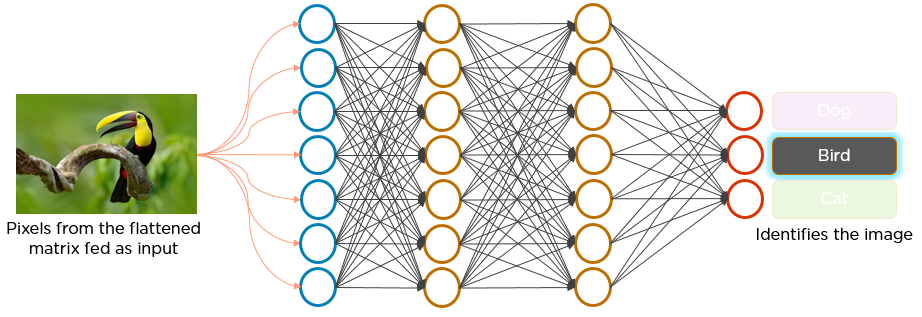
The flattened matrix is fed as input to the **fully connected layer** to classify the image.



**Fig 3.10 Flattened Image fed to Fully Connected Layer**



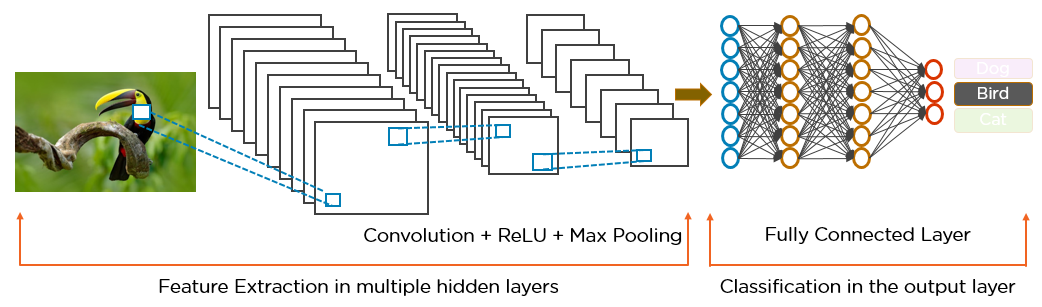
**Fig 3.11 Full CNN Structure**



**Fig 3.12 Feeding Bird Image to the CNN**

**Here’s how exactly CNN recognizes a bird:**

* The pixels from the image are fed to the convolutional layer that performs the convolution operation
* It results in a convolved map
* The convolved map is applied to a ReLU function to generate a rectified feature map
* The image is processed with multiple convolutions and ReLU layers for locating the features
* Different pooling layers with various filters are used to identify specific parts of the image
* The pooled feature map is flattened and fed to a fully connected layer to get the final output

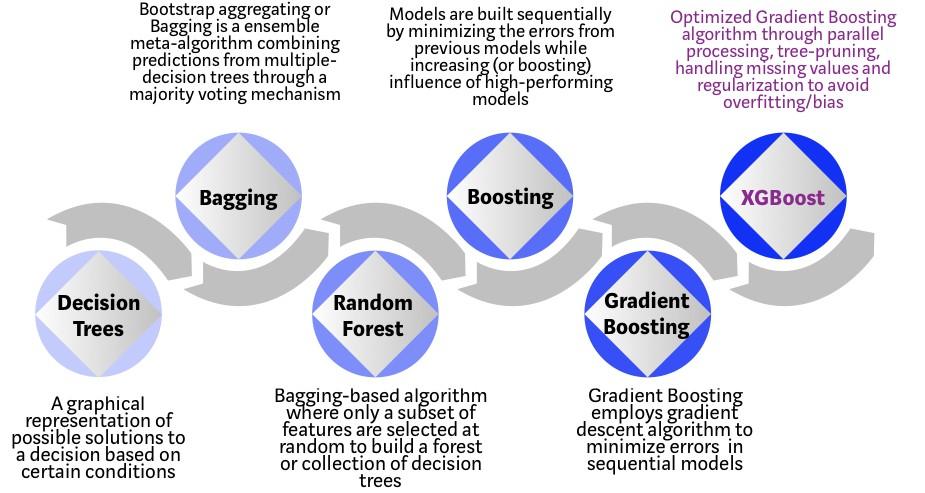


**Fig 3.13 Two Different parts of a CNN Structure**

# **3.2 XGBoost Algorithm**

# **3.2.1 What is XGBoost?**

[XGBoost](https://xgboost.ai/)is a decision-tree-based ensemble Machine Learning algorithm that uses a [gradient boosting](https://en.wikipedia.org/wiki/Gradient_boosting) framework. In prediction problems involving unstructured data (images, text, etc.) artificial neural networks tend to outperform all other algorithms or frameworks. However, when it comes to small-to-medium structured/tabular data, decision tree based algorithms are considered best-in-class right now. Please see the chart below for the evolution of tree-based algorithms over the years.



**Fig 3.14 Bagging and Boosting Techniques**

XGBoost algorithm was developed as a research project at the University of Washington. [Tianqi Chen and Carlos Guestrin](https://arxiv.org/pdf/1603.02754.pdf) presented their paper at SIGKDD Conference in 2016 and caught the Machine Learning world by fire. Since its introduction, this algorithm has not only been credited with winning numerous Kaggle competitions but also for being the driving force under the hood for several cutting-edge industry applications. As a result, there is a strong community of data scientists contributing to the XGBoost open source projects with ~350 contributors and ~3,600 commits on [GitHub](https://github.com/dmlc/xgboost/). The algorithm differentiates itself in the following ways:

1. A wide range of applications: Can be used to solve regression, classification, ranking, and user-defined prediction problems.
2. Portability: Runs smoothly on Windows, Linux, and OS X.
3. Languages: Supports all major programming languages including C++, Python, R, Java, Scala, and Julia.
4. Cloud Integration: Supports AWS, Azure, and Yarn clusters and works well with Flink, Spark, and other ecosystems.

**3.2.2 How to build an intuition for XGBoost?**

Decision trees, in their simplest form, are easy-to-visualize and fairly interpretable algorithms but building intuition for the next-generation of tree-based algorithms can be a bit tricky. See below for a simple analogy to better understand the evolution of tree-based algorithms.

Imagine that you are a hiring manager interviewing several candidates with excellent qualifications. Each step of the evolution of tree-based algorithms can be viewed as a version of the interview process.

1. Decision Tree: Every hiring manager has a set of criteria such as education level, number of years of experience, interview performance. A decision tree is analogous to a hiring manager interviewing candidates based on his or her own criteria.

2. Bagging: Now imagine instead of a single interviewer, now there is an interview panel where each interviewer has a vote. Bagging or bootstrap aggregating involves combining inputs from all interviewers for the final decision through a democratic voting process.

3. Random Forest: It is a bagging-based algorithm with a key difference wherein only a subset of features is selected at random. In other words, every interviewer will only test the interviewee on certain randomly selected qualifications (e.g. a technical interview for testing programming skills and a behavioral interview for evaluating non-technical skills).

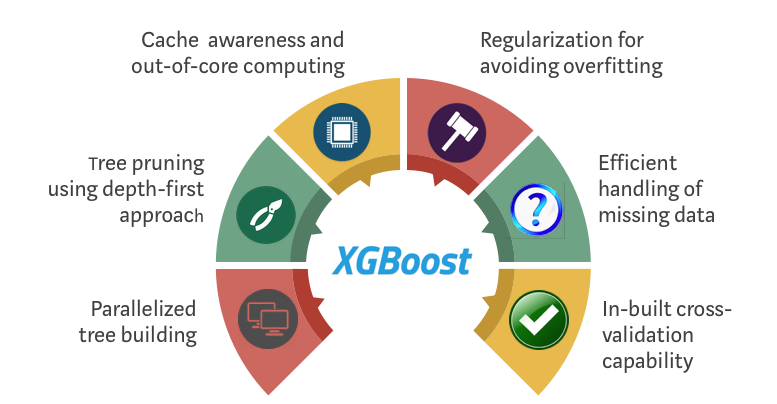
4. Boosting: This is an alternative approach where each interviewer alters the evaluation criteria based on feedback from the previous interviewer. This ‘boosts’ the efficiency of the interview process by deploying a more dynamic evaluation process.

5. Gradient Boosting: A special case of boosting where errors are minimized by gradient descent algorithm e.g. the strategy consulting firms leverage by using case interviews to weed out less qualified candidates.

6. XGBoost: Think of XGBoost as gradient boosting on ‘steroids’ (well it is called ‘Extreme Gradient Boosting’ for a reason!). It is a perfect combination of software and hardware optimization techniques to yield superior results using less computing resources in the shortest amount of time.

**3.2.3 Why does XGBoost perform so well?**

XGBoost and Gradient Boosting Machines (GBMs) are both ensemble tree methods that apply the principle of boosting weak learners ([CARTs](https://www.datasciencecentral.com/profiles/blogs/introduction-to-classification-regression-trees-cart) generally) using the gradient descent architecture. However, XGBoost improves upon the base GBM framework through systems optimization and algorithmic enhancements.



**Fig 3.15 XGBoost**

**3.2.4 How XGBoost optimizes standard GBM algorithm**

**3.2.4.1 System Optimization:**

**Parallelization**: XGBoost approaches the process of sequential tree building using [parallelized](http://zhanpengfang.github.io/418home.html) implementation. This is possible due to the interchangeable nature of loops used for building base learners; the outer loop that enumerates the leaf nodes of a tree, and the second inner loop that calculates the features. This nesting of loops limits parallelization because without completing the inner loop (more computationally demanding of the two), the outer loop cannot be started. Therefore, to improve run time, the order of loops is interchanged using initialization through a global scan of all instances and sorting using parallel threads. This switch improves algorithmic performance by offsetting any parallelization overheads in computation.

**Tree Pruning**: The stopping criterion for tree splitting within GBM framework is greedy in nature and depends on the negative loss criterion at the point of split. XGBoost uses ‘max\_depth’ parameter as specified instead of criterion first, and starts pruning trees backward. This ‘depth-first’ approach improves computational performance significantly.

**Hardware Optimization**: This algorithm has been designed to make efficient use of hardware resources. This is accomplished by cache awareness by allocating internal buffers in each thread to store gradient statistics. Further enhancements such as ‘out-of-core’ computing optimize available disk space while handling big data-frames that do not fit into memory.

**3.2.4.2 Algorithmic Enhancements:**

**1. Regularization:** It penalizes more complex models through both LASSO (L1) and Ridge (L2) [regularization](https://towardsdatascience.com/l1-and-l2-regularization-methods-ce25e7fc831c) to prevent overfitting.

**2. Sparsity Awareness:** XGBoost naturally admits sparse features for inputs by automatically ‘learning’ best missing value depending on training loss and handles different types of [sparsity patterns](https://www.kdnuggets.com/2017/10/xgboost-concise-technical-overview.html) in the data more efficiently.

**3. Weighted Quantile Sketch**: XGBoost employs the distributed [weighted Quantile Sketch algorithm](https://arxiv.org/pdf/1603.02754.pdf) to effectively find the optimal split points among weighted datasets.

**4. Cross-validation**: The algorithm comes with built-in [cross-validation](https://towardsdatascience.com/cross-validation-in-machine-learning-72924a69872f) method at each iteration, taking away the need to explicitly program this search and to specify the exact number of boosting iterations required in a single run.

## 3.3 Bagging and Random Forest Ensemble Algorithms for Machine Learning

Random Forest is one of the most popular and most powerful machine learning algorithms. It is a type of [ensemble machine learning algorithm](https://machinelearningmastery.com/ensemble-methods-for-deep-learning-neural-networks/) called Bootstrap Aggregation or bagging.

In this post you will discover the Bagging ensemble algorithm and the Random Forest algorithm for predictive modeling. After reading this post you will know about:

1. The bootstrap method for estimating statistical quantities from samples.

2. The Bootstrap Aggregation algorithm for creating multiple different models from a single training dataset.

3. The Random Forest algorithm that makes a small tweak to Bagging and results in a very powerful classifier.

## 3.3.1 Bootstrap Method

Before we get to Bagging, let’s take a quick look at an important foundation technique called the bootstrap.

The bootstrap is a powerful statistical method for estimating a quantity from a data sample. This is easiest to understand if the quantity is a descriptive statistic such as a mean or a standard deviation.

Let’s assume we have a sample of 100 values (x) and we’d like to get an estimate of the mean of the sample.

We can calculate the mean directly from the sample as:

mean(x) = 1/100 \* sum(x)

We know that our sample is small and that our mean has error in it. We can improve the estimate of our mean using the bootstrap procedure:

1. Create many (e.g. 1000) random sub-samples of our dataset with replacement (meaning we can select the same value multiple times).

2. Calculate the mean of each sub-sample.

3. Calculate the average of all of our collected means and use that as our estimated mean for the data.

For example, let’s say we used 3 resamples and got the mean values 2.3, 4.5 and 3.3. Taking the average of these we could take the estimated mean of the data to be 3.367.

This process can be used to estimate other quantities like the standard deviation and even quantities used in machine learning algorithms, like learned coefficients.

## 3.3.2 Bootstrap Aggregation (Bagging)

Bootstrap Aggregation (or Bagging for short), is a simple and very powerful ensemble method.

An ensemble method is a technique that combines the predictions from multiple machine learning algorithms together to make more accurate predictions than any individual model.

Bootstrap Aggregation is a general procedure that can be used to reduce the variance for those algorithm that have high variance. An algorithm that has high variance are decision trees, like classification and regression trees (CART).

Decision trees are sensitive to the specific data on which they are trained. If the training data is changed (e.g. a tree is trained on a subset of the training data) the resulting decision tree can be quite different and in turn the predictions can be quite different.

Bagging is the application of the Bootstrap procedure to a high-variance machine learning algorithm, typically decision trees.

Let’s assume we have a sample dataset of 1000 instances (x) and we are using the CART algorithm. Bagging of the CART algorithm would work as follows.

1. Create many (e.g. 100) random sub-samples of our dataset with replacement.

2. Train a CART model on each sample.

3. Given a new dataset, calculate the average prediction from each model

Ensemble learning, in general, is a model that makes predictions based on a number of different models. By combining individual models, the ensemble model tends to be more flexible(less bias) and less data-sensitive(less variance).

Two most popular ensemble methods are bagging and boosting.

**1. Bagging**: Training a bunch of individual models in a parallel way. Each model is trained by a random subset of the data

**2. Boosting**:Training a bunch of individual models in a sequential way. Each individual model learns from mistakes made by the previous model.

**3.4 Random Forest**

Random forest is an ensemble model using bagging as the ensemble method and decision tree as the individual model.

Let’s take a closer look at the magic of the randomness:

Step 1: Select n (e.g. 1000) random subsets from the training set

Step 2: Train n (e.g. 1000) decision trees

1. One random subset is used to train one decision tree

2. The optimal splits for each decision tree are based on a random subset of features (e.g. 10 features in total, randomly select 5 out of 10 features to split)

Step 3: Each individual tree predicts the records/candidates in the test set, independently.

Step 4: Make the final prediction

For each candidate in the test set, Random Forest uses the class (e.g. cat or dog) with the majority vote as this candidate’s final prediction.

**3.4.1 AdaBoost (Adaptive Boosting)**

AdaBoost is a boosting ensemble model and works especially well with the decision tree. Boosting model’s key is learning from the previous mistakes, e.g. misclassification data points.

AdaBoost learns from the mistakes by increasing the weight of misclassified data points.

Let’s illustrate how AdaBoost adapts.

**Step 0**: Initialize the weights of data points. if the training set has 100 data points, then each point’s initial weight should be 1/100 = 0.01.

**Step 1**: Train a decision tree

**Step 2**: Calculate the weighted error rate (e) of the decision tree. The weighted error rate (e) is just how many wrong predictions out of total and you treat the wrong predictions differently based on its data point’s weight. The higher the weight, the more the corresponding error will be weighted during the calculation of the (e).

**Step 3**: Calculate this decision tree’s weight in the ensemble

The weight of this tree = learning rate \* log ((1 — e) / e)

1. The higher weighted error rate of a tree, the less decision power the tree will be given during the later voting

2. The lower weighted error rate of a tree, the higher decision power the tree will be given during the later voting

**Step 4**: Update weights of wrongly classified points

The weight of each data point =

1. If the model got this data point correct, the weight stays the same

2. If the model got this data point wrong, the new weight of this point = old weight \* np.exp(weight of this tree)

Note: The higher the weight of the tree (more accurate this tree performs), the more boost (importance) the misclassified data point by this tree will get. The weights of the data points are normalized after all the misclassified points are updated.

**Step 5**: Repeat Step 1(until the number of trees we set to train is reached)

**Step 6**: Make the final prediction

The AdaBoost makes a new prediction by adding up the weight (of each tree) multiply the prediction (of each tree). Obviously, the tree with higher weight will have more power of influence the final decision.

**3.4.2 Gradient Boosting**

Gradient boosting is another boosting model. Remember, boosting model’s key is learning from the previous mistakes.

Gradient Boosting learns from the mistake — residual error directly, rather than update the weights of data points.

Let’s illustrate how Gradient Boost learns.

**Step 1**: Train a decision tree

**Step 2**: Apply the decision tree just trained to predict

**Step 3**: Calculate the residual of this decision tree, save residual errors as the new y

**Step 4**: Repeat Step 1 (until the number of trees we set to train is reached)

**Step 5**: Make the final prediction

The Gradient Boosting makes a new prediction by simply adding up the predictions (of all trees).

**CHAPTER 4**

**DISEASES**

**4.1. Covid-19**

Coronaviruses are a family of viruses that can cause illnesses such as the common cold, severe acute respiratory syndrome (SARS) and Middle East respiratory syndrome (MERS). In 2019, a new coronavirus was identified as the cause of a disease outbreak that originated in China.

The virus is now known as the severe acute respiratory syndrome coronavirus 2 (SARS-CoV-2). The disease it causes is called coronavirus disease 2019 (COVID-19). In March 2020, the World Health Organization (WHO) declared the COVID-19 outbreak a pandemic.

Public health groups, including the U.S. Centers for Disease Control and Prevention (CDC) and WHO, are monitoring the pandemic and posting updates on their websites. These groups have also issued recommendations for preventing and treating the illness.

**4.1.1 Symptoms**

Signs and symptoms of coronavirus disease 2019 (COVID-19) may appear two to 14 days after exposure. This time after exposure and before having symptoms is called the incubation period. Common signs and symptoms can include:

* Fever
* Cough
* Tiredness

Early symptoms of COVID-19 may include a loss of taste or smell.

Other symptoms can include:

* Shortness of breath or difficulty breathing
* Muscle aches
* Chills
* Sore throat
* Runny nose
* Headache
* Chest pain
* Pink eye (conjunctivitis)
* Nausea
* Vomiting
* Diarrhea
* Rash

This list is not all inclusive. Children have similar symptoms to adults and generally have mild illness.

The severity of COVID-19 symptoms can range from very mild to severe. Some people may have only a few symptoms, and some people may have no symptoms at all. Some people may experience worsened symptoms, such as worsened shortness of breath and pneumonia, about a week after symptoms start.

People who are older have a higher risk of serious illness from COVID-19, and the risk increases with age. People who have existing medical conditions also may have a higher risk of serious illness. Certain medical conditions that may increase the risk of serious illness from COVID-19 include:

* Serious heart diseases, such as heart failure, coronary artery disease or cardiomyopathy
* Cancer
* Chronic obstructive pulmonary disease (COPD)
* Type 1 or type 2 diabetes
* Overweight, obesity or severe obesity
* High blood pressure
* Smoking
* Chronic kidney disease
* Sickle cell disease or thalassemia
* Weakened immune system from solid organ transplants
* Pregnancy
* Asthma
* Chronic lung diseases such as cystic fibrosis or pulmonary fibrosis
* Liver disease
* Dementia
* Down syndrome
* Weakened immune system from bone marrow transplant, HIV or some medications
* Brain and nervous system conditions
* Substance use disorders

## 4.1.2 Causes

Infection with the new coronavirus (severe acute respiratory syndrome coronavirus 2, or SARS-CoV-2) causes coronavirus disease 2019 (COVID-19).

The virus that causes COVID-19 spreads easily among people, and more continues to be discovered over time about how it spreads. Data has shown that it spreads mainly from person to person among those in close contact (within about 6 feet, or 2 meters). The virus spreads by respiratory droplets released when someone with the virus coughs, sneezes, breathes, sings or talks. These droplets can be inhaled or land in the mouth, nose or eyes of a person nearby.

In some situations, the COVID-19 virus can spread by a person being exposed to small droplets or aerosols that stay in the air for several minutes or hours — called airborne transmission. It's not yet known how common it is for the virus to spread this way.

It can also spread if a person touches a surface or object with the virus on it and then touches his or her mouth, nose or eyes, but the risk is low.

Some reinfections of the virus that causes COVID-19 have happened, but these have been uncommon.

**4.1.3 Risk factors**

Risk factors for COVID-19 appear to include:

* Close contact (within 6 feet, or 2 meters) with someone who has COVID-19
* Being coughed or sneezed on by an infected person

### **4.1.4 Diagnosis**

Tests for etiological diagnosis may be direct, identifying genetic material of SARS-CoV-2, or indirect, determining the humoral immune response to SARS-CoV-2.

The most commonly used method for identifying genetic material from SARS-CoV-2 is real-time polymerase chain reaction (RT-PCR). This method involves reverse transcription of the genetic material of the virus (RNA) to complementary DNA (cDNA), followed by amplification of some regions of the cDNA. Probes (DNA/RNA marked sequences to identify the genetic target in the material) and primers (DNA/RNA sequences that promote replication of the genetic material found in the sample) were created after the SARS-CoV-2 genome was sequenced. Several serial amplification cycles are performed to identify these targets: the more cycles are needed, the lower the viral load of the material under study.[8](https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7456621/#bib0040)

Four regions of the SARS-CoV-2 genome have been targeted: RdRp gene (RNA-dependent RNA polymerase), genes from structural proteins E (virus envelope) and N (virus nucleocapsid), and ORF1ab gene (open reading frame 1a and 1b).[3](https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7456621/#bib0015),[8](https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7456621/#bib0040) Kits using different regions of the genome are commercially available. The sequential use of different probes and primers for the RdRp, E and N genes, known as the Charité-Berlin Institute protocol, presents good sensitivity and specificity.[9](https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7456621/#bib0045) There are other proposed protocols that follow the same logic of sequential use of probes and primes for different genetic targets.[10](https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7456621/#bib0050)

Regardless of the method used, the sensitivity and specificity of the different RT-PCR kits are not 100%. This is considered the gold standard for diagnosis of SARS-CoV-2 infection, but its sensitivity is estimated to be approximately 70% and specificity, 95%.[11](https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7456621/#bib0055),[12](https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7456621/#bib0060) Many factors can interfere with the results, whether related to the virus, to the method itself (the collection procedure and handling of the material), or even to the viral load of the sample (type of material collected, duration of symptoms, and disease severity).[13](https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7456621/#bib0065)

Mutations in the virus genome can render the probes and primers obsolete, producing false negative results. To date, SARS-CoV-2 has undergone mutations, but without implications for the RT-PCR detection. Mismatch between primers and probes can also lead to false negative results, and ideally more than one region of the virus genome should be simultaneously or sequentially amplified.[13](https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7456621/#bib0065)

Factors related to the collection procedure and handling of the material are often responsible for false negative results. Dacron or polyester swabs should be used and immersed immediately after collection in appropriate and refrigerated storage medium. The material should be kept under refrigeration and quickly sent to the laboratory.[10](https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7456621/#bib0050),[13](https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7456621/#bib0065)

**4.2 Brain Tumor**

A brain tumor is a mass or growth of abnormal cells in your brain.

Many different types of brain tumors exist. Some brain tumors are noncancerous (benign), and some brain tumors are cancerous (malignant). Brain tumors can begin in your brain (primary brain tumors), or cancer can begin in other parts of your body and spread to your brain (secondary, or metastatic, brain tumors).

How quickly a brain tumor grows can vary greatly. The growth rate as well as location of a brain tumor determines how it will affect the function of your nervous system.

Brain tumor treatment options depend on the type of brain tumor you have, as well as its size and location.

**4.2.1 Symptoms**

The signs and symptoms of a brain tumor vary greatly and depend on the brain tumor's size, location and rate of growth.

General signs and symptoms caused by brain tumors may include:

* New onset or change in pattern of headaches
* Headaches that gradually become more frequent and more severe
* Unexplained nausea or vomiting
* Vision problems, such as blurred vision, double vision or loss of peripheral vision
* Gradual loss of sensation or movement in an arm or a leg
* Difficulty with balance
* Speech difficulties
* Confusion in everyday matters
* Personality or behavior changes
* Seizures, especially in someone who doesn't have a history of seizures
* Hearing problems

## 4.2.2 Causes

### Brain tumors that begin in the brain Primary brain tumors originate in the brain itself or in tissues close to it, such as in the brain-covering membranes (meninges), cranial nerves, pituitary gland or pineal gland.

Primary brain tumors begin when normal cells acquire errors (mutations) in their DNA. These mutations allow cells to grow and divide at increased rates and to continue living when healthy cells would die. The result is a mass of abnormal cells, which forms a tumor.

In adults, primary brain tumors are much less common than are secondary brain tumors, in which cancer begins elsewhere and spreads to the brain.

Many different types of primary brain tumors exist. Each gets its name from the type of cells involved. Examples include:

* **Gliomas.** These tumors begin in the brain or spinal cord and include astrocytomas, ependymomas, glioblastomas, oligoastrocytomas and oligodendrogliomas.
* **Meningiomas.** A meningioma is a tumor that arises from the membranes that surround your brain and spinal cord (meninges). Most meningiomas are noncancerous.
* **Acoustic neuromas (schwannomas).** These are benign tumors that develop on the nerves that control balance and hearing leading from your inner ear to your brain.
* **Pituitary adenomas.** These are mostly benign tumors that develop in the pituitary gland at the base of the brain. These tumors can affect the pituitary hormones with effects throughout the body.
* **Medulloblastomas.** These are the most common cancerous brain tumors in children. A medulloblastoma starts in the lower back part of the brain and tends to spread through the spinal fluid. These tumors are less common in adults, but they do occur.
* **Germ cell tumors.** Germ cell tumors may develop during childhood where the testicles or ovaries will form. But sometimes germ cell tumors affect other parts of the body, such as the brain.
* **Craniopharyngiomas.** These rare, noncancerous tumors start near the brain's pituitary gland, which secretes hormones that control many body functions. As the craniopharyngioma slowly grows, it can affect the pituitary gland and other structures near the brain.

**4.2.3 Risk factors**

In most people with primary brain tumors, the cause of the tumor is not clear. But doctors have identified some factors that may increase your risk of a brain tumor.

Risk factors include:

* **Exposure to radiation.** People who have been exposed to a type of radiation called ionizing radiation have an increased risk of brain tumor. Examples of ionizing radiation include radiation therapy used to treat cancer and radiation exposure caused by atomic bombs.
* **Family history of brain tumors.** A small portion of brain tumors occurs in people with a family history of brain tumors or a family history of genetic syndromes that increase the risk of brain tumors.

**4.2.4 Diagnosis**

If it's suspected that you have a brain tumor, your doctor may recommend a number of tests and procedures, including:

* **A neurological exam.** A neurological exam may include, among other things, checking your vision, hearing, balance, coordination, strength and reflexes. Difficulty in one or more areas may provide clues about the part of your brain that could be affected by a brain tumor.
* **Imaging tests.** Magnetic resonance imaging (MRI) is commonly used to help diagnose brain tumors. In some cases a dye may be injected through a vein in your arm during your MRI study.

A number of specialized MRI scan components — including functional MRI, perfusion MRI and magnetic resonance spectroscopy — may help your doctor evaluate the tumor and plan treatment.

Sometimes other imaging tests are recommended, including computerized tomography (CT). Positron emission tomography (PET) may be used for brain imaging, but is generally not as useful for creating images of brain cancer as it is for other types of cancer.

* **Tests to find cancer in other parts of your body.** If it's suspected that your brain tumor may be a result of cancer that has spread from another area of your body, your doctor may recommend tests and procedures to determine where the cancer originated. One example might be a CT or PET scan to look for signs of lung cancer.
* **Collecting and testing a sample of abnormal tissue (biopsy).** A biopsy can be performed as part of an operation to remove the brain tumor, or a biopsy can be performed using a needle.

**4.3 Breast Cancer**

Breast cancer is cancer that forms in the cells of the breasts.

After skin cancer, breast cancer is the most common cancer diagnosed in women in the United States. Breast cancer can occur in both men and women, but it's far more common in women.

Substantial support for breast cancer awareness and research funding has helped created advances in the diagnosis and treatment of breast cancer. Breast cancer survival rates have increased, and the number of deaths associated with this disease is steadily declining, largely due to factors such as earlier detection, a new personalized approach to treatment and a better understanding of the disease.

**4.3.1 Symptoms**

Signs and symptoms of breast cancer may include:

* A breast lump or thickening that feels different from the surrounding tissue
* Change in the size, shape or appearance of a breast
* Changes to the skin over the breast, such as dimpling
* A newly inverted nipple
* Peeling, scaling, crusting or flaking of the pigmented area of skin surrounding the nipple (areola) or breast skin
* Redness or pitting of the skin over your breast, like the skin of an orange

## 4.3.2 Causes

Doctors know that breast cancer occurs when some breast cells begin to grow abnormally. These cells divide more rapidly than healthy cells do and continue to accumulate, forming a lump or mass. Cells may spread (metastasize) through your breast to your lymph nodes or to other parts of your body.

Breast cancer most often begins with cells in the milk-producing ducts (invasive ductal carcinoma). Breast cancer may also begin in the glandular tissue called lobules (invasive lobular carcinoma) or in other cells or tissue within the breast.

Researchers have identified hormonal, lifestyle and environmental factors that may increase your risk of breast cancer. But it's not clear why some people who have no risk factors develop cancer, yet other people with risk factors never do. It's likely that breast cancer is caused by a complex interaction of your genetic makeup and your environment.

**4.3.3 Risk factors**

A breast cancer risk factor is anything that makes it more likely you'll get breast cancer. But having one or even several breast cancer risk factors doesn't necessarily mean you'll develop breast cancer. Many women who develop breast cancer have no known risk factors other than simply being women.

Factors that are associated with an increased risk of breast cancer include:

* **Being female.** Women are much more likely than men are to develop breast cancer.
* **Increasing age.** Your risk of breast cancer increases as you age.
* **A personal history of breast conditions.** If you've had a breast biopsy that found lobular carcinoma in situ (LCIS) or atypical hyperplasia of the breast, you have an increased risk of breast cancer.
* **A personal history of breast cancer.** If you've had breast cancer in one breast, you have an increased risk of developing cancer in the other breast.
* **A family history of breast cancer.** If your mother, sister or daughter was diagnosed with breast cancer, particularly at a young age, your risk of breast cancer is increased. Still, the majority of people diagnosed with breast cancer have no family history of the disease.
* **Inherited genes that increase cancer risk.** Certain gene mutations that increase the risk of breast cancer can be passed from parents to children. The most well-known gene mutations are referred to as BRCA1 and BRCA2. These genes can greatly increase your risk of breast cancer and other cancers, but they don't make cancer inevitable.
* **Radiation exposure.** If you received radiation treatments to your chest as a child or young adult, your risk of breast cancer is increased.
* **Obesity.** Being obese increases your risk of breast cancer.
* **Beginning your period at a younger age.** Beginning your period before age 12 increases your risk of breast cancer.
* **Beginning menopause at an older age.** If you began menopause at an older age, you're more likely to develop breast cancer.
* **Having your first child at an older age.** Women who give birth to their first child after age 30 may have an increased risk of breast cancer.
* **Having never been pregnant.** Women who have never been pregnant have a greater risk of breast cancer than do women who have had one or more pregnancies.
* **Postmenopausal hormone therapy.** Women who take hormone therapy medications that combine estrogen and progesterone to treat the signs and symptoms of menopause have an increased risk of breast cancer. The risk of breast cancer decreases when women stop taking these medications.
* **Drinking alcohol.** Drinking alcohol increases the risk of breast cancer.

### **4.3.4 Diagnosis**

### Tests and procedures used to diagnose breast cancer include:

* **Breast exam.** Your doctor will check both of your breasts and lymph nodes in your armpit, feeling for any lumps or other abnormalities.
* **Mammogram.** A mammogram is an X-ray of the breast. Mammograms are commonly used to screen for breast cancer. If an abnormality is detected on a screening mammogram, your doctor may recommend a diagnostic mammogram to further evaluate that abnormality.
* **Breast ultrasound.** Ultrasound uses sound waves to produce images of structures deep within the body. Ultrasound may be used to determine whether a new breast lump is a solid mass or a fluid-filled cyst.
* **Removing a sample of breast cells for testing (biopsy).** A biopsy is the only definitive way to make a diagnosis of breast cancer. During a biopsy, your doctor uses a specialized needle device guided by X-ray or another imaging test to extract a core of tissue from the suspicious area. Often, a small metal marker is left at the site within your breast so the area can be easily identified on future imaging tests.

Biopsy samples are sent to a laboratory for analysis where experts determine whether the cells are cancerous. A biopsy sample is also analyzed to determine the type of cells involved in the breast cancer, the aggressiveness (grade) of the cancer, and whether the cancer cells have hormone receptors or other receptors that may influence your treatment options.

* **Breast magnetic resonance imaging (MRI).** An MRI machine uses a magnet and radio waves to create pictures of the interior of your breast. Before a breast MRI, you receive an injection of dye. Unlike other types of imaging tests, an MRI doesn't use radiation to create the images.

**4.4 Alzheimer**

Alzheimer's disease is a progressive neurologic disorder that causes the brain to shrink (atrophy) and brain cells to die. Alzheimer's disease is the most common cause of dementia — a continuous decline in thinking, behavioral and social skills that affects a person's ability to function independently.

Approximately 5.8 million people in the United States age 65 and older live with Alzheimer's disease. Of those, 80% are 75 years old and older. Out of the approximately 50 million people worldwide with dementia, between 60% and 70% are estimated to have Alzheimer's disease.

The early signs of the disease include forgetting recent events or conversations. As the disease progresses, a person with Alzheimer's disease will develop severe memory impairment and lose the ability to carry out everyday tasks.

Medications may temporarily improve or slow progression of symptoms. These treatments can sometimes help people with Alzheimer's disease maximize function and maintain independence for a time. Different programs and services can help support people with Alzheimer's disease and their caregivers.

There is no treatment that cures Alzheimer's disease or alters the disease process in the brain. In advanced stages of the disease, complications from severe loss of brain function — such as dehydration, malnutrition or infection — result in death.

## 4.4.1 Symptoms

Memory loss is the key symptom of Alzheimer's disease. Early signs include difficulty remembering recent events or conversations. As the disease progresses, memory impairments worsen and other symptoms develop.

At first, a person with Alzheimer's disease may be aware of having difficulty remembering things and organizing thoughts. A family member or friend may be more likely to notice how the symptoms worsen.

Brain changes associated with Alzheimer's disease lead to growing trouble with:

### **Memory**

Everyone has occasional memory lapses, but the memory loss associated with Alzheimer's disease persists and worsens, affecting the ability to function at work or at home.

People with Alzheimer's may:

* Repeat statements and questions over and over
* Forget conversations, appointments or events, and not remember them later
* Routinely misplace possessions, often putting them in illogical locations
* Get lost in familiar places
* Eventually forget the names of family members and everyday objects
* Have trouble finding the right words to identify objects, express thoughts or take part in conversations

### **Thinking and reasoning**

Alzheimer's disease causes difficulty concentrating and thinking, especially about abstract concepts such as numbers.

Multitasking is especially difficult, and it may be challenging to manage finances, balance checkbooks and pay bills on time. Eventually, a person with Alzheimer's may be unable to recognize and deal with numbers.

### **Making judgments and decisions**

Alzheimer's causes a decline in the ability to make reasonable decisions and judgments in everyday situations. For example, a person may make poor or uncharacteristic choices in social interactions or wear clothes that are inappropriate for the weather. It may be more difficult to respond effectively to everyday problems, such as food burning on the stove or unexpected driving situations.

### **Planning and performing familiar tasks**

Once-routine activities that require sequential steps, such as planning and cooking a meal or playing a favorite game, become a struggle as the disease progresses. Eventually, people with advanced Alzheimer's often forget how to perform basic tasks such as dressing and bathing.

### **Changes in personality and behavior**

Brain changes that occur in Alzheimer's disease can affect moods and behaviors. Problems may include the following:

* Depression
* Apathy
* Social withdrawal
* Mood swings
* Distrust in others
* Irritability and aggressiveness
* Changes in sleeping habits
* Wandering
* Loss of inhibitions
* Delusions, such as believing something has been stolen

## 4.4.2 Causes

The exact causes of Alzheimer's disease aren't fully understood. But at a basic level, brain proteins fail to function normally, which disrupts the work of brain cells (neurons) and triggers a series of toxic events. Neurons are damaged, lose connections to each other and eventually die.

Scientists believe that for most people, Alzheimer's disease is caused by a combination of genetic, lifestyle and environmental factors that affect the brain over time.

Less than 1% of the time, Alzheimer's is caused by specific genetic changes that virtually guarantee a person will develop the disease. These rare occurrences usually result in disease onset in middle age.

The damage most often starts in the region of the brain that controls memory, but the process begins years before the first symptoms. The loss of neurons spreads in a somewhat predictable pattern to other regions of the brains. By the late stage of the disease, the brain has shrunk significantly.

Researchers trying to understand the cause of Alzheimer's disease are focused on the role of two proteins:

* **Plaques.** Beta-amyloid is a fragment of a larger protein. When these fragments cluster together, they appear to have a toxic effect on neurons and to disrupt cell-to-cell communication. These clusters form larger deposits called amyloid plaques, which also include other cellular debris.
* **Tangles.** Tau proteins play a part in a neuron's internal support and transport system to carry nutrients and other essential materials. In Alzheimer's disease, tau proteins change shape and organize themselves into structures called neurofibrillary tangles. The tangles disrupt the transport system and are toxic to cells.

## 4.4.3 Risk factors

### **Age**

Increasing age is the greatest known risk factor for Alzheimer's disease. Alzheimer's is not a part of normal aging, but as you grow older the likelihood of developing Alzheimer's disease increases.

One study, for example, found that annually there were four new diagnoses per 1,000 people ages 65 to 74, 32 new diagnoses per 1,000 people ages 75 to 84, and 76 new diagnoses per 1,000 people age 85 and older.

### **Family history and genetics**

Your risk of developing Alzheimer's is somewhat higher if a first-degree relative — your parent or sibling — has the disease. Most genetic mechanisms of Alzheimer's among families remain largely unexplained, and the genetic factors are likely complex.

### **Down syndrome**

Many people with Down syndrome develop Alzheimer's disease. This is likely related to having three copies of chromosome 21 — and subsequently three copies of the gene for the protein that leads to the creation of beta-amyloid. Signs and symptoms of Alzheimer's tend to appear 10 to 20 years earlier in people with Down syndrome than they do for the general population.

### **Sex**

There appears to be little difference in risk between men and women, but, overall, there are more women with the disease because they generally live longer than men.

### **Mild cognitive impairment**

Mild cognitive impairment (MCI) is a decline in memory or other thinking skills that is greater than normal for a person's age, but the decline doesn't prevent a person from functioning in social or work environments.

### **Head trauma**

People who've had a severe head trauma have a greater risk of Alzheimer's disease. Several large studies found that in people age 50 years or older who had a traumatic brain injury (TBI), the risk of dementia and Alzheimer's disease increased. The risk increases in people with more-severe and multiple TBIs. Some studies indicate that the risk may be greatest within the first six months to two years after the TBI.

### **Air pollution**

Studies in animals have indicated that air pollution particulates can speed degeneration of the nervous system. And human studies have found that air pollution exposure — particularly from traffic exhaust and burning wood — is associated with greater dementia risk.

### **Excessive alcohol consumption**

Drinking large amounts of alcohol has long been known to cause brain changes. Several large studies and reviews found that alcohol use disorders were linked to an increased risk of dementia, particularly early-onset dementia.

### **Poor sleep patterns**

Research has shown that poor sleep patterns, such as difficulty falling asleep or staying asleep, are associated with an increased risk of Alzheimer's disease.

### **Lifestyle and heart health**

Research has shown that the same risk factors associated with heart disease may also increase the risk of Alzheimer's disease. These include:

* Lack of exercise
* Obesity
* Smoking or exposure to secondhand smoke
* High blood pressure
* High cholesterol
* Poorly controlled type 2 diabetes

These factors can all be modified. Therefore, changing lifestyle habits can to some degree alter your risk. For example, regular exercise and a healthy low-fat diet rich in fruits and vegetables are associated with a decreased risk of developing Alzheimer's disease.

### **Lifelong learning and social engagement**

## Studies have found an association between lifelong involvement in mentally and socially stimulating activities and a reduced risk of Alzheimer's disease. Low education levels — less than a high school education — appear to be a risk factor for Alzheimer's disease.

## 4.4.4 Diagnosis

An important part of diagnosing Alzheimer's disease includes being able to explain your symptoms, as well as perspective from a close family member or friend about symptoms and their impact on daily life. Additionally, a diagnosis of Alzheimer's disease is based on tests your doctor administers to assess memory and thinking skills.

Laboratory and imaging tests can rule out other potential causes or help the doctor better identify the disease causing dementia symptoms.

But Alzheimer's disease is only diagnosed with complete certainty after death, when microscopic examination of the brain reveals the characteristic plaques and tangles.

### **Tests**

A diagnostic work-up would likely include the following tests:

### Physical and neurological exam

Your doctor will perform a physical exam and likely assess overall neurological health by testing the following:

* Reflexes
* Muscle tone and strength
* Ability to get up from a chair and walk across the room
* Sense of sight and hearing
* Coordination
* Balance

### **Lab tests**

Blood tests may help your doctor rule out other potential causes of memory loss and confusion, such as a thyroid disorder or vitamin deficiencies.

### **Mental status and neuropsychological testing**

Your doctor may give you a brief mental status test to assess memory and other thinking skills. Longer forms of neuropsychological testing may provide additional details about mental function compared with people of a similar age and education level. These tests can help establish a diagnosis and serve as a starting point to track the progression of symptoms in the future.

### **Brain imaging**

Images of the brain are now used chiefly to pinpoint visible abnormalities related to conditions other than Alzheimer's disease — such as strokes, trauma or tumors — that may cause cognitive change. New imaging applications — currently used primarily in major medical centers or in clinical trials — may enable doctors to detect specific brain changes caused by Alzheimer's.

Imaging of brain structures include the following:

* **Magnetic resonance imaging (MRI).** MRI uses radio waves and a strong magnetic field to produce detailed images of the brain. While they may show brain shrinkage of brain regions associated with Alzheimer's disease, MRI scans also rule out other conditions. An MRI is generally preferred to a CT scan for the evaluation of dementia.
* **Computerized tomography (CT).** A CT scan, a specialized X-ray technology, produces cross-sectional images (slices) of your brain. It's usually used to rule out tumors, strokes and head injuries.

Imaging of disease processes can be performed with positron emission tomography (PET). During a PET scan, a low-level radioactive tracer is injected into the blood to reveal a particular feature in the brain. PET imaging may include the following:

* **Fluorodeoxyglucose (FDG) PET** scans show areas of the brain in which nutrients are poorly metabolized. Identifying patterns of degeneration — areas of low metabolism — can help distinguish between Alzheimer's disease and other types of dementia.
* **Amyloid PET imaging**can measure the burden of amyloid deposits in the brain. This imaging is primarily used in research but may be used if a person has unusual or very early onset of dementia symptoms.
* **Tau PET imaging,** which measures the burden of neurofibrillary tangles in the brain, is generally used in the research setting.

In special circumstances, such as rapidly progressive dementia, dementia with atypical features or early-onset dementia, other tests may be used to measure abnormal beta-amyloid and tau in the cerebrospinal fluid.

**4.5 Diabetes**

Diabetes mellitus refers to a group of diseases that affect how your body uses blood sugar (glucose). Glucose is vital to your health because it's an important source of energy for the cells that make up your muscles and tissues. It's also your brain's main source of fuel.

The underlying cause of diabetes varies by type. But, no matter what type of diabetes you have, it can lead to excess sugar in your blood. Too much sugar in your blood can lead to serious health problems.

Chronic diabetes conditions include type 1 diabetes and type 2 diabetes. Potentially reversible diabetes conditions include prediabetes and gestational diabetes. Prediabetes occurs when your blood sugar levels are higher than normal, but not high enough to be classified as diabetes. And prediabetes is often the precursor of diabetes unless appropriate measures are taken to prevent progression. Gestational diabetes occurs during pregnancy but may resolve after the baby is delivered.

**4.5.1 Symptoms**

Diabetes symptoms vary depending on how much your blood sugar is elevated. Some people, especially those with prediabetes or type 2 diabetes, may sometimes not experience symptoms. In type 1 diabetes, symptoms tend to come on quickly and be more severe.

Some of the signs and symptoms of type 1 diabetes and type 2 diabetes are:

* Increased thirst
* Frequent urination
* Extreme hunger
* Unexplained weight loss
* Presence of ketones in the urine (ketones are a byproduct of the breakdown of muscle and fat that happens when there's not enough available insulin)
* Fatigue
* Irritability
* Blurred vision
* Slow-healing sores
* Frequent infections, such as gums or skin infections and vaginal infections

Type 1 diabetes can develop at any age, though it often appears during childhood or adolescence. Type 2 diabetes, the more common type, can develop at any age, though it's more common in people older than 40.

## 4.5.2 Causes

### **Causes of type 1 diabetes**

The exact cause of type 1 diabetes is unknown. What is known is that your immune system — which normally fights harmful bacteria or viruses — attacks and destroys your insulin-producing cells in the pancreas. This leaves you with little or no insulin. Instead of being transported into your cells, sugar builds up in your bloodstream.

Type 1 is thought to be caused by a combination of genetic susceptibility and environmental factors, though exactly what those factors are is still unclear. Weight is not believed to be a factor in type 1 diabetes.

### **Causes of prediabetes and type 2 diabetes**

In prediabetes — which can lead to type 2 diabetes — and in type 2 diabetes, your cells become resistant to the action of insulin, and your pancreas is unable to make enough insulin to overcome this resistance. Instead of moving into your cells where it's needed for energy, sugar builds up in your bloodstream.

Exactly why this happens is uncertain, although it's believed that genetic and environmental factors play a role in the development of type 2 diabetes too. Being overweight is strongly linked to the development of type 2 diabetes, but not everyone with type 2 is overweight.

### **Causes of gestational diabetes**

During pregnancy, the placenta produces hormones to sustain your pregnancy. These hormones make your cells more resistant to insulin.

Normally, your pancreas responds by producing enough extra insulin to overcome this resistance. But sometimes your pancreas can't keep up. When this happens, too little glucose gets into your cells and too much stays in your blood, resulting in gestational diabetes.

## 4.5.3 Risk factors

Risk factors for diabetes depend on the type of diabetes.

### **Risk factors for type 1 diabetes**

Although the exact cause of type 1 diabetes is unknown, factors that may signal an increased risk include:

* **Family history.** Your risk increases if a parent or sibling has type 1 diabetes.
* **Environmental factors.** Circumstances such as exposure to a viral illness likely play some role in type 1 diabetes.
* **The presence of damaging immune system cells (autoantibodies).** Sometimes family members of people with type 1 diabetes are tested for the presence of diabetes autoantibodies. If you have these autoantibodies, you have an increased risk of developing type 1 diabetes. But not everyone who has these autoantibodies develops diabetes.
* **Geography.** Certain countries, such as Finland and Sweden, have higher rates of type 1 diabetes.

### **Risk factors for prediabetes and type 2 diabetes**

Researchers don't fully understand why some people develop prediabetes and type 2 diabetes and others don't. It's clear that certain factors increase the risk, however, including:

* **Weight.** The more fatty tissue you have, the more resistant your cells become to insulin.
* **Inactivity.** The less active you are, the greater your risk. Physical activity helps you control your weight, uses up glucose as energy and makes your cells more sensitive to insulin.
* **Family history.** Your risk increases if a parent or sibling has type 2 diabetes.
* **Race or ethnicity.** Although it's unclear why, certain people — including Black, Hispanic, American Indian and Asian American people — are at higher risk.
* **Age.** Your risk increases as you get older. This may be because you tend to exercise less, lose muscle mass and gain weight as you age. But type 2 diabetes is also increasing among children, adolescents and younger adults.
* **Gestational diabetes.** If you developed gestational diabetes when you were pregnant, your risk of developing prediabetes and type 2 diabetes increases. If you gave birth to a baby weighing more than 9 pounds (4 kilograms), you're also at risk of type 2 diabetes.
* **Polycystic ovary syndrome.** For women, having polycystic ovary syndrome — a common condition characterized by irregular menstrual periods, excess hair growth and obesity — increases the risk of diabetes.
* **High blood pressure.** Having blood pressure over 140/90 millimeters of mercury (mm Hg) is linked to an increased risk of type 2 diabetes.
* **Abnormal cholesterol and triglyceride levels.** If you have low levels of high-density lipoprotein (HDL), or "good," cholesterol, your risk of type 2 diabetes is higher. Triglycerides are another type of fat carried in the blood. People with high levels of triglycerides have an increased risk of type 2 diabetes. Your doctor can let you know what your cholesterol and triglyceride levels are.

### **Risk factors for gestational diabetes**

Pregnant women can develop gestational diabetes. Some women are at greater risk than are others. Risk factors for gestational diabetes include:

* **Age.** Women older than age 25 are at increased risk.
* **Family or personal history.** Your risk increases if you have prediabetes — a precursor to type 2 diabetes — or if a close family member, such as a parent or sibling, has type 2 diabetes. You're also at greater risk if you had gestational diabetes during a previous pregnancy, if you delivered a very large baby or if you had an unexplained stillbirth.
* **Weight.** Being overweight before pregnancy increases your risk.
* **Race or ethnicity.** For reasons that aren't clear, women who are Black, Hispanic, American Indian or Asian American are more likely to develop gestational diabetes.

## 4.5.4 Diagnosis

Symptoms of type 1 diabetes often appear suddenly and are often the reason for checking blood sugar levels. Because symptoms of other types of diabetes and prediabetes come on more gradually or may not be evident, the American Diabetes Association (ADA) has recommended screening guidelines. The ADA recommends that the following people be screened for diabetes:

* **Anyone with a body mass index higher than 25 (23 for Asian Americans), regardless of age,** who has additional risk factors, such as high blood pressure, abnormal cholesterol levels, a sedentary lifestyle, a history of polycystic ovary syndrome or heart disease, and who has a close relative with diabetes.
* **Anyone older than age 45** is advised to receive an initial blood sugar screening, and then, if the results are normal, to be screened every three years thereafter.
* **Women who have had gestational diabetes** are advised to be screened for diabetes every three years.
* **Anyone who has been diagnosed with prediabetes** is advised to be tested every year.

### Tests for type 1 and type 2 diabetes and prediabetes

* **Glycated hemoglobin (A1C) test.** This blood test, which doesn't require fasting, indicates your average blood sugar level for the past two to three months. It measures the percentage of blood sugar attached to hemoglobin, the oxygen-carrying protein in red blood cells.

The higher your blood sugar levels, the more hemoglobin you'll have with sugar attached. An A1C level of 6.5% or higher on two separate tests indicates that you have diabetes. An A1C between 5.7 and 6.4 % indicates prediabetes. Below 5.7 is considered normal.

If the A1C test results aren't consistent, the test isn't available, or you have certain conditions that can make the A1C test inaccurate — such as if you are pregnant or have an uncommon form of hemoglobin (known as a hemoglobin variant) — your doctor may use the following tests to diagnose diabetes:

* **Random blood sugar test.** A blood sample will be taken at a random time. Regardless of when you last ate, a blood sugar level of 200 milligrams per deciliter (mg/dL) — 11.1 millimoles per liter (mmol/L) — or higher suggests diabetes.
* **Fasting blood sugar test.** A blood sample will be taken after an overnight fast. A fasting blood sugar level less than 100 mg/dL (5.6 mmol/L) is normal. A fasting blood sugar level from 100 to 125 mg/dL (5.6 to 6.9 mmol/L) is considered prediabetes. If it's 126 mg/dL (7 mmol/L) or higher on two separate tests, you have diabetes.
* **Oral glucose tolerance test.** For this test, you fast overnight, and the fasting blood sugar level is measured. Then you drink a sugary liquid, and blood sugar levels are tested periodically for the next two hours.

A blood sugar level less than 140 mg/dL (7.8 mmol/L) is normal. A reading of more than 200 mg/dL (11.1 mmol/L) after two hours indicates diabetes. A reading between 140 and 199 mg/dL (7.8 mmol/L and 11.0 mmol/L) indicates prediabetes.

If type 1 diabetes is suspected, your urine will be tested to look for the presence of a byproduct produced when muscle and fat tissue are used for energy because the body doesn't have enough insulin to use the available glucose (ketones). Your doctor will also likely run a test to see if you have the destructive immune system cells associated with type 1 diabetes called autoantibodies.

### **Tests for gestational diabetes**

Your doctor will likely evaluate your risk factors for gestational diabetes early in your pregnancy:

* **If you're at high risk of gestational diabetes** — for example, if you were obese at the start of your pregnancy; you had gestational diabetes during a previous pregnancy; or you have a mother, father, sibling or child with diabetes — your doctor may test for diabetes at your first prenatal visit.
* **If you're at average risk of gestational diabetes,** you'll likely have a screening test for gestational diabetes sometime during your second trimester — typically between 24 and 28 weeks of pregnancy.

**4.6 Pneumonia**

Pneumonia is an infection that inflames the air sacs in one or both lungs. The air sacs may fill with fluid or pus (purulent material), causing cough with phlegm or pus, fever, chills, and difficulty breathing. A variety of organisms, including bacteria, viruses and fungi, can cause pneumonia.

Pneumonia can range in seriousness from mild to life-threatening. It is most serious for infants and young children, people older than age 65, and people with health problems or weakened immune systems.

**4.6.1 Symptoms**

The signs and symptoms of pneumonia vary from mild to severe, depending on factors such as the type of germ causing the infection, and your age and overall health. Mild signs and symptoms often are similar to those of a cold or flu, but they last longer.

Signs and symptoms of pneumonia may include:

* Chest pain when you breathe or cough
* Confusion or changes in mental awareness (in adults age 65 and older)
* Cough, which may produce phlegm
* Fatigue
* Fever, sweating and shaking chills
* Lower than normal body temperature (in adults older than age 65 and people with weak immune systems)
* Nausea, vomiting or diarrhea
* Shortness of breath

Newborns and infants may not show any sign of the infection. Or they may vomit, have a fever and cough, appear restless or tired and without energy, or have difficulty breathing and eating.

## 4.6.2 Causes

Many germs can cause pneumonia. The most common are bacteria and viruses in the air we breathe. Your body usually prevents these germs from infecting your lungs. But sometimes these germs can overpower your immune system, even if your health is generally good.

Pneumonia is classified according to the types of germs that cause it and where you got the infection.

#### Community-acquired pneumonia

Community-acquired pneumonia is the most common type of pneumonia. It occurs outside of hospitals or other health care facilities. It may be caused by:

* **Bacteria.** The most common cause of bacterial pneumonia in the U.S. is Streptococcus pneumoniae. This type of pneumonia can occur on its own or after you've had a cold or the flu. It may affect one part (lobe) of the lung, a condition called lobar pneumonia.
* **Bacteria-like organisms.** Mycoplasma pneumoniae also can cause pneumonia. It typically produces milder symptoms than do other types of pneumonia. Walking pneumonia is an informal name given to this type of pneumonia, which typically isn't severe enough to require bed rest.
* **Fungi.** This type of pneumonia is most common in people with chronic health problems or weakened immune systems, and in people who have inhaled large doses of the organisms. The fungi that cause it can be found in soil or bird droppings and vary depending upon geographic location.
* **Viruses, including COVID-19.** Some of the viruses that cause colds and the flu can cause pneumonia. Viruses are the most common cause of pneumonia in children younger than 5 years. Viral pneumonia is usually mild. But in some cases it can become very serious. Coronavirus 2019 (COVID-19) may cause pneumonia, which can become severe.

#### Hospital-acquired pneumonia

Some people catch pneumonia during a hospital stay for another illness. Hospital-acquired pneumonia can be serious because the bacteria causing it may be more resistant to antibiotics and because the people who get it are already sick. People who are on breathing machines (ventilators), often used in intensive care units, are at higher risk of this type of pneumonia.

#### Health care-acquired pneumonia

Health care-acquired pneumonia is a bacterial infection that occurs in people who live in long-term care facilities or who receive care in outpatient clinics, including kidney dialysis centers. Like hospital-acquired pneumonia, health care-acquired pneumonia can be caused by bacteria that are more resistant to antibiotics.

#### Aspiration pneumonia

Aspiration pneumonia occurs when you inhale food, drink, vomit or saliva into your lungs. Aspiration is more likely if something disturbs your normal gag reflex, such as a brain injury or swallowing problem, or excessive use of alcohol or drugs.

## 4.6.3 Risk factors

Pneumonia can affect anyone. But the two age groups at highest risk are:

* Children who are 2 years old or younger
* People who are age 65 or older

Other risk factors include:

* **Being hospitalized.** You're at greater risk of pneumonia if you're in a hospital intensive care unit, especially if you're on a machine that helps you breathe (a ventilator).
* **Chronic disease.** You're more likely to get pneumonia if you have asthma, chronic obstructive pulmonary disease (COPD) or heart disease.
* **Smoking.** Smoking damages your body's natural defenses against the bacteria and viruses that cause pneumonia.
* **Weakened or suppressed immune system.** People who have HIV/AIDS, who've had an organ transplant, or who receive chemotherapy or long-term steroids are at risk.

**4.6.4 Diagnosis**

Your doctor will start by asking about your medical history and doing a physical exam, including listening to your lungs with a stethoscope to check for abnormal bubbling or crackling sounds that suggest pneumonia.

If pneumonia is suspected, your doctor may recommend the following tests:

* **Blood tests.** Blood tests are used to confirm an infection and to try to identify the type of organism causing the infection. However, precise identification isn't always possible.
* **Chest X-ray.** This helps your doctor diagnose pneumonia and determine the extent and location of the infection. However, it can't tell your doctor what kind of germ is causing the pneumonia.
* **Pulse oximetry.** This measures the oxygen level in your blood. Pneumonia can prevent your lungs from moving enough oxygen into your bloodstream.
* **Sputum test.** A sample of fluid from your lungs (sputum) is taken after a deep cough and analyzed to help pinpoint the cause of the infection.

Your doctor might order additional tests if you're older than age 65, are in the hospital, or have serious symptoms or health conditions. These may include:

* **CT scan.** If your pneumonia isn't clearing as quickly as expected, your doctor may recommend a chest CT scan to obtain a more detailed image of your lungs.
* **Pleural fluid culture.** A fluid sample is taken by putting a needle between your ribs from the pleural area and analyzed to help determine the type of infection.

**4.7 Heart Diseases**

Heart disease describes a range of conditions that affect your heart. Heart diseases include:

* Blood vessel disease, such as coronary artery disease
* Heart rhythm problems (arrhythmias)
* Heart defects you're born with (congenital heart defects)
* Heart valve disease
* Disease of the heart muscle
* Heart infection

Many forms of heart disease can be prevented or treated with healthy lifestyle choices.

## 4.7.1 Symptoms

Heart disease symptoms depend on what type of heart disease you have.

### **Symptoms of heart disease in your blood vessels**

A buildup of fatty plaques in your arteries, or atherosclerosis (ath-ur-o-skluh-ROE-sis) can damage your blood vessels and heart. Plaque buildup causes narrowed or blocked blood vessels that can lead to a heart attack, chest pain (angina) or stroke.

Coronary artery disease symptoms may be different for men and women. For instance, men are more likely to have chest pain. Women are more likely to have other signs and symptoms along with chest discomfort, such as shortness of breath, nausea and extreme fatigue.

**Signs and symptoms can include:**

* Chest pain, chest tightness, chest pressure and chest discomfort (angina)
* Shortness of breath
* Pain, numbness, weakness or coldness in your legs or arms if the blood vessels in those parts of your body are narrowed
* Pain in the neck, jaw, throat, upper abdomen or back

You might not be diagnosed with coronary artery disease until you have a heart attack, angina, stroke or heart failure. It's important to watch for cardiovascular symptoms and discuss concerns with your doctor. Cardiovascular disease can sometimes be found early with regular evaluations.

### **Heart disease symptoms caused by abnormal heartbeats (heart arrhythmias)**

Your heart may beat too quickly, too slowly or irregularly. Heart arrhythmia signs and symptoms can include:

* Fluttering in your chest
* Racing heartbeat (tachycardia)
* Slow heartbeat (bradycardia)
* Chest pain or discomfort
* Shortness of breath
* Lightheadedness
* Dizziness
* Fainting (syncope) or near fainting

### **Heart disease symptoms caused by heart defects**

Serious heart defects that you're born with (congenital heart defects) usually are noticed soon after birth. Heart defect signs and symptoms in children could include:

* Pale gray or blue skin color (cyanosis)
* Swelling in the legs, abdomen or areas around the eyes
* In an infant, shortness of breath during feedings, leading to poor weight gain

Less serious congenital heart defects are often not diagnosed until later in childhood or during adulthood. Signs and symptoms of congenital heart defects that usually aren't immediately life-threatening include:

* Easily getting short of breath during exercise or activity
* Easily tiring during exercise or activity
* Swelling in the hands, ankles or feet

### **Heart disease symptoms caused by diseased heart muscle (cardiomyopathy)**

In early stages of cardiomyopathy, you may have no symptoms. As the condition worsens, symptoms may include:

* Breathlessness with activity or at rest
* Swelling of the legs, ankles and feet
* Fatigue
* Irregular heartbeats that feel rapid, pounding or fluttering
* Dizziness, lightheadedness and fainting

### **Heart disease symptoms caused by heart infection**

Endocarditis is an infection that affects the inner lining of your heart chambers and heart valves (endocardium). Heart infection signs and symptoms can include:

* Fever
* Shortness of breath
* Weakness or fatigue
* Swelling in your legs or abdomen
* Changes in your heart rhythm
* Dry or persistent cough
* Skin rashes or unusual spots

### **Heart disease symptoms caused by heart valve problems (valvular heart disease)**

The heart has four valves — the aortic, mitral, pulmonary and tricuspid valves — that open and close to direct blood flow through your heart. Many things can damage your heart valves, leading to narrowing (stenosis), leaking (regurgitation or insufficiency) or improper closing (prolapse).

Depending on which valve isn't working properly, valvular heart disease signs and symptoms generally include:

* Fatigue
* Shortness of breath
* Irregular heartbeat
* Swollen feet or ankles
* Chest pain
* Fainting (syncope)

## 4.7.2 Causes

Heart disease causes depend on your specific type of heart disease. There are many different types of heart disease. To understand the causes of heart disease, it helps to understand how the heart works.

### **How the heart works:**

**Chambers and valves of the heart**

Your heart is a pump. It's a muscular organ about the size of your fist, located slightly left of center in your chest. Your heart is divided into the right and the left sides.

* The right side of the heart includes the right atrium and ventricle. It collects and pumps blood to the lungs through the pulmonary arteries.
* The lungs give the blood a new supply of oxygen. The lungs also breathe out carbon dioxide, a waste product.
* Oxygen-rich blood then enters the left side of the heart, including the left atrium and ventricle.
* The left side of the heart pumps blood through the largest artery in the body (aorta) to supply tissues throughout the body with oxygen and nutrients.

### **Heart valves**

Four heart valves keep your blood moving the right way by opening only one way and only when they need to. To work properly, the valves must be formed properly, must open all the way and must close tightly so there's no leakage. The four valves are:

* Tricuspid
* Mitral
* Pulmonary
* Aortic

### **Heartbeats**

A beating heart squeezes (contracts) and relaxes in a continuous cycle.

* During contraction (systole), your ventricles squeeze tight, forcing blood into the vessels to your lungs and body.
* During relaxation (diastole), the ventricles are filled with blood coming from the upper chambers (left and right atria).

### **Electrical system**

Your heart's electrical wiring keeps it beating. Your heartbeat controls the continuous exchange of oxygen-rich blood with oxygen-poor blood. This exchange keeps you alive.

* Electrical signals begin high in the upper right chamber (right atrium) and travel through specialized pathways to the ventricles, delivering the signal for the heart to pump.
* The system keeps your heart beating in a coordinated and normal rhythm, which keeps blood flowing.

### **Causes of coronary artery disease**

**Development of atherosclerosis**

A buildup of fatty plaques in your arteries (atherosclerosis) is the most common cause of coronary artery disease. Unhealthy lifestyle habits, such as a poor diet, lack of exercise, being overweight and smoking, can lead to atherosclerosis.

### **Causes of heart arrhythmia**

Common causes of arrhythmias or conditions that can lead to arrhythmias include:

* Coronary artery disease
* Diabetes
* Drug abuse
* Excessive use of alcohol or caffeine
* Heart defects you're born with (congenital heart defects)
* High blood pressure
* Smoking
* Some over-the-counter medications, prescription medications, dietary supplements and herbal remedies
* Stress
* Valvular heart disease

In a healthy person with a normal, healthy heart, it's unlikely for a deadly arrhythmia to develop without some outside trigger, such as an electrical shock or the use of illegal drugs. However, in a heart that's diseased or deformed, the heart's electrical signals may not properly start or travel through the heart, making arrhythmias more likely to develop.

### **Causes of congenital heart defects**

Congenital heart defects usually develop while a baby is in the womb. Heart defects can develop as the heart develops, about a month after conception, changing the flow of blood in the heart. Some medical conditions, medications and genes may play a role in causing heart defects.

Heart defects can also develop in adults. As you age, your heart's structure can change, causing a heart defect.

### **Causes of cardiomyopathy**

The cause of cardiomyopathy, a thickening or enlarging of the heart muscle, may depend on the type:

* **Dilated cardiomyopathy.** The cause of this most common type of cardiomyopathy often is unknown. The condition usually causes the left ventricle to widen. Dilated cardiomyopathy may be caused by reduced blood flow to the heart (ischemic heart disease) resulting from damage after a heart attack, infections, toxins and certain drugs, including those used to treat cancer. It may also be inherited from a parent.
* **Hypertrophic cardiomyopathy.** This type usually is passed down through families (inherited). It can also develop over time because of high blood pressure or aging.
* **Restrictive cardiomyopathy.** This least common type of cardiomyopathy, which causes the heart muscle to become rigid and less elastic, can occur for no known reason. Or it may be caused by diseases, such as connective tissue disorders or the buildup of abnormal proteins (amyloidosis).

### **Causes of heart infection**

A heart infection, such as endocarditis, is caused when germs reach your heart muscle. The most common causes of heart infection include:

* Bacteria
* Viruses
* Parasites

### **Causes of valvular heart disease**

Many things can cause diseases of your heart valves. You may be born with valvular disease, or the valves may be damaged by conditions such as:

* Rheumatic fever
* Infections (infectious endocarditis)
* Connective tissue disorders

## 4.7.3 Risk factors

Risk factors for developing heart disease include:

* **Age.** Growing older increases your risk of damaged and narrowed arteries and a weakened or thickened heart muscle.
* **Sex.** Men are generally at greater risk of heart disease. The risk for women increases after menopause.
* **Family history.** A family history of heart disease increases your risk of coronary artery disease, especially if a parent developed it at an early age (before age 55 for a male relative, such as your brother or father, and 65 for a female relative, such as your mother or sister).
* **Smoking.** Nicotine tightens your blood vessels, and carbon monoxide can damage their inner lining, making them more susceptible to atherosclerosis. Heart attacks are more common in smokers than in nonsmokers.
* **Poor diet.** A diet that's high in fat, salt, sugar and cholesterol can contribute to the development of heart disease.
* **High blood pressure.** Uncontrolled high blood pressure can result in hardening and thickening of your arteries, narrowing the vessels through which blood flows.
* **High blood cholesterol levels.** High levels of cholesterol in your blood can increase the risk of plaque formation and atherosclerosis.
* **Diabetes.** Diabetes increases your risk of heart disease. Both conditions share similar risk factors, such as obesity and high blood pressure.
* **Obesity.** Excess weight typically worsens other heart disease risk factors.
* **Physical inactivity.** Lack of exercise also is associated with many forms of heart disease and some of its other risk factors as well.
* **Stress.** Unrelieved stress may damage your arteries and worsen other risk factors for heart disease.
* **Poor dental health.** It's important to brush and floss your teeth and gums often, and have regular dental checkups. If your teeth and gums aren't healthy, germs can enter your bloodstream and travel to your heart, causing endocarditis.

**4.7.4 Diagnosis**

Your doctor will perform a physical exam and ask about your personal and family medical history. The tests you'll need to diagnose your heart disease depend on what condition your doctor thinks you might have. Besides blood tests and a chest X-ray, tests to diagnose heart disease can include:

* **Electrocardiogram (ECG or EKG).** An ECG is a quick and painless test that records the electrical signals in your heart. It can spot abnormal heart rhythms. You may have an ECG while you're at rest or while exercising (stress electrocardiogram).
* **Holter monitoring.** A Holter monitor is a portable ECG device you wear to continuously record your heart rhythm, usually for 24 to 72 hours. Holter monitoring is used to detect heart rhythm problems that aren't found during a regular ECG exam.
* **Echocardiogram.** This noninvasive exam uses sound waves to produce detailed images of your heart's structure. It shows how your heart beats and pumps blood.
* **Stress test.** This type of test involves raising your heart rate with exercise or medicine while performing heart tests and imaging to check how your heart responds.
* **Cardiac catheterization.** In this test, a short tube (sheath) is inserted into a vein or artery in your leg (groin) or arm. A hollow, flexible and longer tube (guide catheter) is then inserted into the sheath. Using X-ray images on a monitor as a guide, your doctor carefully threads the catheter through the artery until it reaches your heart.

During cardiac catheterization, the pressures in your heart chambers can be measured, and dye can be injected. The dye can be seen on an X-ray, which helps your doctor see the blood flow through your heart, blood vessels and valves to check for problems.

* **Cardiac computerized tomography (CT) scan.** In a cardiac CT scan, you lie on a table inside a doughnut-shaped machine. An X-ray tube inside the machine rotates around your body and collects images of your heart and chest.
* **Cardiac magnetic resonance imaging (MRI).** A cardiac MRI uses a magnetic field and computer-generated radio waves to create detailed images of your heart.

**CONCLUSION**

I believe the trial has shown conclusively that it is both possible and desirable to use Python as the principal teaching language:

* It is Free (as in both cost and source code).
* It is trivial to install on a Windows PC allowing students to take their interest further. For many the hurdle of installing a Pascal or C compiler on a Windows machine is either too expensive or too complicated
* It is a flexible tool that allows both the teaching of traditional procedural programming and modern OOP; It can be used to teach a large number of transferable skills
* It is a real-world programming language that can be andis used in academia and the commercial world
* It appears to be quicker to learn and, in combination with its many libraries, this offers the possibility of more rapid student development allowing the course to be made more challenging and varied
* And most importantly, its clean syntax offers increased understanding and enjoyment for students

Python should be used as the first year teaching language. If used it will be possible to teach students more programming and less of the peculiarities of a particularlanguage. Teaching a mid-level language like C in just one day is inadvisable. Too much time must be spent teaching C and not enough time teaching generic skills to students with no programming experience.

The use of Python as the first year language is not a dead-end. I have tried to emphasise that Python allows the teaching of widely applicable programming concepts. Its use in no way precludes the use of C in a more advanced course. In fact students who go on to use C in later years will have a better grounding in concepts from their introduction to programming than they might have from a C-based introduction. I believe that more students will go on to advanced programming if introduced using Python because introducing programming using C will frustrate and scare off a large number of students.

In conclusion, Python offers the optimum compromise of teach ability and applicability.