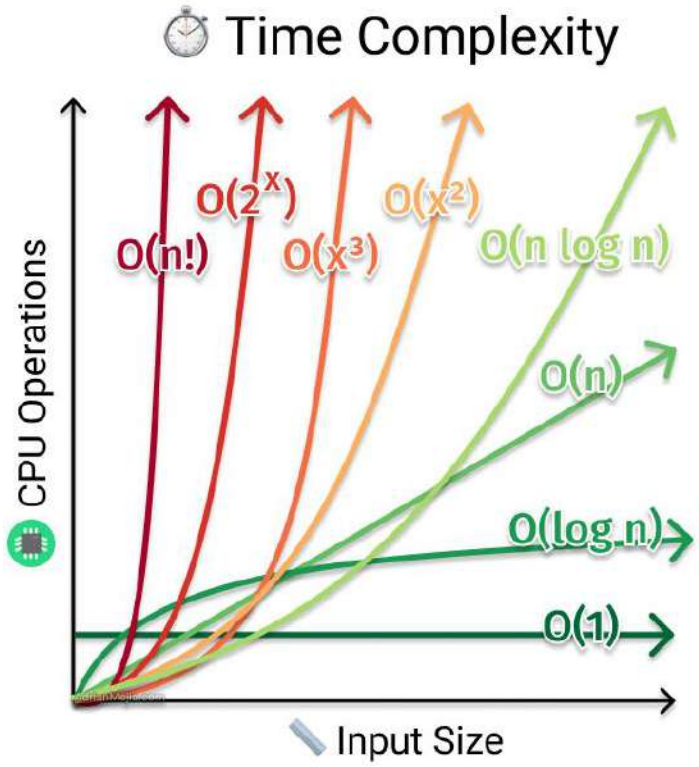
CSE 4810 – Algorithm Engineering

Lab 2 – Lecture Sheet

Algorithmic Complexity and Big-O Notation



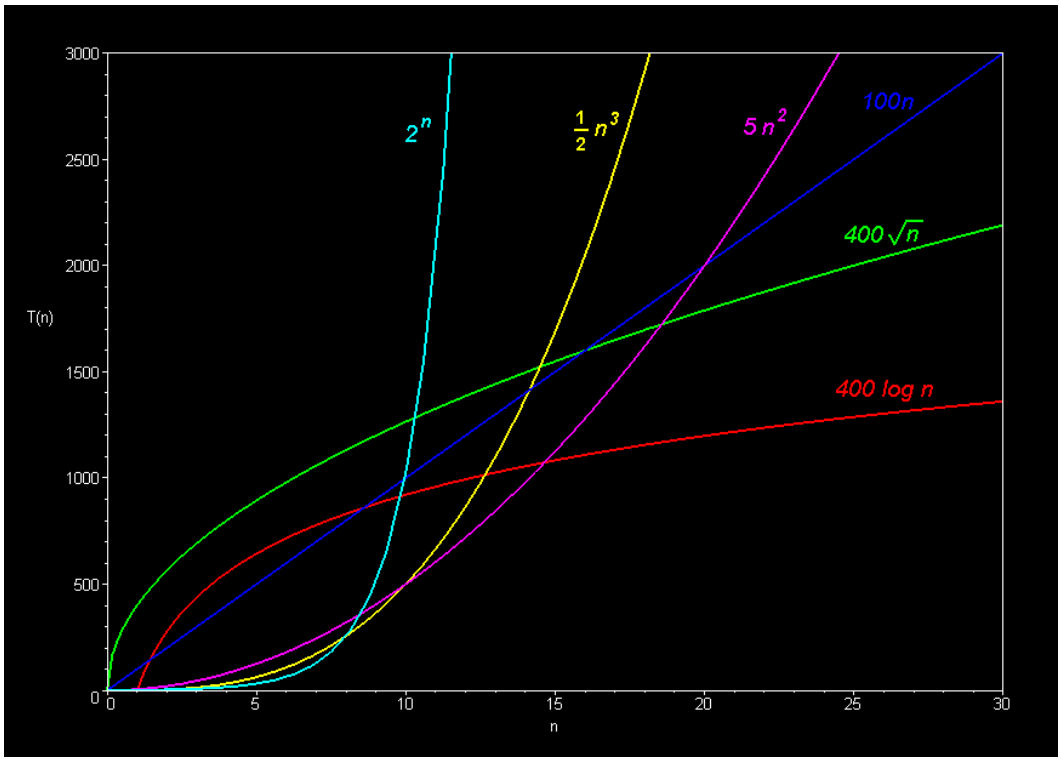
The graph compares the running times of various algorithms.

- Linear -- O(n)
- Quadratic -- O(n^2)
- Cubic -- O(n^3)
- Logarithmic -- O(log n)
- Exponential -- O(2^n)
- Square root -- O(sqrt n)

Example:

$n^a > \log n\,, where\ 0 < a < 1$

$n^a > n \log n\,, where\ a > 1$

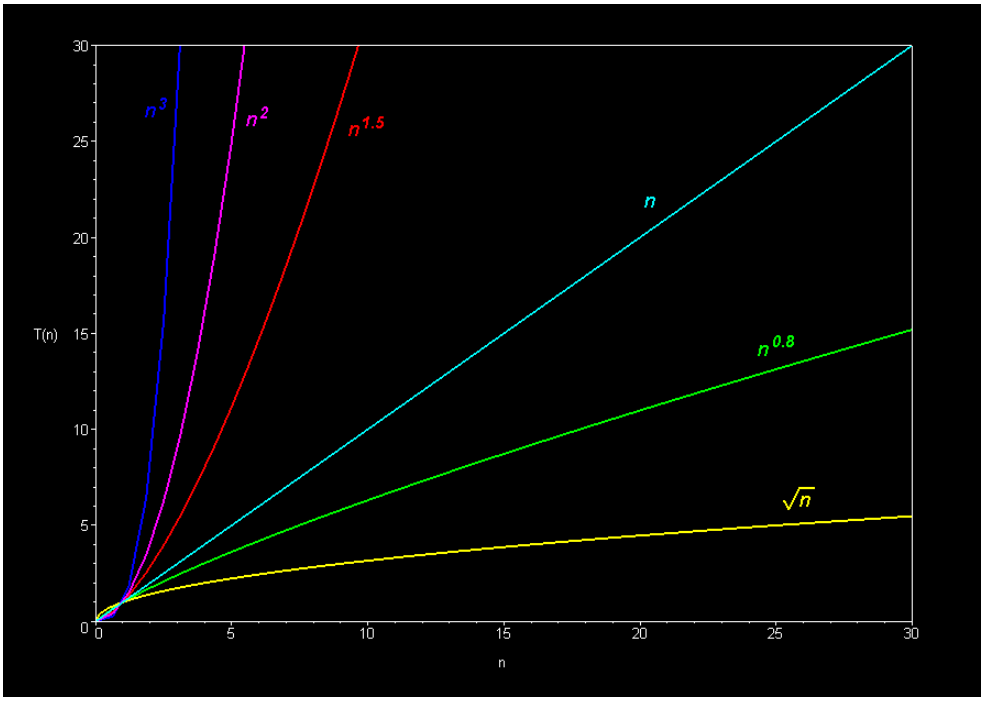


Comparison of algorithms in terms of the maximum problem size they can handle:

Algorithm Complexity	Running time T(n) (measured in seconds on an Apple Delicious computer)	Maximum problem size given 1000 seconds on an Apple Delicious	Computer Speed x 10 Maximum problem size given 1000 seconds on a Power Delicious (or 10,000 seconds on a classic Delicious)
O(n)	100n	n = 10	n = 100 *(x 10 increase)*
$O(n^2)$	$5n^2$	n = 14	n = 45 *(x 3)*
$O(n^3)$	½ n^3	n = 12	n = 27 *(x 2)*
$O(2^n)$	2^n	n = 10	n = 13 *(x 1.3)*
O(sqrt n)	400 sqrt(n)	n = 6	n = 625 *(x 100)*
O(log n)	400 log(n)	n = 12	n = 72 billion *(x 6 billion)*

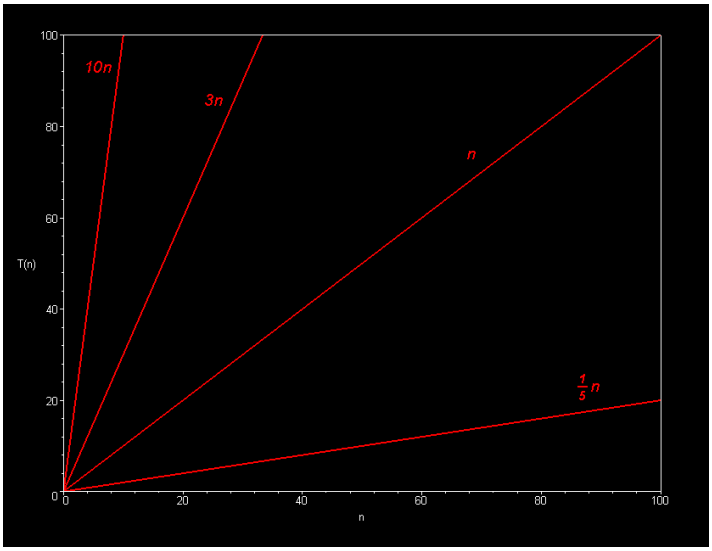
MORAL: Cheaper, faster computers mean bigger problems to solve.
Bigger problems to solve mean efficiency is *more* important.

The basic **shape of a polynomial function** is determined by the **highest valued exponent** in the polynomial (called the ***order*** of the polynomial).

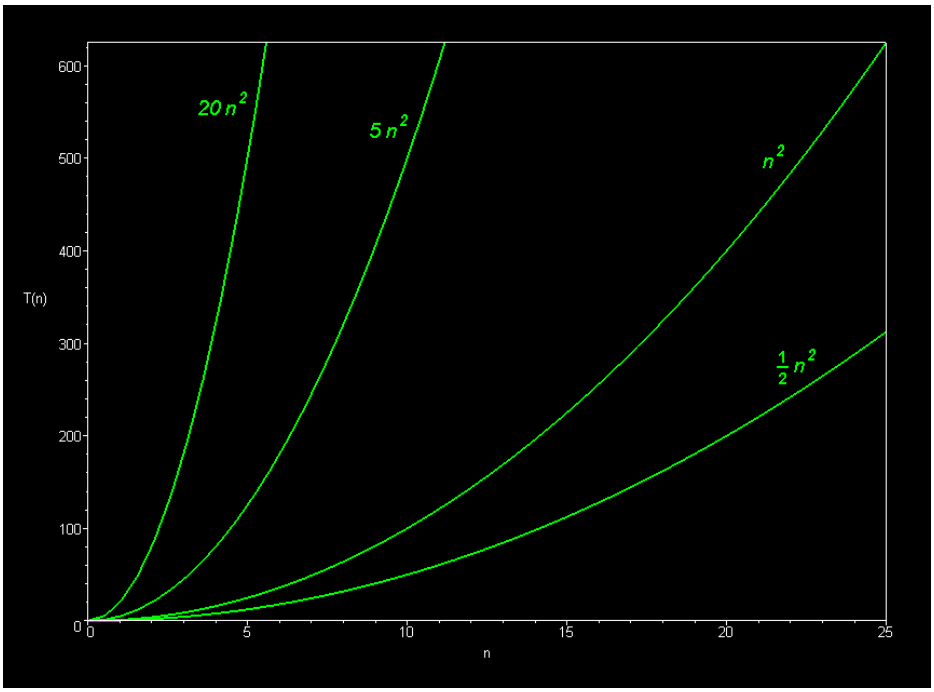


Multiplicative constants do not affect the fundamental shape of a curve. Only the steepness of the curve is affected.

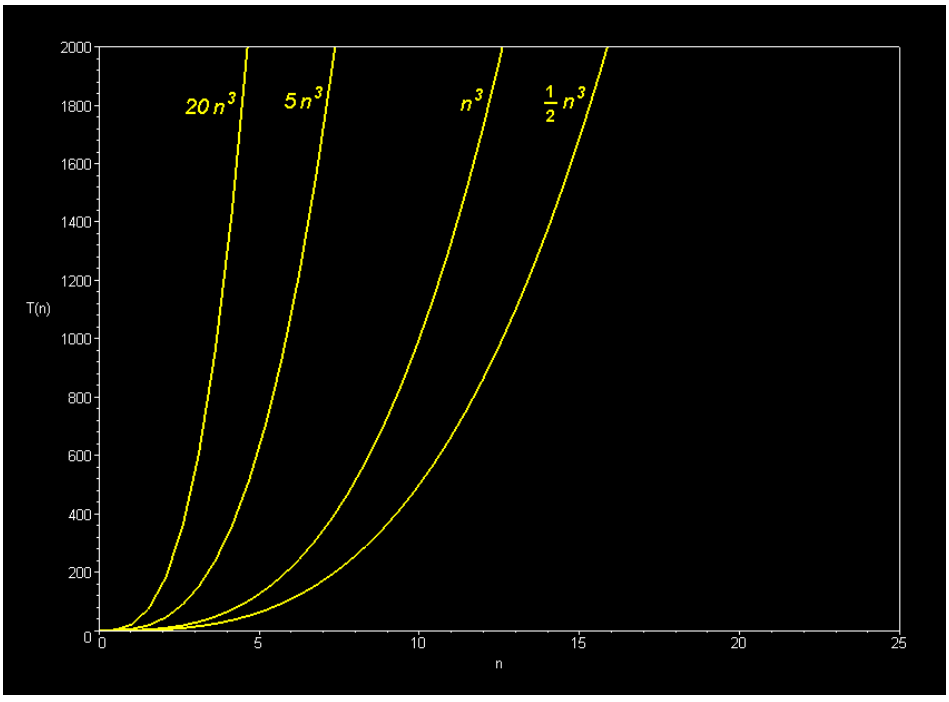
Linear Family



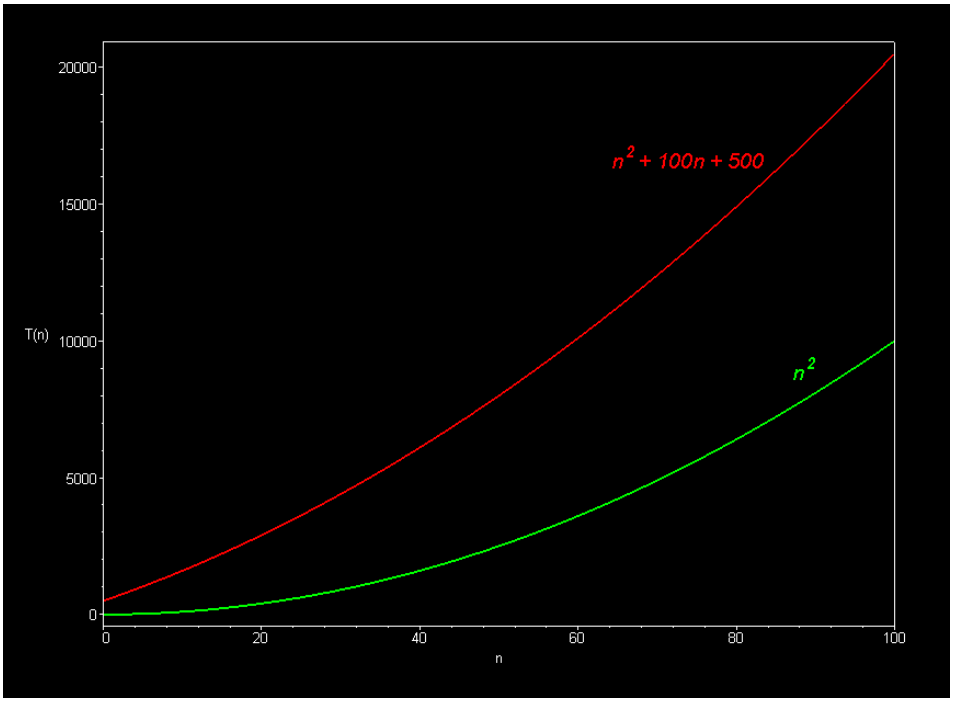
Quadratic Family

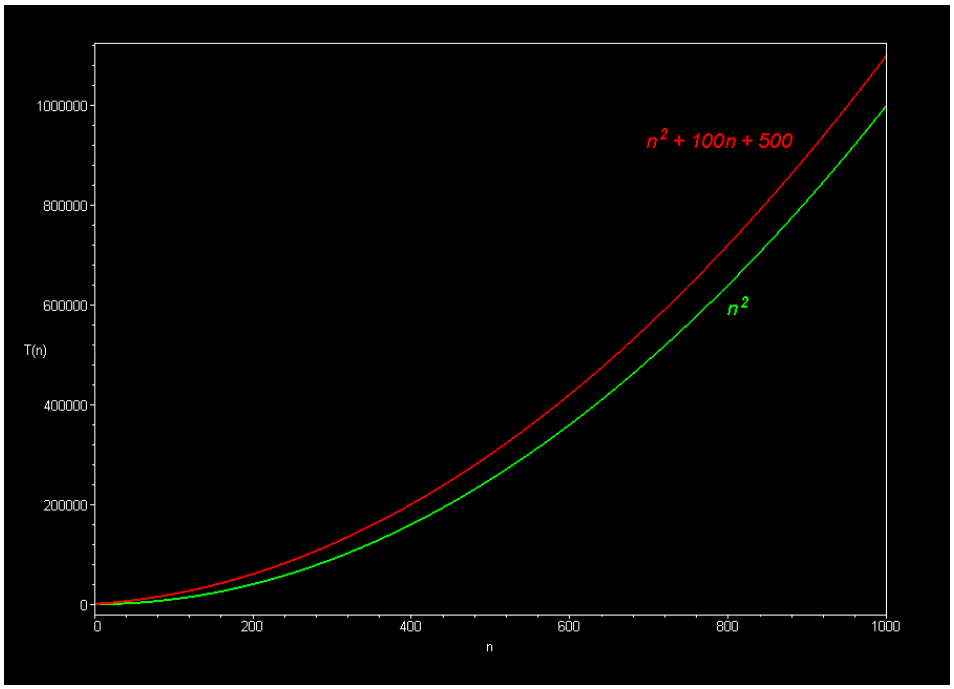


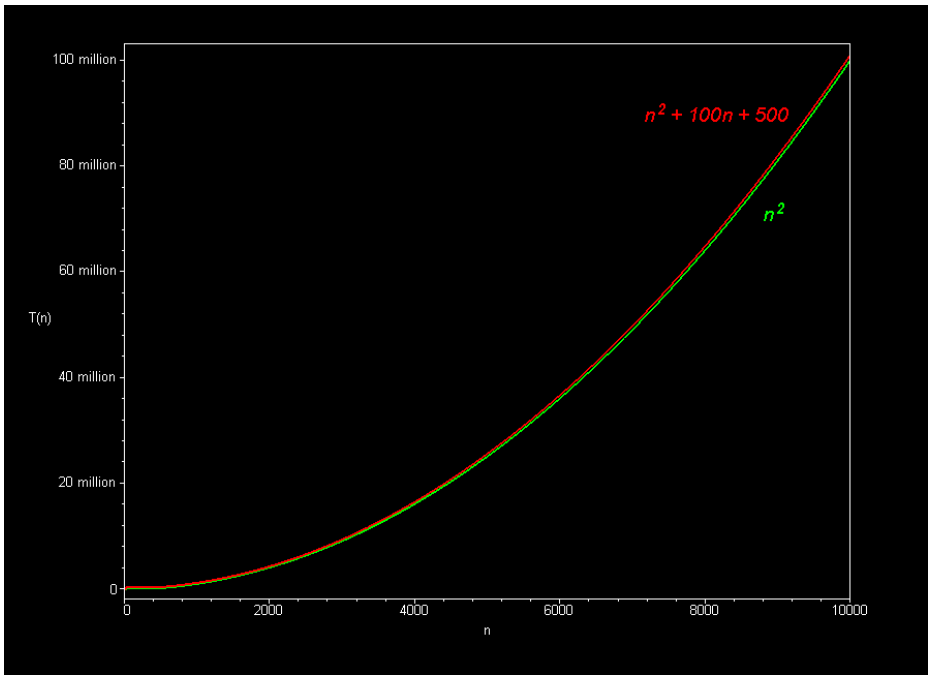
Cubic Family



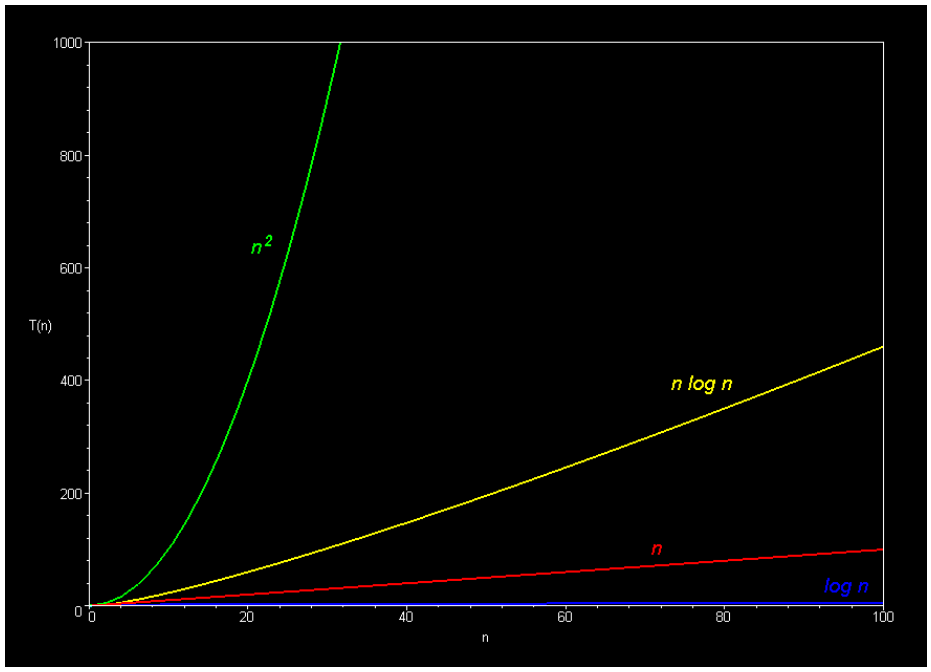
Only the *dominant terms* of a polynomial matter in the long run. Lower-order terms fade to insignificance as the problem size increases.



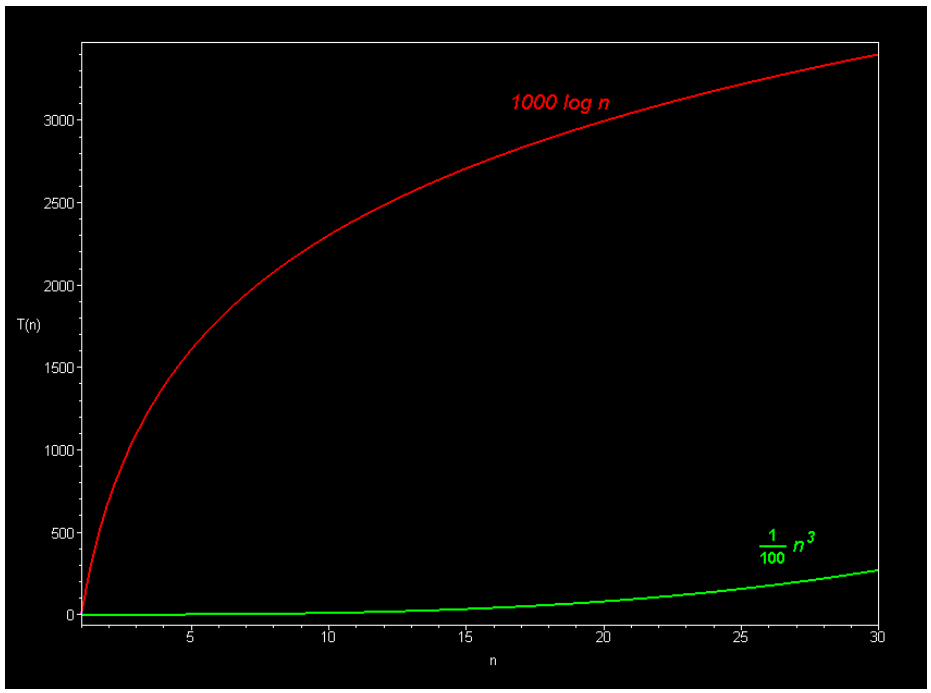


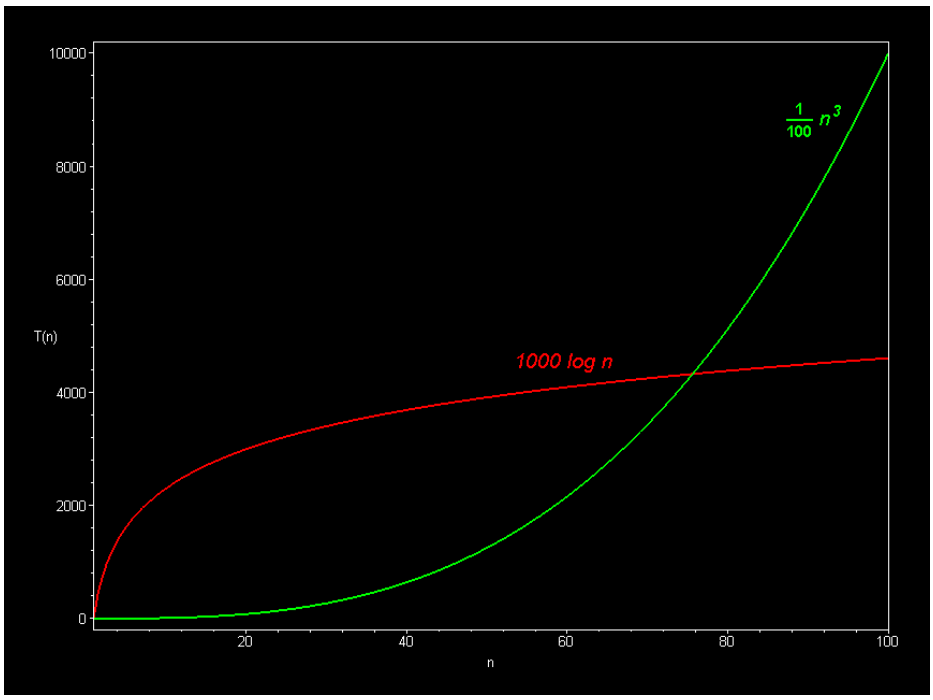


The best sorting algorithms (such as mergesort) run in O(n log n) time. Slower ones (such as bubble sort, selection sort, and insertion sort), take O(n^2) time.



Polynomial curves will always overtake logarithmic curves eventually, when the problem size gets big enough, regardless of the multiplicative constants involved.





Data Structures

A data structure is a specific method to store and organize data in a computer to be easily used to efficiently perform operations on it.

When data is unstructured, it is not organized or doesn't have a defined data model.

Then, it is not suitable for the analysis or operations. Unstructured data is a very common problem. It is estimated that 80% of the world's data is unstructured.

Most organizations collect data and store it in an unorganized way that is not effective in making data easy to use.

There are different types (forms) of data structure, and each type could be effective for some operations but not for others.

So, it is the programmer's job to understand which data structure is suitable to analyze the data efficiently so it can be used to solve the problems or achieve the goal.

Data structures are the fundamentals of software engineering and computer science and are being used in most software systems.

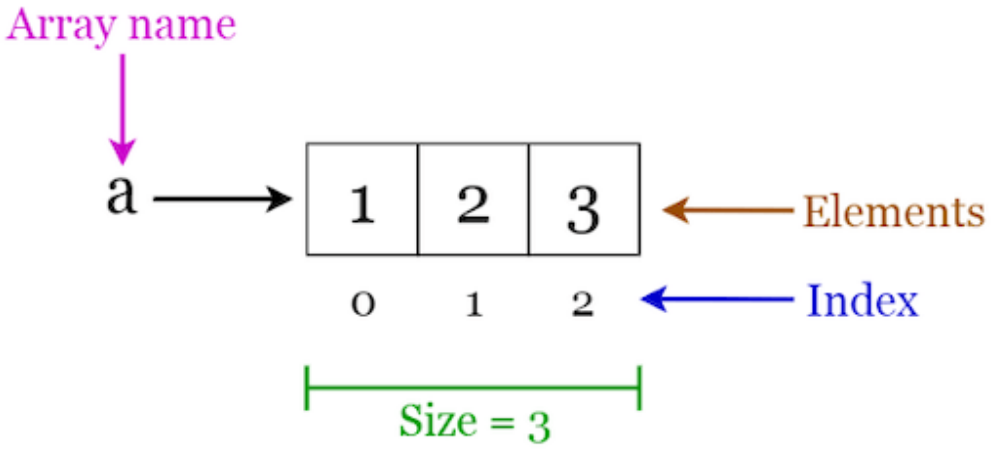
Common types of data structure

Now let us discuss eight of the most common data structures.

1. Arrays

The first in our list of basic data structures is one of the simplest data structures. An array is a fixed-size structure that stores multiple items of the same kind of data sequentially.

An array contains the same but fixed-size data type elements (also known as variables). That's why you can't change the array's size. In an array index, each item starts with "0."



Source of all tables in this article

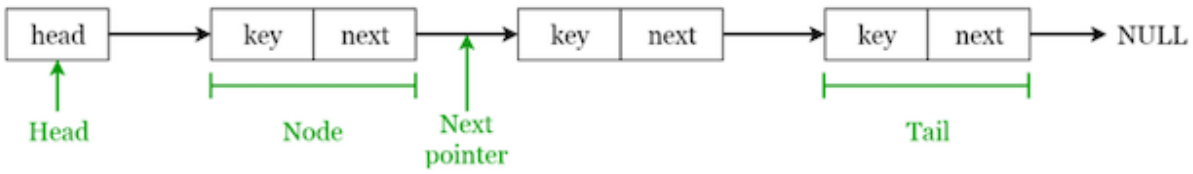
An array has elements inside each container and lined up small containers in a sequence.

As arrays are fixed in size, it is not possible to insert or delete an element to an array.
To add an element, you need to create a new array with increased size (+1), then copy the existing elements and add a new element to it.

They are mostly used as a structure to build more complicated data structures and for sorting algorithms.

2. Linked lists

A linked list is a linear data structure where items are arranged in linear order and linked (connected) to each other. That's why you cannot access random data; you need to access data only in order (sequentially).



In a linked list, the first element in the list is known as "Head," and the last item is known as "Tail."

Each item is called a "node," and each contains a pointer and a key in the linked list. The Pointer takes you to the next Node known as "next."

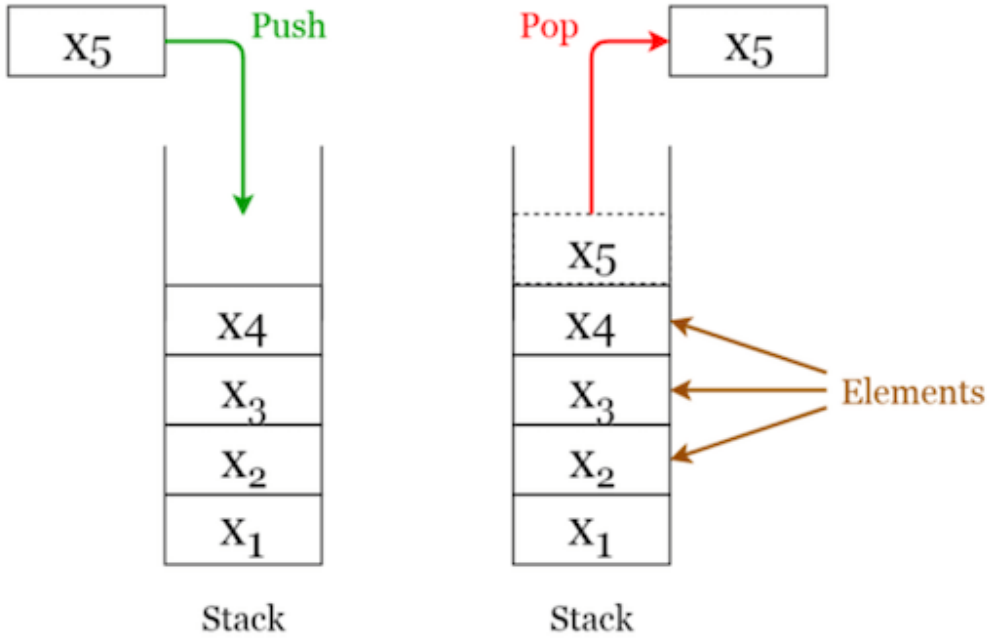
You can traverse each item from head to tail (in forward direction) by creating a single linked list. In the same way, a double-linked list can traverse in both directions; from head to tail (forward direction) and from tail to head (backward direction).

They are used in switching between programs for symbol table management.

Related: How to grow your design business into an agency

3. Stacks

Next on our list of basic data structures is Stack. Stack is also a linear order structure, but it works in a LIFO (Last in First Out) order. That's why it is also known as LIFO structure. It means the last-placed element can be accessed first.

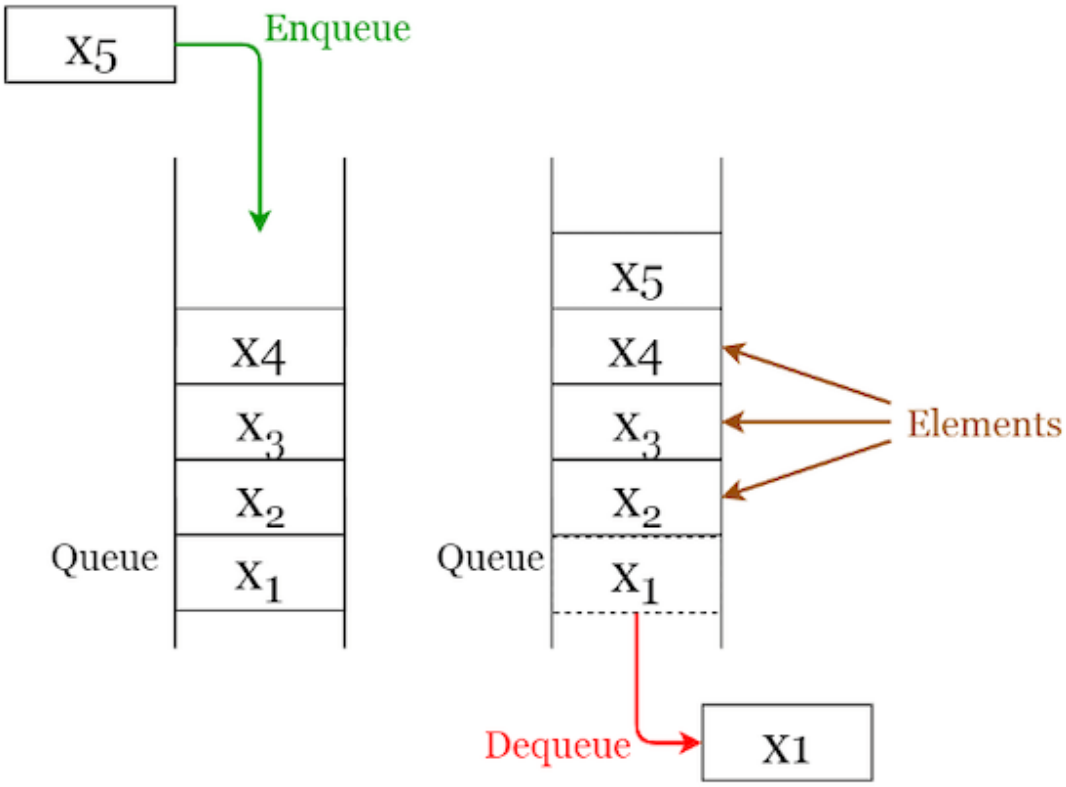


You can push (add a new) or pop (delete) an element on/from the top of the elements, like a stack of plates in the real world.

Stacks are mostly used in recursion programming to implement function calls and mathematical expressions for parsing and evaluations.

4. Queues

Queues are the same as Stack structure, but they don't follow the LIFO model. A queue follows the FIFO (First in First Out) model. This means the first element can be accessed first.



A line of people entering the building is a great example. The first person (starting the line) will enter the building before anyone else, and the person standing in the last of the line will enter in the last.

In the same way, you can add a new element (enqueue) at the end of the structure and dequeue (delete) an element from the starting of the structure.

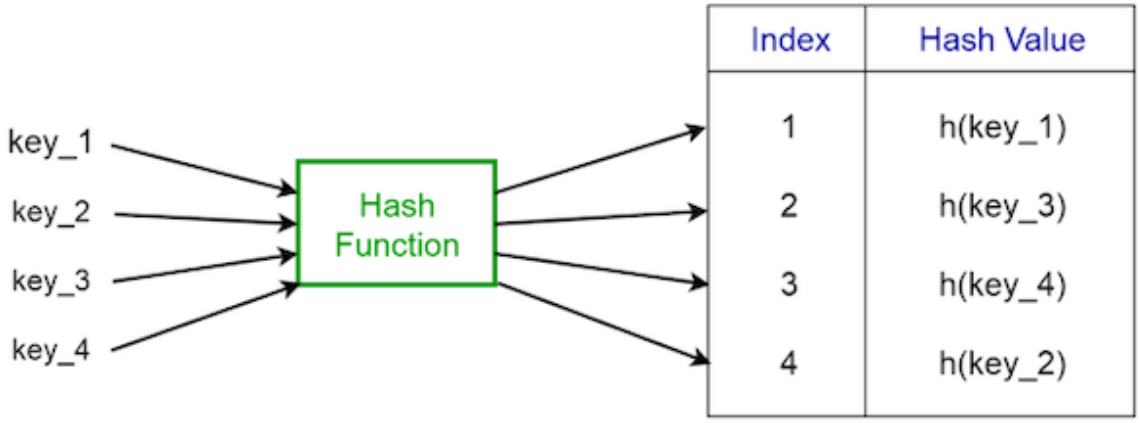
They are mostly used in multithreading to manage threads as well as to execute priority queuing systems.

5. Hash tables

The hash table data structure connects each value with a key and stores them.

You can efficiently lookup values by using a key.

From a group of similar objects, you can easily find a specific object.



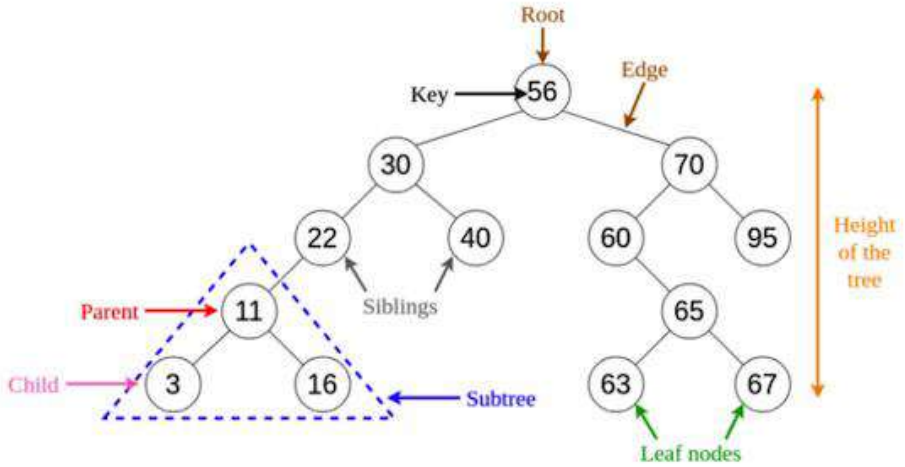
To understand the Hash table, you can take it as a student ID (Key) that a college assigns to students. All the details related to a student can be easily found by using the student ID.

To map any size of data set to one fixed size, the hash table uses the hash function. The values returned by a hash function are known as hash values.

They are mostly used to create associate arrays, database indexes, and a “set.”

6. Trees

Another basic data structure is a Tree. In the tree structure, data is linked together as in the linked list but organized hierarchically, just like the visual representation of a person’s family tree.



There are various types of trees like:

- Binary search tree (BST)
- Red-black tree
- B tree, treap
- AVL tree

And each type is suited for certain applications.

For example, the BST data structure stores values (data) in sorted order.

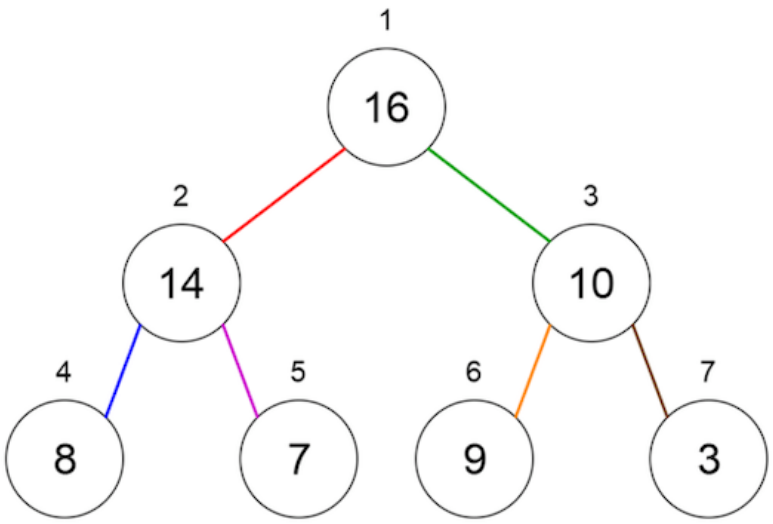
BST structure is widely used in different types of search operations, and other types of tree structures are used to create expression solvers and in wireless networking.

7. Heaps

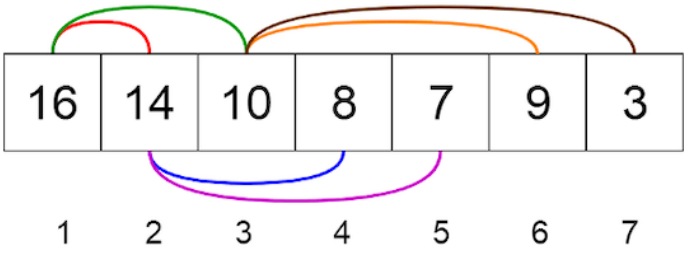
A heap is a specific type of binary tree where the parent nodes are compared to their child nodes, and values are arranged in the nodes accordingly.

A heap can be represented as an array or a binary tree as you can see in the images below:

Binary Tree Representation



Array Representation



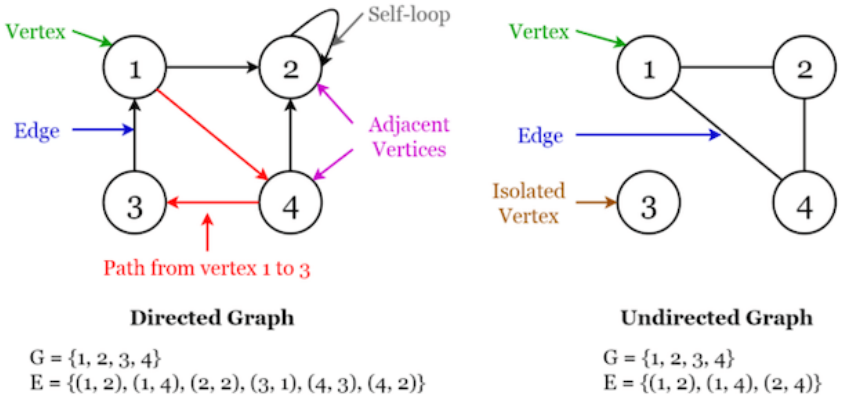
There are two types of heaps:

1. **Min heap**, where the parent's key is equal or less than the keys of its children.
2. **Max heap**, where the parent's key is greater than the keys of its children.

Heaps are widely used to find the largest and smallest values in an array and to create priority queues in algorithms.

8. **Graphs**

A graph is a non-linear and abstract data structure that consists of a fixed (finite) set of nodes or vertices and is connected by a set of edges. Edges are the arcs or lines that simply connect nodes in the graph.



Graphs are great for solving real-world problems, as well as representations of digital networks. They're also used for the representations of the networks like circuit networks.

Time complexities of different data structures

Worst Case

Data structure	Access	Search	Insertion	Deletion
Array	O(1)	O(N)	O(N)	O(N)
Stack	O(N)	O(N)	O(1)	O(1)
Queue	O(N)	O(N)	O(1)	O(1)
Singly Linked list	O(N)	O(N)	O(N)	O(N)
Doubly Linked List	O(N)	O(N)	O(1)	O(1)
Hash Table	O(N)	O(N)	O(N)	O(N)
Binary Search Tree	O(N)	O(N)	O(N)	O(N)
AVL Tree	O(log N)	O(log N)	O(log N)	O(log N)
Binary Tree	O(N)	O(N)	O(N)	O(N)
Red Black Tree	O(log N)	O(log N)	O(log N)	O(log N)

Best Case

Data structure	Access	Search	Insertion	Deletion
Array	O(1)	O(1)	O(1)	O(1)
Stack	O(1)	O(1)	O(1)	O(1)
Queue	O(1)	O(1)	O(1)	O(1)
Singly Linked list	O(1)	O(1)	O(1)	O(1)
Doubly Linked List	O(1)	O(1)	O(1)	O(1)
Hash Table	O(1)	O(1)	O(1)	O(1)
Binary Search Tree	O(log n)	O(log n)	O(log n)	O(log n)
AVL Tree	O(log n)	O(log n)	O(log n)	O(log n)
B Tree	O(log n)	O(log n)	O(log n)	O(log n)
Red Black Tree	O(log n)	O(log n)	O(log n)	O(log n)

Average Case

Data structure	Access	Search	Insertion	Deletion
Array	O(1)	O(N)	O(N)	O(N)
Stack	O(N)	O(N)	O(1)	O(1)
Queue	O(N)	O(N)	O(1)	O(1)
Singly Linked list	O(N)	O(N)	O(1)	O(1)
Doubly Linked List	O(N)	O(N)	O(1)	O(1)
Hash Table	O(1)	O(1)	O(1)	O(1)
Binary Search Tree	O(log N)	O(log N)	O(log N)	O(log N)
AVL Tree	O(log N)	O(log N)	O(log N)	O(log N)
B Tree	O(log N)	O(log N)	O(log N)	O(log N)
Red Black Tree	O(log N)	O(log N)	O(log N)	O(log N)