BIG DATA DEVELOPMENT

INTRO TO ALGORITHMS

ALGORITHM'S WEEK SCHEDULE

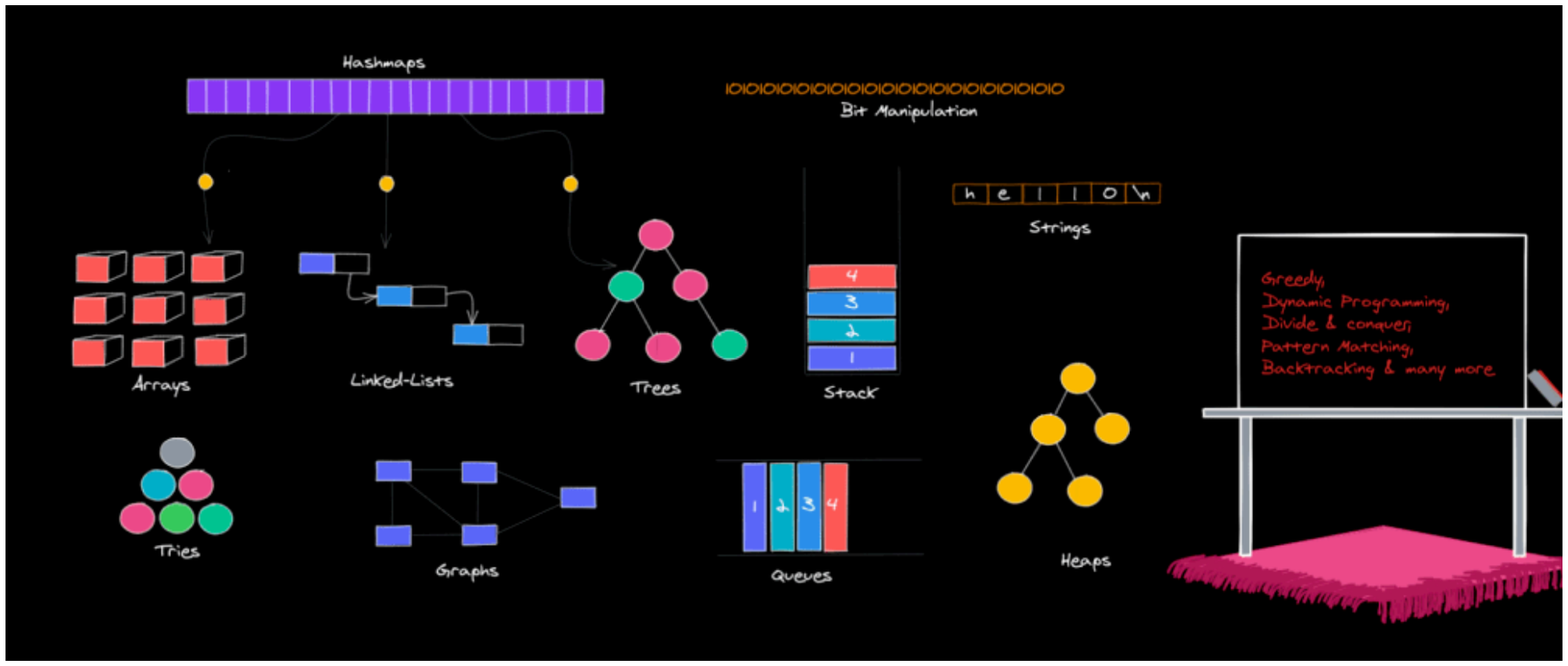
- 1.9 - Office
  - Intro to Algorithms
  - Data structures reading materials
  - Exercises
- 2.9 - Home
  - Sorting algorithms implementation
- 5.9 - Home
  - Graph traversal implementation
  - Topological sorting implementation
- 12.9 - Home
  - Shortest path algorithm implementation
- 13.9 - Office
  - Lectures

AGENDA

- Data Structures
  - Array
  - Linked List
  - Hash Table
  - Tree
    - BST
    - Trie
    - Heap
  - Stack
  - Queue
  - Graph
    - Graph representations
- Big O Notation
  - Selection Sort
  - Importance of Big O Notation
  - Time and space complexity
  - Trade offs
- Algorithms
  - Day to day usages
  - Common algorithms
    - Sorting
    - Shortest Path
    - Minimum Spanning Tree
    - Binary Search
    - BFS & DFS
    - Topological Sort
    - Flood Fill

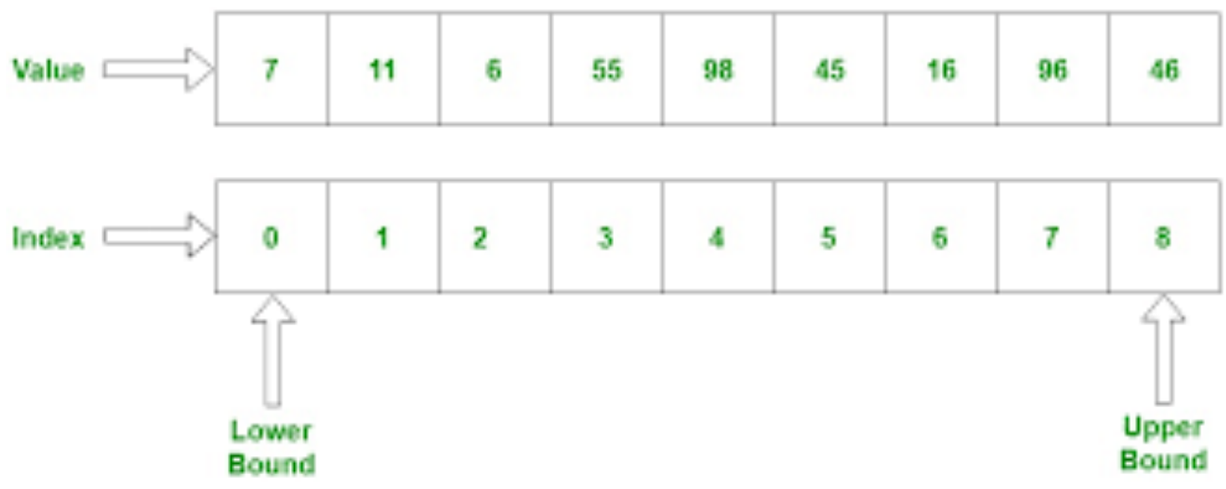
DATA STRUCTURES

- Organizing and storing data in computers in such a way that we can perform operations on the stored data more efficiently.
- Data structures are being used in almost every program or software system that has been developed.



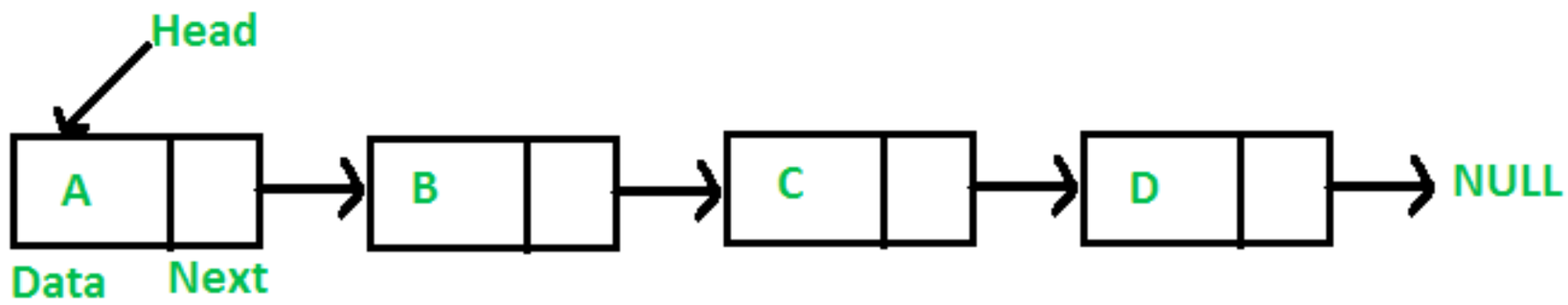
ARRAY

- Array is a data structure consisting of a collection of elements, each identified by at least one array index or key.
- The memory address of the first element of an array is called first address, foundation address, or base address.
- One-dimensional array/Vector
- Matrix/Table



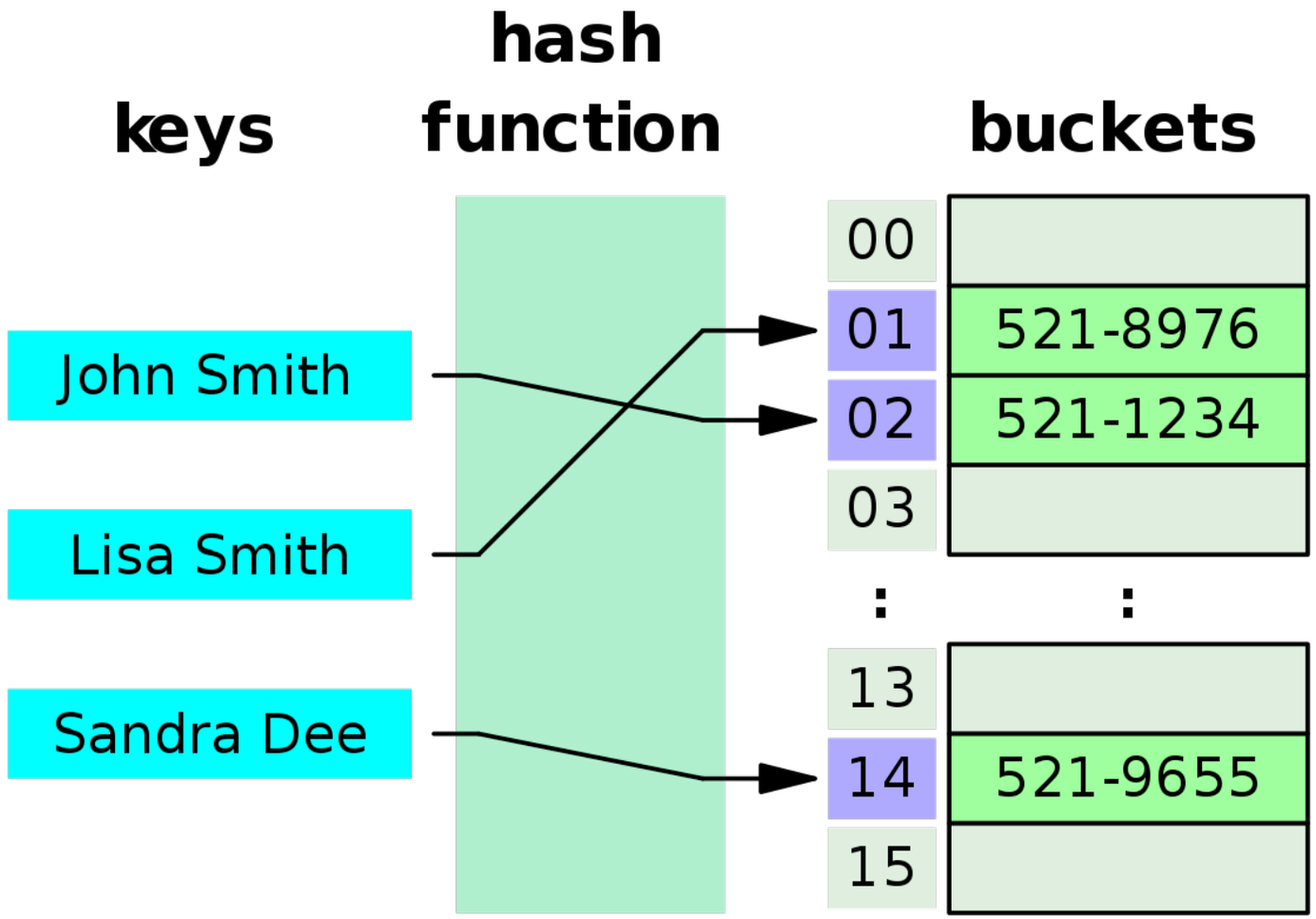
LINKED LIST

- A linear collection of data elements whose order is not given by their physical placement in memory.
- A data structure consisting of a collection of nodes which together represent a sequence.
- Benefits - the list elements can be easily inserted or removed without reallocation or reorganisation of the entire structure.
- Singly linked list
- Doubly linked list
- Circular linked list



HASH TABLE

- Stores values which have keys associated with each of them
- In case of collision
  - Chaining
  - Open addressing
- Applications
  - Implement DB indices
  - Implement Associative Arrays
  - Implement the Set data structure

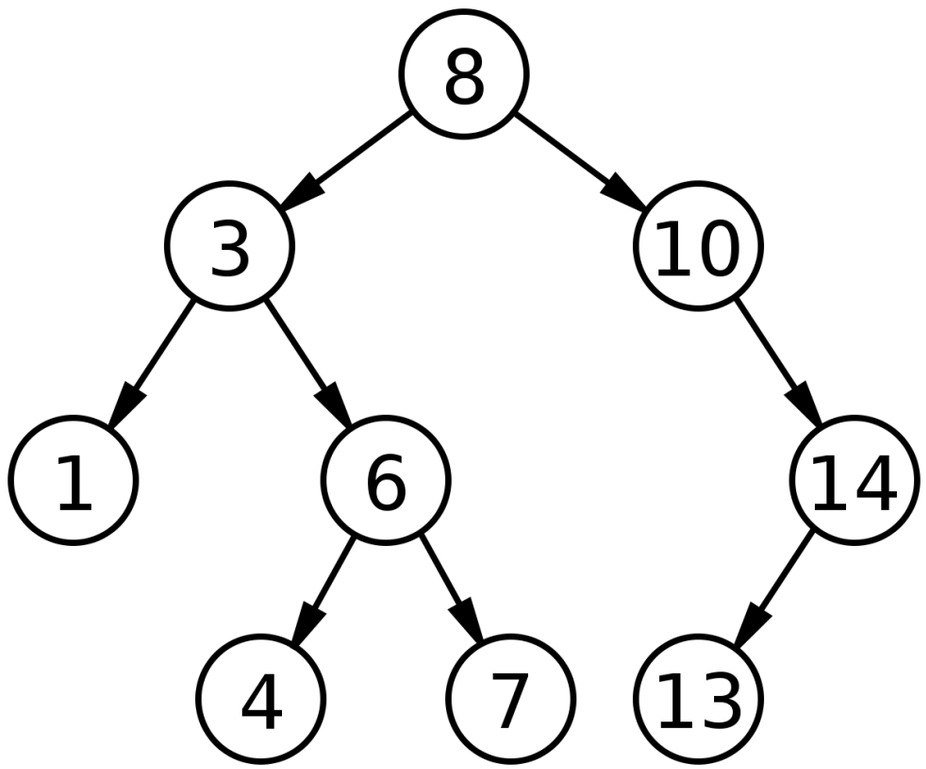


TREE

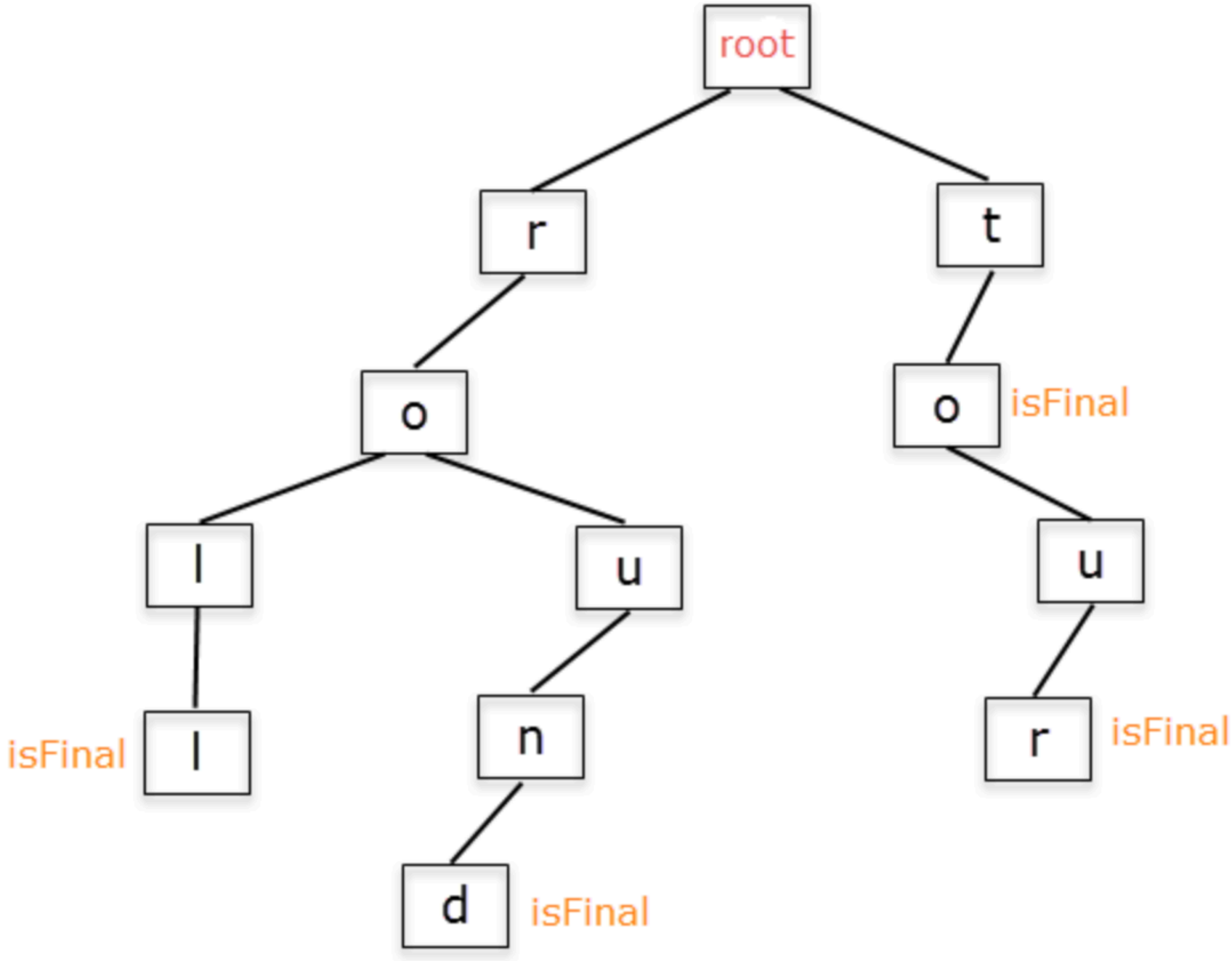
- Abstract data type that simulates a hierarchical tree structure
- A given node only contains the list of its children but does not contain a reference to its parent (if any).
- Usages
  - Document Object Models (XML/HTML)
  - Search trees

BINARY SEARCH TREE (BST)

- The left subtree of a node contains only nodes with keys lesser than the node's key.
- The right subtree of a node contains only nodes with keys greater than the node's key.
- The left and right subtree each must also be a binary search tree.
- There must be no duplicate nodes.

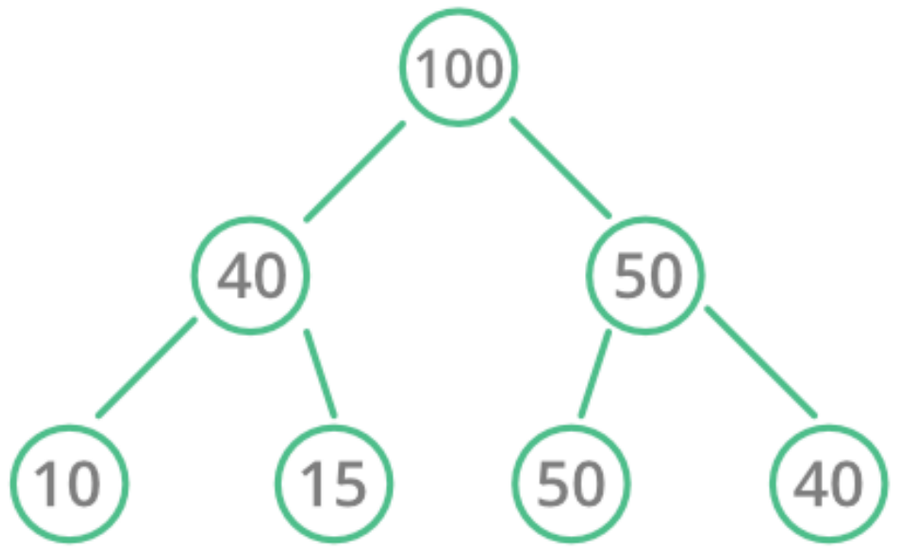


TRIE

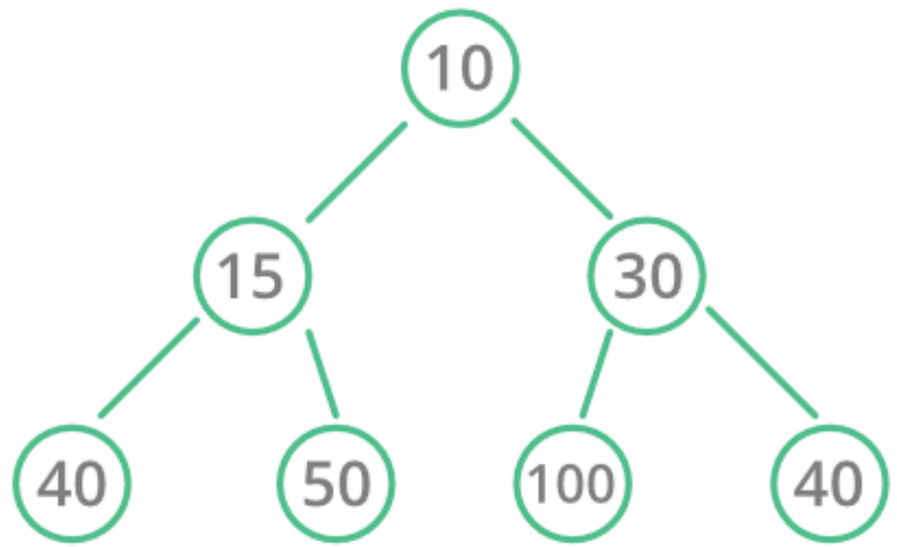


HEAP

- Types
  - Max-Heap
  - Min-Heap
- Applications
  - Heapsort
  - Selection algorithms
  - Graph algorithms
  - Priority Queue
  - Order statistics



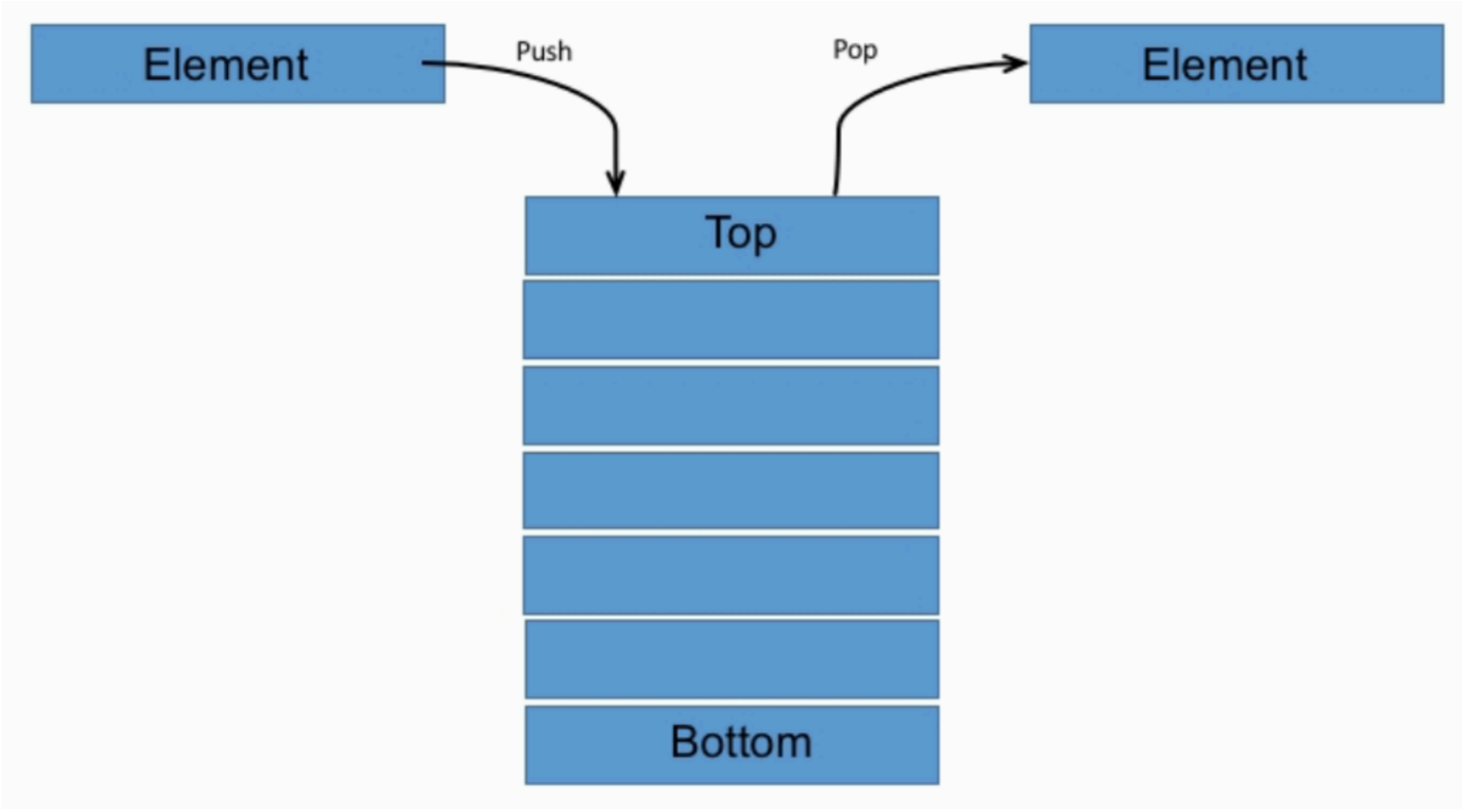
Max Heap



Min Heap

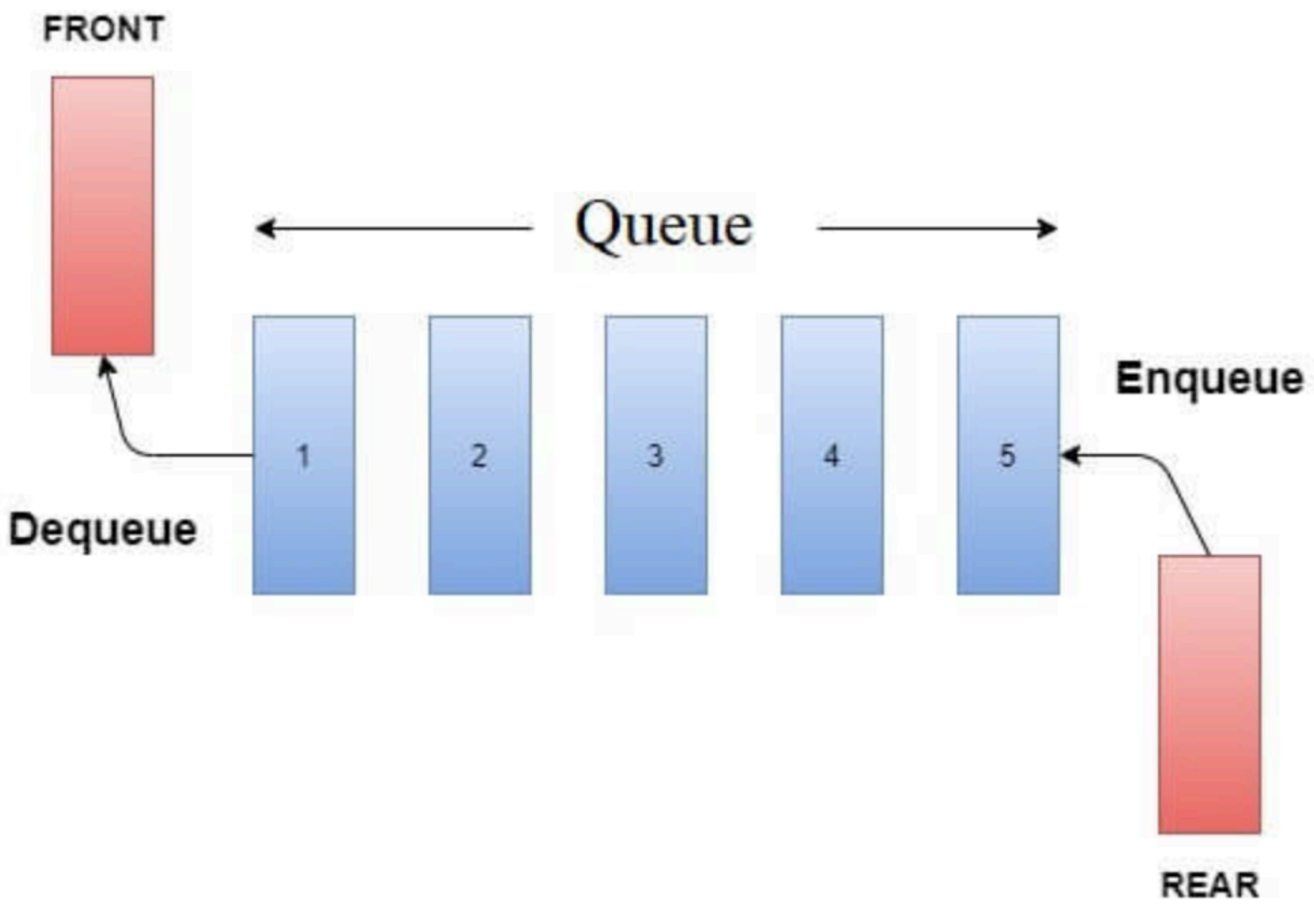
STACK

- Linear data structure which follows a particular order in which the operations are performed
- LIFO (Last In First Out)



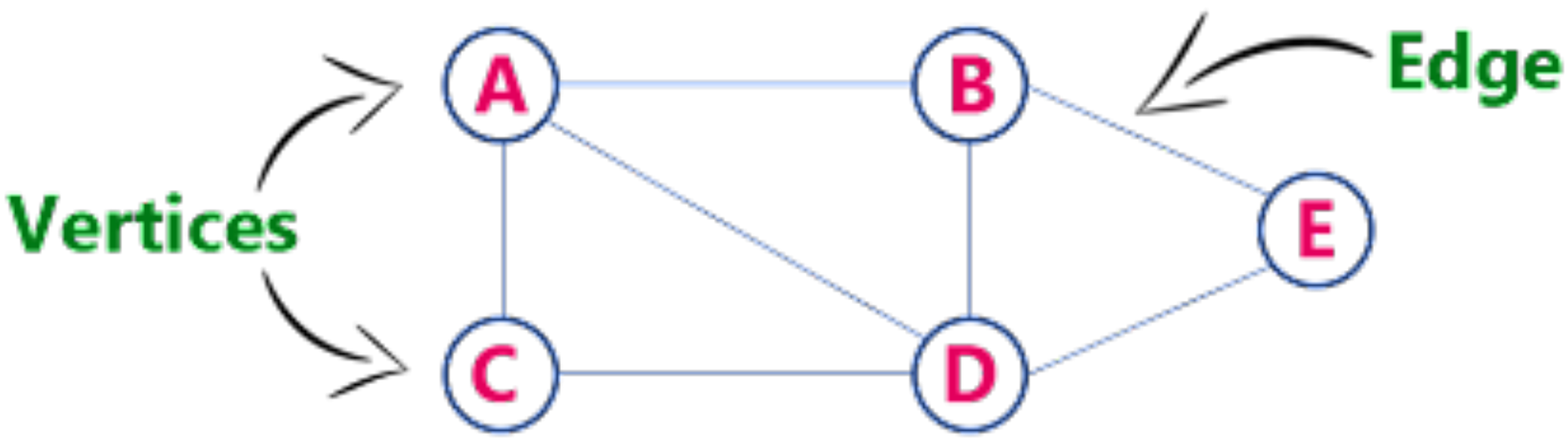
QUEUE

- Linear data structure which follows a particular order in which the operations are performed
- FIFO (First In First Out)



GRAPH

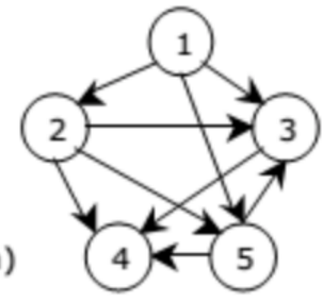
- A graph is a data structure that consists of the following two components:
  - A finite set of vertices (also called as nodes)
  - A finite set of ordered pair of the form (u, v) called as edge



GRAPH REPRESENTATIONS

- Adjacency Matrix
  - 2D array of size V x V where V is the number of vertices in a graph
  - The indices represents the edge from one vertex to the other
  - The value represents the weight

G1:

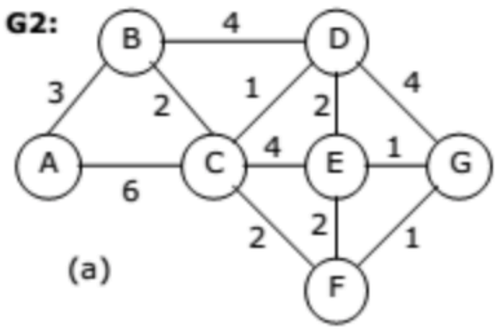


(a)

(b)

	1	**2**	**3**	**4**	**5**
1	0	1	1	0	1
2	0	0	1	1	1
3	0	0	0	1	0
4	0	0	0	0	0
5	0	0	1	1	0

G2:



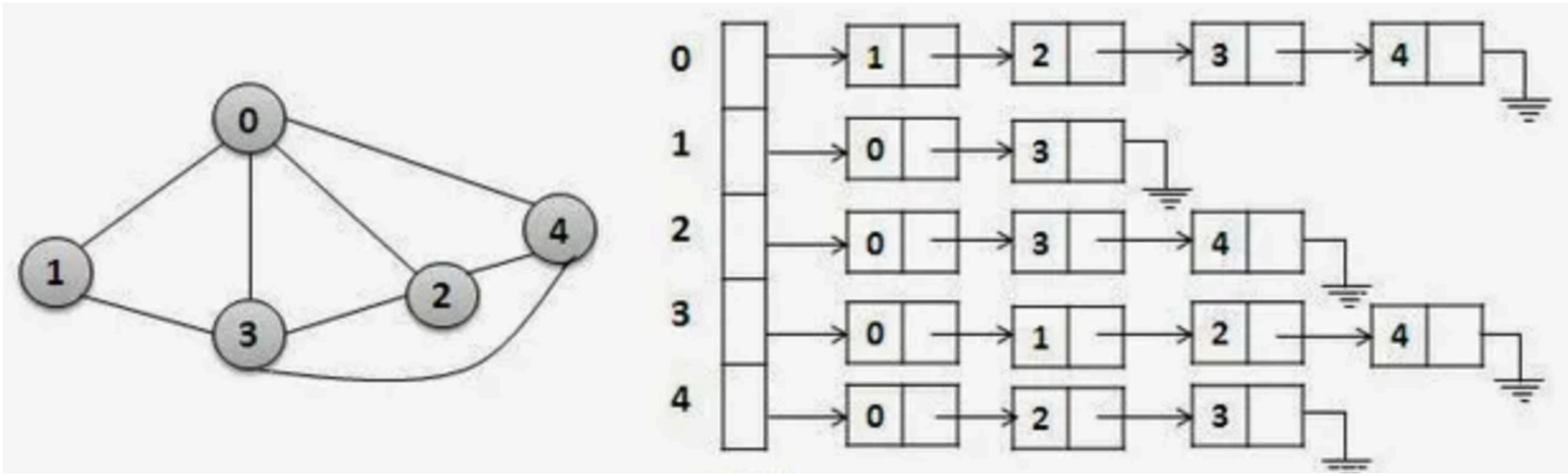
(a)

(b)

	A	**B**	**C**	**D**	**E**	**F**	**G**
A	0	3	6	∝	∝	∝	∝
B	3	0	2	4	∝	∝	∝
C	6	2	0	1	4	2	∝
D	∝	4	1	0	2	∝	4
E	∝	∝	4	2	0	2	1
F	∝	∝	2	∝	2	0	1
G	∝	∝	∝	4	1	1	0

GRAPH REPRESENTATIONS

- Adjacency List
  - The size of the array is equal to the number of vertices
  - The index represents the list of vertices adjacent to the ***i***'th vertex



BIG O NOTATION

- Also called Asymptotic Notations
- Big O Notation describes the complexity of your code using algebraic terms
- The dominant term is the term that grows the fastest
  - $g(n) = n^2 + 5n + 6$

INSERTION SORT EXAMPLE

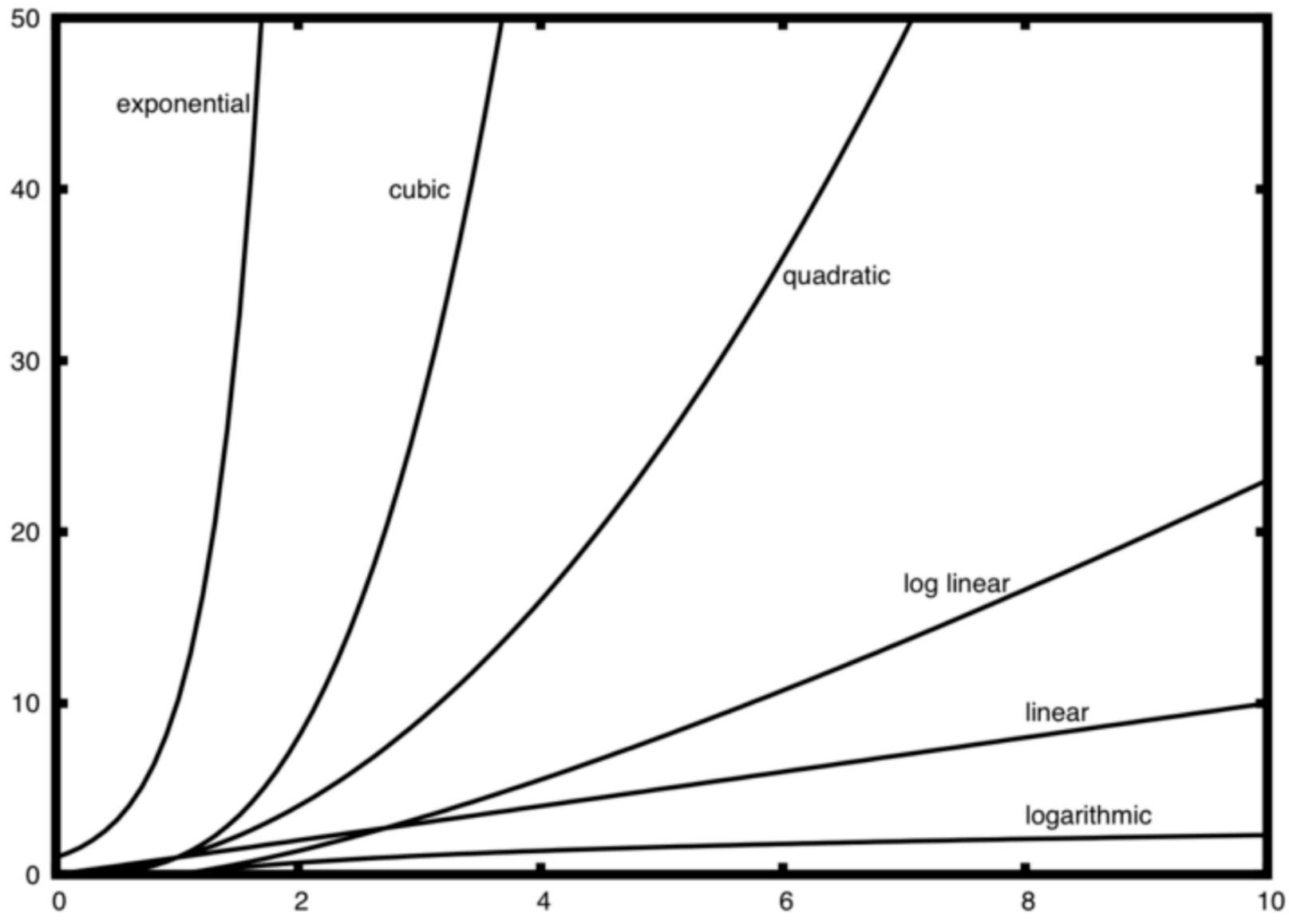
```
insertionSort(List) {
  for(i from 0 to List.Length) {
    SmallestElement = List[i]
    for(j from i to List.Length) {
      if(SmallestElement > List[j]) {
        SmallestElement = List[j]
      }
    }
    Swap(List[i], SmallestElement)
  }
}
```

- 1+ 2 + … + n times = n(n-1)/2 times = $n^2/2 - n/2$
- The dominant term is n^2

IMPORTANCE OF BIG O NOTATION

- Why do we need it?
- Do we actually use it in our day to day?
- With a better CPU and memory, can we stop thinking about it?

IMPORTANCE OF BIG O NOTATION



A RULE OF THUMB

- Top of the art
  - O(1)
- Pretty good
  - O(n*log(n))
- Ok
  - O(n)
- Bad
  - O(n^2)
- Unacceptable
  - O(2^n)
  - O(n!)

TIME AND SPACE COMPLEXITY

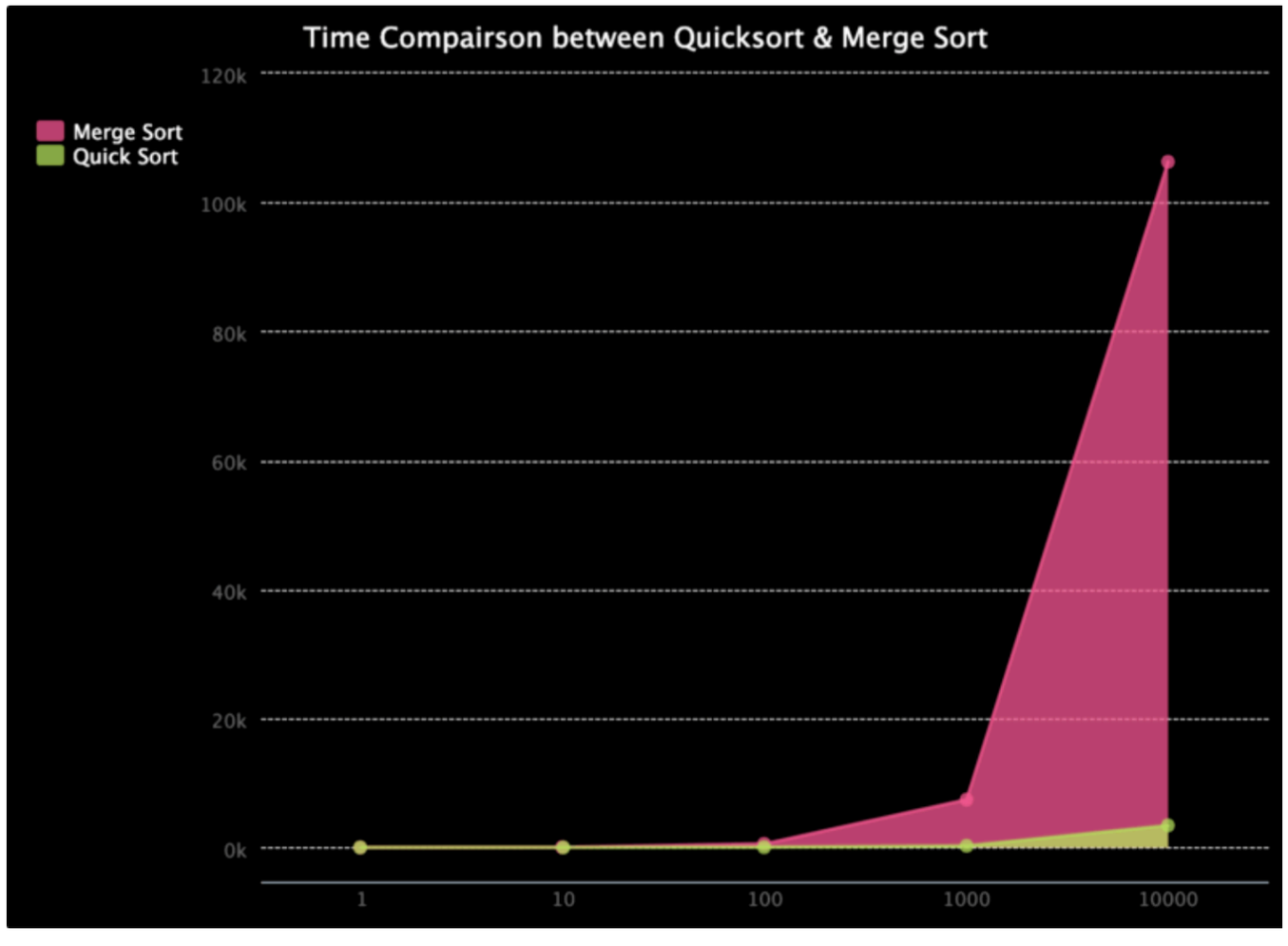
- We always evaluate both time and space complexity
- Trade-offs
  - Count the number of occurrences of an element in an array of integers
  - Get the top n players' score

DIFFERENT CASES IN THE SAME ALGORITHM (INSERTION SORT)

- Best-case time complexity
  - The array is sorted -> O(n)
- Average/Expected-case time complexity
  - Depend on your probability of sorted elements
- Worst-case time complexity
  - None of the elements are sorted -> O(n^2)
- Generally we evaluate the complexity of the worst-case

NOT ALWAYS THE WORST-CASE

- Quick sort worst-case complexity is O(n^2)
- Merge sort worst-case complexity is O(n*log(n))



Array Sorting Algorithms

Algorithm	Time Complexity			Space Complexity
	Best	Average	Worst	Worst
Quicksort	Ω(n log(n))	θ(n log(n))	O(n^2)	O(log(n))
Mergesort	Ω(n log(n))	θ(n log(n))	O(n log(n))	O(n)
Timsort	Ω(n)	θ(n log(n))	O(n log(n))	O(n)
Heapsort	Ω(n log(n))	θ(n log(n))	O(n log(n))	O(1)
Bubble Sort	Ω(n)	θ(n^2)	O(n^2)	O(1)
Insertion Sort	Ω(n)	θ(n^2)	O(n^2)	O(1)
Selection Sort	Ω(n^2)	θ(n^2)	O(n^2)	O(1)
Tree Sort	Ω(n log(n))	θ(n log(n))	O(n^2)	O(n)
Shell Sort	Ω(n log(n))	θ(n(log(n))^2)	O(n(log(n))^2)	O(1)
Bucket Sort	Ω(n+k)	θ(n+k)	O(n^2)	O(n)
Radix Sort	Ω(nk)	θ(nk)	O(nk)	O(n+k)
Counting Sort	Ω(n+k)	θ(n+k)	O(n+k)	O(k)
Cubesort	Ω(n)	θ(n log(n))	O(n log(n))	O(n)

Common Data Structure Operations

Data Structure	Time Complexity								Space Complexity
	Average				Worst				Worst
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion	
Array									
Stack									
Queue									
Singly-Linked List									
Doubly-Linked List									
Hash Table									
Binary Search Tree									

Common Data Structure Operations

Data Structure	Time Complexity								Space Complexity
	Average				Worst				Worst
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion	
Array	Θ(1)	Θ(n)	Θ(n)	Θ(n)	O(1)	O(n)	O(n)	O(n)	O(n)
Stack	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Queue	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Singly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Doubly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Hash Table	N/A	Θ(1)	Θ(1)	Θ(1)	N/A	O(n)	O(n)	O(n)	O(n)
Binary Search Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)

MATH – TRY AT HOME

- Rank following functions from the most complex to the lease complex:

n^{100}	$(\sqrt{2})^{\log n}$	n^2	$\sqrt{n}$	$2^{2^{n+1}}$
n^3	$\log^2 n$	$\log(n!)$	2^{2^n}	$\frac{1}{n^{\log n}}$
$\log n$	$2^{100 \log n}$	$n \cdot 2^n$	$n^{\log \log n}$	$\log_3 5n$
$\binom{n}{2} + n \log n$	$4^{\log n}$	n	$n \log n$	$2^{\log^{1.001} n}$

ALGORITHMS

- What is an algorithm
  - Finite sequence of well-defined, computer-implementable instructions
  - Algorithms are always unambiguous and are used as specifications for performing calculations, data processing, automated reasoning, and other tasks

DAY TO DAY USAGES

- Cook book recipes
- Dressing
- Enter your browser
- Brushing your teeth
- Infinite more..

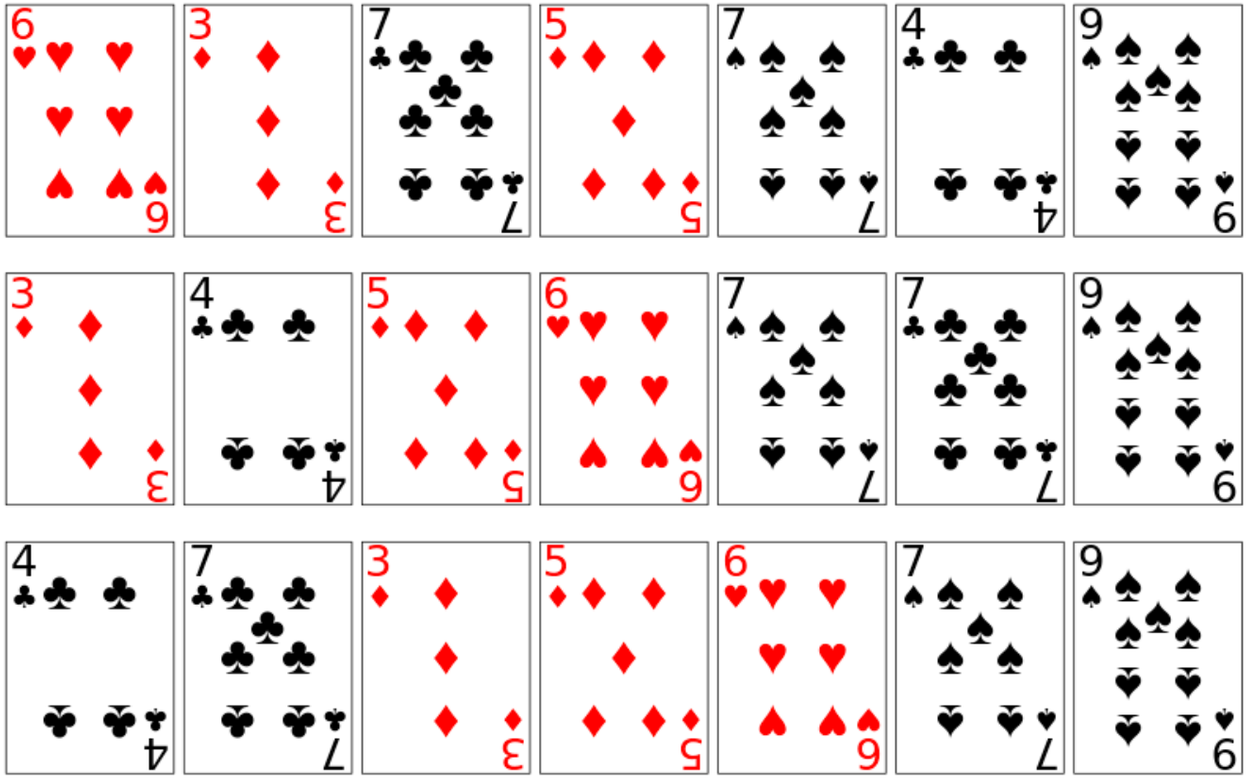
- Software algorithm
  - Summing up all the elements
  - Convert a Dictionary into Array
  - Infinite more..

COMMON ALGORITHMS

- Sorting
- Shortest Path
- Minimal Spanning Tree
- BSA
- Topological sort
- Flood fill
- Random/Shuffle
- BFS & DFS

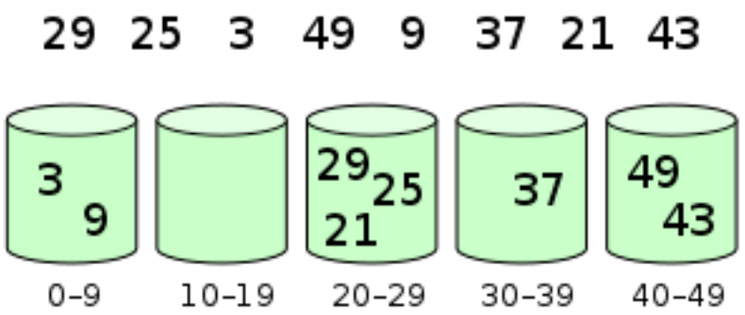
SORTING

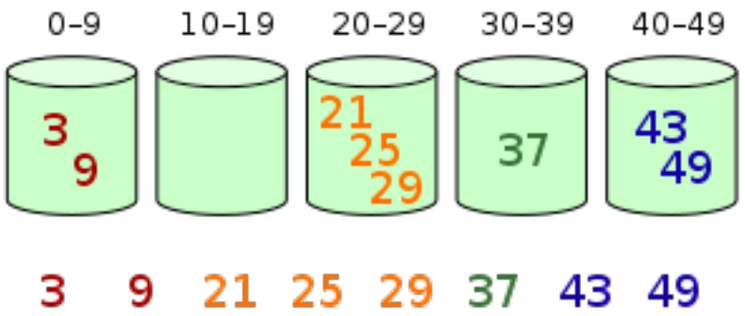
- Comparison sorts
  - Quicksort
  - Merge sort
  - Insertion sort
- Non-comparison sorts
  - Bucket sort
  - Radix sort



- Stability
  - important to preserve order over multiple sorts on the same data set

RADIX SORT





Most significant digit, forward recursive [edit]

Input list, fixed width numeric strings with leading zeros:

[170, 045, 075, 025, 002, 024, 802, 066]

First digit, with brackets indicating buckets:

[{045, 075, 025, 002, 024, 066}, {170}, {802}]

Notice that 170 and 802 are already complete because they are all

Next digit:

[{ {002}, {025, 024}, {045}, {066}, {075} }, 170, 802]

Final digit:

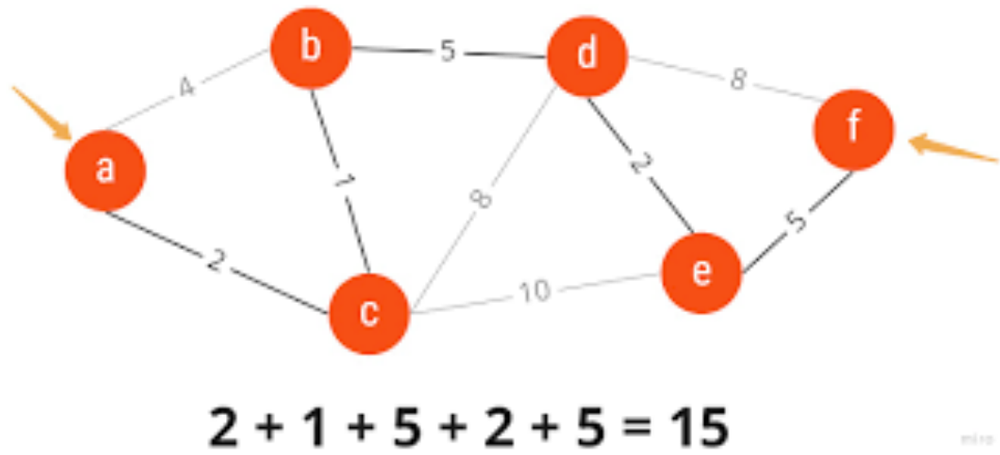
[002, { {024}, {025} }, 045, 066, 075 , 170, 802]

All that remains is concatenation:

[002, 024, 025, 045, 066, 075, 170, 802]

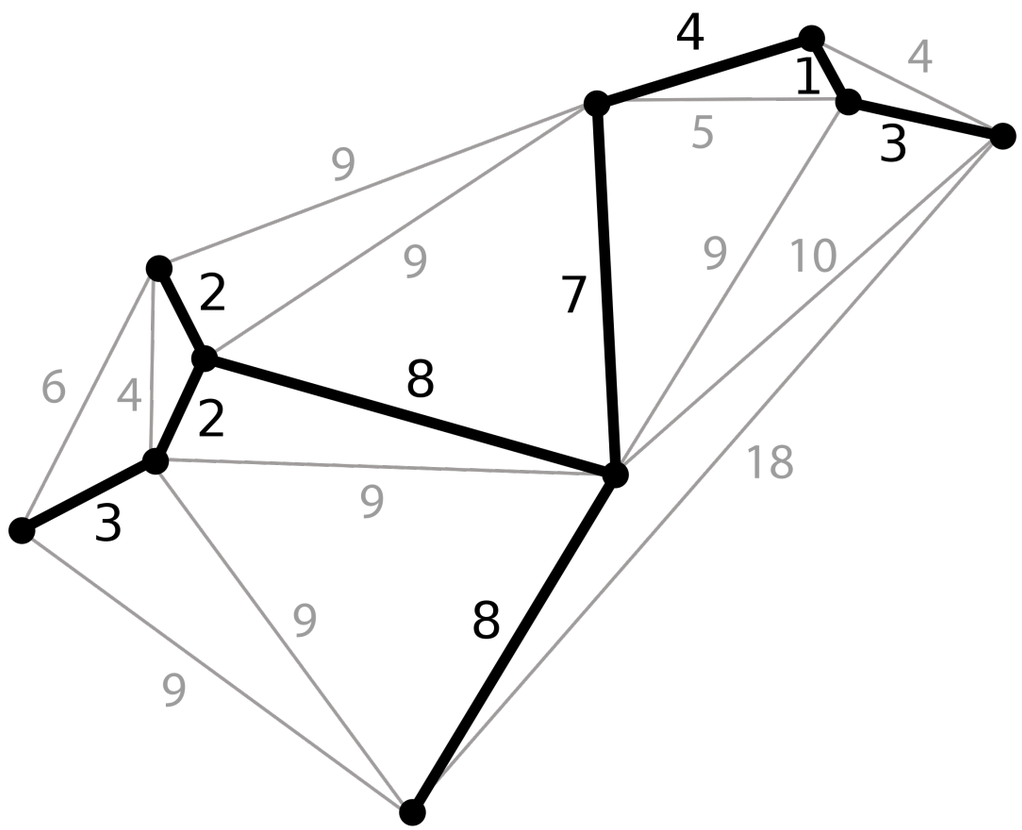
SHORTEST PATH

- Finding a path between two vertices in a graph such that the sum of the weights of its constituent edges is minimized
- Algorithms
  - Dijkstra's algorithm solves the single-source shortest path problem with non-negative edge weight.
  - Bellman–Ford algorithm solves the single-source problem if edge weights may be negative.
  - A* search algorithm solves for single-pair shortest path using heuristics to try to speed up the search.
  - Floyd–Warshall algorithm solves all pairs shortest paths.



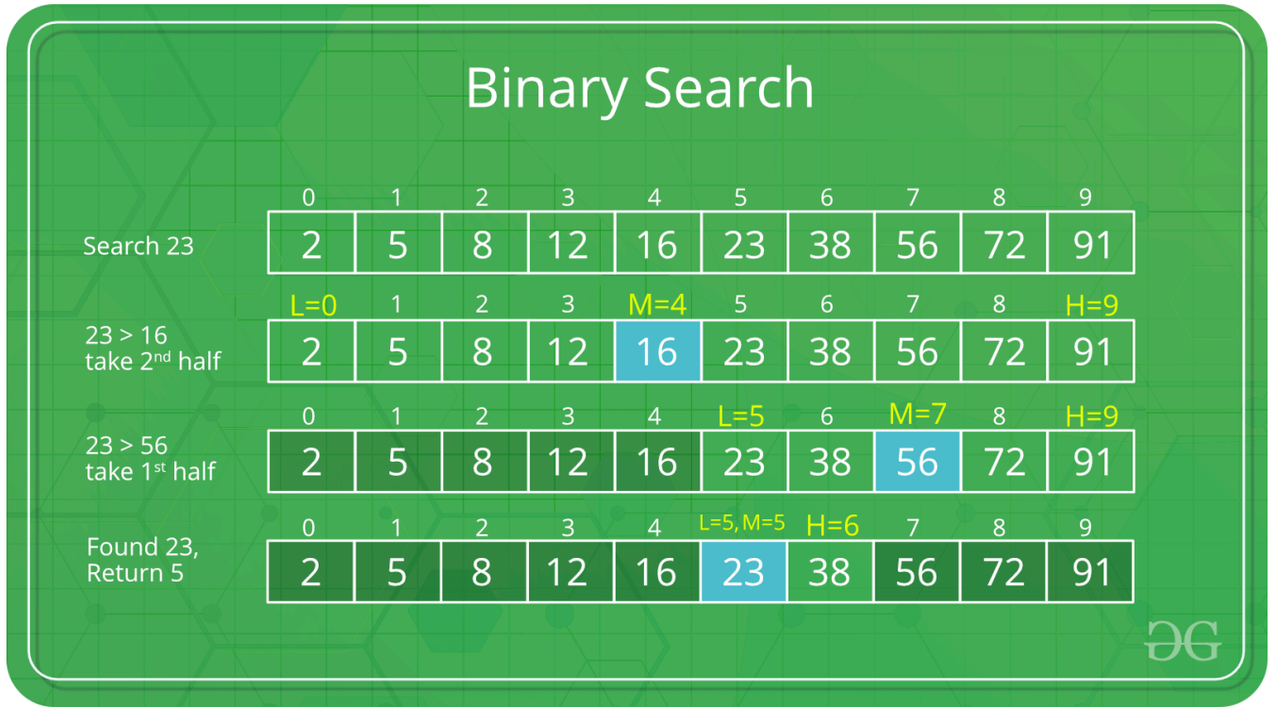
MINIMUM SPANNING TREE (MST)

- Applications
  - Networks in general
    - Computer
    - Telecommunications
    - Transportation
    - Water supply
    - Electrical grids
  - Taxonomy (categorization or classification)
  - Cluster analysis
  - Circuit design
- Algorithms
  - Kruskal's algorithm
  - Prim's algorithm



BINARY SEARCH ALGORITHM (BSA)

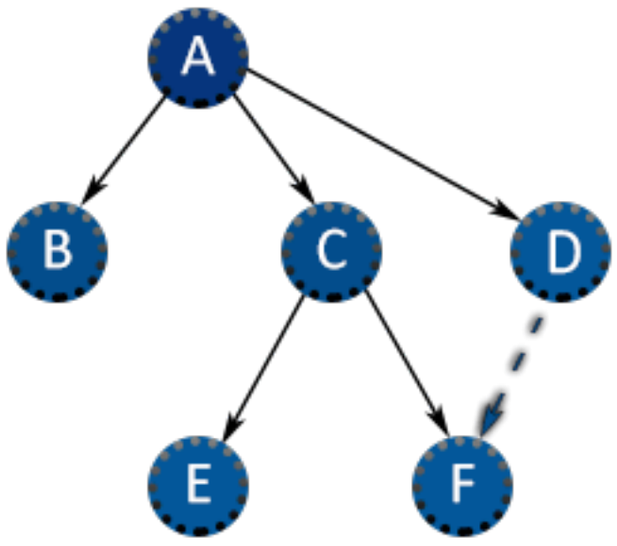
- Search on a sorted array by repeatedly dividing the search interval in half
- Repeatedly check until the value is found or the interval is empty



BFS & DFS

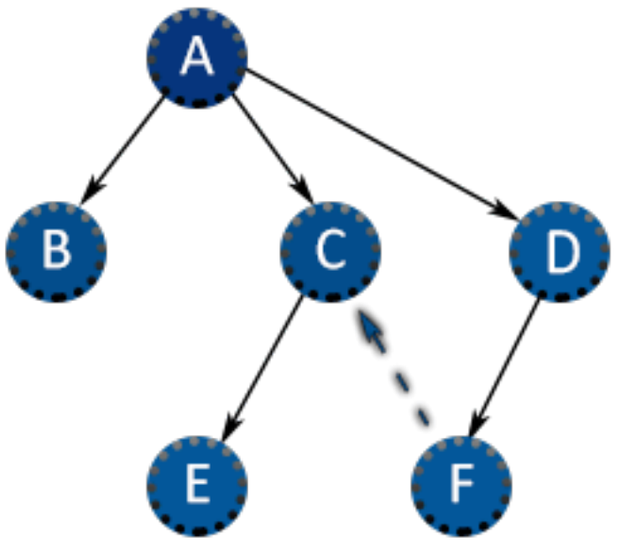
- Ways to traverse on a graph

BFS



A B C D E F

DFS

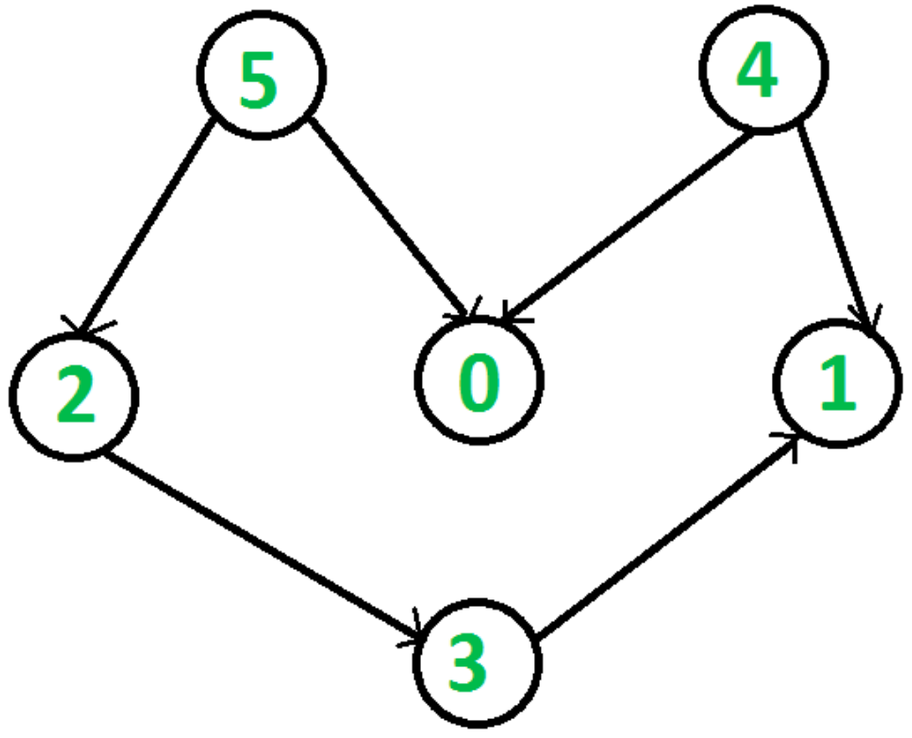


A D F C E B

TOPOLOGICAL SORT

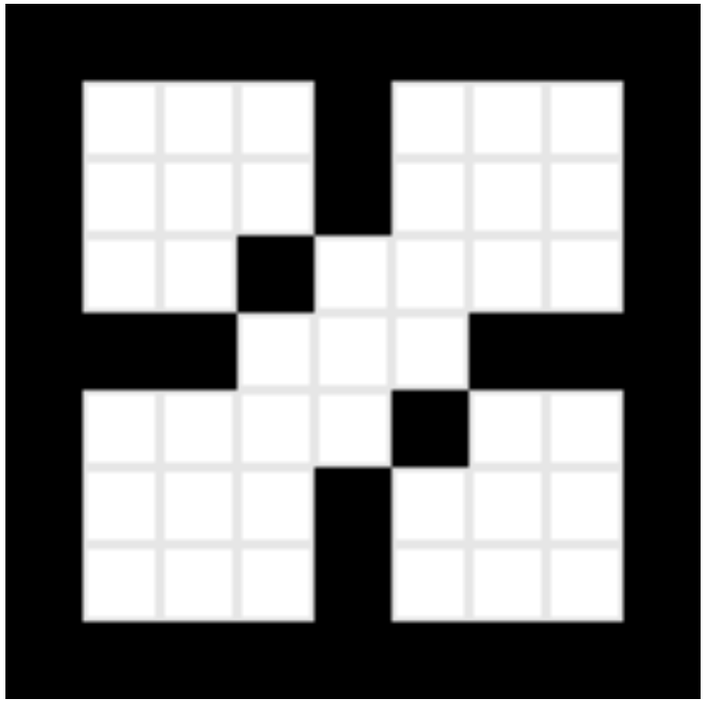
- For Directed Acyclic Graph (DAG) only
- A linear ordering of vertices such that for every directed edge (u, v), vertex u comes before v in the ordering

- Applications
  - Scheduling jobs
  - University course dependencies
  - Dressing app



FLOOD FILL

- Determines and alters the area connected to a given node in a multi-dimensional array with some matching attribute



THANK YOU!

TEXT

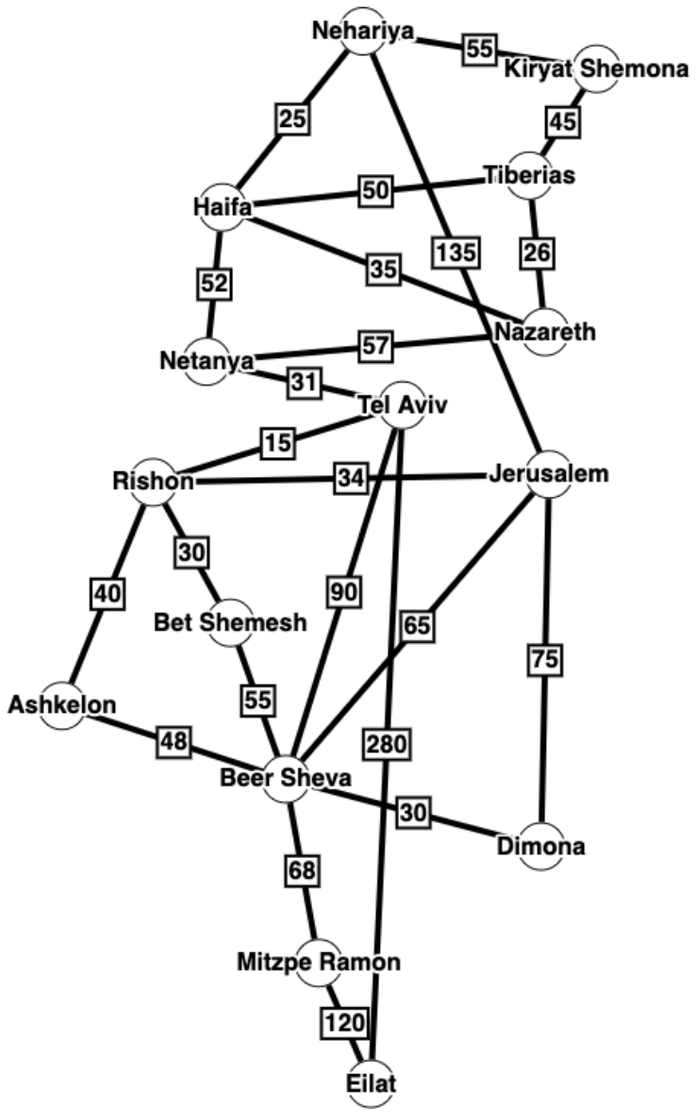
S.No.	**Separate Chaining**	**Open Addressing**
1.	Chaining is Simpler to implement.	Open Addressing requires more computation.
2.	In chaining, Hash table never fills up, we can always add more elements to chain.	In open addressing, table may become full.
3.	Chaining is Less sensitive to the hash function or load factors.	Open addressing requires extra care to avoid clustering and load factor.
4.	Chaining is mostly used when it is unknown how many and how frequently keys may be inserted or deleted.	Open addressing is used when the frequency and number of keys is known.
5.	Cache performance of chaining is not good as keys are stored using linked list.	Open addressing provides better cache performance as everything is stored in the same table.
6.	Wastage of Space (Some Parts of hash table in chaining are never used).	In Open addressing, a slot can be used even if an input doesn't map to it.
7.	Chaining uses extra space for links.	No links in Open addressing

TEXT

Underlying data structure	Lookup or Deletion		Insertion		Ordered
	average	worst case	average	worst case	
Hash table	O(1)	O(*n*)	O(1)	O(*n*)	No
Self-balancing binary search tree	O(log *n*)	O(log *n*)	O(log *n*)	O(log *n*)	Yes
unbalanced binary search tree	O(log *n*)	O(*n*)	O(log *n*)	O(*n*)	Yes
Sequential container of key–value pairs (e.g. association list)	O(*n*)	O(*n*)	O(1)	O(1)	No

TEXT


Kiryat Shemona
Nehariya
25
45
Haifa
50
Tiberias
35
Nazareth
52
57
Netanya
135
31
Tel Aviv
Rishon
40
30
34
Bet Shemesh
Jerusalem
Ashkelon
90
48
65
65
75
Beer Sheva
30
Dimona
68
Mitzpe Ramon
120
Eilat


Nehariya
55
Kiryat Shemona
25
45
Tiberias
Haifa
50
135
26
35
52
Nazareth
57
Netanya
31
Tel Aviv
15
Rishon
34
Jerusalem
30
40
90
Bet Shemesh
65
75
Ashkelon
55
48
280
Beer Sheva
30
Dimona
68
Mitzpe Ramon
120
Eilat