Algorithm

Let's learn about Data Structures and Algorithms

{} - Summary

What are Data Structures and Algorithms ?

{} 1 - What are Data Structures and Algorithms ?

Data Structure: Named location to store and organize data.

```
const array = ['h', 'e', 'l', 'l', 'o'];
```

An array is a data structure that allow us to store a collection of elements at contiguous memory location.

{} 1 - What are Data Structures and Algorithms ?

Algorithm: A set of steps to solve problem.

Given the following problem,
"Take me to school"

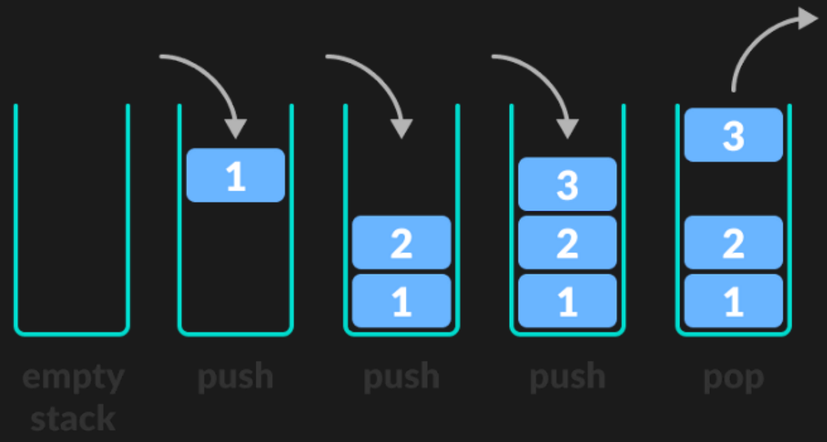
We can have the following algorithm,

1. Start the Car
2. Drive to the school location
3. Drive back to home

Data Structures

{} 2.1 - Data Structures / Stacks

Stack = LIFO data structure (Last-In First-Out)



push(), Add element to the top
pop(), Remove element from the top

Uses of Stacks ?
- Undo / Redo features
- Calling functions (Call Stack)

{} 2.2 - Data Structures / Queues

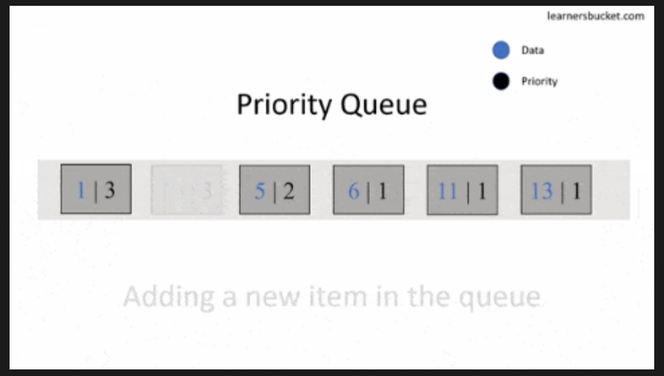
Queues = FIFO data structure (First-In First-Out)

enqueue(), Add element
dequeue(), Remove element

Uses of Queues ?

- Keyboard Buffer (keys appear in the order they're pressed)
- Printer Queue (Print jobs should be completed in order)

{} 2.3 - Data Structures / Priority Queues

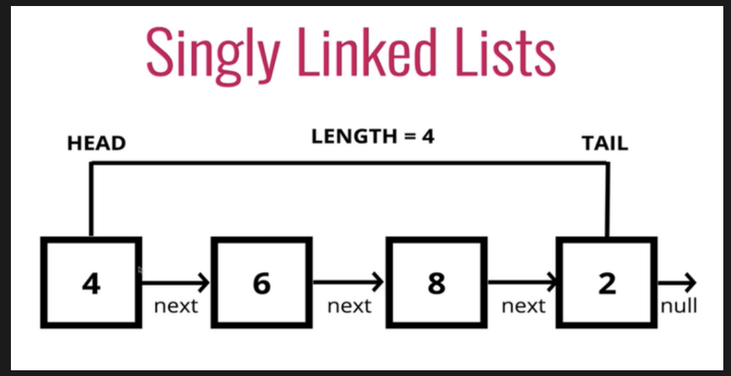


enqueue(), Add element
dequeue(), Remove element

Two ways to deal with priority queue,
- Add element according to their priorities
- Queue element without taking care of the priorities, dequeue() them according to their priorities

Uses of Priority Queue ?
- Load Balancing (Operating System instructions priorities)

{} 2.5 - Data Structures / Linked Lists

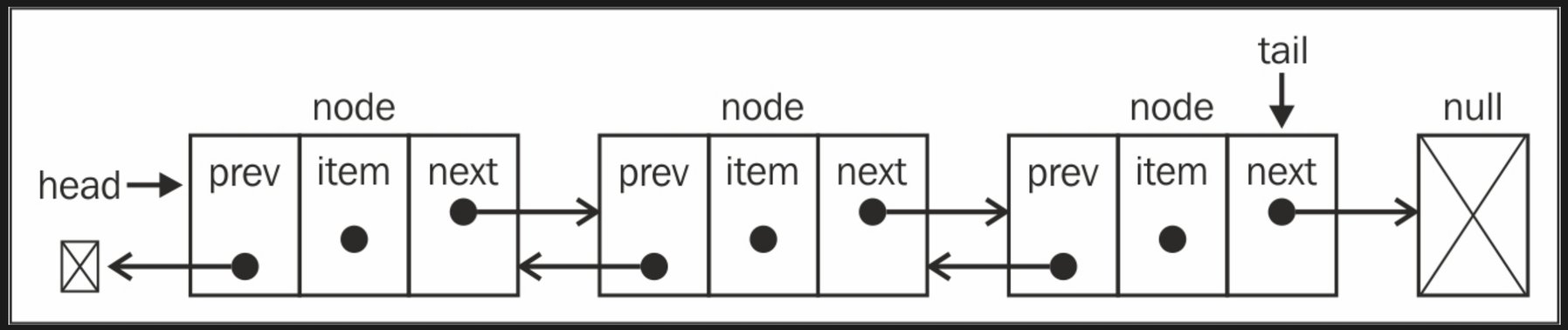


append(element), Add element to the list
insert(position, element), Add element at a given position
removeAt(position), Remove element at a given position

Advantage
- List size increase dynamically
- No need to shift elements while adding a new one.

Disadvantage
- Searching is really slow because we need to iterate over the whole list

{} 2.6 - Data Structures / Doubly Linked List



append(element), Add element to the list
insert(position, element), Add element at a given position
removeAt(position), Remove element at a given position

Advantage
- Reversing the list is easier

Disadvantage
- It use more memory (because of the previous pointer on each node)

Uses
- GPS Navigation
- Spotify Playlist

Big O Notation

{} 3 - Big O Notation

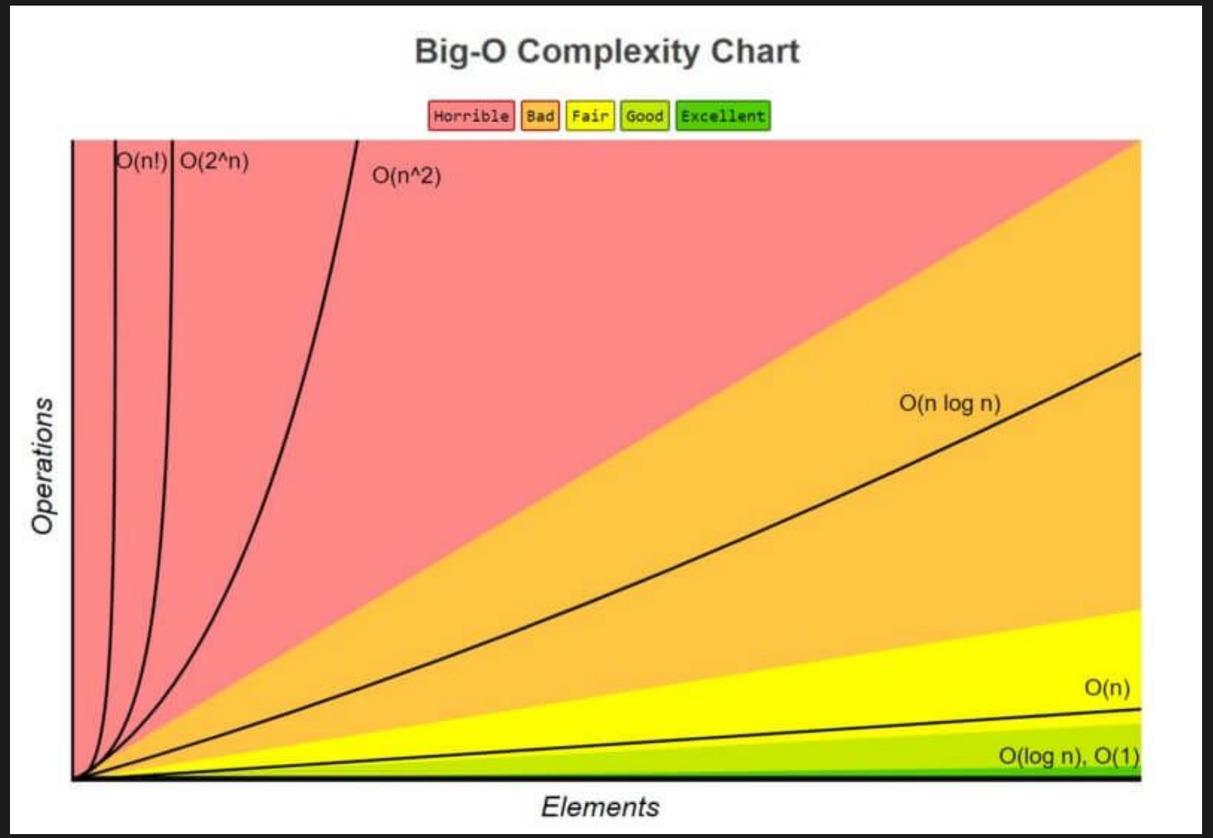
“Big O notation is a mathematical notation that describes the limiting behavior of a function when the argument tends towards a particular value or infinity. It is a member of a family of notations invented by Paul Bachmann, Edmund Landau, and others, collectively called Bachmann–Landau notation or asymptotic notation.”

- Wikipedia's definition of Big O Notation

Big O Notation describe the complexity of your code using algebraic terms.

{} 3 - Big O Notation

O(1) = constant time
O(log n) = logarithmic time
O(n) = linear time
O(n log n) = quasilinear time
O(n^2) = quadratic time
O(n!) = factorial time

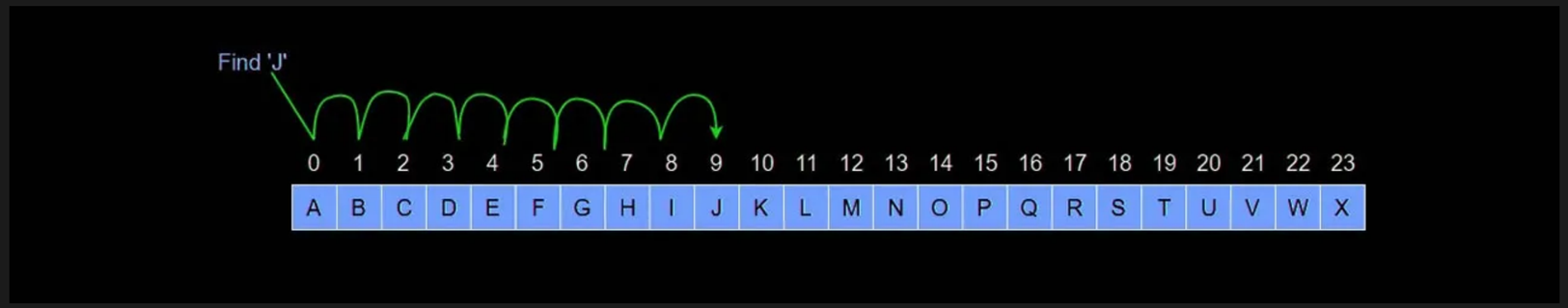


source: https://www.bigocheatsheet.com/

Algorithms

comming soon …

{} 4.1 - Algorithms / Linear Search



Runtime Complexity: O(n)

Disadvantage:

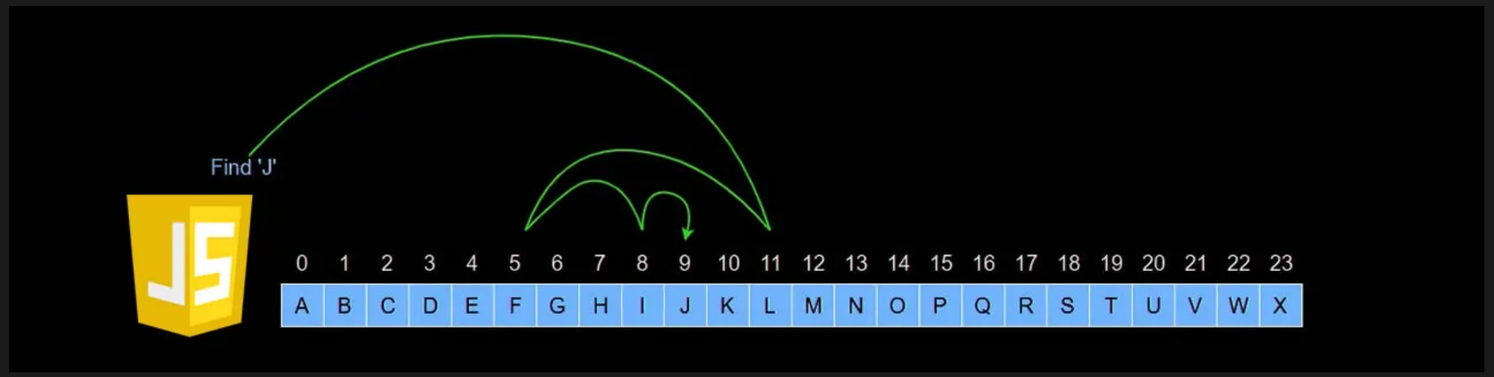
Slow for large data sets

Advantage:

Fast for searches of small to medium data sets

Useful for data structures with sequential access memory (SAM)

{} 4.2 - Algorithms / Binary Search



Requirement: Need a sorted Array.

Runtime Complexity: 0(log n)

SEE: https://medium.com/@samip.sharma963/binary-search-and-its-big-o-3333d13bd6ec)

Disadvantage:

Binary search is based on random access memory (RAM) caching is difficult

It can use recursive wich requires more stack space

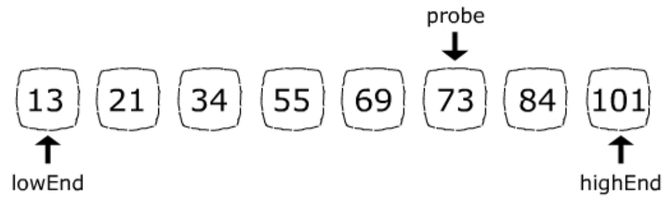
Advantage:

Significantly faster than linear search on large data

"Divide and conquer", It indicates wether the element being searched is before or after the current position. Each step we eliminate half of the array.

{} 4.3 - Algorithms / Interpolation Search

$$probe = lowEnd + \frac{(highEnd - lowEnd) \times (item - data[lowEnd])}{data[highEnd] - data[lowEnd]}$$



Requirement: Need a sorted Array, It perform really well if data is "uniformly" distributed.

Runtime Complexity: O(log(log n))

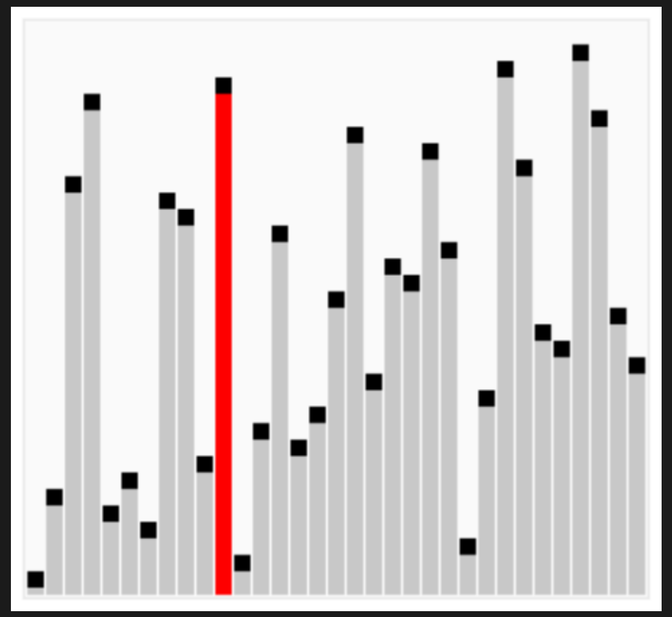
Disadvantage:

If the data is not "uniformly" distributed the Runtime Complexity O(n) for worst cases.

Advantage:

If the data is "uniformly" distributed it goes really fast, even faster than binary search.

{} 4.4 - Algorithms / Bubble Sort



Runtime Complexity: O(n^2)

Ok - Small Dataset
Bad - Large Dataset

Disadvantage:

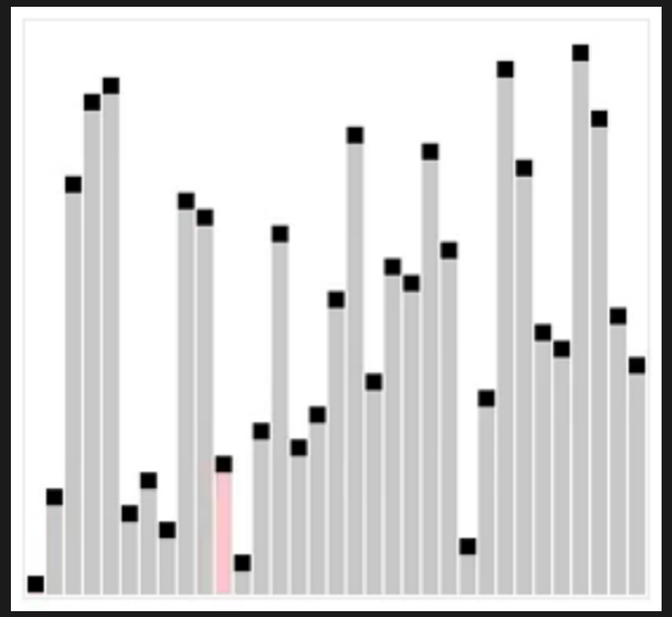
It's a pretty slow algorithm.

Advantage:

Really simple algorithm.

Little memory overhead, and data is already in memory ready to process.

{} 4.5 - Algorithms / Selection Sort



Runtime Complexity: O(n^2)

Ok - Small Dataset
Bad - Large Dataset

Disadvantage:

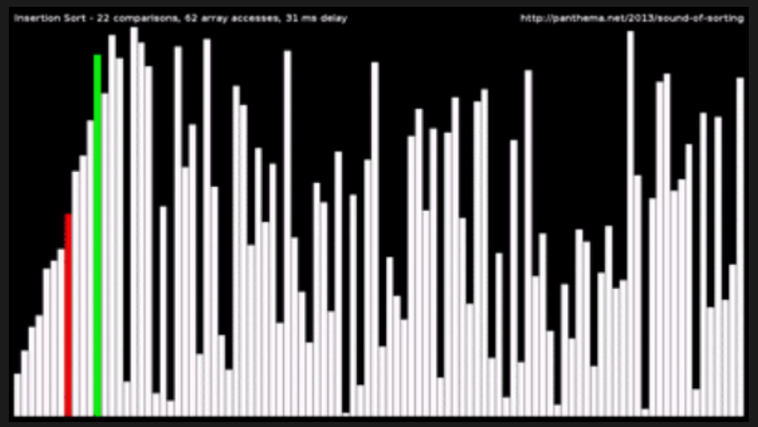
It's a pretty slow algorithm.

Advantage:

Really simple algorithm.

Little memory overhead, and data is already in memory ready to process.

{} 4.6 - Algorithms / Insertion Sort



Runtime Complexity: O(n^2)

Ok - Small Dataset

Bad - Large Dataset

Disadvantage:

It's a pretty slow algorithm.

Advantage:

Little memory overhead, and data is already in memory ready to process.