JavaScript Data Structures

**1. Array**

An **array** is an object used to store a collection. This collection could be anything: numbers, objects, more arrays, etc. In JavaScript, arrays are used to store multiple values in one variable that can be accessed through an index (which starts from zero). The values stored in the array are called **elements** and the number of elements stored in an array is typically referred to as the array’s **length.**

**Advantages**

* Simple to create and use.
* Foundational building block for complex data structures

**Disadvantages**

* Fixed size
* Expensive to insert/delete or resequence values
* Inefficient to sort

**Applications**

* Basic spreadsheets
* Within complex structures such as hash tables

**2. Queue**

Queues are conceptually similar to stacks; both are sequential structures, but queues process elements in the order they were entered rather than the most recent element. As a result, queues can be thought of as a FIFO (First In, First Out) version of stacks. These are helpful as a buffer for requests, storing each request in the order it was received until it can be processed.

**Advantages**

* Dynamic size
* Orders data in the order it was received
* Low runtime

**Disadvantages**

* Can only retrieve the oldest element

**Applications**

* Effective as a buffer when receiving frequent data
* Convenient way to store order-sensitive data such as stored voicemails
* Ensures the oldest data is processed first

**3. Linked List**

Linked lists are a data structure which, unlike the previous three, does not use physical placement of data in memory. This means that, rather than indexes or positions, linked lists use a referencing system: elements are stored in nodes that contain a pointer to the next node, repeating until all nodes are linked. This system allows efficient insertion and removal of items without the need for reorganization.

**Advantages**

* Efficient insertion and removal of new elements
* Less complex than restructuring an array

**Disadvantages**

* Uses more memory than arrays
* Inefficient to retrieve a specific element
* Inefficient to traverse the list backward

**Applications**

* Best used when data must be added and removed in quick succession from unknown locations

**4. Trees**

Trees are another relation-based data structure, which specialize in representing hierarchical structures. Like a linked list, nodes contain both elements of data and pointers marking its relation to immediate nodes.

Each tree has a “root” node, off of which all other nodes branch. The root contains references to all elements directly below it, which are known as its “child nodes”. This continues, with each child node, branching off into more child nodes.

Nodes with linked child nodes are called internal nodes while those without child nodes are external nodes. A common type of tree is the “binary search tree” which is used to easily search stored data. These search operations are highly efficient, as its search duration is dependent not on the number of nodes but on the number of levels down the tree.

**Advantages**

* Ideal for storing hierarchical relationships
* Dynamic size
* Quick at insert and delete operations
* In a binary search tree, inserted nodes are sequenced immediately.
* Binary search trees are efficient at searches; length is only O(height).

**Disadvantages**

* Slow to rearrange nodes
* Child nodes hold no information about their parent node
* Binary search trees are not as fast as the more complicated hash table
* Binary search trees can degenerate into linear search (scanning all elements) if not implemented with balanced sub-trees.

**Applications**

* Storing hierarchical data such as a file location.
* Binary search trees are excellent for tasks needing searching or ordering of data.

**5. Graphs**

Graphs are a relation-based data structure helpful for storing web-like relationships. Each node, or vertex, as they’re called in graphs, has a title (A, B, C, etc.), a value contained within, and a list of links (called edges) it has with other vertices.

**Advantages**

* Can quickly convey visuals over text
* Usable to model a diverse number of subjects so long as they contain a relational structure

**Disadvantages**

* At a higher level, text can be time-consuming to convert to an image.
* It can be difficult to see the existing edges or how many edges a given vertex has connected to it

**Applications**

* Network representations
* Modelling social networks, such as Facebook.

**6. Hash table**

Hash tables are a complex data structure capable of storing large amounts of information and retrieving specific elements efficiently. This data structure relies on the concept of key/value pairs, where the “key” is a searched string and the “value” is the data paired with that key.

Each searched key is converted from its string form into a numerical value, called a hash, using a predefined hash function. This hash then points to a storage bucket – a smaller subgroup within the table. It then searches the bucket for the originally entered key and returns the value associated with that key.

**Advantages**

* Key can be in any form, while array’s indices must be integers
* Highly efficient search function
* Constant number of operations for each search
* Constant cost for insertion or deletion operations

**Disadvantages**

* Collisions: an error caused when two keys convert to the same hash code or two hash codes point to the same value.
* These errors can be common and often require an overhaul of the hash function.

**Applications**

* Database storage
* Address lookups by name

Each hash table can be very different, from the types of the keys and values, to the way their hash functions work. Due to these differences and the multi-layered aspects of a hash table, it is nearly impossible to encapsulate so generally.