Learn Data Structures and Algorithms

Free

Java

- Data Structure is a named location that can be used to store and organize data.

- Algorithm is a collection of steps to solve a problem

- A call stack is a stack data structure that stores information about the active subroutines of a computer program.

Content:

1. What are Static Arrays
2. Dynamic Arrays.
3. Bigh O Notation
4. Linear Search
5. Binary Search
6. Interpolation Search
7. Stack
8. Queue
9. Priority Queue
10. Linked lists.
11. Linked lists vs ArrayList
12. Bubble Sort
13. Selection Sort
14. Insertion Sort
15. Recursion
16. Merge Sort
17. Quick Sort
18. Hash Table (Hashing, Bucket, Collision)
19. Graphs.
20. Adjacency Matrix
21. Adjacency list
22. Depth First Search
23. Breadth First Search
24. Tree
25. Binary Tree
26. Binary Search Tree
27. Tree Traversal (Inorder, pre order, post order).
28. Calculate execution Time.

~~Dynamic~~

Static Array stores elements in consecutive memory addresses.

Accessing an element takes $O(1)$ time.

Searching of any element takes $O(n)$ time

Adding/Deleting any element takes $O(n)$ time

Dynamic Array called as ArrayList in Java, Vector in C++, Array in Java script and List in Python.

~~Once the~~ Dynamic Array has its own static Array.

Once the static Array reaches its capacity, a new static array is added with an increased capacity.

The capacity is increased 1.5-2 times.

~~Advantages~~

Random access in $O(1)$, Good locality reference & data cache utilization since continuous memory location.

Easy to insert or delet at the end.

~~Disadvantages~~

wastes more memory, Shifting is time consuming $O(n)$,

Expanding/Shrinking the array is time consuming $O(n)$

No predefined functions in Java.

Bigh O Notation stands for "How code slows as data grows".

time ↓

$O(n)$ → linear time

$O(1)$ → constant time.

$O(\log n)$ → linear time
Increasingly less time to complete
(more the data, lesser the time).

position according to comparision.

$O(n \log n)$ → Quasilinear time, slows down with the increase in data

$O(n^2)$ → quadratic time, more data more increase in time.

$O(n!)$ → factorial time, Extremely slow.

---

linear Search we Iterate through a collection one element at a time.

runtime complexity : $O(n)$

Disadvantage:
Slow for large data sets

Advantages:
Fast for small to medium data sets
Does not need to sort
Useful for ds with no random access.

Binary Search search algorithm that finds the position of a target value within a sorted array. Half of the array is eliminated during each step. (Sorted array needed)

runtime complexity: $O(\log n)$

In Java we can use binarySearch( ) method that takes two arguments, the array & the target element to be searched. This method returns index of the element found, -1 otherwise.

Interpolation search improvement over binary search best used for uniformly distributed data guesses where a value be based on calculated probe results if probe is incorrect, search area is narrowed and a new probe is calculated

average case complexity: $O(\log(\log(n)))$

wost case : $O(n)$ (Sorted array needed)

$$probe = low + (high - low) * \frac{(value - array[low])}{(array[high] - array[low])}$$

if (array[probe]=value) ; index

if (array[probe] < value) low= probe +1

if (array[probe] < value) high= probe -1

Stack is LIFO Data Structure - Last In First Out

Push() to add to the top

Pop() to remove from the top.

Stack is a Linear Data Structure

Functions in a Stack : push, pop, peek, empty, search

Peek() used to return the topmost element of stack

Search() will return the index of the element passed as an argument if present, else -1.

Note: The Index of the stack begins with 1 for search. So the top index is 1.

Queue is FIFO Data Structure - First In First Out

Queue is a collection designed for holding elements prior to processing.

Queue is a linear data structure.

Enqueue(element) → Adds the element to the queue

Dequeue(element) → Removes the element from the queue.

Queue in Java [No inbuilt - can be built using LinkedList]

offer() → To add the element into Queue

poll() → To remove the element from Queue

peek() → Examines & returns the element in and pointing the HEAD.

isEmpty() → Checks if the Queue is Empty or not.

size() → Returns the number of elements

contains() → Returns true if the argument element is present in the Queue. else false.

Priority Queue is a FIFO data structure that serves elements with the highest priorities first before elements with lower priority.

In PriorityQueue, if the elements are numbers, then they automatically are arranged in ascending order.

To make it in descending order, we have to change or pass on the args Collections.reverseOrder()

For strings, they are arranged alphabetically.

Linked Lists

Do not have index, non continuous address location.

Last element pointer is NULL.

First element HEAL (First)

last element TAIL.(LAST)

In Java, LinkedList by default are Dably Linked list.

LinkedList.push() → Adds the element to the HEAD

pop() → Pops or removes the element at HEAD.

offer() → Adds the element to the HEAD.

poll() → Pops or removes the element at HEAD.

add(x,y) → will add y in the index x.

remove(y) → will remove the element y from the linked list

indexOf(y) → returns the position of the element in the linked list starting with 0.

peekFirst() → returns HEAD (without removing)

peekLast() → returns TAIL

addFirst() → Adds an element to the HEAD.

addLast() → Adds an element to the TAIL

removeFirst() → returns HEAD (By removing)

removeLast() → returns TAIL

LinkedList is a data structure that stores a series of nodes where each Nodes store in 2 parts (data + address).

Nodes are Non-consecutive memory locations.

Elements are linked used pointers.

Single Linked List

[data | address] → [data | address] · -- [data | address]

Doubly Linked List

[address | data | address] ↔ [address | data | address] ... [address | data | ad

Advantages

Dynamic memory allocation, operations are easy with O(1)

No memory loss.

Disadvantages:

Greater memory usage, No random access of elements, Searching takes more time with O(n)

LinkedList vs ArrayList (Dynamic Array)

For get() → ArrayList is faster than LinkedList

get() → returns an element of args position.

For remove() → LinkedList is faster than ArrayList for elements to ~~remove~~ in the ~~middle~~. Start

Arraylist is faster than Linked List for elements in the middle, end.

Bubble Sort

Time complexity = $O(n^2)$ Space complexity = $O(1)$

pairs of adjacent elements are compared and the elements swapped if they are not in order.

Not suggested for larger data.

Advantages

Simple, Stable, In-place, Suitable for partially sorted arrays.

Disadvantages:

Slow speed, less efficient.

Selection Sort

Time complexity = $O(n^2)$ Space-complexity = $O(1)$

Search through an array and keep track of the minimum value during each iteration. At the end of each iteration swap variables.

Advantages

- Easy to understand, Efficient for small data sets, Fewer swaps, In place sorting, can select k^{th} smallest or largest element without fully sorting

Disadvantages

Inefficient for large datasets, Not stable, Makes a lot of comparisions, Might need to be replaced.

Insertion Sort

After comparing elements to the left shift elements to the right to make room to insert a value

time complexity = $O(n^2)$, Best case $O(n)$

- Space complexity = $O(1)$

Advantages

less steps than bubble sort

Disadvantages

Bad for big dataset.

Recursion when a thing is defined in terms of itself.

A recursive method is the one that calls itself.

Advantages

Easier to read/write

Easier to debug.

Disadvantages

Slower sometimes

Uses more memory.

Needs a base case & a recursive case.

Merge Sort divide and conquer algorithm.

Time complexity = O(nlogn) Space complexity = O(n)

Steps:

First divide the array into half, right array & left array. move the elements into it.

~~Sort the~~ & ~~If the~~ elements are ~~greater than~~ the middle element,

Recursively repeat the first step, left, right then merge.

merge function takes left, right & full array.

Advantages

Fixed time complexity of O(nlogn)

Stable

Efficient for linked lists.

Easy to understand.

~~Disadvantages~~

Need more space

less efficient for small

Cannot modify original

Performance doesn't improve even if sorted.

Quick Sort Pivot Approach.

Steps:
First get the pivot in a position such that,
elements in the left are lesser than pivot.
elements in the right are more than pivot.
Recursively call the quick sort function.

Quick sort algorithm moves smaller elements to left of a pivot. recursively dividing array in 2 partitions

run-time complexity: Best case, Avg case = $O(n \log(n))$

Worst case = $O(n^2)$

space complexity = $O(\log(n))$.

Hash Table

Java
put( ) → Takes two args (key, value) & adds to table

keySet( ) → Returns the set of keys from the table

get( ) → Takes one arg (key) & returns the value.

remove( ) → Takes one arg (key) & removes the entry.

hashCode( ) → returns the hascode, % capacity. of which we can obtain the index (for Int). Different datatypes have different value of hashCode

Hash Table is a data structure that stores unique key values.
Each key/value pair is known as an Entry.

Fast insertion, look up, deletion of key/value pairs.
Not ideal for small datasets, great with large.

Hashing takes a key & computes an integer.
In a Hashtable, we use the hash % capacity to calculate an index number.

bucket is an indexed storage location for one or more entries, can store multiple entries in case of collision.

Collision occurs because hash function generates the same index for more than one key.

Runtime complexity:

Best Case O(1) [No collisions]
Worst Case O(n) [Exclusive collisions]

Advantages

Fast access.
Efficient insert/delete
Efficient space if used correctly.
Flexible.

Disadvantages

Collision handling.
Hash function dependency
Unordered data.
Wastes memory.
When has many collisions, performance significantly decreases.

Graphs is a non-linear data structure that represents relationship between objects. It's made up of vertices (nodes) and edges, which connect the vertices.

Types:

- Directed : graph with ordered pairs of vertices, also known as edges, arrows or arcs.
- undirect : A graph with unordered pairs of vertices, also known as edges.
- Weighted ; A graph with edges that have values.

Adjacency Matrix An array to store 1's / 0's to represent edges.

# of rows = # of unique nodes

# of Colums = # of unique nodes

runtime complexity to check an Edge: O(1)

space complexity : $O(v^2)$ $v \rightarrow$ vertices.

If the edge exist's between ~~the row of i and col~~ i and j, then the element in the adjacency matrix A[i][j] = 1 else A[i][j] = 0.

Adjacency list An array/arrayList of LinkedLists.

Each LinkedList has a unique node at the head. All adjacent neighbors to that node are added to that node's LinkedList

runtime complexity to check an Edge: O(v)

space complexity: O(v+e) v → vertex, e → edge

Depth First Search a search algorithm for traversing a tree or graph data structure

Steps:

1. Pick a route
2. keep going until you reach a dead end, or a previously visited node.
3. Backtrack to last node that has unvisited adjacent neighbors.

Uses ~~Stack~~ Array/Stack. Traverse a graph branch by branch. children are visited before siblings.

Done by using an Adjacency Matrix.

Better if destination is on average far from the start.

Breadth First Search a search algorithm for traversing a tree or graph data structure. This is done one "level" at a time, rather than one "branch" at a time

Done by using the Adjacency Matrix.

Uses Queue. Better if destination is on average close to start.

Traverse a graph level by level.

Siblings are visited before children

Tree a non linear data structure where nodes are organized in a hierarchy.

Subtree : a smaller tree held within a larger tree

size of the tree = number of nodes.

depth of a node = number of edges below the root node

height of a node = number of edges above furthest leaf node

Binary Tree is a tree where each node has children not more than 2.

Binary Search Tree A tree data structure where each node is greater than it's left child, but less than it's right.

benefit: easy to locate a node when they are in this order

Time complexity: best case - O(log n)

worst case - O(n)

Space complexity: O(n)

Tree Traversal process of visiting all the nodes in a tree

Inorder → Leftnode + Root Node + rightnode

Post order → leftnode + rightnode + Root Node

Pre order → RootNode + leftnode + rightnode.

Calculate execution time

millisecond -10^{-3} s

micro second -10^{-6} s

nano second - 10^{-9} s.