Introduction to Data Structures and Algorithms.

Wednesday, May 26, 2021 2:17 AM

A data structure organizes and stores data. Each of them have their own strengths and weaknesses.

For example Array DS can search for an element quite fast when we know the index but when you know the data instead of the index then search is very slow.

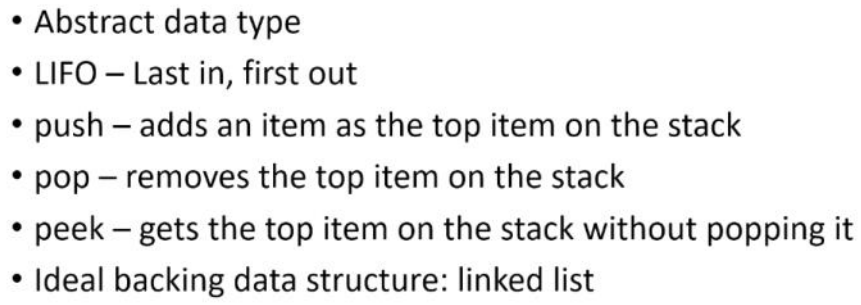
So, which is the best Data Structure ??

Ans. "it depends" on what data you want to store and how is it going to be accessed.

Algorithm is a set of steps required to accomplish a task. Implementation is the code we write for an algo.

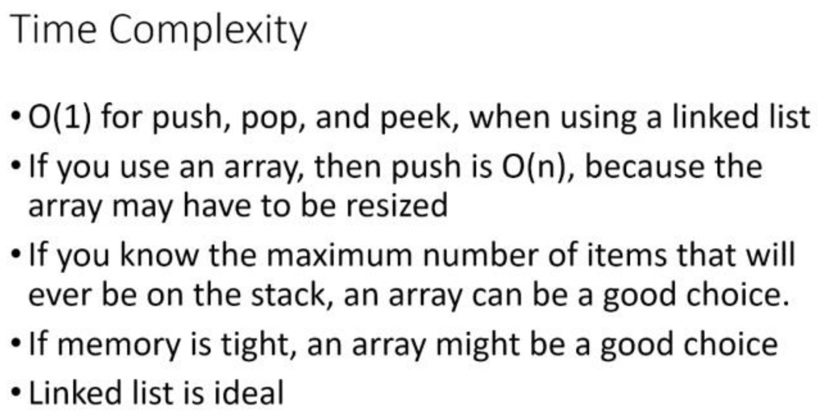
Stacks

Friday, June 11, 2021 7:09 PM



- Abstract data type
- LIFO – Last in, first out
- push – adds an item as the top item on the stack
- pop – removes the top item on the stack
- peek – gets the top item on the stack without popping it
- Ideal backing data structure: linked list

Function calls use Stack.
Ideal DS for Stack is Linked List bcz we just need to add/remove/see from top position ie one of the end of the list O(1) time complexity me ye sab ho jaega.
Array se bhi kar sakte he but usme top ko right end of array manna padega to achieve O(1) par phir bhi System.arraycopy() will be O(n).

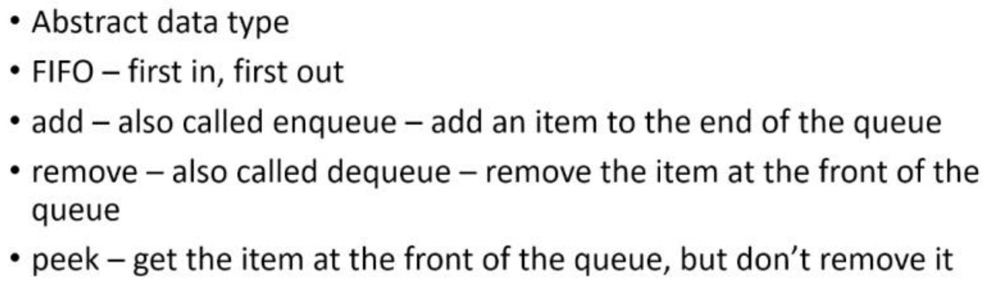


Time Complexity

- O(1) for push, pop, and peek, when using a linked list
- If you use an array, then push is O(n), because the array may have to be resized
- If you know the maximum number of items that will ever be on the stack, an array can be a good choice.
- If memory is tight, an array might be a good choice
- Linked list is ideal

Queues

Saturday, June 12, 2021 12:05 PM



- Abstract data type
- FIFO – first in, first out
- add – also called enqueue – add an item to the end of the queue
- remove – also called dequeue – remove the item at the front of the queue
- peek – get the item at the front of the queue, but don’t remove it

It can be implemented using **Linked list** (ideal, always O(1)) and Array List (if we have to resize then O(n), otherwise O(1)), time complexity is similar as that of stack.

Array List me ek cheez aur he we have to use Circular Array List, otherwise memory management is poor.

Big-Oh Notation

Wednesday, May 26, 2021 2:20 AM

Comparing running times of 2 algos is not the apt way to compare them.

Rather we use number of steps taken by that algo as it doesn't depend on hardware, this is call Time Complexity.

We also have Space/Memory Complexity, memory required for algo, nowadays memory is cheap that's why don't really care for this.

For Time Complexity we use worst case which gives us an upper bound of time required.

Definition of Big-O -

For a given function f(n) we denote it's big Oh notation by O(g(n)). **Mathematically, f(n) = O(g(n)) read as — "f of n is big oh of g of n", if and only if there exist positive constant c and n_0 such that $0 \leq f(n) \leq cg(n)$ for all n, $n \geq n_0$**

Big-O Notation is a way of expressing complexity, related to the number of steps an algo has to go through.

O(an expression). Pronounced as O of.

Example - Let tot. num. of steps req. = (2*n + 2), n is the total number of inputs, here +2 remains constant as n changes so does 2*, so they are not included in Time Complexity

Big-O is O(n) this is linear time complexity.

Remember for the above example Big-O could also be O(n^2), O(n^3) …. etc.

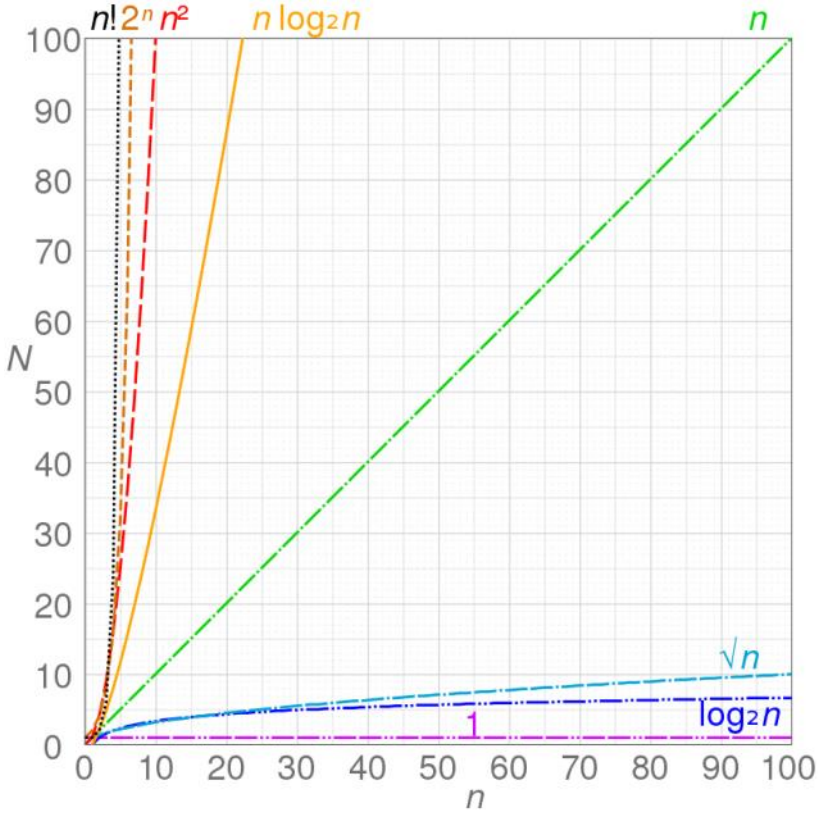
Some time complexities are -

1. O(1) - Constant
2. O($\log_2$ n) - Logarithmic
3. O(n) - Linear
4. O(n*$\log_2$ n) - n log star n
5. O(n^2) - Quadratic

Remember the base of log is 2.

The above order is from Best (1) to Worst (5) in terms of time required / time complexity.

Graph for some Big-O vs n (number of inputs) -


n!
2^n
n^2
n log₂ n
n
√n
log₂ n
1
N
n
0
10
20
30
40
50
60
70
80
90
100

Big-Omega Notation

Thursday, May 27, 2021 6:17 PM

$\Omega(g(n))$ = {f(n): there exist a positive constants c and n0 such that $0 \leq c\, g(n) \leq f(n)$ for all $n \geq$ n0}

It's used to find lower bound for time / space required.

Big-Theta Notation

Thursday, May 27, 2021 6:18 PM

$\Theta(g(n))$ = {f(n) : there exist a positive constants c1, c2 and n0 such that $0 \leq c1\ g(n) \leq f(n) \leq c2\ g(n)$ for all $n \geq n0$}

It's used to find a range for time / space required.

Array as DS

Wednesday, May 26, 2021 2:24 AM

Arrays are stored in memory as contiguous block of memory. Each element of array is of the same size.

When we create an array of say Object class or String class, then it seems such that each element can have different length b ut this is not how things are, actually for primitives the actual data is stored in the array element whereas for (non -primitives) Objects etc their reference is stored rather than the actual data and references are of same size.

If base address of array (address of arr[0]) is x and each element of array is of size y then address of arr[i] is (x + i*y).

Advantages of Arrays -

1. Arrays are efficient when we have to find element by index, Big-O for this is O(1) constant time complexity.
2. Also they are memory efficient as we only have to store our data and nothing else.

Disadvantages of Arrays -

1. Arrays are inefficient when we have to find element without knowing its index, Big-O for this is O(n) linear time complexity.

Some Big-Os for Array operations -

1. Adding elements to a full array - O(n) , pehle ke naya bada array bana padega fir usme sare ke sare n elements copy karenge by looping n times so linear time comlexity ayegi.
2. Adding an element to the last of an array that has space - O(1) , only 1 step is required --> arr[last index + 1] = data;
3. Insertion/Deletion at specific index - O(n), just think about its worst case and figure out num. of steps required.
4. Updation at a particular index - O(1), only 1 step is required --> arr[index] = data;

Lists

Wednesday, May 26, 2021 2:26 AM

List is basically a sequential arrangement of elements.

Intro to Abstract Datatypes -

1. Don't dictate how data is actually stored in memory / how its organized.
2. Dictate the operation we can perform on that data.

Concrete DS like an Array ke liye JDK me Class he "Arrays" name ki, but for an Abstract Data Type say ArrayList, Linked List etc have an Interface rather than a Class as they don't want to tell us how data is stored in memory.

Vector vs **ArrayList** -

Vector is synchronized whereas ArrayList is unsynchronized, matlab agar mai multiple threads se ArrayList me Write/Modify operation kara to mujhe thread conflict mil sakta he, Note - multiple threads me sirf read karne ke liye there is no issue. Whereas for Vector this problem is not there.

So, if you want thread safety use Vector and if not then use ArrayList. Note - ArrayList me bhi MANUALLY thread synchronization kar sakte he.

Apart from this difference ArrayList and Vector are quite identical.

So, why did the ArrayList class came in the picture ?? It's due to the reason that synchronization comes with a slow down in performance.

ArrayList aur Vector dono are made from AbstractList.

Singly LinkedList -

- In each element of a LinkedList we have to store data as well as a reference to the next element, this is a problem we are tight on memory.
- LinkedList has a head/fistNode field that points to the 1st element of the list.
- The 1st element's next field points to the 2nd element, 2nd element's next field points to 3rd ……………….
- Last node pe null ka reference hota he to tell us that the list has ended.

Serach for linked list has O(n) linear as we will have to iterate.

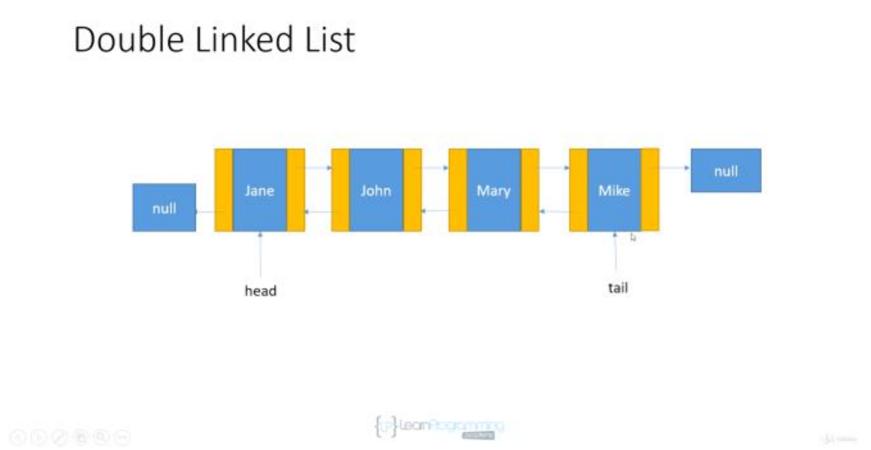
The insertion/deletion/updation operation for a linked list have O(1) constant time complexity for adding/deleting/updating at/from index = 0.

As for insertion/deletion/updation operation at some index other than 0 we will have to iterate through the list to get to that element hence it has O(n) linear time complexity.

Singly Linked List is best used when we want to add/delete/update elements from the front of our list.

Linked list ka "firstNode" parameter is not a normal node, it just points to the index = 0 node, it's NOT the index = 0 node.

Doubly LinkedList -



Double Linked List

Doubly Linked List is best used when we want to add/delete/update elements from the front or back of our list.

if we want to do any of these 3 operation in the middle, it would again be O(n) as we'll have to iterate.

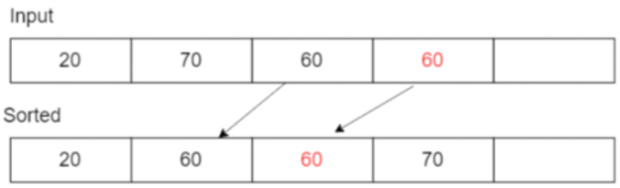
Basics of Algorithms

17 June 2021 08:51

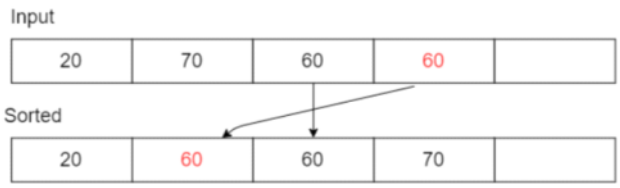
★ In-place Algorithm – An Algorithm who's extra memory requirement (excluding memory to store input) doesn't depend on the input size is called an In-place Algorithm. For sort Algorithm, extra space req. Should be independent of the number of elements to be sorted.
Hindi – Input ko store karne ke alawa apko jitni memory chahiye apki algorithm run karne ke liye that should be independent of input.

Internal Sort – All the data to be sorted is stored in the computer's Main Memory (RAM), no secondary memory is required.
External Sort – Some of the data to be sorted is in external / secondary memory.

Stable Sort - Relative ordering of duplicate items is preserved.



Unstable Sort – Relative ordering of duplicate items is NOT preserved.



★ Stable Sorts are preferred over Unstable Sorts.

Radix Sort

Friday, July 9, 2021 9:23 PM

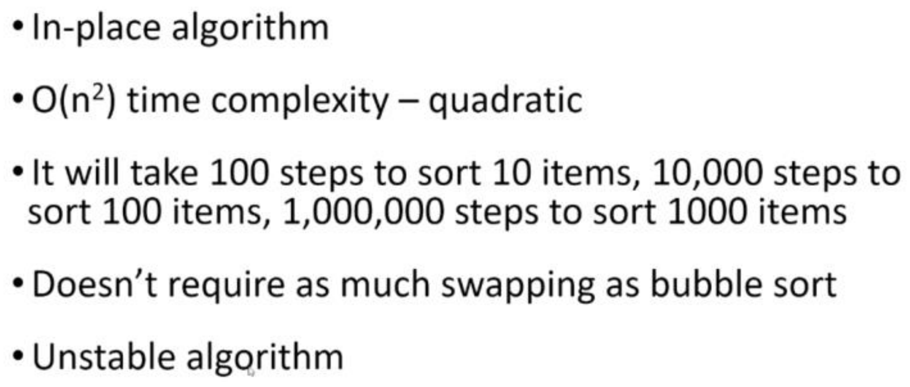
https://www.geeksforgeeks.org/radix-sort/

The idea of Radix Sort is to do digit by digit sort starting from least significant digit to most significant digit. Radix sort uses counting sort as a subroutine to sort.

Time complexity – O(n) can be achieved, refer the article.

Selection Sort

Saturday, June 26, 2021 7:21 PM



- In-place algorithm
- O(n^2) time complexity – quadratic
- It will take 100 steps to sort 10 items, 10,000 steps to sort 100 items, 1,000,000 steps to sort 1000 items
- Doesn't require as much swapping as bubble sort
- Unstable algorithm

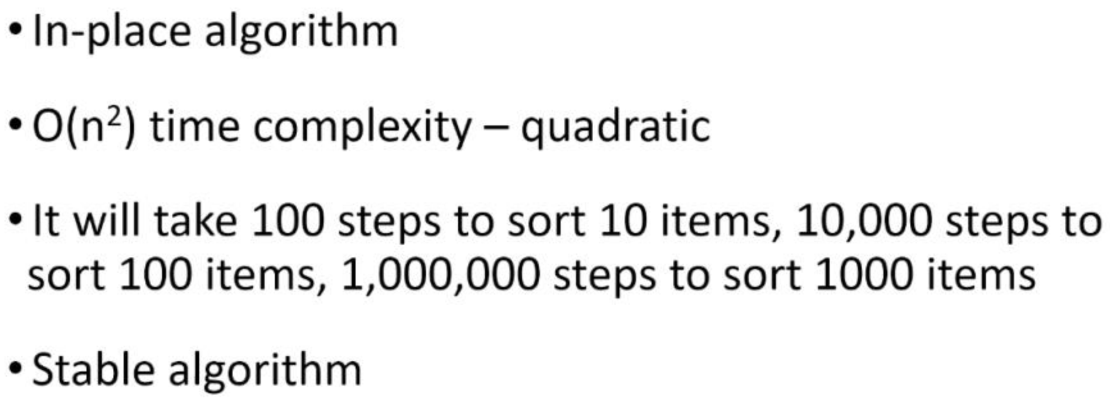
Default Selection Sort algorithm is UNSTABLE, could be made stable by using Insertion sort.

Time Complexity: O(n^2) ALWAYS, as there are two nested loops.
Auxiliary Space: O(1)
In Place : Yes

GENRALLY, Selection Sort performs better than Bubble Sort.

Insertion Sort

Saturday, June 26, 2021 7:53 PM



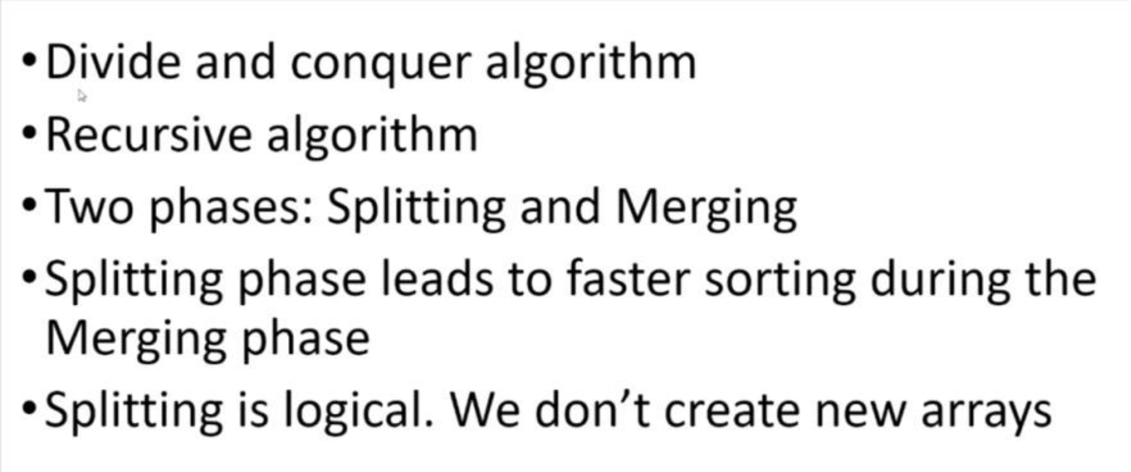
- In-place algorithm
- $O(n^2)$ time complexity – quadratic
- It will take 100 steps to sort 10 items, 10,000 steps to sort 100 items, 1,000,000 steps to sort 1000 items
- Stable algorithm

Worst and Average Case Time Complexity: $O(n^2)$. Worst case occurs when array is reverse sorted.
Best Case Time Complexity: O(n). Best case occurs when array is already sorted.
Auxiliary Space: O(1)

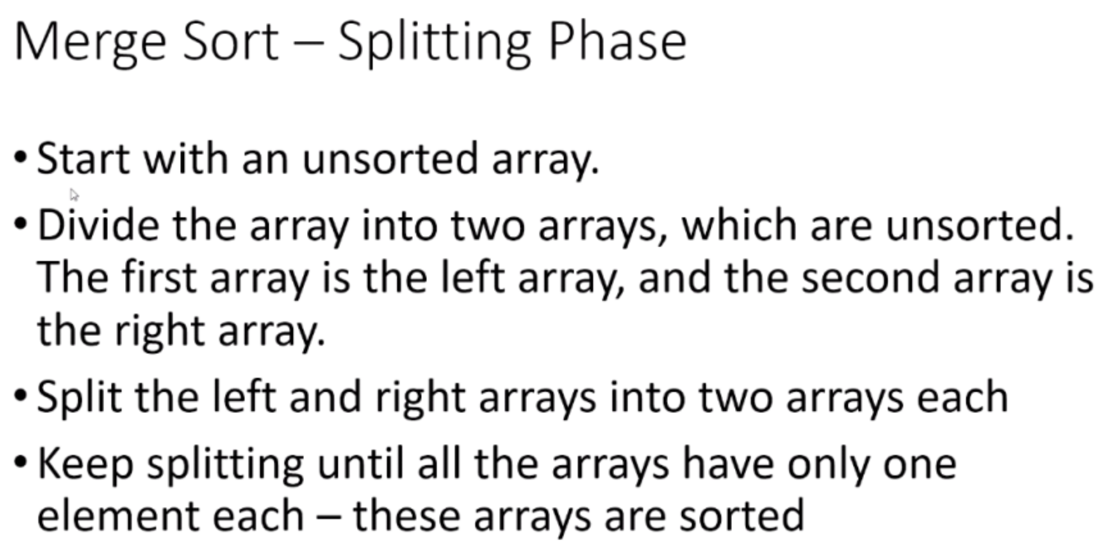
It's a stable sort algo.

Merge Sort

Tuesday, June 29, 2021 9:10 AM

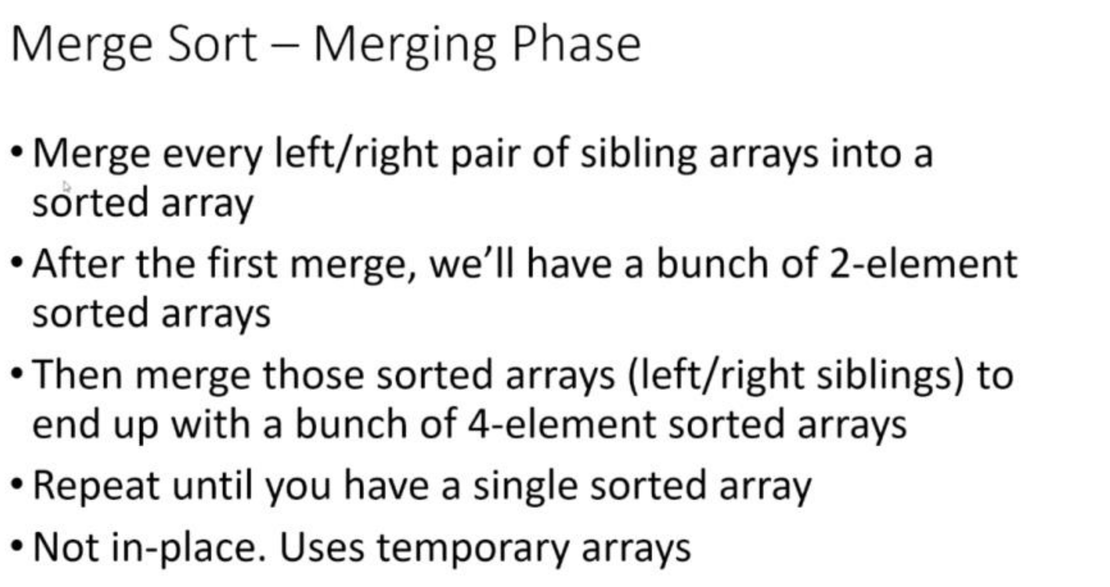


- Divide and conquer algorithm
- Recursive algorithm
- Two phases: Splitting and Merging
- Splitting phase leads to faster sorting during the Merging phase
- Splitting is logical. We don't create new arrays



Merge Sort – Splitting Phase

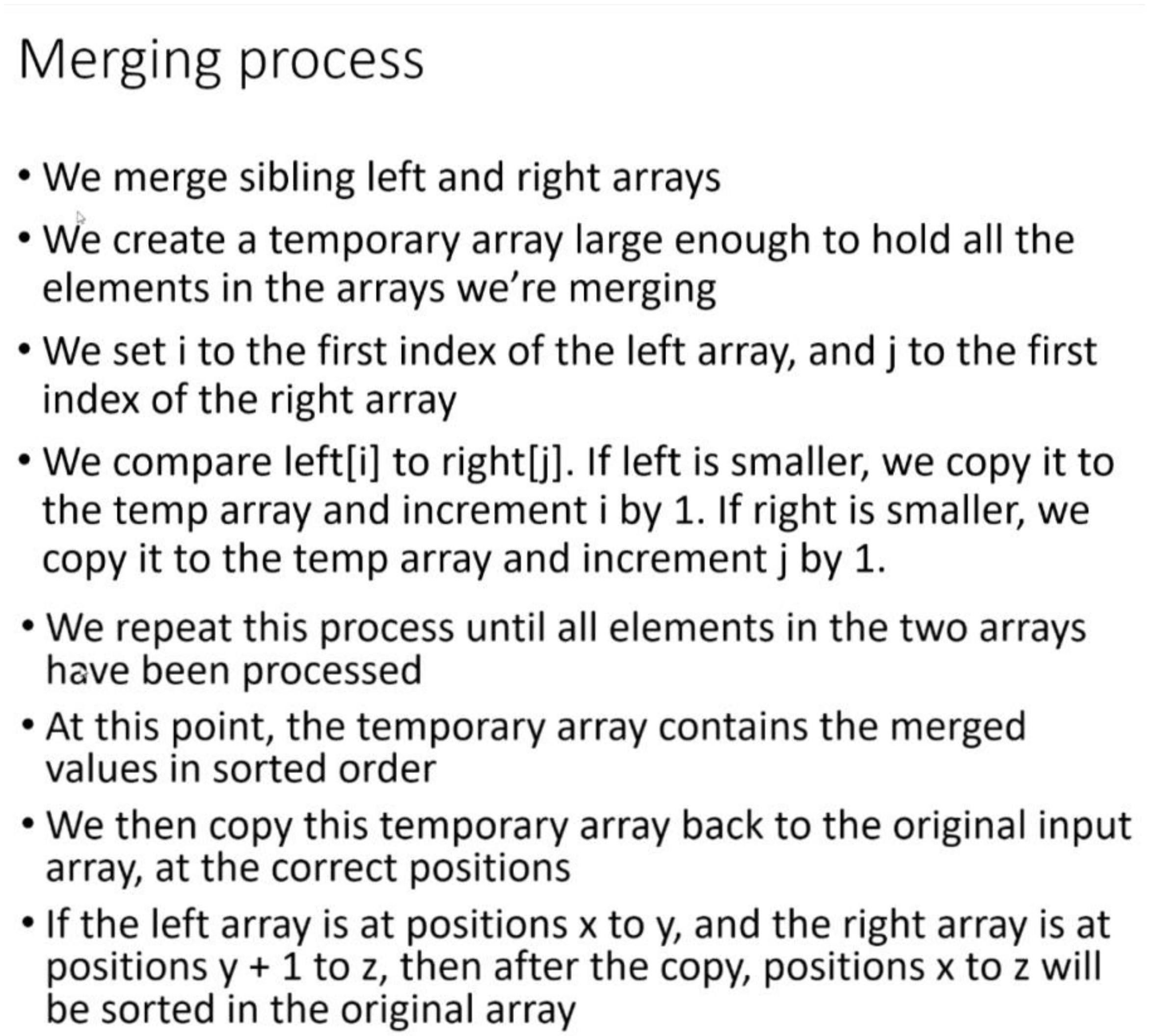
- Start with an unsorted array.
- Divide the array into two arrays, which are unsorted. The first array is the left array, and the second array is the right array.
- Split the left and right arrays into two arrays each
- Keep splitting until all the arrays have only one element each – these arrays are sorted



Merge Sort – Merging Phase

- Merge every left/right pair of sibling arrays into a sorted array
- After the first merge, we'll have a bunch of 2-element sorted arrays
- Then merge those sorted arrays (left/right siblings) to end up with a bunch of 4-element sorted arrays
- Repeat until you have a single sorted array
- Not in-place. Uses temporary arrays

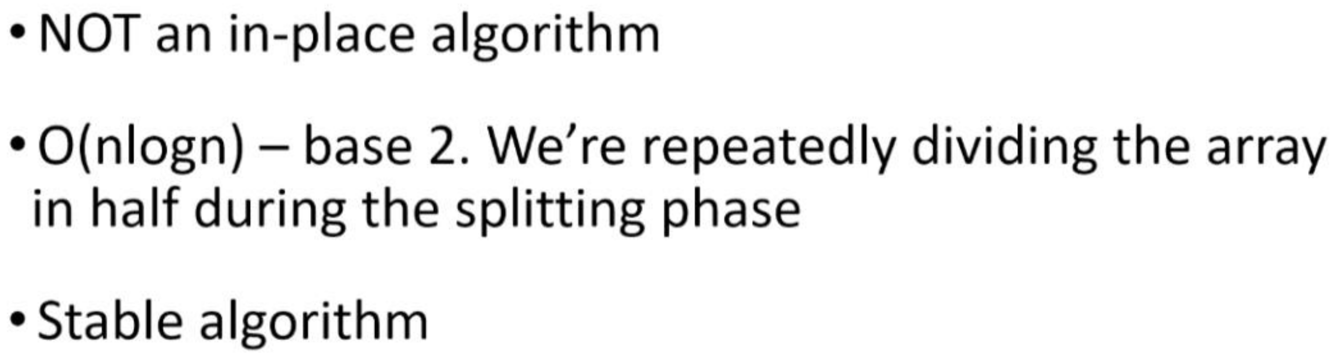
<u>Following will be done in the merging step -</u>



Merging process

- We merge sibling left and right arrays
- We create a temporary array large enough to hold all the elements in the arrays we're merging
- We set i to the first index of the left array, and j to the first index of the right array
- We compare left[i] to right[j]. If left is smaller, we copy it to the temp array and increment i by 1. If right is smaller, we copy it to the temp array and increment j by 1.
- We repeat this process until all elements in the two arrays have been processed
- At this point, the temporary array contains the merged values in sorted order
- We then copy this temporary array back to the original input array, at the correct positions
- If the left array is at positions x to y, and the right array is at positions y + 1 to z, then after the copy, positions x to z will be sorted in the original array

Conclusions about Merge Sort -



- NOT an in-place algorithm
- O(nlogn) – base 2. We're repeatedly dividing the array in half during the splitting phase
- Stable algorithm

Quick Sort

Monday, July 5, 2021 6:33 PM

Its in-place but not stable.

Time Complexity -

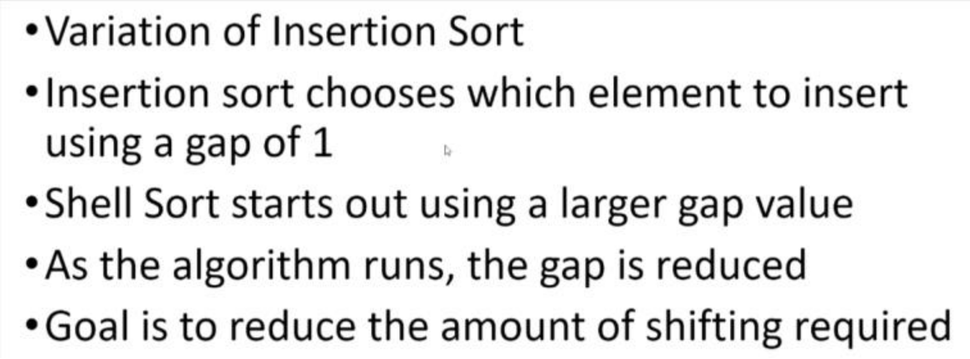
- Worst Case - O(n^2)
- Average Case and Best Case – O($n\log_2 n$)

Quick Sort is proffered over Merge Sort as its in-place.

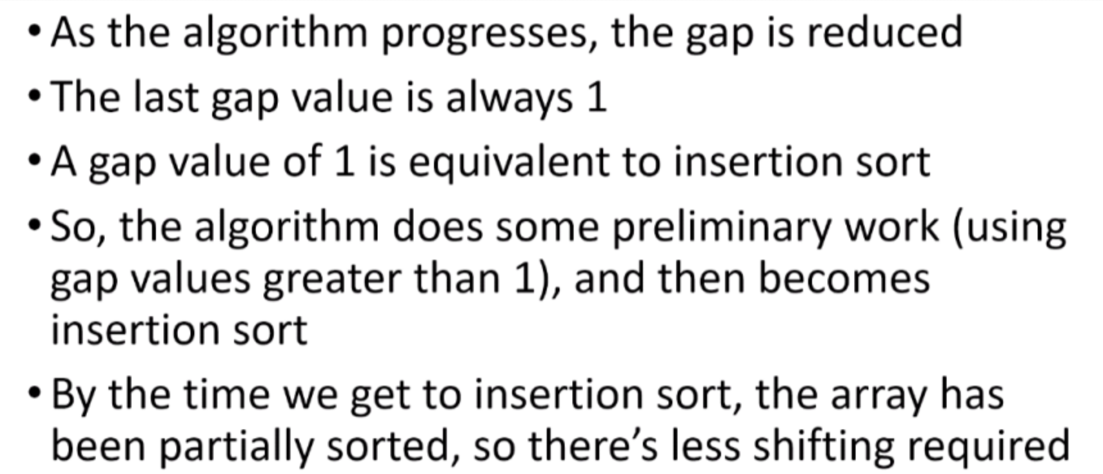
JDK sort method implements Dual-Pivot Quicksort.

Shell Sort

Friday, July 2, 2021 7:19 PM

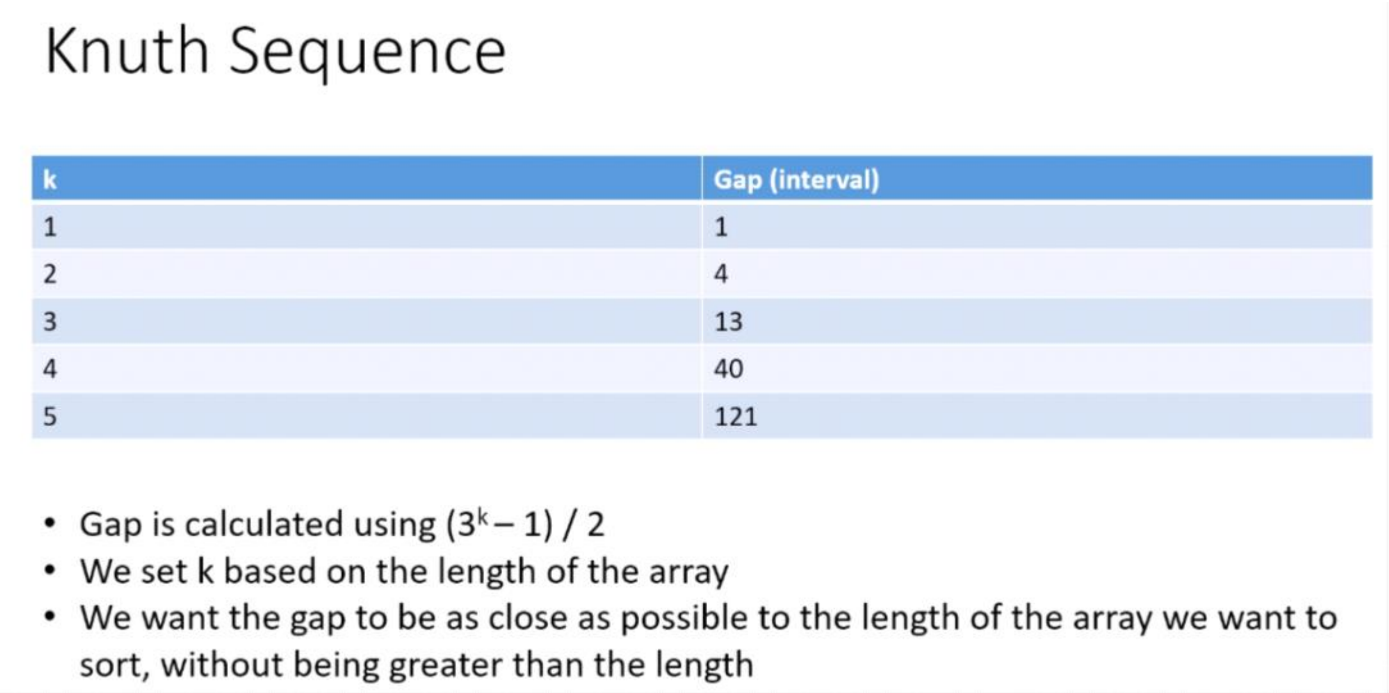


- Variation of Insertion Sort
- Insertion sort chooses which element to insert using a gap of 1
- Shell Sort starts out using a larger gap value
- As the algorithm runs, the gap is reduced
- Goal is to reduce the amount of shifting required



- As the algorithm progresses, the gap is reduced
- The last gap value is always 1
- A gap value of 1 is equivalent to insertion sort
- So, the algorithm does some preliminary work (using gap values greater than 1), and then becomes insertion sort
- By the time we get to insertion sort, the array has been partially sorted, so there's less shifting required

For different gap sequences we have different time complexities of the algorithm, we use the following sequence -

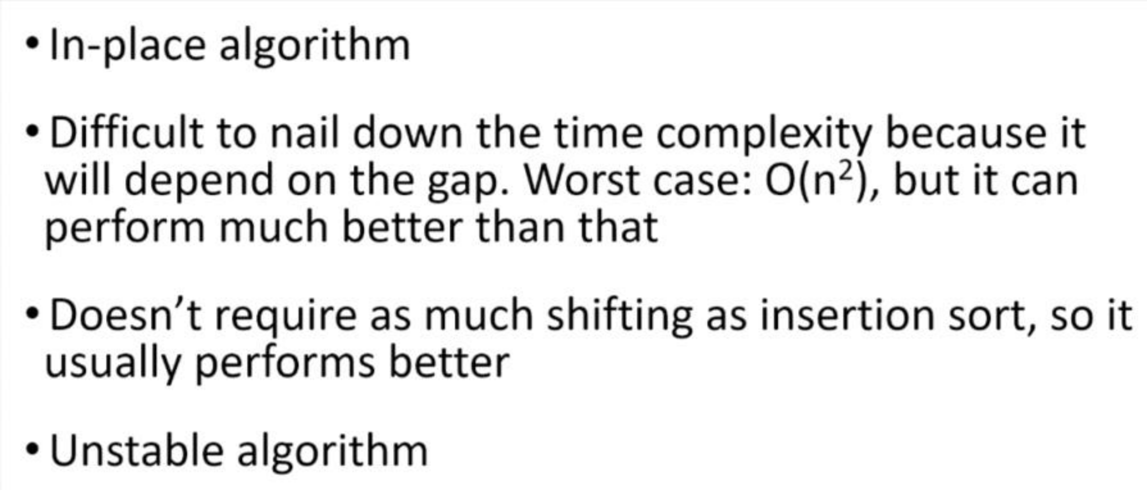


Knuth Sequence

k	Gap (interval)
1	1
2	4
3	13
4	40
5	121

- Gap is calculated using $(3^k - 1) / 2$
- We set k based on the length of the array
- We want the gap to be as close as possible to the length of the array we want to sort, without being greater than the length

Obviously, gap can't be greater than the size of the array.



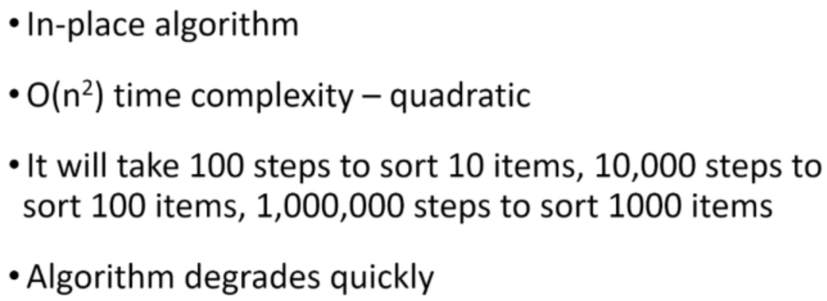
- In-place algorithm
- Difficult to nail down the time complexity because it will depend on the gap. Worst case: O(n^2), but it can perform much better than that
- Doesn’t require as much shifting as insertion sort, so it usually performs better
- Unstable algorithm

★ Shell Sort is unstable because during the preliminary sorting it's possible to change order of same-valued elements.

Time complexity of Shell Sort – we have 3 nested loops.
The "count how many loops there are to get the time complexity" tip works most of the time, but there are exceptions, and shell sort is one of them. The complexity will depend on the way the gap value is being calculated. Remember that one of the loops is controlling the gap value, not iterating over the elements in the array. The complexity of the implementation I showed you is actually n^1.5. Some implementations perform better than that; others worse.

Bubble Sort

Saturday, June 26, 2021 5:47 PM



- In-place algorithm
- O(n^2) time complexity – quadratic
- It will take 100 steps to sort 10 items, 10,000 steps to sort 100 items, 1,000,000 steps to sort 1000 items
- Algorithm degrades quickly

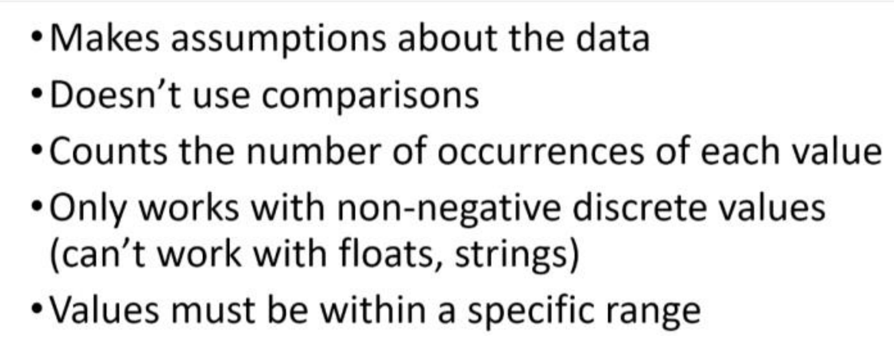
Worst and Average Case Time Complexity: O(n^2). Worst case occurs when array is reverse sorted.
Best Case Time Complexity: O(n). Best case occurs when array is already sorted.
Auxiliary Space: O(1)

Note – The above time complexities are for an optimized implementation of Bubble Sort, for general / original Bubble Sort all the cases have O(n^2) Time Complexity.

It's a stable sort algorithm.

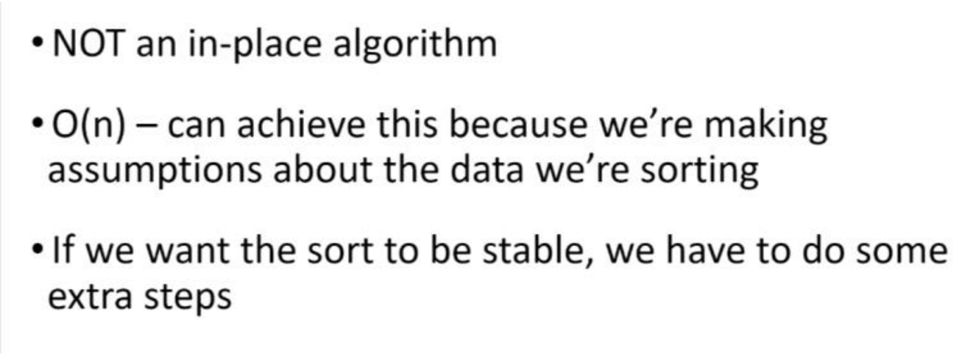
Counting Sort

Tuesday, July 6, 2021 9:38 AM



- Makes assumptions about the data
- Doesn't use comparisons
- Counts the number of occurrences of each value
- Only works with non-negative discrete values (can't work with floats, strings)
- Values must be within a specific range

- Whole Numbers (Integers) are discrete.
- Range must be reasonable not something like 0 to a million.
- Comparison ni karenge matlab we will not use >, < operators.



- NOT an in-place algorithm
- O(n) – can achieve this because we're making assumptions about the data we're sorting
- If we want the sort to be stable, we have to do some extra steps

It's NOT in-place bcz we require a counting array whose size is equal to the number of discrete elements in the range. Counting sort is efficient if the range of input data is not significantly greater than the number of objects to be sorted.

Counting sort uses a partial hashing to count the occurrence of the data object in O(1).

Counting sort CAN BE extended to - (Both achieved in implementation GOG)

1. Work for negative inputs.
2. Be Stable.

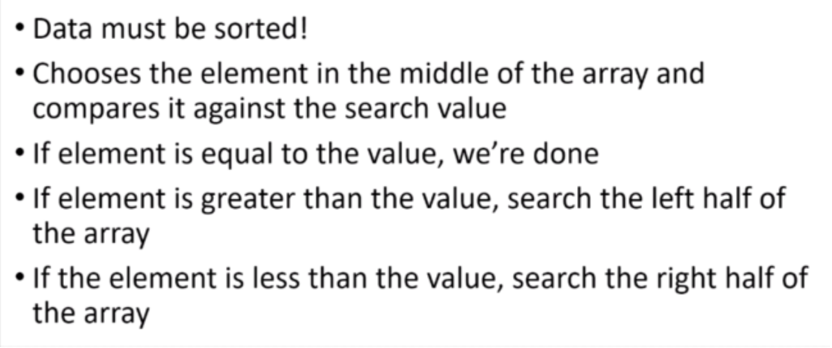
GOG - Time Complexity: O(n+k) where n is the number of elements in input array and k is the range of input.
Auxiliary Space: O(n+k)

Search Algorithms

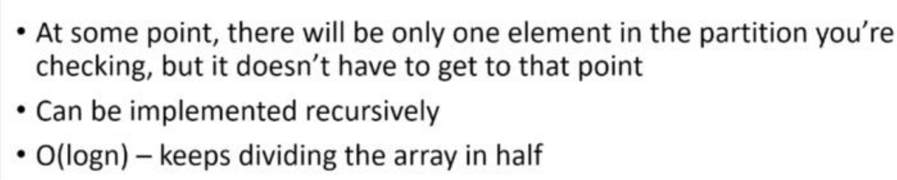
10 July 2021 10:19

Linear Search - O(n)

Binary Search -



- Data must be sorted!
- Chooses the element in the middle of the array and compares it against the search value
- If element is equal to the value, we're done
- If element is greater than the value, search the left half of the array
- If the element is less than the value, search the right half of the array



- At some point, there will be only one element in the partition you're checking, but it doesn't have to get to that point
- Can be implemented recursively
- O(logn) – keeps dividing the array in half

(log base at 2)

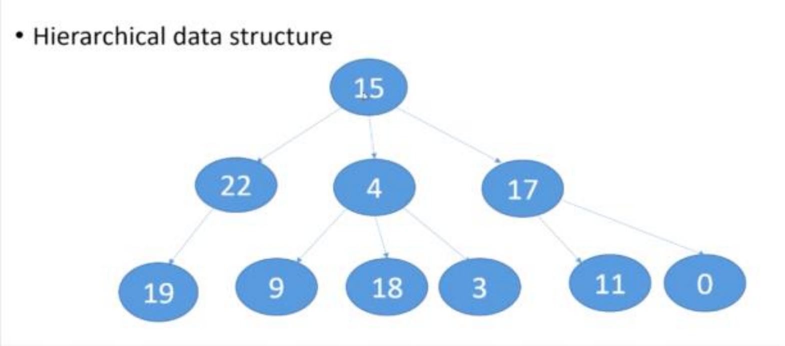
Binary search can be implemented recursively as well as iteratively. Iterative is preferred as overhead in case of recursive function calls isn't present. If memory is an issue then use recursive.

Trees

10 July 2021 13:05

Some people call trees DS others call it ADT, it's a grey area.

Normal Tree -



- Every circle in the above diagram of a tree is called a **Node.** Nodes can have children. Each non-Root Node has 1 and only 1 parent.
- **Leaf Nodes** (19,9,18,3,11,0) are such where our tree's corresponding branch terminates, so a leaf node has no child nodes.
- Every tree has a special Node called **Root,** Root node doesn't have a parent.
- Each above arrow is called an **Edge** of the tree. Points from parent to child.
- **Sub-Tree of a Node** is that node + all of its child nodes / descendants. Matlab wo node aur uske niche ka sara. So, except a singleton tree all the trees contain subtrees.
- A **Path** is a sequence of nodes required to go from one node to another node. Ek path me cycle allowed ni hoti, meaning you cant have a path that crosses a node more than once. **Root Path** matlab ek aisa path jo kisi node ko root node tak le jai.
- We say that node A is an ancestor of node B if node A comes in the root path of node B.
- **Depth of a Node** is the number of edges between that node and the Root node. Ex - Depth of node 18 is 2, Depth of node 15 is 0 as it is the root itself, Depth of node 22 is 1.
- **Height of a Node** is the number of edges along the longest path from that node to a leaf. Leaf Node has a Height = 0. **Height of a Tree** is the Height of the Root node.
- **Level of a Tree** is a collection of all Nodes that have the same Depth. 15 (Root) is at Level 0, {22,4,17} is at Level 1 and {19,9,18,3,11,0}is at Level 2. Simply Level 0 (root) ke immediate children are at level 1 and grandchildren at level 2.
- A singleton tree has only one node that is the Root node. (Normal tree me minimum 1 node (Root wala) ana chahiye)
- 0 nodes wala normal tree isn't allowed.

22 is a descendant of 15, 15 is parent of 22.

We use trees when we have a hierarchical ordering of data.

Ex - 1. Classes in java form a tree, as each child class can only extend 1 parent class, but a parent class can have multiple child classes. Root node of java classes is the Object class.

Ex - 2. File hierarchy (each drive), a parent directory can have multiple child directories (ek folder ke andar kai sare sub folders ho sakte he), and each child directory has only 1 parent directory (ek sub folder ko contain karne wala sirf ek hi parent folder hoga).

Binary Trees

10 July 2021 17:27