Module 2 - Material Notes

Search Algorithms

Search algorithms are methods used to find a specific item (or items) in a collection of items. These methods can be used on a variety of data structures, including arrays, lists, and graphs. The efficiency of a search algorithm is typically measured by its time complexity, often expressed in Big O notation.

Here are some key terms and concepts related to search algorithms:

Linear Search: This is the simplest search algorithm. It works by sequentially checking each element in the list until it finds a match or exhausts all possibilities. It's best used with small lists or unsorted data. The time complexity of a linear search is O(n).

Binary Search: This algorithm works on ***sorted lists***. It begins by comparing the target value to the middle element of the list. If they are not equal, the half in which the target cannot lie is eliminated, and the search continues on the remaining half until it's successful or the remaining half is empty. The time complexity of a binary search is O(log n).

Remember, the choice of a search algorithm often depends on the specifics of the problem, including the structure and size of the data set, as well as the specific requirements of the task.

Sorting Algorithms

For a visual explanation check out (**link (https://www.toptal.com/developers/sorting-algorithms)**)

Sorting algorithms are methods used to arrange a list of elements in a certain order, typically numerical or lexicographical. They play a fundamental role in many areas of computing. The efficiency of a sorting algorithm is usually measured by its time complexity, often expressed in Big O notation.

Here are some key terms and concepts related to sorting algorithms:

1. Insertion Sort Insertion Sort works by maintaining a sorted sublist in the given array and repeatedly inserting the next item in the right position within this sorted sublist. It's similar to how you might sort playing cards in your hand.

- Time complexity: Average and worst case time complexity is $O(n^2)$, where n is the number of items being sorted. Best case is O(n) when the input is already sorted.

2. Selection Sort Selection Sort works by repeatedly finding the minimum (or maximum, if sorting in descending order) from a list and putting it at the beginning of the list. The 'selection' process is done using a simple linear scan.

- Time complexity: The time complexity of Selection Sort is $O(n^2)$ for all cases (best, average, and worst) because it always scans all elements to find the minimum/maximum.

3. Bubble Sort Bubble Sort repeatedly steps through the list, compares adjacent elements and swaps them if they are in the wrong order. This process is repeated until no more swaps are needed.

- Time complexity: Bubble Sort has a worst-case and average time complexity of $O(n^2)$, where n is the number of items being sorted. The best case is O(n), which occurs when the input is already sorted.

4. Shell Sort Shell Sort is a generalized version of insertion sort. It first sorts elements far apart from each other and successively reduces the interval between the elements to be compared. It's a more efficient version of insertion sort.

- Time complexity: The time complexity of Shell Sort depends on the gap sequence chosen. The worst case time complexity of shell sort is $O(n^2)$, but with a good choice of gaps, it can perform in O(n log n).

5. Bucket Sort Bucket Sort works by distributing the elements of an array into a number of 'buckets'. Each bucket is then sorted individually, either using a different sorting algorithm, or recursively applying the bucket sort algorithm.

- Time complexity: The computational complexity for Bucket Sort is linear, i.e., O(n+k) in the average case, where n is the number of elements, and k is the number of buckets. The worst case complexity is $O(n^2)$ when all elements end up in the same bucket, essentially making it a slow sorting algorithm like bubble sort.

6. Merge Sort Merge Sort is an efficient, stable sorting algorithm based on the divide and conquer method. It divides unsorted list into N sublists, each containing one element, then repeatedly merges sublists to produce newly sorted sublists until there is only one sublist remaining.

- Time complexity: The time complexity for all cases (best, average, and worst) is O(n log n), where n is the number of items being sorted.

7. Quick Sort Quick Sort is a highly efficient sorting algorithm and is based on the partitioning of array of data into smaller arrays. A large array is partitioned into two arrays, one of which holds values smaller than the specified value, say pivot (derived from the original array), and another array holds values greater than the pivot value. Quick sort partitions an array and then calls itself recursively twice to sort the two resulting subarrays.

- Time complexity: The average case time complexity of Quick Sort is O(n log n). However, in the worst case, it can go up to $O(n^2)$, when the input is either sorted in increasing or decreasing order.

Each sorting algorithm has its own strengths and weaknesses, and the best choice of algorithm can depend on the details of the problem, such as the size of the list, the extent to which the list is already somewhat ordered, and the need for stability in the sort.

Some links:

https://www.geeksforgeeks.org/bubble-sort/ (https://www.geeksforgeeks.org/bubble-sort/)

https://www.geeksforgeeks.org/quick-sort/ (https://www.geeksforgeeks.org/quick-sort/)

https://www.geeksforgeeks.org/merge-sort/ (https://www.geeksforgeeks.org/merge-sort/)

https://www.geeksforgeeks.org/counting-sort/ (https://www.geeksforgeeks.org/counting-sort/)

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

https://www.geeksforgeeks.org/insertion-sort/ (https://www.geeksforgeeks.org/insertion-sort/)

https://www.programiz.com/dsa/shell-sort (https://www.programiz.com/dsa/shell-sort)

https://www.google.com/search?q=shell+sort+visual+explanation&rlz=1C1RXQR_enUS985US985&oq=shell+sort+visual+explanation&aqs=chrome.0.69i59.2599j1j7&sourceid=chrome&ie=UTF-8#fpstate=ive&vld=cid:696e76da,vid:lViqgakt-Eg (https://www.google.com/search?q=shell+sort+visual+explanation&rlz=1C1RXQR_enUS985US985&oq=shell+sort+visual+explanation&aqs=chrome.0.69i59.2599j1j7&sourceid=chrome&ie=UTF-8#fpstate=ive&vld=cid:696e76da,vid:lViqgakt-Eg)