Search & Sorting Algorithms Study Notes

Linear Search

Linear Search is a sequential search algorithm that checks each element in a list one by one until a match is found or the entire list has been searched. It is simple to implement but can be inefficient for large datasets since it has a time complexity of O(n). It takes longer as the n data size increases.

Time Complexity: `O(n)` **Linear**

Watch this 2-min video: https://youtu.be/19hcyQN8J7o?si=HyZ3Uc_m2gFKARNk

Binary Search

Binary Search is a more efficient searching algorithm that requires a sorted list as input. It works by repeatedly dividing the search space in half, comparing the middle element with the target value, and eliminating half of the remaining elements based on the comparison. This divide-and-conquer approach gives Binary Search a time complexity of O(log n), making it much faster than Linear Search for large datasets.

Time Complexity: `O(log n)` **Logarithmic**

Watch this 4-min video: https://www.youtube.com/watch?v=fDKIpRe8GW4

Bubble Sort

Bubble Sort is a simple sorting algorithm that repeatedly steps through the list, compares adjacent elements, and swaps them if they are in the wrong order. While easy to understand and implement, it has a time complexity of $O(n^2)$, making it inefficient for large datasets. The algorithm gets its

name from the way smaller elements "bubble" to the top of the list with each iteration.

Time Complexity: `O(n²)` **Quadratic**

Watch this 2-min video: https://youtu.be/xli_FI7CuzA?si=4_WVI3IlaIV5KVJE

Selection Sort

Selection Sort is a sorting algorithm that works by repeatedly finding the minimum element from the unsorted portion of the list and placing it at the beginning. The algorithm maintains two subarrays: one that is sorted and one that is unsorted. Like Bubble Sort, Selection Sort has a time complexity of $O(n^2)$, but it generally performs better than Bubble Sort as it makes fewer swaps.

Time Complexity: `O(n²)` **Quadratic**

Watch this 3-min video: https://www.youtube.com/watch?v=g-PGLbMth_g

Insertion Sort

Insertion Sort is a sorting algorithm that builds the final sorted array one element at a time. It works by taking elements from the unsorted part and inserting them into their correct position in the sorted part of the array. While it also has a time complexity of $O(n^2)$, Insertion Sort performs well on small datasets and is particularly efficient when dealing with arrays that are already partially sorted.

Time Complexity: `O(n²)` **Quadratic**

Watch this 2-min video: https://youtu.be/JU767SDMDvA?si=i_MQ_hXZdxvJ2R7t

Merge Sort

Merge Sort is a highly efficient divide-and-conquer sorting algorithm that works by dividing the array into smaller subarrays, sorting them, and then merging them back together. It consistently performs well with a time complexity of O(n log n), making it much faster than the simpler sorting algorithms for large datasets. Although it requires additional memory space, Merge Sort is stable and predictable, making it a popular choice for sorting linked lists and in situations where stable sorting is required.

Time Complexity: `O(n log n)` **Linearithmic**

Watch this 3-min video: https://youtu.be/4VqmGXwpLqc?si=4w--nqBk8P3yoITx

Quick Sort

Quick Sort is another efficient divide-and-conquer sorting algorithm that works by selecting a 'pivot' element and partitioning the array around it, with smaller elements placed before the pivot and larger elements after it. This process is recursively applied to the sub-arrays until the entire array is sorted. While Quick Sort has an average time complexity of $O(n \log n)$ and performs extremely well in practice due to its efficient cache usage, its worst-case time complexity is $O(n^2)$, though this can be mitigated with proper pivot selection strategies.

Time Complexity: `O(n²)` **Quadratic**

Watch this 4-min video: https://www.youtube.com/watch?v=Hoixgm4-P4M

Bucket Sort

Bucket Sort is a distribution-based sorting algorithm that works by distributing elements into a number of buckets, then sorting these buckets individually. It assumes uniform distribution of the input data across buckets and performs well when this assumption holds true. The algorithm has an average time complexity of $O(n + k)$, where k is the number of buckets.

Time Complexity: `O(n + k)` **Linear** (where k is the number of buckets)

Watch this short video: https://youtu.be/H5WT4×4crdI?si=v2B8A6w0E3C6oW4B

Heap Sort

Heap Sort is a comparison-based sorting algorithm that uses a binary heap data structure to sort elements. It works by first building a max-heap from the input array, then repeatedly extracting the maximum element and rebuilding the heap until all elements are sorted. While it has a consistent time complexity of $O(n \log n)$ and sorts in-place, it typically performs slower than Quick Sort in practice due to poor cache performance.

Time Complexity: `O(n log n)` **Linearithmic**

Watch this 4-min video: https://youtu.be/2DmK_H7IdTo