Sorting Notes

Merge Sort

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

- Divide-Sort-Merge

  The recurrence relation of merge sort is:

  $$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 2T\left(\frac{n}{2}\right) + \Theta(n) & \text{if } n > 1 \end{cases}$$

- Time Complexity:
  - **Best Case:** O(n log n), When the array is already sorted or nearly sorted.
  - **Average Case:** O(n log n), When the array is randomly ordered.
  - **Worst Case:** O(n log n), When the array is sorted in reverse order.
- Sorting large datasets
- sorting Linked lists.
- Stability
- Guaranteed O(N logN)
- Not in-place
- Slower than QuickSort in general

Insertion Sort

https://www.geeksforgeeks.org/insertion-sort-algorithm/

- Move from left to right-compare current with all previous-find its correct place

  Summary of Insertion Sort Comparisons

  $$T_{\text{best}}(n) = n - 1 \quad (\Theta(n))$$
  $$T_{\text{avg}}(n) = \frac{n^2-n}{4} \quad (\Theta(n^2))$$
  $$T_{\text{worst}}(n) = \frac{n^2-n}{2} \quad (\Theta(n^2))$$

-
- Time Complexity
  - **Best case**: O(n), If the list is already sorted, where n is the number of elements in the list.
  - **Average case**: O(n2), If the list is randomly ordered
  - **Worst case:** O(n2), If the list is in reverse order
- Stable
- Efficient for small lists and nearly sorted lists (**main point**)
- in-place
- Used as a subroutine in Bucket Sort
- incremental approach.

Quick Sort

https://www.geeksforgeeks.org/quick-sort-algorithm/

- Choose a Pivot-rearrange the Array left and right around pivot-repeat recursively
-
  - **Best case** (perfectly balanced partitions every time):

    Recurrence:

    $$T_{\text{best}}(n) = 2\,T\!\left(\tfrac{n}{2}\right) + cn$$

    By the Master theorem,

    $$T_{\text{best}}(n) = n\log_2 n + O(n) = \Theta(n\log n).$$

  - **Average case** (random pivot across all splits):

    $$T_{\text{avg}}(n) = \frac{2}{n}\sum_{i=0}^{n-1} T(i) + cn \approx 2\,T\!\left(\tfrac{n}{2}\right) + cn = \Theta(n\log n).$$

  - **Worst case** (always picks smallest or largest pivot):

    $$T_{\text{worst}}(n) = T(n-1) + T(0) + cn = T(n-1) + cn = \sum_{i=1}^{n} ci = \frac{cn(n+1)}{2} = \Theta(n^2).$$

- Time Complexity:
  - **Best Case**: (Ω(n log n)), Occurs when the pivot element divides the array into two equal halves.
  - **Average Case**: (θ(n log n)), On average, the pivot divides the array into two parts, but not necessarily equal.
  - **Worst Case:** (O(n^2)), Occurs when the smallest or largest element is always chosen as the pivot (e.g., sorted arrays).
- efficient on large data sets.
- low overhead
- Cache Friendly
- not stable
- in-place

Heap Sort

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

- comparison-based
- buid a heap-swap root and last node-call heapify on new root
- Heap Sort $\quad T = O(n) + O(n\log n) \Rightarrow \Theta(n\log n)$
- **Time Complexity:** O(n log n) in all cases
- in-place

- not stable
- 2-3 times slower than QuickSort
- Inefficient

Counting Sort

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

- non-comparison-based
- count frequencies-count cumulative frequencies-place elements in output array using counts
- Counting Sort $T = O(n + k)$
- Time Complexity: O(N+M), where N and M are the size of inputArray[] and countArray[] respectively.
    - **Worst-case:** O(N+k).
    - **Average-case:** O(N+k).
    - **Best-case:** O(N+k).
- Stable
- faster than all comparison-based
- stable
- doesn't work on decimal
- not In-place sorting
- used as a subroutine in Radix Sort
- used in Bucket Sort

Radix Sort

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

- linear sorting algorithm
- processing digit by digit
- sort the array by Least Significant digit in 1st iteration-then sort digit by digit until Most SD-use counting sort for individual digit sorting
- non-comparative
- Radix Sort $T = d\,(n + k)$
- time complexity of O(d * (n + b)), where d is the number of digits, n is the number of elements, and b is the base of the number system being used
- faster for large datasets, especially when the keys have many digits
- not as efficient for small datasets.
- Stable
- Not in-place

Bucket Sort

https://www.geeksforgeeks.org/bucket-sort-2/

- dividing elements into various groups, or buckets by uniformly distributing the elements
- Create buckets array-Take elements from input-multiply by bucket array size-apply floor-put this input element to this index of buckets array (use linked list for conflicted index)-concatenate buckets
- Sort elements in each bucket using stable algorithm (commonly used insertion sort because of small no of elements in a bucket)
- Bucket Sort $T = O(n) + O(n^2/b) + O(n)$
- Time Complexity
  - Worst Case: $O(n^2)$ hen one bucket gets all the elements
  - Best Case: O(n + k) when every bucket gets equal number of elements
- stable if the internal sorting algorithm used to sort it is also stable
- not in-place