Sorting

Types of sorting

- Internal Sorting: If the data to be sorted in small and can be kept in main memory and sorting is performed, then it is called as internal sorting.
- External Sorting: If the data is large which cant be placed in main memory at a time, then the data that is currently being sorted is brought in main memory and rest is on secondary memory. This type of sorting is external sorting.
- **We are discussing only Internal Sorting as part of our syllabus.**

Classification of Sorting Algorithm

- **By Number of Comparisons**: In this method, sorting algorithms are classified based on the number of comparisons.
- For comparison based sorting algorithms, best case behavior is O(nlogn) and worst case behavior is O(n^2).
- Comparison-based sorting algorithms evaluate the elements of the list by key comparison operation and need at least O(nlogn) comparisons for most inputs.

- **By Number of Swaps:** In this method, sorting algorithms are categorized by the number of swaps.

- **By Memory Usage:** Some sorting algorithms are such, that they need O(1) or O(logn) memory to create auxiliary locations for sorting the data temporarily.

- **By Recursion:** Sorting algorithms are either recursive [quick sort] or non-recursive [selection sort, and insertion sort], and there are some algorithms which use both (merge sort).

- **By Stability:** Sorting algorithm is stable if the relative order of original list is maintained in case of duplicate key values.

- **By Adaptability:** With a few sorting algorithms, the complexity changes based on pre-sortedness [quick sort]: presortedness of the input affects the running time. Algorithms that take this into account are known to be adaptive

Bubble-Sort

- Bubble-Sort is the slowest algorithm for sorting, but it is heavily used, as it is easy to implement.

- In Bubble-Sort, we compare each pair of adjacent values.

- We want to sort values in increasing order so if the second value is less than the first value then we swap these two values. Otherwise, we will go to the next pair.

- We will have N number of passes to get the array completely sorted. After the first pass, the largest value will be in the rightmost position.

Bubble Sort: Performance

Worst case complexity : O(n^2)
Best case complexity (Improved version) : O(n)
Average case complexity (Basic version) : O(n^2)
Worst case space complexity : O(1) auxiliary

Selection-Sort

- Selection sort is an in-place sorting algorithm.
- Selection sort works well for small files. It is used for sorting the files with very large values and small keys.
- Here selection is made based on keys and swaps are made only when required.
- Advantages • Easy to implement • In-place sort (requires no additional storage space)
- Disadvantages • Time complexity is $O(n^2)$

Algorithm

1. Find the minimum value in the list

2. Swap it with the value in the current position

3. Repeat this process for all the elements until the entire array is sorted This algorithm is called selection sort since it repeatedly selects the smallest element.

Performance of selection sort

- Worst case complexity : O(n^2)
- Best case complexity : O(n^2)
- Average case complexity : O(n^2)
- Worst case space complexity: O(1) auxiliar

Insertion Sort

- It is a simple and efficient comparison sort.
- Each iteration removes an element from the input data and inserts it into the correct position in the list being sorted.
- The choice of the element being removed from the input is random and this process is repeated until all input elements have gone through.

Advantages

• Simple implementation

• Efficient for small data

• Adaptive: If the input list is presorted [may not be completely] then insertions sort takes O(n + d), where d is the number of inversions

• Practically more efficient than selection and bubble sorts, even though all of them have O(n 2) worst case complexity

• Stable: Maintains relative order of input data if the keys are same • In-place: It requires only a constant amount O(1) of additional memory space

Performance of insertion sort

- Worst case complexity : $O(n^2)$
- Best case complexity : $O(n^2)$
- Average case complexity : $O(n^2)$
- Worst case space complexity: $O(n^2)$ $O(1)$ auxiliar

Merge Sort

- Merge sort is based divide and conquer strategy.
- Can be implemented as external sorting, when the dataset is so big that it is impossible to load the whole dataset into memory, here sorting is done in chunks.
- Merging is the process of combining two sorted files to make one bigger sorted file.
- Selection is the process of dividing a file into two parts: k smallest elements and n – k largest elements.
- Selection and merging are opposite operations
    - selection splits a list into two lists
    - merging joins two files to make one file

Performance

- Worst case complexity : O(nlogn)
- Best case complexity : O(nlogn)
- Average case complexity : O(nlogn)
- Worst case space complexity: O(n) auxiliary

Quick Sort

- It is an example of a divide-and-conquer algorithmic technique.
- It is also called partition exchange sort.
- It uses recursive calls for sorting the elements, and it is one of the famous algorithms among comparison-based sorting algorithms.
- Divide: The array A[low ...high] is partitioned into two non-empty sub arrays A[low ...q] and A[q + 1... high], such that each element of A[low ... high] is less than or equal to each element of A[q + 1... high]. The index q is computed as part of this partitioning procedure.
- Conquer: The two sub arrays A[low ...q] and A[q + 1 ...high] are sorted by recursive calls to Quick sort.

Algorithm

- The recursive algorithm consists of four steps:

1) If there are one or no elements in the array to be sorted, return.

2) Pick an element in the array to serve as the “pivot” point. (Usually the left-most element in the array is used.)

3) Split the array into two parts – one with elements larger than the pivot and the other with elements smaller than the pivot.

4) Recursively repeat the algorithm for both halves of the original array.

Performance

- Worst case Complexity: O(n^2)
- Best case Complexity: O(nlogn)
- Average case Complexity: O(nlogn)
- Worst case space Complexity: O(1)