Sorting

Tuesday, 24 January 2023 1:36 PM

- Sorting algorithms
  - Insertion Sort - some segment will be sorted and remaining will be unsorted sta
    - *Insertion Sort* is one such online algorithm
    - https://www.geeksforgeeks.org/insertion-sort/
  - Selection Sort
    - As the name indicates, it picks the smallest/largest element from unsorte respective index
    - https://www.geeksforgeeks.org/selection-sort/
  - Heap Sort
    - Note: Heap always make sure that maximum element is at the top, but it of the elements in increasing/decreasing order..
    - So the crux..should be like heapify the initial array first..
    - Start removing top element and replace it with last element..
      - Apply heapify and repeart
    - **Advanced version of selection sort**
    - It internally uses heapify logic to sort the elements
    - Since heap structure can be maintained in the array. It does not required
    - In-place sorting algorithm
    - https://www.geeksforgeeks.org/heap-sort/

- https://www.geeksforgeeks.org/difference-between-insertion-sort-and-selection-sort/
  https://www.tutorialandexample.com/quick-sort-vs-merge-sort
  https://www.geeksforgeeks.org/time-complexities-of-all-sorting-algorithms/

Algorithm	Time Complexity			Space Complexit
	Best	Average	Worst	Worst
Selection Sort	Ω(n^2)	θ(n^2)	O(n^2)	O(1)
Bubble Sort	Ω(n)	θ(n^2)	O(n^2)	O(1)
Insertion Sort	Ω(n)	θ(n^2)	O(n^2)	O(1)
Heap Sort	Ω(n log(n))	θ(n log(n))	O(n log(n))	O(1)
Quick Sort	Ω(n log(n))	θ(n log(n))	O(n^2)	O(n)

te

d array and fit it in its

doesn't guarantee the order

auxillary arrays

y

Merge Sort	Ω(n log(n))	θ(n log(n))	O(n log(n))	O(n)
Bucket Sort	Ω(n +k)	θ(n +k)	O(n^2)	O(n)
Radix Sort	Ω(nk)	θ(nk)	O(nk)	O(n + k)
Count Sort	Ω(n +k)	θ(n +k)	O(n +k)	O(k)
Shell Sort	Ω(n log(n))	θ(n log(n))	O(n^2)	O(1)
Tim Sort	Ω(n)	θ(n log(n))	O(n log (n))	O(n)
Tree Sort	Ω(n log(n))	θ(n log(n))	O(n^2)	O(n)
Cube Sort	Ω(n)	θ(n log(n))	O(n log(n))	O(n)

Sorting:

Basis for comparison	Quick Sort	Merge Sor
The partition of elements in the array	The splitting of a array of elements is in any ratio, not necessarily divided into half.	In the merg parted into
Worst case complexity	O(n^2)	O(nlogn)
Works well on	It works well on smaller array	It operates
Speed of execution	It work faster than other sorting algorithms for small data set like Selection sort etc	It has a con size of data
Additional storage space requirement	Less(In-place)	More(not In
Efficiency	Inefficient for larger arrays	More efficie
Sorting method	Internal	External
Stability	Not Stable	Stable
Preferred for	for Arrays	for Linked L
Locality of reference	good	poor

Quick Sort	**Merge Sort**

rt

ge sort, the array is
just 2 halves (i.e. n/2).

fine on any size of array

sistent speed on any

n-place)

ent

Lists

The idea is to select a random pivot element and place all the elements smaller than the pivot on its left side and all the elements larger than the pivot on its right side in the array. After that, repeat the same process for the left and right subarrays separately.	The idea is to divide a given subarrays from the middle a process for the subarrays to size becomes equal to 1 and elements by comparing with created arrays in the correct
Its best and average time complexities are O(nlogn), but in the worst case, it becomes equal to O(n^2). This is when it selects the smallest or largest as the pivot element every time.	Its best, average, and worst- are O(nlogn).
In the worst case, it behaves like the subtract and conquer technique. We can conclude that it depends on the content.	It does not depend on the c depends on the structure. It the 'divide and conquer' tec
Its space complexity is O(1) as it requires no extra space.	Its space complexity, in the O(n).
It is an internal sorting technique.	It is an external sorting tech
It is not a stable sorting technique. The order of identical elements does not remain preserved.	It is a stable sorting techniq identical elements remains
It is less efficient compared to merge sort.	More efficient than the quic