*Examples to explain sorting algorithms

Let's look at 3 algorithms for explanation: SelectionSort, QuickSort, RadixSort

SelectionSort: This algorithm works similar but in opposite direction to the BubbleSort algorithm. The time complexity for this algorithm is O(N**2) and it is consisted of 2 main parts - Comparison part and Swap part. First algorithm starts with comparison from zero index element of an array and iterates over array to look for smaller number than first element of an array. If smaller number is found here swap part will come in to play. It will replace first element of an array with the found number. So, minimum number of array will be in the extreme left. Here first iteration ends and loop runs for second time this time starting from second element of an array. Same operations will be executed for second time at last finishing with the replacement of second smallest number with index one element. This process will continue as long as the length of array.

Next Algorithm is QuickSort. Let's say we are given an array that we should sort. To do that we need a pivot number which will be most times middle point of array and it will split the whole list into left and right arrays.And we sort the array in a manner that elements greater than pivot are to the right and elements little than pivot are to the left. Then process continues until the list is sorted thanks to the recursive nature of quicksort algorithm.

RadixSort: We have an array of numbers with the max number of 3 digits and the task is to sort them. The significance of numbers changes from least significant digit to the most significant digit of number. First, we will sort the input array according to the one's digit (least significant digit). Then, we will sort numbers according to ten's digit. Finally, we sort according to hundred's digit.

*Benefits of stacks

Stacks are able to do operations in just O (1) time complexity. Stacks are used to solve problems like checking balancing of symbols. We can also use them in order to

evaluate arithmetic expressions. Stacks use LIFO structure means that, last element added will be first to be out. Stacks are used for a number of things. One very common usage is to keep track of the return address during function call. Stacks are also used to pass data between functions. Also undo and redo operations in word processors are available thanks to stacks.

*Difference between a stack and queue

The biggest different between a stack and queue lays in their structure. Queue uses FIFO structure which stands for First in First out. Stacks because of their structure accept new elements from top and remove element from top as well. However, queues use FIFO structure and new items are added after the last element and elements are deleted from the top.

*Different forms of queues

List-based queue

Stack-based queue

Node-based queue

Double-ended queue

*Why should I use Stack or Queue data structures instead of arrays and lists and when should I use them?

Arrays are good for random access. However, sometimes we would need to use a data structure which obeys to the structure of either FIFO or LIFO. Stacks and queues are ideal to use in this case. Stacks and queues are also efficient from memory management side. Arrays can allocate large amount of memory and waste it, on the other hand, stacks and queues use memory more efficient

*What is the significance of Stack being a recursive data structure?

One of the main applications of stacks are implementing function calls (also recursion).

Generally recursive data structures call itself. And recursive problems can be broken down into recursive and base cases. Stack is recursive data structure because of its structure LIFO. LIFO is like recursion the last called function comes out first from the program