Question 1:

Please explain divide-and-conquer approach.

Answer:

The divide-and-conquer approach is a fundamental problem-solving strategy in computer science. It is used to solve complex problems by breaking them down into smaller ones.

The first step is to divide the original problem into smaller, more manageable subproblems. This division should be such that each subproblem is a smaller instance of the same problem as the original, or at least closely related.

Next, you solve each of the subproblems independently. This can be done recursively, meaning that each subproblem is further divided into even smaller subproblems until a base case is reached where the problem is simple enough to be solved directly.

Question 2:

Please name 3 algorithms that typically follow/use divide-and-conquer approach.

Answer:

1. Quick sort
2. Merge sort
3. Heap sort

Question 3:

Illustrate the operation of merge sort on the array.

A = (3, 41, 52, 26, 38, 57, 9, 49)

Answer:

(3, 41, 52, 26, 38, 57, 9, 49)

(3, 41, 52, 26) (38, 57, 9, 49)

(3, 41) (52, 26) (38, 57) (9, 49)

(3) (41) (52) (26) (38) (57) (9) (49)

(3, 41) (26, 52) (38, 57) (9, 49)

(3, 26, 41, 52) (9, 38, 49, 57)

(3, 9, 26, 38, 41, 49, 52, 57) Sorted array.

Question 4:

On which case (scenario) you need to choose quicksort instead of merge sort?

Provide an explanation!

Answer:

Quicksort is often preferred when you need a sorting algorithm with lower space complexity, good average-case performance, and potential adaptability to partially sorted data. Merge Sort is a more stable and predictable choice with guaranteed worst-case performance, making it a better option when worst-case time complexity and data stability are crucial.

Let’s say Space complexity,

Quicksort is an in-place sorting algorithm, meaning it sorts the elements within the original array without requiring additional memory for merging or copying elements to auxiliary arrays. It uses a small amount of stack space for recursive calls, but its primary advantage is that it doesn't require much additional memory overall.

Merge Sort, on the other hand, requires additional memory for merging sorted subarrays. This additional memory can be a significant concern when dealing with large datasets. If you're working with limited memory resources, Quicksort may be a better choice due to its lower space complexity.

Question 5:

On which case (scenario) you need to use heapsort?

Provide an explanation!

Answer:

We can consider using Heapsort when you need a sorting algorithm with guaranteed worst-case time complexity, limited memory usage, and the option to make it stable. It's particularly well-suited for scenarios where worst-case performance and memory efficiency are critical.

Let’s say,

Heapsort guarantees a worst-case time complexity of O(n log n). This means that regardless of the initial order of the input data, Heapsort will always perform efficiently and consistently in terms of time complexity. Unlike Quicksort, which can have a worst-case time complexity of O(n^2) in certain scenarios, Heapsort avoids such worst-case scenarios. If you have critical applications where predictable and guaranteed performance is essential, Heapsort is a good choice.