Module 4 - Divide and Conquer Algorithm

1. Understanding Divide and Conquer Approach

Concepts:

Definition: A problem-solving approach that divides a problem into smaller sub-problems, solves each sub-problem independently, and then combines the solutions to solve the original problem.

Process:

Divide/Break: Break the problem into smaller sub-problems.

Conquer/Solve: Solve the sub-problems independently.

Merge/Combine: Combine the solutions of the sub-problems to form the solution of the original problem.

Examples:

Merge Sort
Quick Sort
Binary Search
Strassen's Matrix Multiplication

2. Merge Sort

Concepts:

Description: A sorting technique based on the divide and conquer approach.

Steps:

Divide the array into two halves.
Recursively sort each half.
Merge the sorted halves.

```
void merge(int arr[], int l, int m, int r) {
    int n1 = m - l + 1;
    int n2 = r - m;

    int L[n1], R[n2];

    for (int i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for (int j = 0; j < n2; j++)
        R[j] = arr[m + 1 + j];

    int i = 0, j = 0, k = l;
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        } else {
            arr[k] = R[j];
            j++;
        }
        k++;
    }
    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }
    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }
}

void mergeSort(int arr[], int l, int r) {
    if (l < r) {
        int m = l + (r - l) / 2;
        mergeSort(arr, l, m);
        mergeSort(arr, m + 1, r);
        merge(arr, l, m, r);
    }
}
```

Analogy: Think of sorting a deck of cards. You split the deck into smaller piles, sort each pile, and then merge them back together in order.

3. Quick Sort

Concepts:

Description: A highly efficient sorting algorithm based on partitioning an array into smaller arrays.

Steps:

Choose a pivot element.

Partition the array into two parts: elements less than the pivot and elements greater than the pivot.

Recursively apply the same logic to the sub-arrays.

```
int partition(int arr[], int low, int high) {
    int pivot = arr[high];
    int i = low - 1;
    for (int j = low; j <= high - 1; j++) {
        if (arr[j] < pivot) {
            i++;
            std::swap(arr[i], arr[j]);
        }
    }
    std::swap(arr[i + 1], arr[high]);
    return i + 1;
}

void quickSort(int arr[], int low, int high) {
    if (low < high) {
        int pi = partition(arr, low, high);
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}
```

Analogy: Imagine organizing books on a shelf by repeatedly selecting a book as a pivot, placing all smaller books to the left and larger ones to the right, and repeating the process for each section.

4. Binary Search

Concepts:
Description: A fast search algorithm with a run-time complexity of O(log n).

Steps:
Compare the target value to the middle element of the array.
If the target value matches, return the index.
If the target value is less, repeat the search on the left sub-array.
If the target value is greater, repeat the search on the right sub-array.

Example:

```
int binarySearch(int arr[], int l, int r, int x) {
   while (l <= r) {
      int m = l + (r - l) / 2;
      if (arr[m] == x)
         return m;
      if (arr[m] < x)
         l = m + 1;
      else
         r = m - 1;
   }
   return -1;
```

Strassen's Matrix Multiplication

Concepts:
Description: An efficient algorithm for matrix multiplication using divide and conquer.

Steps:
Divide matrices into sub-matrices.
Compute seven products recursively.
Combine the results to form the final product.