Day - 7 20/11/24

1. **Quick Sort:**

```
class Solution {
public:
    void quickSort(vector<int>& nums, int low, int high) {
        if (high <= low) return;
        int pivotIndex = partition(nums, low, high);
        quickSort(nums, low, pivotIndex - 1);
        quickSort(nums, pivotIndex + 1, high);
    }
private:
    int partition(vector<int>& nums, int low, int high) {
        int pivotVal = nums[high];
        int i = low - 1;
        for (int j = low; j < high; j++) {
            if (nums[j] < pivotVal) {
                i++;
                swap(nums[i], nums[j]);
            }
        }
        i++;
        swap(nums[i], nums[high]);
        return i;
    }
};
```

Time Complexity: O(n log n)

2. **Merge Sort:**

```
class Solution {
  public:
    void merge(vector<int>& arr, int l, int m, int r) {
        int n1 = m - l + 1;
        int n2 = r - m;
        vector<int> L(n1), R(n2);
        for (int i = 0; i < n1; i++)
            L[i] = arr[l + i];
        for (int i = 0; i < n2; i++)
            R[i] = arr[m + 1 + i];
        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(vector<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);
    }
}
};
```

Time Complexity: O(n log n)

3. **Decode Ways:**

```
class Solution {
public:
    int numDecodings(string s) {
        int n = s.length();
        if (s[0] == '0') return 0;
        vector<int> dp(n + 1, 0);
        dp[0] = 1;
        dp[1] = 1;
        for (int i = 2; i <= n; i++) {
            if (s[i - 1] != '0') {
                dp[i] += dp[i - 1];
            }
            if (s[i - 2] == '1' || (s[i - 2] == '2' && s[i - 1] <= '6')) {
                dp[i] += dp[i - 2];
            }
        }
        return dp[n];
    }
};
```

Time Compllexity: O(n)

4. **Jump Game 2:**

```
class Solution {
public:
    int jump(vector<int>& nums) {
        int njumps = 0;
        int maxindex = 0;
        int currentreach = 0;
        for(int i=0; i<nums.size()-1; i++)
        {
            maxindex = max(maxindex, i+nums[i]);
            if(i==currentreach)
            {
                njumps++;
                currentreach = maxindex;
            }
        }
        return njumps;

    }
};
```

Time Complexity: O(n)

5. **Interpolation Search:**

```
int interpolationSearch(int arr[], int size, int value) {
    int low = 0, high = size - 1;
    while (low <= high && value >= arr[low] && value <= arr[high]) {
        int probe = low + (high - low) * (value - arr[low]) / (arr[high] - arr[low]);
        if (arr[probe] == value)
            return probe;
        if (arr[probe] < value)
            low = probe + 1;
        else
            high = probe - 1;
    }
    return -1;
}
```

Time Complexity: O(log log n)

6. **Ternary Search:**

```
int interpolationSearch(int arr[], int size, int value) {
    int low = 0, high = size - 1;
```

```
    while (low <= high && value >= arr[low] && value <= arr[high]) {
        int probe = low + (high - low) * (value - arr[low]) / (arr[high] - arr[low]);

        if (arr[probe] == value)
            return probe;

        if (arr[probe] < value)
            low = probe + 1;
        else
            high = probe - 1;
    }
    return -1;
}
```

Time Complexity: O(log3n)

7. <u>3sum Closest</u>

```
class Solution {
public:
    int threeSumClosest(vector<int>& nums, int target) {
        int closestSum = INT_MAX / 2;
        for (int i = 0; i < nums.size() - 2; i++) {
            for (int j = i + 1; j < nums.size() - 1; j++) {
                for (int k = j + 1; k < nums.size(); k++) {
                    int currentSum = nums[i] + nums[j] + nums[k];
                    if (abs(target - currentSum) < abs(target - closestSum)) {
                        closestSum = currentSum;
                    }
                }
            }
        }
        return closestSum;
    }
};
```

Time Complexity:O(n^3)