Recursion Problems Solutions

Here is the **complete write-up** of the recursion problems and their solutions, rewritten exactly as you requested (without emojis), and including a **fully implemented version** of one of them at the end:

Recursion Problems and Solutions

---

1. Fibonacci Series

Approach: Recursive

- Use the formula: `F(n) = F(n-1) + F(n-2)`
- Base cases: `F(0) = 0`, `F(1) = 1`

Pseudocode:

```cpp
int fibonacci(int n) {
    if (n <= 1) return n;
    return fibonacci(n - 1) + fibonacci(n - 2);
}
```

Dry Run:
Input: n = 5
`fibonacci(5) = fibonacci(4) + fibonacci(3)`
`fibonacci(4) = fibonacci(3) + fibonacci(2)`
Output: 0, 1, 1, 2, 3, 5

- **Time Complexity:** O(2^N)
- **Space Complexity:** O(N) (recursion stack)

---

2. Climbing Stairs

Approach: Recursive

- Use the relation: `ways(n) = ways(n-1) + ways(n-2)`
- Base cases: `ways(0) = 1`, `ways(1) = 1`

Pseudocode:

```cpp
int climbStairs(int n) {
    if (n <= 1) return 1;
    return climbStairs(n - 1) + climbStairs(n - 2);
}
```

Dry Run:
Input: n = 4
`climbStairs(4) = climbStairs(3) + climbStairs(2)`
`climbStairs(3) = climbStairs(2) + climbStairs(1)`
Output: 5

- **Time Complexity:** O(2^N)
- **Space Complexity:** O(N)

---

3. Binary Search (Recursive Implementation)

Approach: Recursive

- Divide the array into halves and recursively search left or right

Pseudocode:

```cpp
cpp

int binarySearch(vector<int>& arr, int left, int right, int target) {
    if (left > right) return -1;
    int mid = left + (right - left) / 2;
    if (arr[mid] == target) return mid;
    if (arr[mid] > target) return binarySearch(arr, left, mid - 1, target);
    return binarySearch(arr, mid + 1, right, target);
}
```

Dry Run:
Input: arr = [1, 2, 3, 4, 5], target = 4
mid = 2 → arr[mid] = 3 → search right
mid = 3 → arr[mid] = 4 → found
Output: 3

- **Time Complexity:** O(log N)
- **Space Complexity:** O(log N)

---

4. Reverse a String

Approach: Recursive

- Swap the first and last characters and recursively reverse the substring

Pseudocode:

```cpp
cpp

void reverseString(string& s, int left, int right) {
    if (left >= right) return;
    swap(s[left], s[right]);
    reverseString(s, left + 1, right - 1);
}
```

Dry Run:
Input: s = "hello"
swap(h, o), swap(e, l)
Output: "olleh"

- **Time Complexity:** O(N)
- **Space Complexity:** O(N)

---

5. Palindrome in a String

Approach: Recursive

- Compare the first and last characters and check recursively

Pseudocode:

```
cpp

bool isPalindrome(string& s, int left, int right) {
    if (left >= right) return true;
    if (s[left] != s[right]) return false;
    return isPalindrome(s, left + 1, right - 1);
}
```

Dry Run:
Input: s = "racecar"
Compare r==r, a==a, c==c
Output: true

- **Time Complexity:** O(N)
- **Space Complexity:** O(N)

---

6. Merge Sort

Approach: Recursive

- Split the array into halves, recursively sort, and merge

Pseudocode:

```
cpp

void merge(vector<int>& arr, int left, int mid, int right) {
    vector<int> temp;
    int i = left, j = mid + 1;
    while (i <= mid && j <= right) {
        if (arr[i] < arr[j]) temp.push_back(arr[i++]);
        else temp.push_back(arr[j++]);
    }
    while (i <= mid) temp.push_back(arr[i++]);
    while (j <= right) temp.push_back(arr[j++]);
    for (int k = left; k <= right; ++k) {
        arr[k] = temp[k - left];
    }
}
```

```
void mergeSort(vector<int>& arr, int left, int right) {
    if (left >= right) return;
    int mid = left + (right - left) / 2;
    mergeSort(arr, left, mid);
    mergeSort(arr, mid + 1, right);
    merge(arr, left, mid, right);
}
```

Dry Run:
Input: arr = [4, 2, 1, 3]
Split: [4, 2], [1, 3] → Merge: [2, 4], [1, 3] → [1, 2, 3, 4]
Output: [1, 2, 3, 4]

- **Time Complexity:** O(N log N)
- **Space Complexity:** O(N)

---

7. Quick Sort

Approach: Recursive

- Partition the array and recursively sort left and right halves

Pseudocode:

```
cpp

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

void quickSort(vector<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);
    }
}
```

Dry Run:
Input: arr = [4, 2, 1, 3]
Choose pivot = 3
Rearrange → Recursive calls
Output: [1, 2, 3, 4]

- **Time Complexity:** O(N log N) (average)
- **Space Complexity:** O(log N)

Fully Implemented Example (C++): Reverse a String

```
cpp

#include <iostream>
#include <string>
using namespace std;

void reverseString(string& s, int left, int right) {
    if (left >= right) return;
    swap(s[left], s[right]);
    reverseString(s, left + 1, right - 1);
}

int main() {
    string str = "hello";
    reverseString(str, 0, str.length() - 1);
    cout << "Reversed String: " << str << endl;
    return 0;
}
```

Output:

```
arduino

Reversed String: olleh
```

Let me know if you'd like to implement another example or generate test cases for these.