Recursion in C++ | Pravin Nandankar

# Definition:

Recursion is a programming technique where a function calls itself in order to solve a problem. The recursive function generally has two parts:  
1. Base Case: The condition under which the function stops calling itself.  
2. Recursive Case: The part where the function calls itself with a different argument.

# Example of a Recursive Function:

#include <iostream>  
using namespace std;  
  
int factorial(int n) {  
 if (n <= 1) return 1; // Base case  
 else return n \* factorial(n - 1); // Recursive case  
}  
  
int main() {  
 int num = 5;  
 cout << "Factorial of " << num << " is " << factorial(num) << endl;  
 return 0;  
}

# Advantages of Recursion:

- Simplifies code for problems that can be broken into similar sub-problems.  
- Easier to write and understand for problems like tree traversals, sorting algorithms, etc.

# Disadvantages of Recursion:

- Can lead to excessive memory use and stack overflow if not implemented correctly.  
- Generally slower than iterative solutions due to function call overhead.

# Questions

## Easy

1. Factorial   
int factorial(int n) {  
 if (n <= 1) return 1; // Base case  
 else return n \* factorial(n - 1); // Recursive case  
}

2. Fibonacci Series  
int fibonacci(int n) {  
 if (n <= 1) return n; // Base case  
 else return fibonacci(n - 1) + fibonacci(n - 2); // Recursive case  
}

3.**Array Based**

int sumofn(int n) { if(n == 0) { return 0; } return n + sumofn(n-1);}int sumOfArray(vector<int> &v, int n) { if (n == 0) return 0; return v[n-1] + sumOfArray(v, n - 1);}bool search(vector<int> v,int key,int n) { if(n == 0) return false; if(v[n-1] == key) return true; return search(v,key,n-1);}bool searchCh(string v,char ch,int n) { if(n == -1) return false; if(v[n] == ch) return true; return searchCh(v,ch,n-1);}

## Medium

3. Sum of Digits  
- Problem: Write a recursive function to find the sum of digits of a number.  
- Code:

#include <iostream>  
using namespace std;  
  
int sumOfDigits(int n) {  
 if (n == 0) return 0; // Base case  
 else return n % 10 + sumOfDigits(n / 10); // Recursive case  
}  
  
int main() {  
 int num = 1234;  
 cout << "Sum of digits of " << num << " is " << sumOfDigits(num) << endl;  
 return 0;  
}

4. Reverse a String  
- Problem: Write a recursive function to reverse a given string.   
void reverseString(string& str, int start, int end) {  
 if (start >= end) return; // Base case  
 swap(str[start], str[end]); // Swap characters  
 reverseString(str, start + 1, end - 1); // Recursive case  
}  
  
int main() {  
 string str = "hello";  
 reverseString(str, 0, str.length() - 1);  
 cout << "Reversed string is " << str << endl;  
 return 0;  
}

## Hard

5. Towers of Hanoi  
- Problem: Write a recursive function to solve the Towers of Hanoi problem.  
- Code:

#include <iostream>  
using namespace std;  
  
void towersOfHanoi(int n, char from\_rod, char to\_rod, char aux\_rod) {  
 if (n == 1) {  
 cout << "Move disk 1 from rod " << from\_rod << " to rod " << to\_rod << endl;  
 return; // Base case  
 }  
 towersOfHanoi(n - 1, from\_rod, aux\_rod, to\_rod); // Move n-1 disks from from\_rod to aux\_rod  
 cout << "Move disk " << n << " from rod " << from\_rod << " to rod " << to\_rod << endl;  
 towersOfHanoi(n - 1, aux\_rod, to\_rod, from\_rod); // Move n-1 disks from aux\_rod to to\_rod  
}  
  
int main() {  
 int n = 3; // Number of disks  
 towersOfHanoi(n, 'A', 'C', 'B'); // A, B and C are names of rods  
 return 0;  
}

6. Permutations of a String  
- Problem: Write a recursive function to generate all permutations of a given string.  
- Code:

#include <iostream>  
using namespace std;  
  
void permute(string str, int l, int r) {  
 if (l == r) {  
 cout << str << endl; // Base case  
 } else {  
 for (int i = l; i <= r; i++) {0

.  
 swap(str[l], str[i]); // Swap characters  
 permute(str, l + 1, r); // Recursive case  
 swap(str[l], str[i]); // Backtrack  
 }  
 }  
}  
  
int main() {  
 string str = "ABC";  
 permute(str, 0, str.length() - 1);  
 return 0;  
}