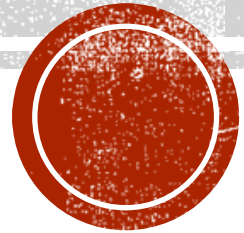
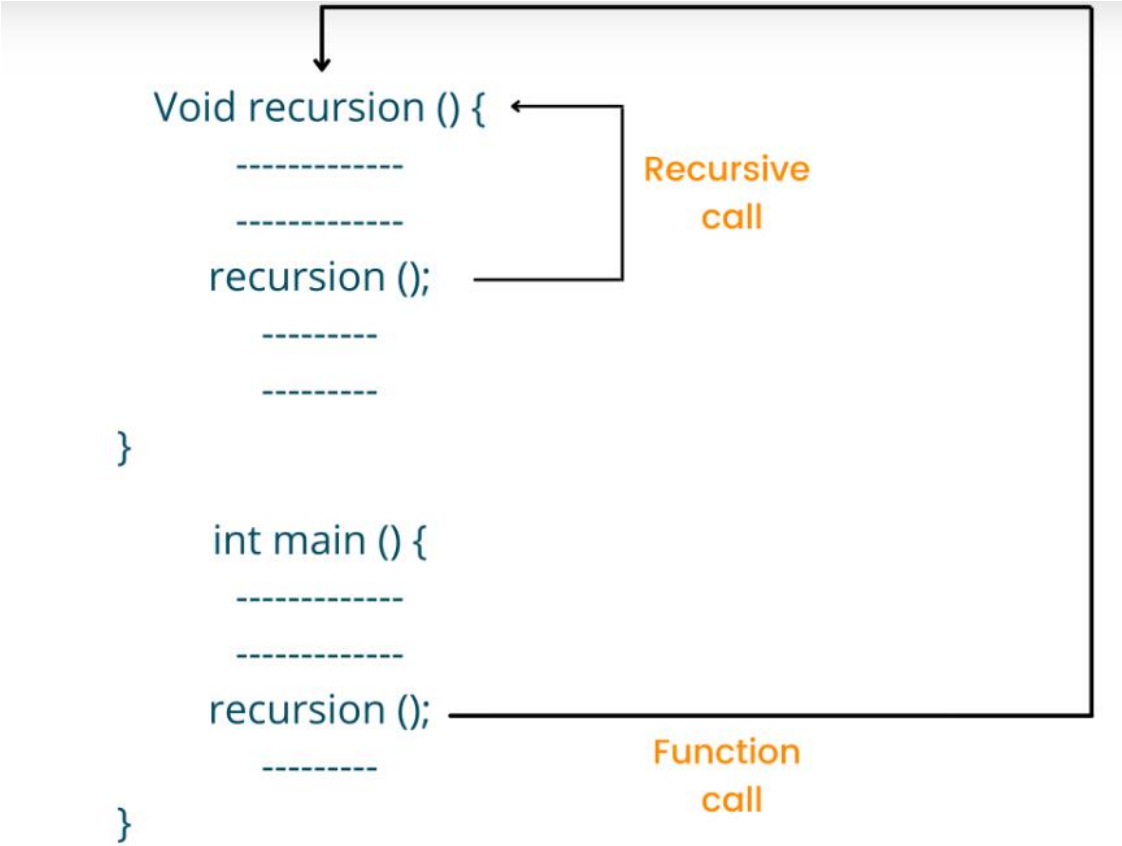
DATA STRUCTURES AND ALGORITHMS



RECURSION

- The process in which a function calls itself directly or indirectly is called recursion
- The corresponding function is called a recursive function.
- A recursive function solves a particular problem by calling a copy of itself
- Recursion has the ability of solving smaller subproblems of the original problems
- Many more recursive calls can be generated as and when required.
- It is essential to know that we should provide a certain case in order to terminate this recursion process
- So we can say that every time the function calls itself with a simpler version of the original problem

HOW RECURSION WORKS?



PROPERTIES OF RECURSION

- Recursion is an amazing technique with the help of which we can reduce the length of our code and make it easier to read and write.
- Performing the same operations multiple times with different inputs
- In every step, we try smaller inputs to make the problem smaller
- Base condition is needed to stop the recursion otherwise infinite loop will occur

ADVANTAGES OF RECURSION

- Less number code lines are used in the recursion program and hence the code looks shorter and cleaner.
- Recursion is easy to approach to solve the problems involving data structure and algorithms like graph and tree
- Recursion helps to reduce the time complexity
- It helps to reduce unnecessary calling of the function
- It helps to solve the stack evolutions and prefix, infix, postfix evaluation
- Recursion is the best method to define objects that have repeated structural forms

DISADVANTAGES OF RECURSION

- It consumes a lot of stack space
- It takes more time to process the program
- If an error is accrued in the program, it is difficult to debug the error in comparison to the iterative program.

RECURSION VS ITERATION

SR No.	Recursion	Iteration
1)	Terminates when the base case becomes true.	Terminates when the condition becomes false.
2)	Used with functions.	Used with loops.
3)	Every recursive call needs extra space in the stack memory.	Every iteration does not require any extra space.
4)	Smaller code size.	Larger code size.

HOW RECURSIVE FUNCTION ARE STORED IN MEMORY?

- Recursion uses more memory, because the recursive function adds to the stack with each recursive call
- It keeps the values there until the call is finished
- The recursive function uses LIFO (LAST IN FIRST OUT) Structure just like the stack data structure

BASICS OF RECURSION

- There are two cases in the recursion/ recursive programs:
  - Base case
  - Recursive case
- In the recursive program, the solution to the base case is provided
- Base case terminates the recursive function
- Solution to the bigger problem is expressed in terms of smaller problems

FACTORIAL OF A NUMBER

Iterative Approach

```
for(i=1; i<=number; i++)
    fact=fact*i;
```

Recursive Approach

```
int fact(int n){
if(n > 1)
        return n*fact(n-1);
else
        return 1;
}
```

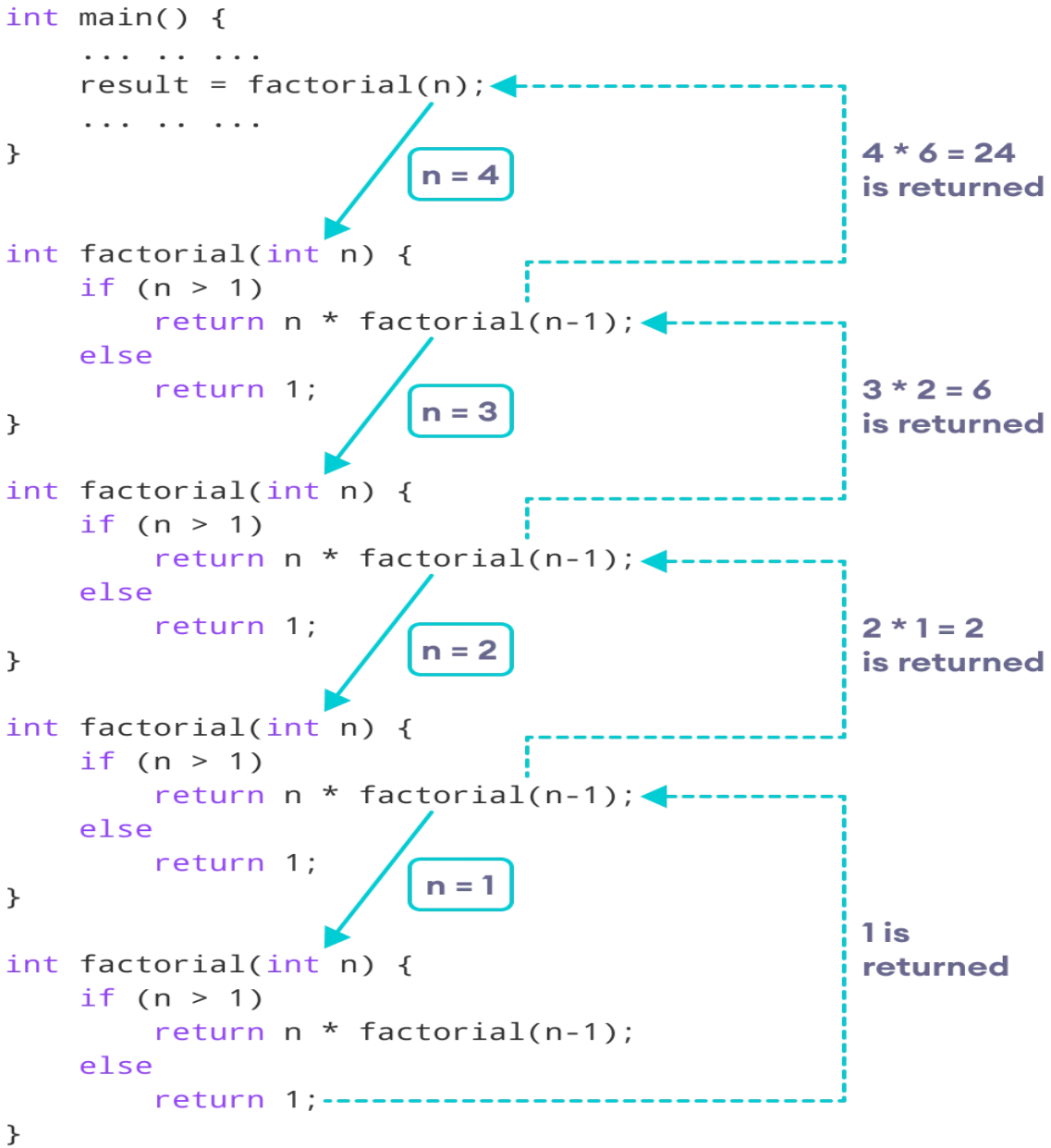
SUM OF NUMBERS FROM 1-5

Iterative Approach

```
int main()

{
  int result = 0;

 for(i=1; i<= 5; i++)

       result = result + 1;

           cout<<result;

}
```

Recursive Approach

```
int sum(int k) {
  if (k > 0) {
    return k + sum(k - 1);
  } else {
    return 0;
  }
}

int main() {
  int result = sum(5);
  cout << result;
  return 0;
}
```


int main() {
... .
result = factorial(n);
... .
}
n = 4
int factorial(int n) {
if (n > 1)
return n * factorial(n-1);
else
return 1;
}
n = 3
int factorial(int n) {
if (n > 1)
return n * factorial(n-1);
else
return 1;
}
n = 2
int factorial(int n) {
if (n > 1)
return n * factorial(n-1);
else
return 1;
}
n = 1
int factorial(int n) {
if (n > 1)
return n * factorial(n-1);
else
return 1;
}
4 * 6 = 24
is returned
3 * 2 = 6
is returned
2 * 1 = 2
is returned
1 is
returned