Design and Analysis of Algorithms

Lecture - 5

Success is always inevitable with Hard Work and Perseverance

N. Ravitha Rajalakshmi

Learning Objective

- Derive recurrence relation for assessing time complexity of recursive functions
- Methods to solve recurrence relation

Recurrence relation

- A function which can be described in terms of its value on smaller inputs.

$$x(\boldsymbol{n}) = \begin{cases} x(\boldsymbol{n-1}) + 1, & n > 1 \\ 1, & n = 1 \end{cases}$$

Recursion

- A function that calls itself

- Why does it call itself ?

    I have to do infinite computations (same computations)

    But , I do not want to write it infinite times

    I wanted to represent with finite statement

    - Niklaus Wirth, Computer Scientist

Find sum (of n numbers)

= nth number + Find sum (of n-1 numbers)

Find Factorial (of n)

= n * Find Factorial (of n-1)

- How long it calls itself?

At some place you should stop

When the answer for a problem instance is known, stop there

Find sum (of 0 numbers) = 0

Find Factorial (of 1) = 1

- When to compose the solution?

When the control returns , do the necessary computations

Find sum (of n numbers)

= nth number + Find sum (of n-1 numbers)

Find Factorial (of n)

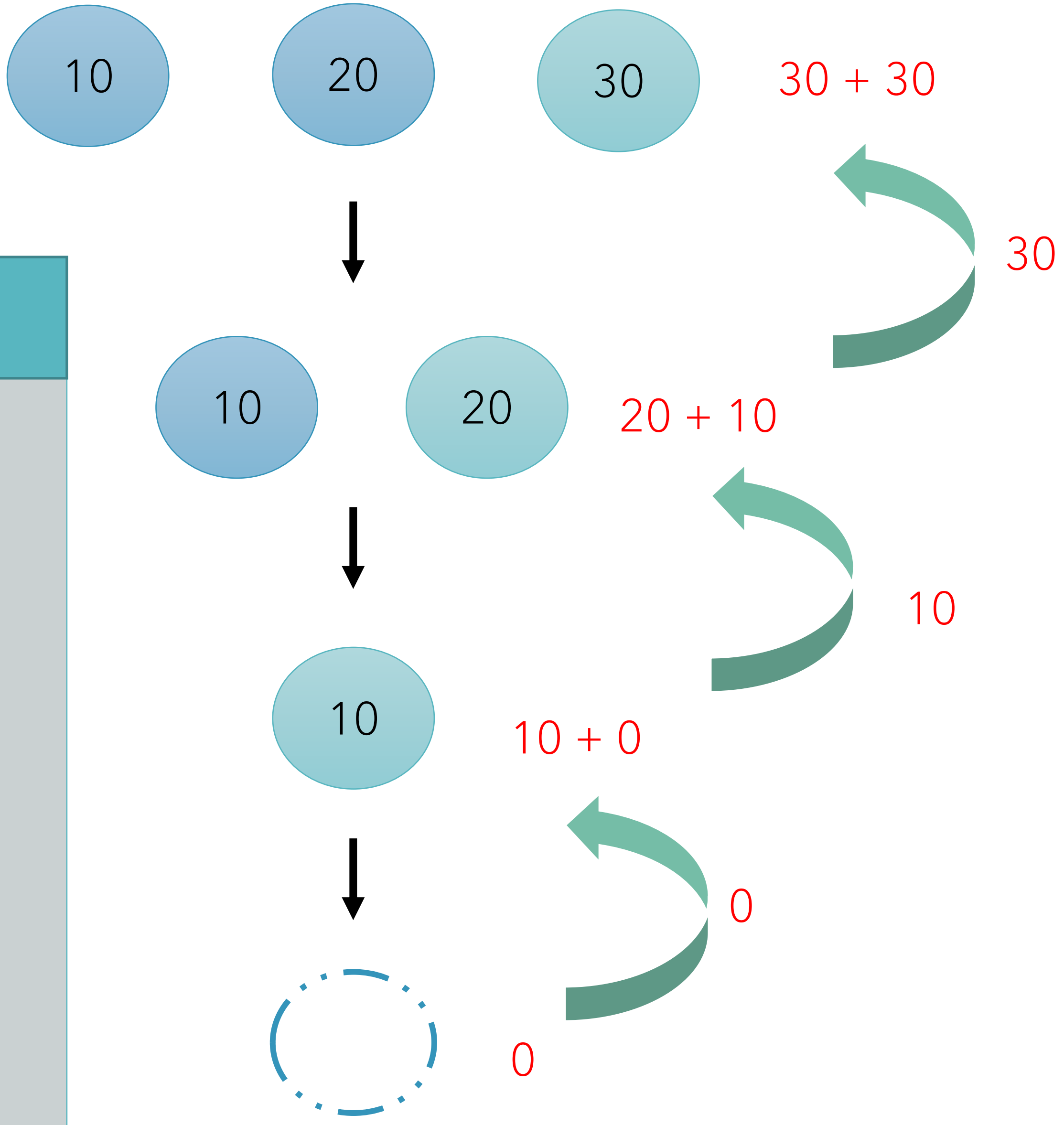
= n * Find Factorial (of n-1)

Recursion

Function FindSum (a, n)

 # a is array # n – number of elements

```
if (n==0){
        return 0;
}
else {
        res = FindSum(a, n-1);
        res = a[n] + res;
        return res;
}
```



Code Structure

Function FindSum (a, n)

 # a is array # n – number of elements

```
if (n==0){
        return 0;
}
else {
        res = FindSum(a, n-1);
        res = a[n] + res;
        return res;
}
```

Base Case

Recursive Case

Recursive call where problem instance is reduced

Recursive Case

```
# a is array  # n – number of
elements
        if (n==0){
                return 0;
        }
        else {

                res = FindSum(a, n-1);
                res = a[n] + res;
                return res;
        }
```

Works with largest case first

Computation occur before recursion

Computation occur after recursion

Works with smallest case first

Pause & Think

- What will be the output of the function when invoked with value of n as 5?

Function print (n)

```
# n – number
      if (n==0){
                  return ;
      }
      else {
                  print(n-1);
                  cout<<n;
      }
```

1   2   3   4   5

5   4   3   2   1

Dilemma

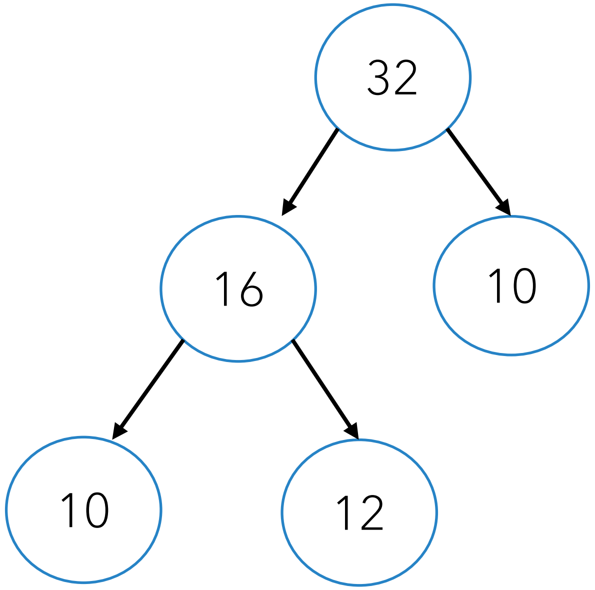
- Two programming constructs for performing repetitive tasks

Iteration (Loops) vs Recursion

Complex Functions can be expressed easily

Problem

32 16 10 12 10



Function preorder(n)

```
# n - node
    if (n!=NULL){
            return ;
    }
    else {
            cout<n->data;
            preorder(n->left);
            preorder(n->right);
    }
```

Dilemma

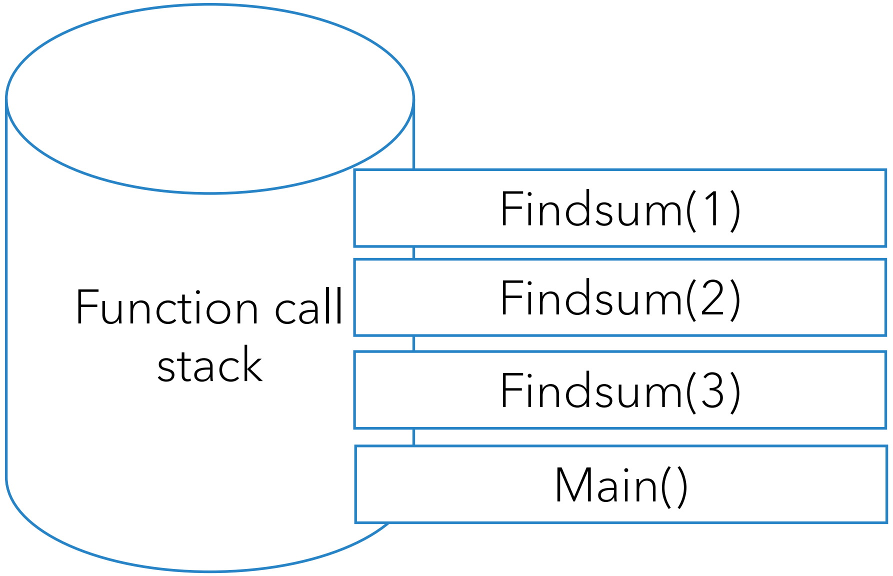
- Two programming constructs for performing repetitive tasks

Iteration (Loops) vs Recursion

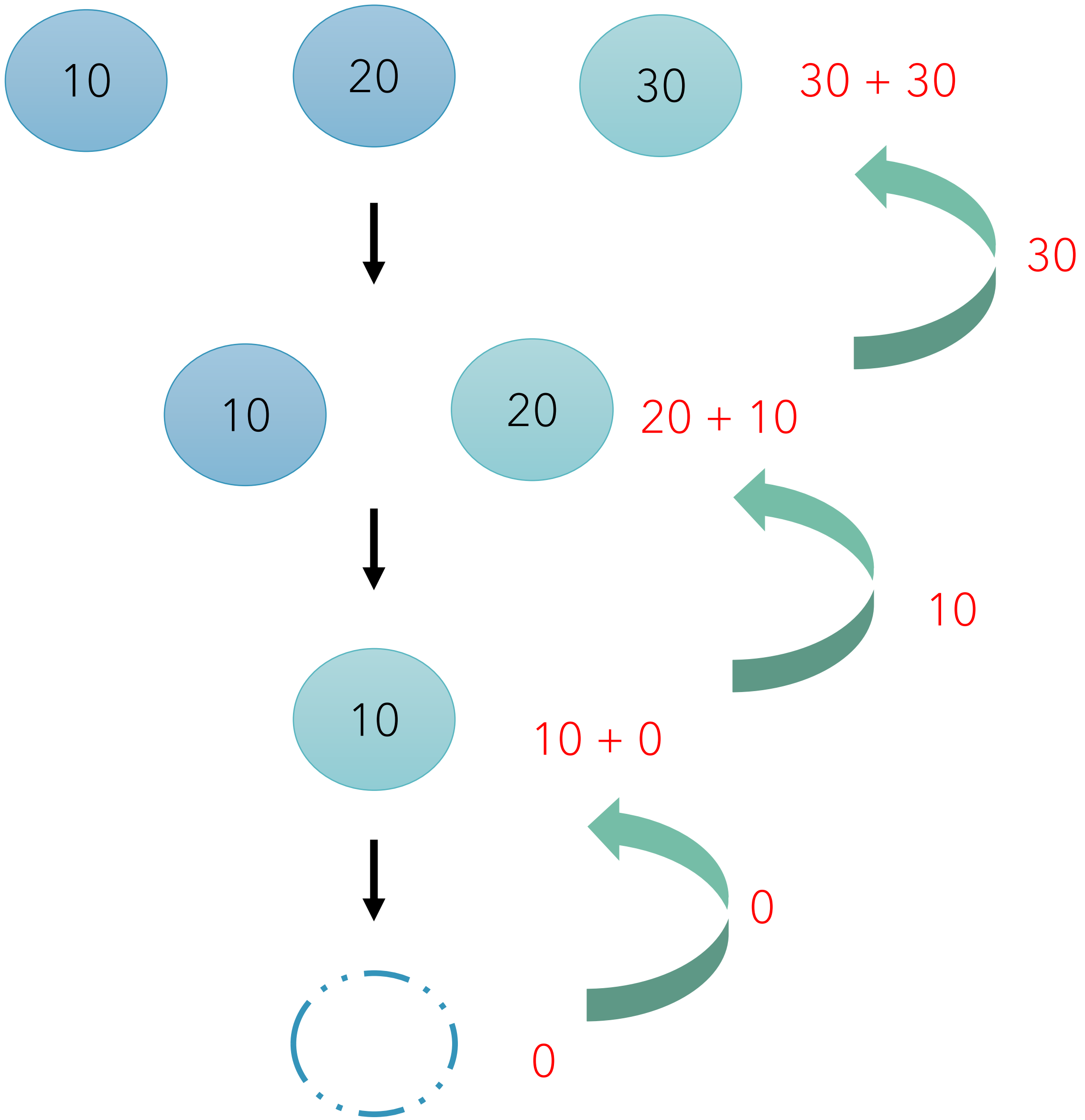
Complex Functions can be expressed easily

takes a lot of memory space

Recursion



Function call stack
Findsum(1)
Findsum(2)
Findsum(3)
Main()



10
20
30
30 + 30
30
10
20
20 + 10
10
10
10 + 0
0
0

Pause & Think

- What does an activation record contain??

    Parameters , Local variables , return address

Growth of Function call stack

- Some Functions take exponential function calls
  - Classic Example is Fibonacci Series
- Which can be avoided with iteration
- Only very few problems !!

Analysis of Recursive Functions

- $T(n) \approx C(n)$
- Input size n
- Basic Operation
- Count can be directly identified as it involves recursion
- T(n) is expressed using recurrence relation

Recurrence relation

- A function which can be described in terms of its value on smaller inputs.

$$x(\boldsymbol{n}) = \begin{cases} x(\boldsymbol{n-1}) + 1, & n > 1 \\ 1, & n = 1 \end{cases}$$

Function FindSum (a, n)

```
# a is array  # n – number of
elements
    if (n==0){
            return 0;
    }
    else {
            res = FindSum(a, n-1);
            res = a[n] + res;
            return res;
    }
```

Input Size
Length of array

Basic Operation
Addition

$$T(n) = \begin{cases} T(n-1) + 1, & n > 0 \\ 0, & n = 0 \end{cases}$$

Solving Recurrences

- Iteration Method
  - Method of forward / backward substitution

- Recursion tree Method

- Master's theorem

Iteration Method

- Solve by backward substitution

$T(n) = T(n-1) + 1$

$= (T(n-2) + 1) + 1$

$= (T(n-2) + 2)$

$= (T(n-3) + 1) + 2$

$= (T(n-3) + 3)$

$= (T(n-4) + 1) + 3$

$= (T(n-4) + 4)$

Generic Equation = $(T(n-i) + i)$

- Use base case to resolve the characteristic equation

$T(0) = 0$

Generic Equation = $(T(n-i) + i)$

$n - i = 0$

$\mathrm{i} = n$

Substitute $\mathrm{i} = n$ in generic equation

$= \mathrm{T}(0) + n$

$= 0 + n$

T(n) $= O(n)$

Summary

- Discussed the working of recursive functions
- Basic Analysis Framework for recursion is studied
- Different methods of solving recurrence relation

Thank You
Happy Learning

Success is always inevitable with Hard Work and Perseverance