Recursion

- Recursion – the phenomenon of a definition or algorithm that uses itself in the definition or the algorithm
- A (mathematical) function is defined recursively if its definition consists of two parts:
  - One or more *base* (or *anchor* or *stopping*) case(s), in which the value(s) of the function is/are specified in terms of one or more values of the parameter(s)
  - An *inductive* or *recursive* step, in which the function's value for the current value of the parameter(s) is defined in terms of previously defined function values and/or parameter values
  - *E.g.* (factorial):
    - Definition: $$n! = \begin{cases} 1 & \text{if } n = 0 \\ 1 \times 2 \times \cdots \times n = n \times (n-1)! & \text{if } n > 0 \end{cases}$$
    - Anchor or base or stopping case: **`0! = 1`**
    - Inductive or recursive step: for **`n > 0`**, **`n! = n*(n-1)!`**
- A *recursive algorithm* implemented in C++ → recursive function
  - A function that *calls itself* (directly or indirectly)

Recursion

```
#include <iostream>
#include <cstdlib>
using namespace std;

int factorial(int n);

int main()
{
   int num;

   cout << "To compute n!, enter n (integer only): ";
   cin >> num;
   cout << num << "! = " << factorial(num) << endl;

   return EXIT_SUCCESS;
}
```

(continued)

Recursion

```
int factorial(int n)
{
   if (n == 0)                        // base case
      return 1;
   else if (n > 0)                    // inductive step
      return n * factorial(n-1);
   else                               // invalid parameter
   {
      cout << "n! is not defined for negative n" << endl;
      return -1;
   }
}
```


Recursion

```
5! = 5 * 4! = 5 * 24 = 120
         4! = 4 * 3! = 4 * 6 = 24
                  3! = 3 * 2! = 3 * 2 = 6
                           2! = 2 * 1! = 2 * 1 = 2
                                    1! = 1 * 0! = 1 * 1 = 1
                                             0! = 1
```

Recursion

requirements

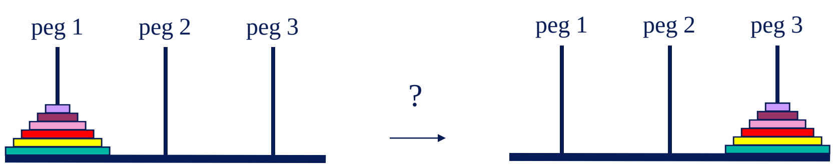
Before attempting to solve a problem recursively, make sure the following four requirements are met:

- The problem must be ***decomposable into sub-problems*** that are themselves ***simpler versions of the same problem*** that you are trying to solve in the first place
- ***Base case(s)*** must ***exist***
- As the problem gets smaller, it must ***approach*** (***get nearer to***) the ***base case(s)***
- ***Base case(s)*** must eventually be ***reached***

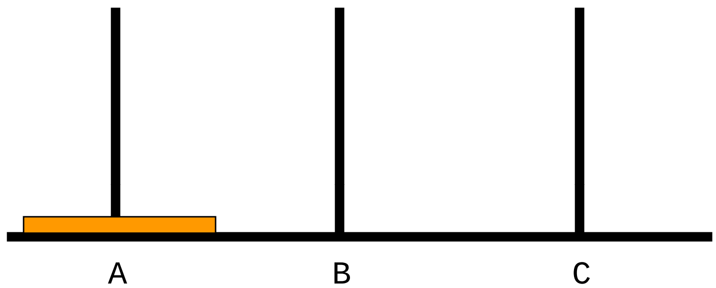
A Classic Problem in Recursion

"The Towers of Hanoi"

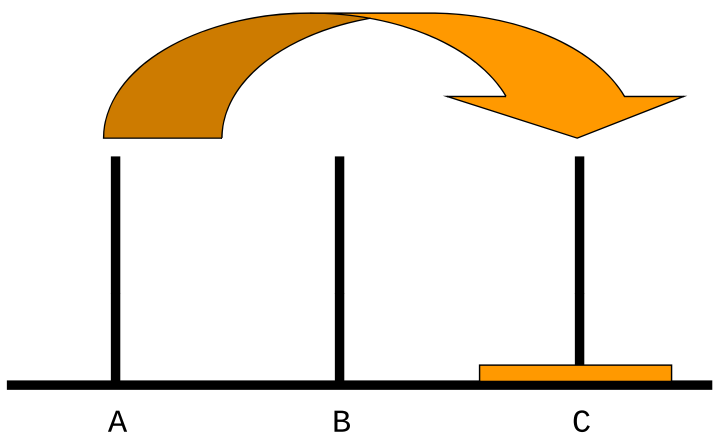
- A puzzle in which disks are moved from one peg to another according to the following set of rules:
  - When a disk is moved from one peg it must be placed on another
  - Only one disk may be moved at a time, and it must be the top disk on a peg
  - A larger disk may never be placed on top of a smaller disk
- Legend has it that…
  - Priests in the Temple of Bramah were given a puzzle consisting of a golden platform with 3 diamond needles, on which were placed 64 golden disks
  - The priests were to move one disk per day, following the above rules
  - When they had successfully completed their task, time would end



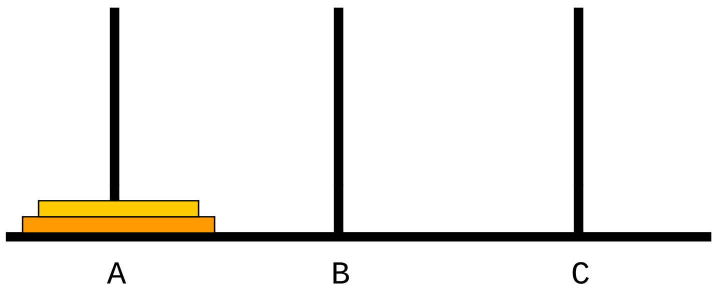
Case of One-Disk Tower



Case of One-Disk Tower

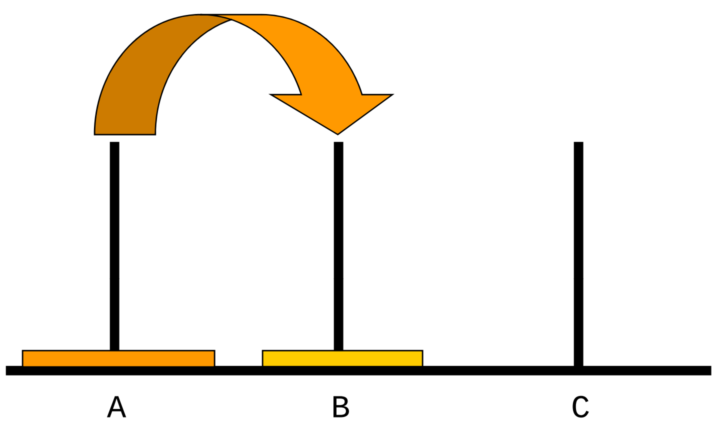


Case of Two-Disk Tower



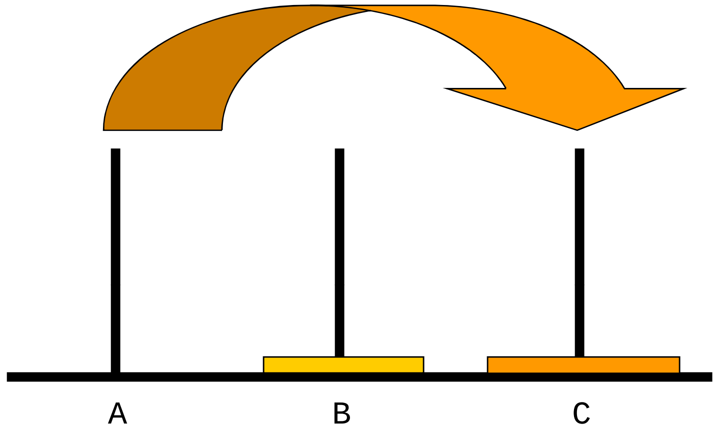
Case of Two-Disk Tower

- Move 1:



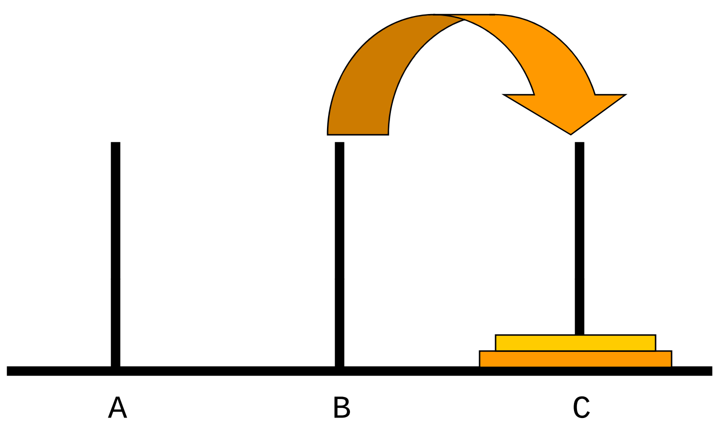
Case of Two-Disk Tower

- Move 2:

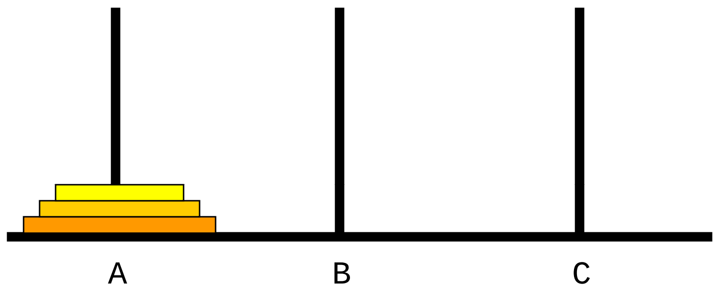


Case of Two-Disk Tower

- Move 3:

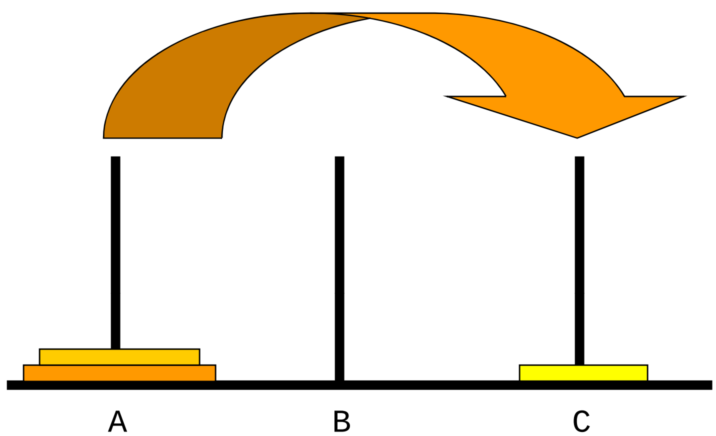


Case of Three-Disk Tower



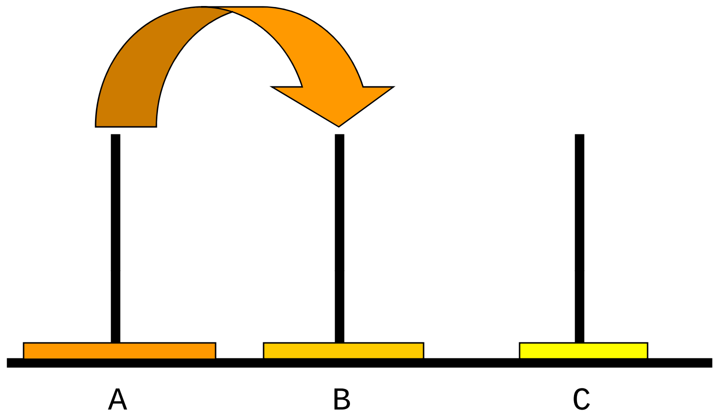
Case of Three-Disk Tower

- Move 1:



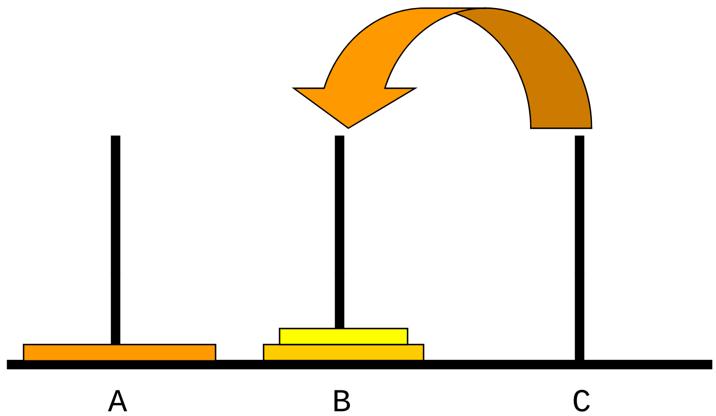
Case of Three-Disk Tower

- Move 2:



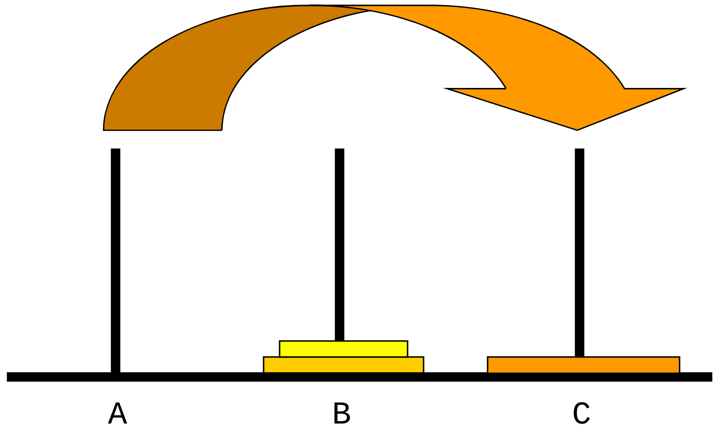
Case of Three-Disk Tower

- Move 3:



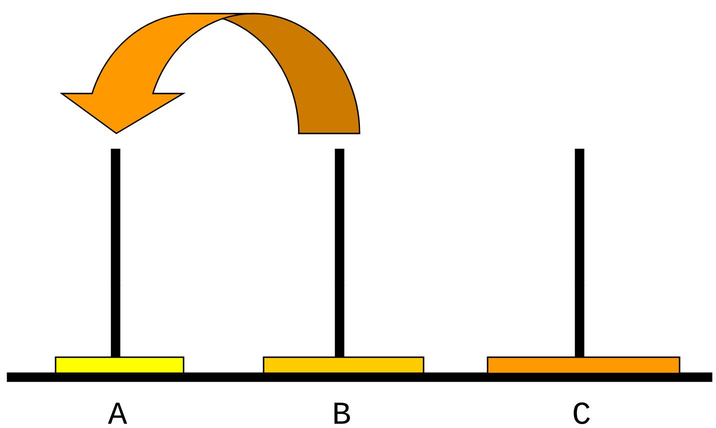
Case of Three-Disk Tower

- Move 4:



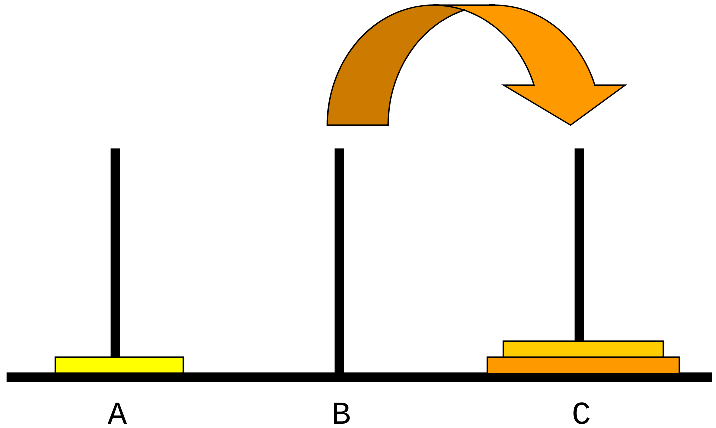
Case of Three-Disk Tower

- Move 5:



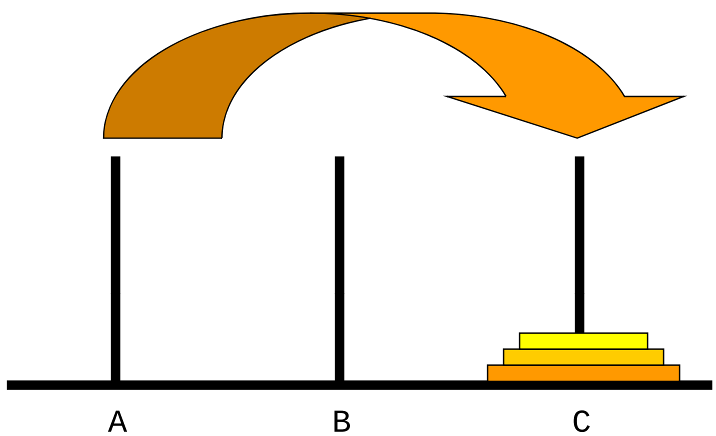
Case of Three-Disk Tower

- Move 6:



Case of Three-Disk Tower

- Move 7:



Simplifying the Case of Three-Disk Tower

- ***Step 1***
  - Move top 2 disks from A to B using C as auxiliary
- ***Step 2***
  - Move the remaining largest disk from A to C
- ***Step 3***
  - Move the 2 disks from B to C using A as auxiliary
- Thus, the problem for 3 disks involves
  - A recursive step → moving 2 disks
  - A stopping step (base case) → a one disk move

Generalizing to Case of *n*-Disk Tower

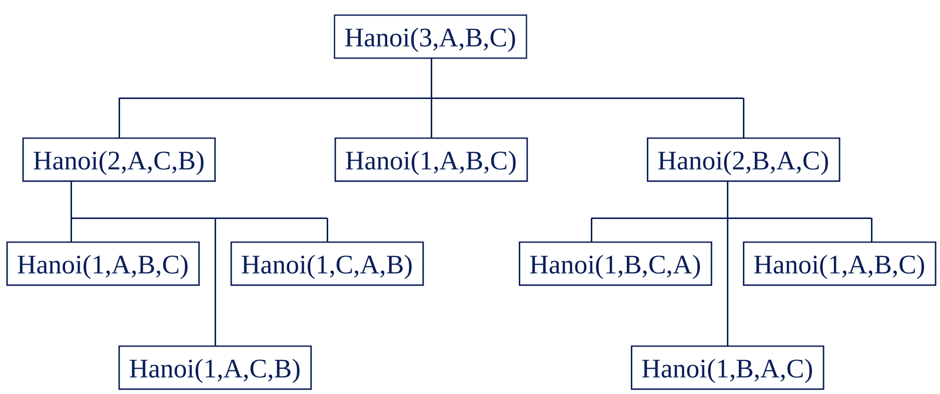
- ***Step 1***
  - Move top (n-1) disks from A to B using C as auxiliary
- ***Step 2***
  - Move the remaining largest disk from A to C
- ***Step 3***
  - Move the (n-1) disks from B to C using A as auxiliary

- Moving (n-1) disks from one tower to another is a smaller problem similar to the original problem that can be solved the same way → recursively

Towers of Hanoi

```
void Hanoi(int n, char fromPeg, char auxPeg, char toPeg)
{
   if (n > 0)
   {
      if (n == 1)
         cout << "Move disk from " << fromPeg << " to "
              << toPeg << endl;
      else
      {
         Hanoi(n-1, fromPeg, toPeg, auxPeg);
         cout << "Move disk from " << fromPeg << " to "
              << toPeg << endl;
         Hanoi(n-1, auxPeg, fromPeg, toPeg);
      }
   }
}
```


Recursive Call Structure



How Many Total Moves for Given *n*?

- If we call Hanoi with …, it takes …

1 disk	Hanoi(1, 'A', 'B', 'C')	1 move
2 disks	Hanoi(2, 'A', 'B', 'C')	3 moves
3 disks	Hanoi(3, 'A', 'B', 'C')	7 moves
4 disks	Hanoi(4, 'A', 'B', 'C')	15 moves
5 disks	Hanoi(5, 'A', 'B', 'C')	31 moves
.	.	
.	.	
.	.	
20 disks	Hanoi(20, 'A', 'B', 'C')	1,048,575 moves
21 disks	Hanoi(21, 'A', 'B', 'C')	2,097,151 moves

What Space and Time Costs?

- Space cost is O(n) and time cost is O(2^n)
- If our machine executes 400 million instructions/second, then ...

# of Disks	Execution Time (seconds)
20	.0025 second
30	2.5 seconds
40	42 minutes
50	29.2 days
60	80 years
70	80,000 years
80	83 million years

- So, at 1 disk per day, it would take approximately 400,000,000*365.4*24*60*60 = 12,600,000,000,000,000 years or about a million times the current estimated age of the universe

Some Remarks about Preceding Analysis

- Time cost is $O(2^n)$
  - (About as bad as we have ever seen Big-O gets)
- Nevertheless, it gets the job done with a minimal amount of code

Recursion vs Iteration

- Iteration can be used in place of recursion
  - An iterative algorithm uses a *looping* (repetition, iteration) construct
  - A recursive algorithm uses a *branching* (function-call, invocation) construct
- Recursive solutions are usually *less efficient* than iterative solutions
  - In terms of both *time* and *space*
  - Mainly due to costly stack operations
- Recursive solutions bear the *risk of stack overflow*
  - Size of stack is finite
- Recursion can greatly *simplify the solution of certain problems*
  - Often resulting in concise and elegant algorithms (thus source code)

Recursion

Tracing Exercise

Shown below are the definitions for functions **F1**, **F2** and **F3**:

```
void F1(int n)
{
   if (n>=1 && n<= 8)
   {
      cout << n;
      F1(n – 1);
   }
   else
      cout << "*";
}
```

```
void F2(int n)
{
   if (n>=1 && n<=8)
   {
      F2(n - 1);
      cout << n;
   }
   else
      cout << "*";
}
```

```
void F3(int n)
{
   if (n>=1 && n<=8)
   {
      cout << n;
      F3(n + 1);
      cout << n;
   }
   else
      cout << "*";
}
```

- What output is produced by the function call **F1(3)**?
- What output is produced by the function call **F1(7)**?
- What output is produced by the function call **F2(3)**?
- What output is produced by the function call **F2(7)**?
- What output is produced by the function call **F3(3)**?
- What output is produced by the function call **F3(7)**?

Recursion

Coding Exercises

- Write a ***recursive*** function to find the sum
  `1 + 2 + 3 + … + n`
  given **`n`** (a nonzero, positive integer)
- Write a ***recursive*** function to find the sum of the first **`n`** odd integers
  `1 + 3 + 5 + … + (2*n - 1)`
  given **`n`** (a nonzero, positive integer)
- Given below is the body of the **`main`** function of a program that is to read in a line of text on a character-by-character basis and then display the characters in reverse order. It calls the function **`Reverse`** that uses ***recursion*** to carry out the reversal of the characters. Write the function definition for **`Reverse`**.

```
int main()
{
   cout << "Enter a line of text below:" << endl;
   Reverse();
   cout << endl;
   return EXIT_SUCCESS;
}
```


Textbook Readings

- Chapter 9
  - Section 9.1

Extras

- Lecture note from another of my (recent) classes
  - CS 2318: Assembly Language

MIPS32 AL – More On Functions
(quick review on recursion)

- Toward a general "definition" of recursion:

  A term used to describe the…

  characteristic feature of a method of dealing with a subject…

  whereby…

  part of the method involves other subjects of the same kind as the subject under consideration
- Examples of "dealing with a subject" *recursively*:
  - Defining a *tree* in terms of *(sub-)trees*
  - Evaluating **`n!`** using **`(n - 1)!`**, **`(n - 2)!`**, **`...`**
  - Calling a function that's an instance of the one being implemented

MIPS32 AL – More On Functions
(quick review on recursion)

- An important role played by recursion in computer science:
  - (recall that computer science is about ***problem solving***)
  - *Divide-and-conquer* problem-solving strategy
    - Solve given problem by solving smaller problems of the same type
- C++ implementation (as function) of associated algorithm:
  - Function body has call(s) to same function → *recursive* function
- Recursive function is usually *less efficient* (resource-wise)
  - Compared to it's *iterative* counterpart
    - (any difficulties involved in obtaining the iterative version aside)
  - Due to *overhead associated with function calls*
- Recursion indispensable for certain important problems
  - (*e.g.*: traversing/processing *non-linear* data structures like trees)
  - Mightily difficult (if not impossible) to deal with iteratively
  - Where recursion finds its niche

MIPS32 AL – More On Functions
(quick review on recursion)

- For recursion to be useful/successful problem-solving tool, insofar as it means using the *divide-and-conquer* strategy, certain conditions must apply:
  - Problem is decomposable into *smaller problems* of the *same type*
  - At least a *base case* exists
    - Also called *anchor case*, *stopping case*, …
  - Each recursive step *makes progress* toward a base case
  - Base case(s) will *eventually be reached*
- Fatal error can result if method is not properly applied
  - *Infinite recursion*
  - *Stack overflow*

MIPS32 AL – More On Functions
(quick review on recursion)

- Main hurdle students face when applying recursion:
  - Express problem in terms of smaller problems of the same type
- Key to success:
  - Think divide-and-conquer
    - Do only a small part yourself
  - Have faith on others to (together) do the rest
- A simple problem we'll solve/implement recursively
  - Sum numbers from 1 to N (for N >= 1)
- 3 other relatively simple problems (for practice):
  - Flip contents of an array: {1, 2, 3, 4, 5} becomes {5, 4, 3, 2, 1}
  - Search if an array contains a value that matches a given value
  - Determine if an array contains any duplicates
- (You wish they were always so simple!)

MIPS32 AL – More On Functions
(a recursive function example)

- Sum from 1 to N (for N >= 1)

```
int SumToN(int N) // N >= 1 & not too big
{
   if (N < 2)
       return 1;
   else
       return N + SumToN(N - 1);
}
```

- **`SumToN`** is both caller and callee
- How should we implement in MIPS assembly?

MIPS32 AL – More On Functions
(a recursive function example)

- Sum from 1 to N (for N >= 1)

```
int SumToN(int N) // N >= 1 & not too big
{
   if (N < 2)
        return 1;
   else
        return N + SumToN(N – 1);
}
```

- `SumToN` is both caller and callee
- How should we implement it in MIPS assembly?
- **Good news: no new things to be learned**
  - **We implement it just like any other (non-leaf) function**

MIPS32 AL – More On Fns
(a recursive function e.g.)

- Sum from 1 to N

```
SumToN:
         addiu $sp, $sp, -32
         sw $ra, 28 ($sp)
         sw $fp, 24 ($sp)
         addiu $fp, $sp, 32

         slti $t0, $a0, 2
         beq $t0, $0, recur
         addi $v0, $0, 1
         j done
recur:   sw $a0, 0($fp)
         addi $a0, $a0, -1
         jal  SumToN
         lw $a0, 0($fp)
         add $v0, $v0, $a0

done:    lw $fp, 24($sp)
         lw $ra, 28($sp)
         addiu $sp, $sp, 32
         jr $ra
```

```
             .data
str1:        .asciiz "Desired N: "
str2:        .asciiz "Sum to N = "
             .text
             .globl main
main:
             addiu $sp, $sp, -32
             sw $ra, 28 ($sp)
             sw $fp, 24 ($sp)
             addiu $fp, $sp, 32

             la $a0, str1
             li $v0, 4
             syscall
             li $v0, 5
             syscall
             j quitTest
repeat:      move $a0, $v0
             jal SumToN
             move $t0, $v0
             la $a0, str2
             li $v0, 4
             syscall
             move $a0, $t0
             li $v0, 1
             syscall
             li $a0, '\n'
             li $v0, 11
             syscall
             la $a0, str1
             li $v0, 4
             syscall
             li $v0, 5
             syscall
quitTest:    bgtz $v0, repeat

             lw $fp, 24($sp)
             lw $ra, 28($sp)
             addiu $sp, $sp, 32
             li $v0, 10
             syscall
```

MIPS32 AL – More On F^n_s

(a recursive function e.g.)

- Let's trace *sum from 1 to 3*

```
SumToN:
        addiu $sp, $sp, -32
        sw $ra, 28 ($sp)
        sw $fp, 24 ($sp)
        addiu $fp, $sp, 32

        slti $t0, $a0, 2
        beq $t0, $0, recur
        addi $v0, $0, 1
        j done
recur:  sw $a0, 0($fp)
        addi $a0, $a0, -1
        jal  SumToN
        lw $a0, 0($fp)
        add $v0, $v0, $a0

done:   lw $fp, 24($sp)
        lw $ra, 28($sp)
        addiu $sp, $sp, 32
        jr $ra
```

$a0 = 3

```
          .data
str1:     .asciiz "Desired N: "
str2:     .asciiz "Sum to N = "
          .text
          .globl main
main:
          addiu $sp, $sp, -32
          sw $ra, 28 ($sp)
          sw $fp, 24 ($sp)
          addiu $fp, $sp, 32

          la $a0, str1
          li $v0, 4
          syscall
          li $v0, 5
          syscall
          j quitTest
repeat:   move $a0, $v0
          jal SumToN
          move $t0, $v0
          la $a0, str2
          li $v0, 4
          syscall
          move $a0, $t0
          li $v0, 1
          syscall
          li $a0, '\n'
          li $v0, 11
          syscall
          la $a0, str1
          li $v0, 4
          syscall
          li $v0, 5
          syscall
quitTest: bgtz $v0, repeat

          lw $fp, 24($sp)
          lw $ra, 28($sp)
          addiu $sp, $sp, 32
          li $v0, 10
          syscall
```


MIPS32 AL – More On F^n_s

(a recursive function e.g.)

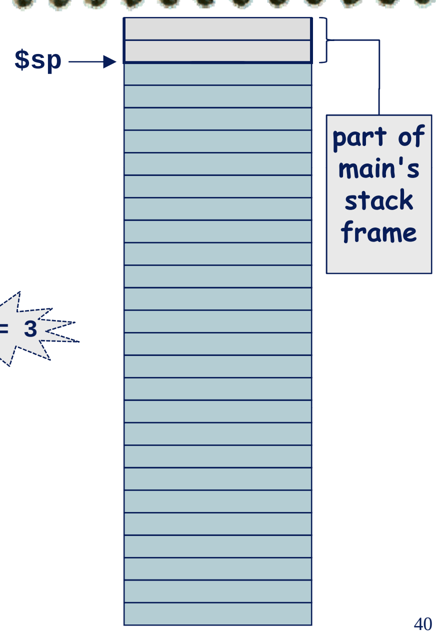
- Let's trace *sum from 1 to 3*

```
SumToN:
        addiu $sp, $sp, -32
        sw $ra, 28 ($sp)
        sw $fp, 24 ($sp)
        addiu $fp, $sp, 32

        slti $t0, $a0, 2
        beq $t0, $0, recur
        addi $v0, $0, 1
        j done
recur:  sw $a0, 0($fp)
        addi $a0, $a0, -1
        jal  SumToN
        lw $a0, 0($fp)
        add $v0, $v0, $a0

done:   lw $fp, 24($sp)
        lw $ra, 28($sp)
        addiu $sp, $sp, 32
        jr $ra
```

$a0 = 3



MIPS32 AL – More On F$^{n}_{s}$

(a recursive function e.g.)

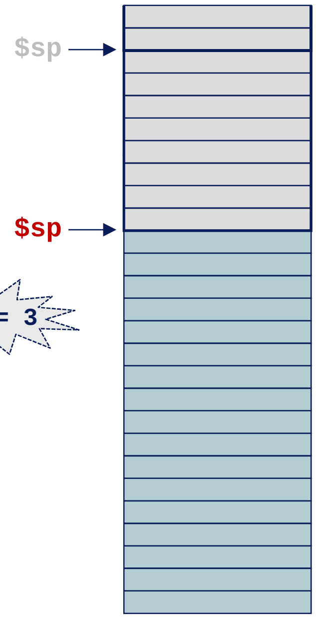
- SumToN(3)

```
SumToN:
        addiu $sp, $sp, -32
        sw $ra, 28 ($sp)
        sw $fp, 24 ($sp)
        addiu $fp, $sp, 32

        slti $t0, $a0, 2
        beq $t0, $0, recur
        addi $v0, $0, 1
        j done
recur:  sw $a0, 0($fp)
        addi $a0, $a0, -1
        jal  SumToN
        lw $a0, 0($fp)
        add $v0, $v0, $a0

done:   lw $fp, 24($sp)
        lw $ra, 28($sp)
        addiu $sp, $sp, 32
        jr $ra
```

$a0 = 3



MIPS32 AL – More On F$^{n}_{s}$

(a recursive function e.g.)

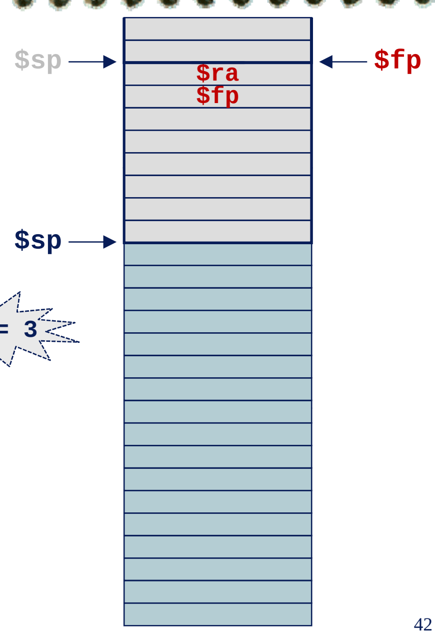
- **SumToN(3)**

```
SumToN:
        addiu $sp, $sp, -32
        sw $ra, 28 ($sp)
        sw $fp, 24 ($sp)
        addiu $fp, $sp, 32

        slti $t0, $a0, 2
        beq $t0, $0, recur
        addi $v0, $0, 1
        j done
recur:  sw $a0, 0($fp)
        addi $a0, $a0, -1
        jal  SumToN
        lw $a0, 0($fp)
        add $v0, $v0, $a0

done:   lw $fp, 24($sp)
        lw $ra, 28($sp)
        addiu $sp, $sp, 32
        jr $ra
```

$a0 = 3



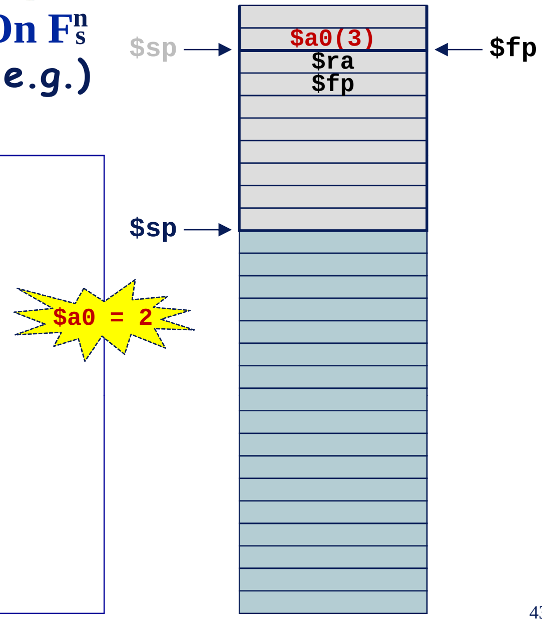
MIPS32 AL – More On F^n_s
(a recursive function e.g.)

- SumToN(3)

```
SumToN:
         addiu $sp, $sp, -32
         sw $ra, 28 ($sp)
         sw $fp, 24 ($sp)
         addiu $fp, $sp, 32

         slti $t0, $a0, 2
         beq $t0, $0, recur
         addi $v0, $0, 1
         j done
recur:   sw $a0, 0($fp)
         addi $a0, $a0, -1
         jal  SumToN
         lw $a0, 0($fp)
         add $v0, $v0, $a0

done:    lw $fp, 24($sp)
         lw $ra, 28($sp)
         addiu $sp, $sp, 32
         jr $ra
```



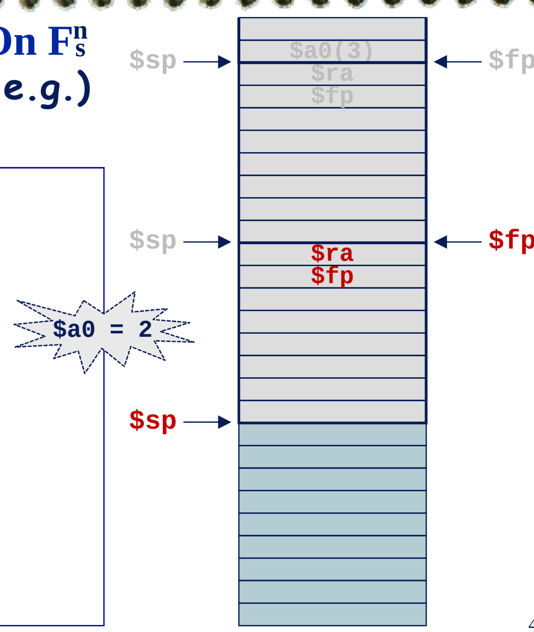
MIPS32 AL – More On F^n_s
(a recursive function e.g.)

- SumToN(2)

```
SumToN:
         addiu $sp, $sp, -32
         sw $ra, 28 ($sp)
         sw $fp, 24 ($sp)
         addiu $fp, $sp, 32

         slti $t0, $a0, 2
         beq $t0, $0, recur
         addi $v0, $0, 1
         j done
recur:   sw $a0, 0($fp)
         addi $a0, $a0, -1
         jal  SumToN
         lw $a0, 0($fp)
         add $v0, $v0, $a0

done:    lw $fp, 24($sp)
         lw $ra, 28($sp)
         addiu $sp, $sp, 32
         jr $ra
```



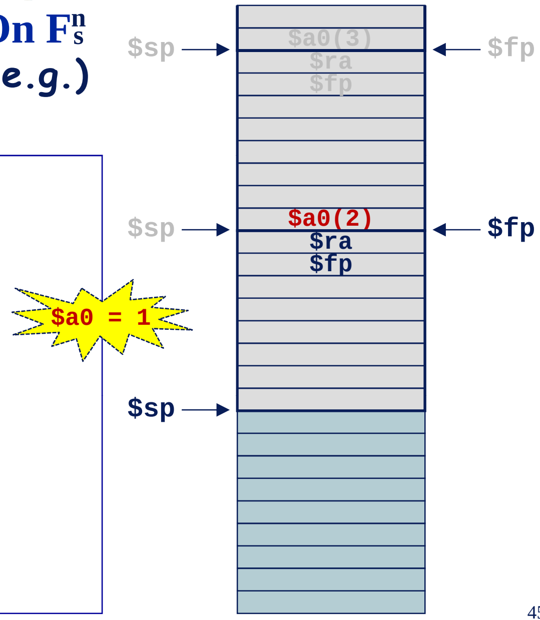
MIPS32 AL – More On F$^{n}_{s}$
(a recursive function e.g.)

- SumToN(2)

```
SumToN:
        addiu $sp, $sp, -32
        sw $ra, 28 ($sp)
        sw $fp, 24 ($sp)
        addiu $fp, $sp, 32

        slti $t0, $a0, 2
        beq $t0, $0, recur
        addi $v0, $0, 1
        j done
recur:  sw $a0, 0($fp)
        addi $a0, $a0, -1
        jal  SumToN
        lw $a0, 0($fp)
        add $v0, $v0, $a0

done:   lw $fp, 24($sp)
        lw $ra, 28($sp)
        addiu $sp, $sp, 32
        jr $ra
```



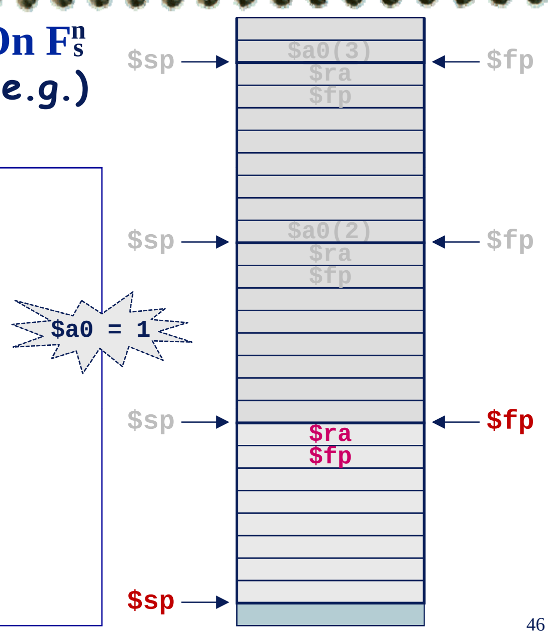
MIPS32 AL – More On F$^{n}_{s}$
(a recursive function e.g.)

- SumToN(1)

```
SumToN:
        addiu $sp, $sp, -32
        sw $ra, 28 ($sp)
        sw $fp, 24 ($sp)
        addiu $fp, $sp, 32

        slti $t0, $a0, 2
        beq $t0, $0, recur
        addi $v0, $0, 1
        j done
recur:  sw $a0, 0($fp)
        addi $a0, $a0, -1
        jal  SumToN
        lw $a0, 0($fp)
        add $v0, $v0, $a0

done:   lw $fp, 24($sp)
        lw $ra, 28($sp)
        addiu $sp, $sp, 32
        jr $ra
```



MIPS32 AL – More On F^n_s (a recursive function e.g.)

- SumToN(1)

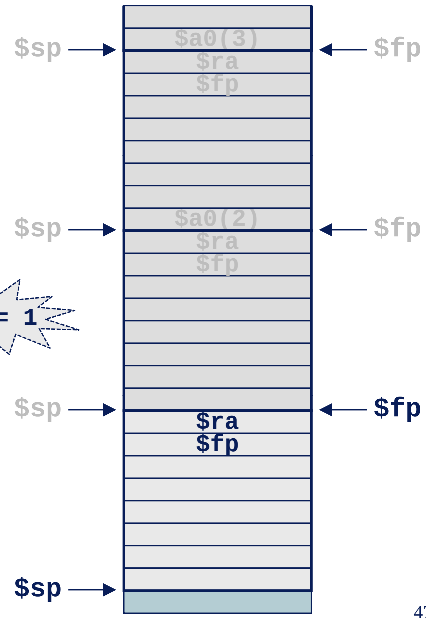
$v0 = 1

```
SumToN:
            addiu $sp, $sp, -32
            sw $ra, 28 ($sp)
            sw $fp, 24 ($sp)
            addiu $fp, $sp, 32

            slti $t0, $a0, 2
            beq $t0, $0, recur
            addi $v0, $0, 1
            j done
recur:      sw $a0, 0($fp)
            addi $a0, $a0, -1
            jal  SumToN
            lw $a0, 0($fp)
            add $v0, $v0, $a0

done:       lw $fp, 24($sp)
            lw $ra, 28($sp)
            addiu $sp, $sp, 32
            jr $ra
```

$a0 = 1



MIPS32 AL – More On F^n_s (a recursive function e.g.)

- SumToN(1)

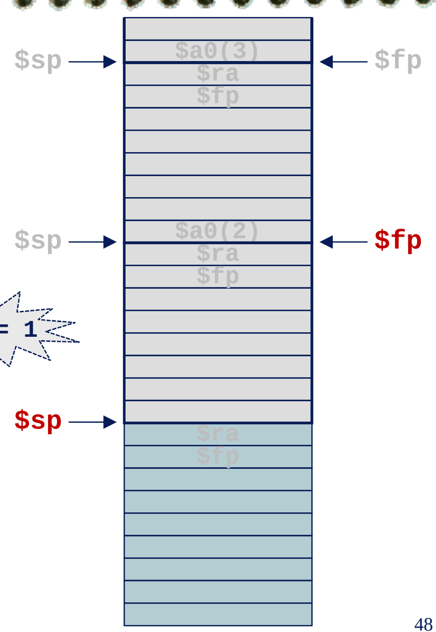
$v0 = 1

```
SumToN:
            addiu $sp, $sp, -32
            sw $ra, 28 ($sp)
            sw $fp, 24 ($sp)
            addiu $fp, $sp, 32

            slti $t0, $a0, 2
            beq $t0, $0, recur
            addi $v0, $0, 1
            j done
recur:      sw $a0, 0($fp)
            addi $a0, $a0, -1
            jal  SumToN
            lw $a0, 0($fp)
            add $v0, $v0, $a0

done:       lw $fp, 24($sp)
            lw $ra, 28($sp)
            addiu $sp, $sp, 32
            jr $ra
```

$a0 = 1



MIPS32 AL – More On F^n_s
(a recursive function e.g.)

- SumToN(2)

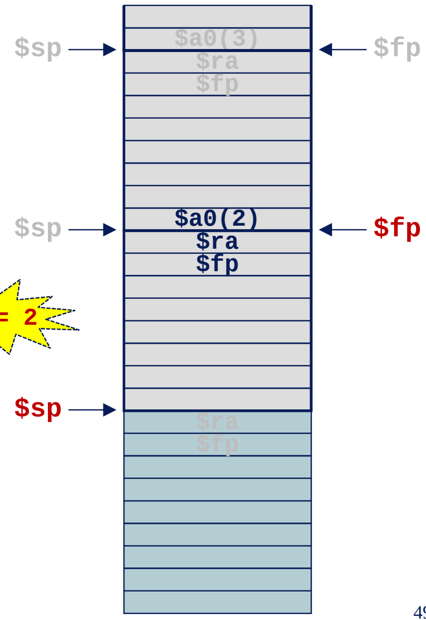
$v0 = 3

```
SumToN:
        addiu $sp, $sp, -32
        sw $ra, 28 ($sp)
        sw $fp, 24 ($sp)
        addiu $fp, $sp, 32

        slti $t0, $a0, 2
        beq $t0, $0, recur
        addi $v0, $0, 1
        j done
recur:  sw $a0, 0($fp)
        addi $a0, $a0, -1
        jal  SumToN
        lw $a0, 0($fp)
        add $v0, $v0, $a0

done:   lw $fp, 24($sp)
        lw $ra, 28($sp)
        addiu $sp, $sp, 32
        jr $ra
```

$a0 = 2



MIPS32 AL – More On F^n_s
(a recursive function e.g.)

- SumToN(2)

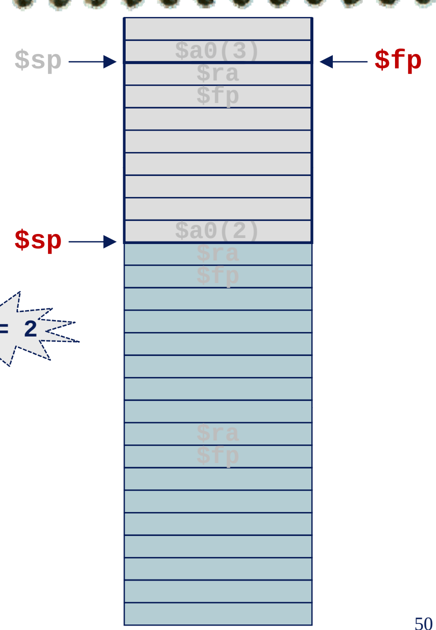
$v0 = 3

```
SumToN:
        addiu $sp, $sp, -32
        sw $ra, 28 ($sp)
        sw $fp, 24 ($sp)
        addiu $fp, $sp, 32

        slti $t0, $a0, 2
        beq $t0, $0, recur
        addi $v0, $0, 1
        j done
recur:  sw $a0, 0($fp)
        addi $a0, $a0, -1
        jal  SumToN
        lw $a0, 0($fp)
        add $v0, $v0, $a0

done:   lw $fp, 24($sp)
        lw $ra, 28($sp)
        addiu $sp, $sp, 32
        jr $ra
```

$a0 = 2



MIPS32 AL – More On Fns
(a recursive function e.g.)

- SumToN(3)

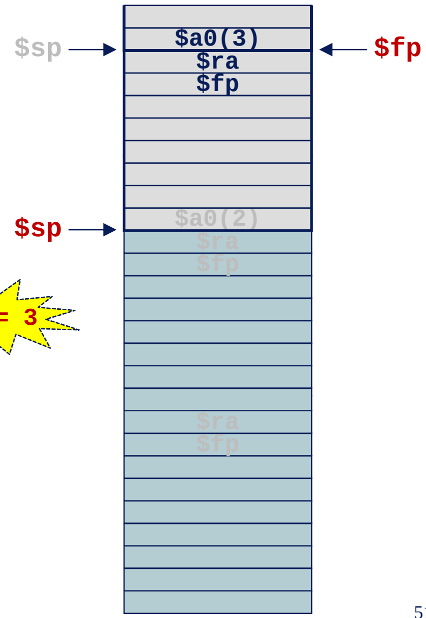
$v0 = 6

```
SumToN:
        addiu $sp, $sp, -32
        sw $ra, 28 ($sp)
        sw $fp, 24 ($sp)
        addiu $fp, $sp, 32

        slti $t0, $a0, 2
        beq $t0, $0, recur
        addi $v0, $0, 1
        j done
recur:  sw $a0, 0($fp)
        addi $a0, $a0, -1
        jal  SumToN
        lw $a0, 0($fp)
        add $v0, $v0, $a0

done:   lw $fp, 24($sp)
        lw $ra, 28($sp)
        addiu $sp, $sp, 32
        jr $ra
```

$a0 = 3



MIPS32 AL – More On Fns
(a recursive function e.g.)

- SumToN(3)

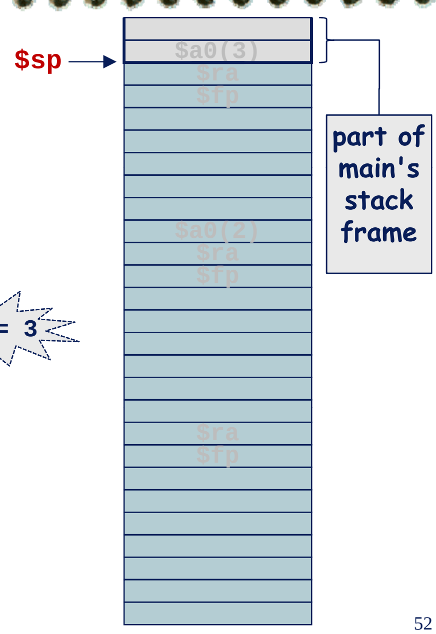
$v0 = 6

```
SumToN:
        addiu $sp, $sp, -32
        sw $ra, 28 ($sp)
        sw $fp, 24 ($sp)
        addiu $fp, $sp, 32

        slti $t0, $a0, 2
        beq $t0, $0, recur
        addi $v0, $0, 1
        j done
recur:  sw $a0, 0($fp)
        addi $a0, $a0, -1
        jal  SumToN
        lw $a0, 0($fp)
        add $v0, $v0, $a0

done:   lw $fp, 24($sp)
        lw $ra, 28($sp)
        addiu $sp, $sp, 32
        jr $ra
```

$a0 = 3



MIPS32 AL – More On F^n_s

(a recursive function e.g.)

- Let's trace ***sum f***

$v0 = 6

```
SumToN:
        addiu $sp, $sp, -32
        sw $ra, 28 ($sp)
        sw $fp, 24 ($sp)
        addiu $fp, $sp, 32

        slti $t0, $a0, 2
        beq $t0, $0, recur
        addi $v0, $0, 1
        j done
recur:  sw $a0, 0($fp)
        addi $a0, $a0, -1
        jal  SumToN
        lw $a0, 0($fp)
        add $v0, $v0, $a0

done:   lw $fp, 24($sp)
        lw $ra, 28($sp)
        addiu $sp, $sp, 32
        jr $ra
```

$a0 = 3

```
            .data
str1:       .asciiz "Desired N: "
str2:       .asciiz "Sum to N = "
            .text
            .globl main
main:
            addiu $sp, $sp, -32
            sw $ra, 28 ($sp)
            sw $fp, 24 ($sp)
            addiu $fp, $sp, 32

            la $a0, str1
            li $v0, 4
            syscall
            li $v0, 5
            syscall
            j quitTest
repeat:     move $a0, $v0
            jal SumToN
            move $t0, $v0
            la $a0, str2
            li $v0, 4
            syscall
            move $a0, $t0
            li $v0, 1
            syscall
            li $a0, '\n'
            li $v0, 11
            syscall
            la $a0, str1
            li $v0, 4
            syscall
            li $v0, 5
            syscall
quitTest:   bgtz $v0, repeat

            lw $fp, 24($sp)
            lw $ra, 28($sp)
            addiu $sp, $sp, 32
            li $v0, 10
            syscall
```

$t0 = 6