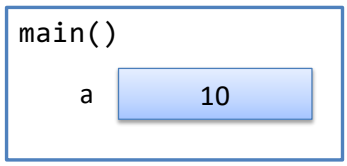
```
/*
 *  university of tehran
 *  school of ece
 *
 *  advanced programming
 *  ramtin khosravi
 */
#define LECTURES  4,5

int main()
{
    cout << "Functions, "
         << "Recursion\n";
}
```

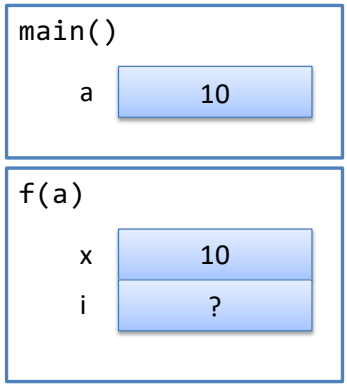

Call-by-Value

```
1.  void f(int x)
2.  {
3.      int i = 5;
4.      x = i + 1;
5.  }

6.  int main()
7.  {
8.      int a = 10;
9.      f(a);
10.     cout << a << "\n";
11. }
```



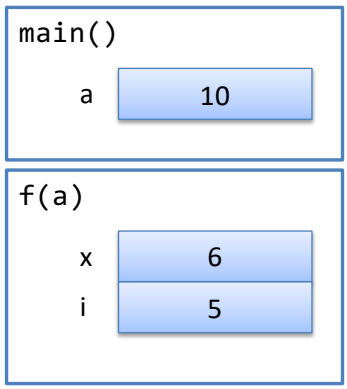
Call-by-Value

```
1.  void f(int x)
2.  {
3.      int i = 5;
4.      x = i + 1;
5.  }

6.  int main()
7.  {
8.      int a = 10;
9.      f(a);
10.     cout << a << "\n";
11. }
```



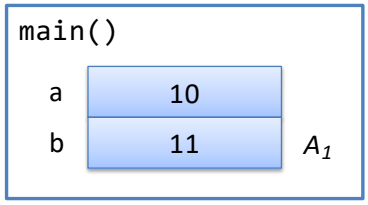
Call-by-Value

```
1.  void f(int x)
2.  {
3.      int i = 5;
4.      x = i + 1;
5.  }

6.  int main()
7.  {
8.      int a = 10;
9.      f(a);
10.     cout << a << "\n";
11. }
```



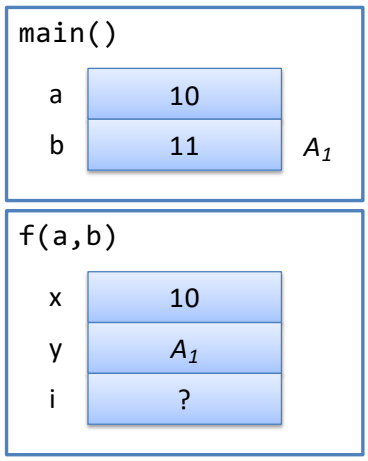
Call-by-Reference

```
1.  void f(int x, int& y)
2.  {
3.      int i = 5;
4.      x = i + 1;
5.      y = 18;
6.  }

7.  int main()
8.  {
9.      int a = 10;
10.     int b = 11;
11.     f(a, b);
12.     cout << a << b;
13. }
```



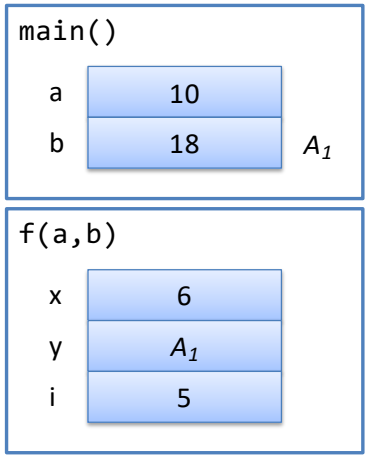
Call-by-Reference

```
1.  void f(int x, int& y)
2.  {
3.      int i = 5;
4.      x = i + 1;
5.      y = 18;
6.  }

7.  int main()
8.  {
9.      int a = 10;
10.     int b = 11;
11.     f(a, b);
12.     cout << a << b;
13. }
```



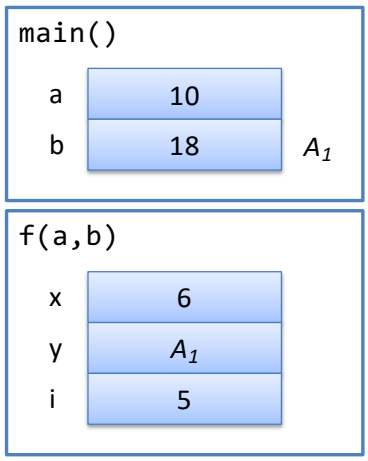
Call-by-Reference

```
1.  void f(int x, int& y)
2.  {
3.      int i = 5;
4.      x = i + 1;
5.      y = 18;
6.  }

7.  int main()
8.  {
9.      int a = 10;
10.     int b = 11;
11.     f(a, b);
12.     cout << a << b;
13. }
```



Call-by-Reference

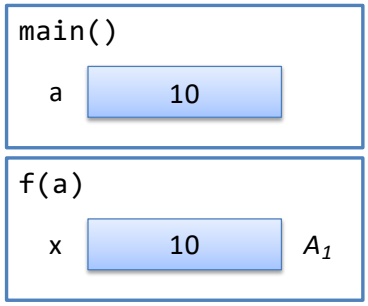
```
1.  void f(int x, int& y)
2.  {
3.      int i = 5;
4.      x = i + 1;
5.      y = 18;
6.  }

7.  int main()
8.  {
9.      int a = 10;
10.     int b = 11;
11.     f(a, b);
12.     cout << a << b;
13. }
```



Is it possible to write f(a, a+1)?

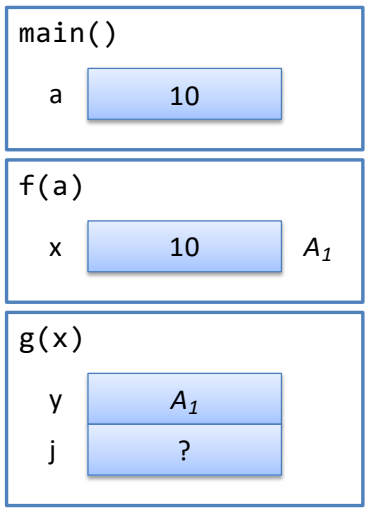
Nested Calls

```
1.  void g(int& y) {
2.      int j = 2;
3.      y = j * 3;
4.  }

5.  void f(int x) {
6.      g(x);
7.  }

8.  int main() {
9.      int a = 10;
10.     f(a);
11.     cout << a << "\n";
12. }
```



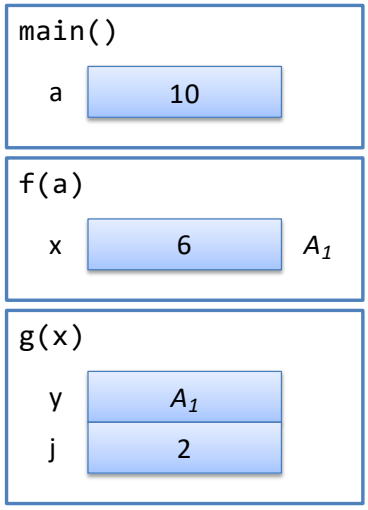
Nested Calls

```
1.  void g(int& y) {
2.      int j = 2;
3.      y = j * 3;
4.  }

5.  void f(int x) {
6.      g(x);
7.  }

8.  int main() {
9.      int a = 10;
10.     f(a);
11.     cout << a << "\n";
12. }
```



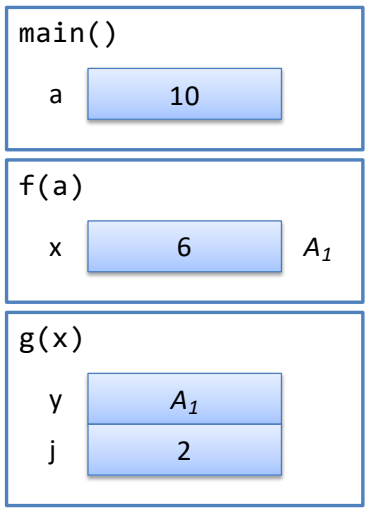
Nested Calls

```
1.  void g(int& y) {
2.      int j = 2;
3.      y = j * 3;
4.  }

5.  void f(int x) {
6.      g(x);
7.  }

8.  int main() {
9.      int a = 10;
10.     f(a);
11.     cout << a << "\n";
12. }
```



Nested Calls

```
1.  void g(int& y) {
2.      int j = 2;
3.      y = j * 3;
4.  }

5.  void f(int x) {
6.      g(x);
7.  }

8.  int main() {
9.      int a = 10;
10.     f(a);
11.     cout << a << "\n";
12. }
```



What happens if we have **`void f(int& x)`***?*

Accessing Variables

```
1. void f(int n) { n=7; }
```

```
movl $0x7, 0x8(%ebp)
```

```
2. void f(int& n) { n=7; }
```

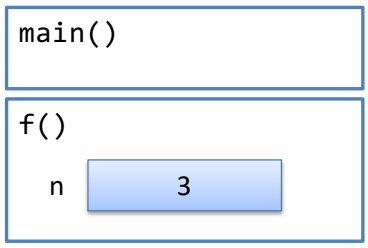
```
mov  0x8(%ebp), %eax
movl $0x7, (%eax)
```

```
3. int n;
   void f() { n=7; }
```

```
movl $0x7,0x804a01c
```

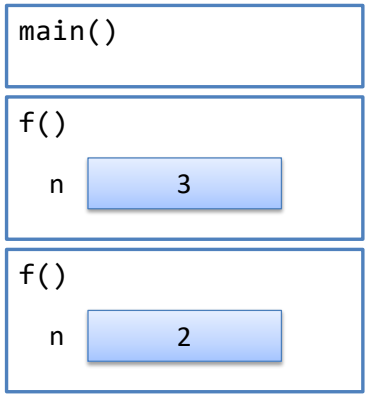

Recursive Calls

```
1. void f(int n) {
2.     if (n == 1)
3.         return;
4.     f(n - 1);
5. }

6. int main() {
7.     f(3);
8. }
```



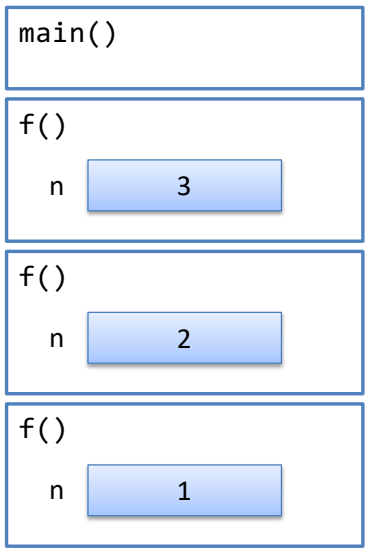
Recursive Calls

```
1.  void f(int n) {
2.      if (n == 1)
3.              return;
4.      f(n - 1);
5.  }

6.  int main() {
7.      f(3);
8.  }
```



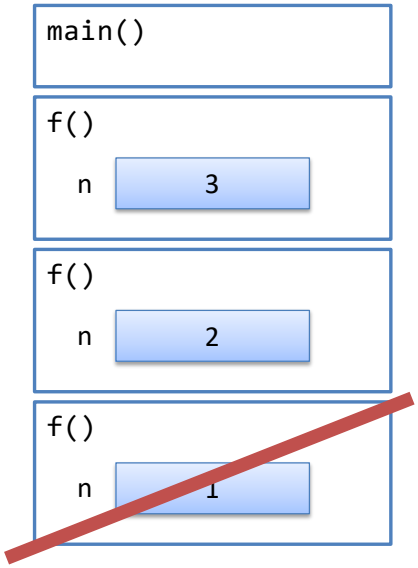
Recursive Calls

```
1.  void f(int n) {
2.      if (n == 1)
3.              return;
4.      f(n - 1);
5.  }

6.  int main() {
7.      f(3);
8.  }
```



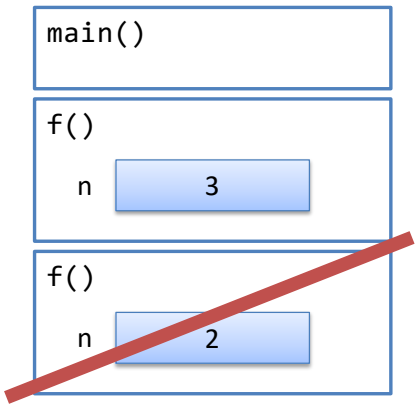
Recursive Calls

```
1.  void f(int n) {
2.      if (n == 1)
3.          return;
4.      f(n - 1);
5.  }

6.  int main() {
7.      f(3);
8.  }
```



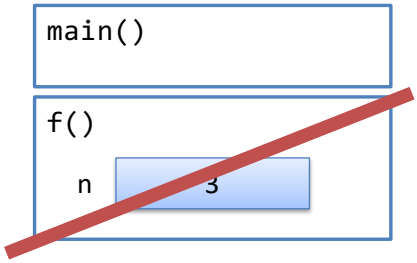
Recursive Calls

```
1.  void f(int n) {
2.      if (n == 1)
3.          return;
4.      f(n - 1);
5.  }

6.  int main() {
7.      f(3);
8.  }
```



Recursive Calls

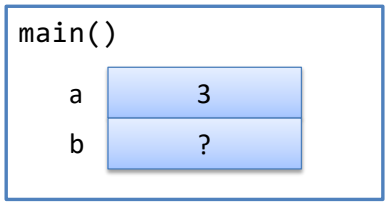
```
1. void f(int n) {
2.     if (n == 1)
3.             return;
4.     f(n - 1);
5. }

6. int main() {
7.     f(3);
8. }
```



Recursive Computations

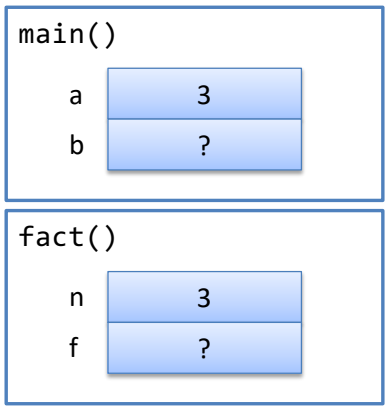
```
1. int fact(int n) {
2.     if (n <= 1)
3.             return 1;
4.     int f = fact(n - 1);
5.     return f * n;
6. }

7. int main() {
8.     int a = 3;
9.     int b = fact(a);
10. }
```

Recursive Computations




```
1.  int fact(int n) {
2.      if (n <= 1)
3.              return 1;
4.      int f = fact(n - 1);
5.      return f * n;
6.  }

7.  int main() {
8.      int a = 3;
9.      int b = fact(a);
10. }
```

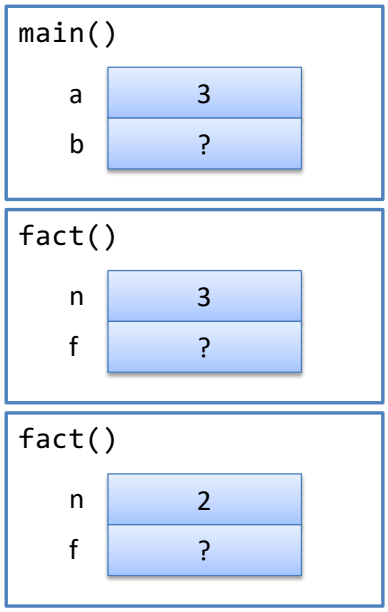

Recursive Computations




```
1.  int fact(int n) {
2.      if (n <= 1)
3.              return 1;
4.      int f = fact(n - 1);
5.      return f * n;
6.  }

7.  int main() {
8.      int a = 3;
9.      int b = fact(a);
10. }
```

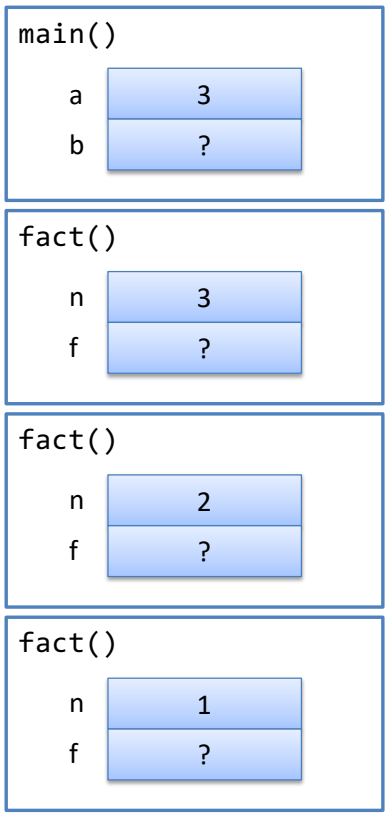
Recursive Computations

```
1.  int fact(int n) {
2.      if (n <= 1)
3.              return 1;
4.      int f = fact(n - 1);
5.      return f * n;
6.  }

7.  int main() {
8.      int a = 3;
9.      int b = fact(a);
10. }
```



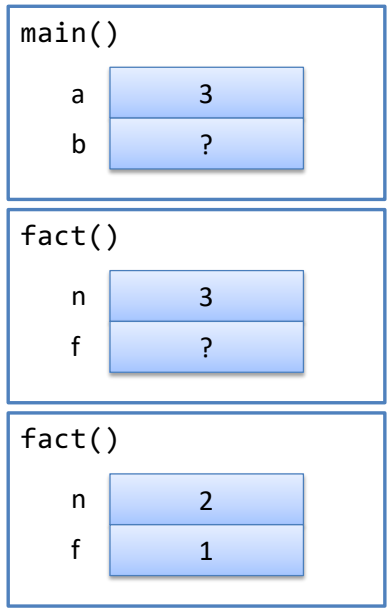
Recursive Computations

```
1.  int fact(int n) {
2.      if (n <= 1)
3.              return 1;
4.      int f = fact(n - 1);
5.      return f * n;
6.  }

7.  int main() {
8.      int a = 3;
9.      int b = fact(a);
10. }
```



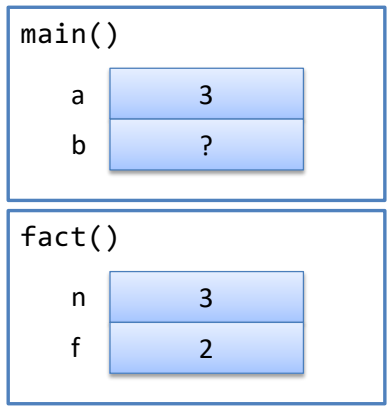
Recursive Computations

```
1.  int fact(int n) {
2.      if (n <= 1)
3.              return 1;
4.      int f = fact(n - 1);
5.      return f * n;
6.  }

7.  int main() {
8.      int a = 3;
9.      int b = fact(a);
10. }
```

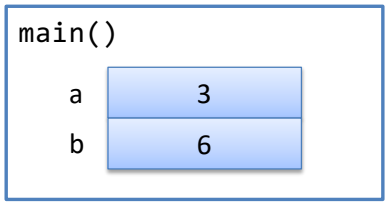


Recursive Computations

```
1.  int fact(int n) {
2.      if (n <= 1)
3.              return 1;
4.      int f = fact(n - 1);
5.      return f * n;
6.  }

7.  int main() {
8.      int a = 3;
9.      int b = fact(a);
10. }
```

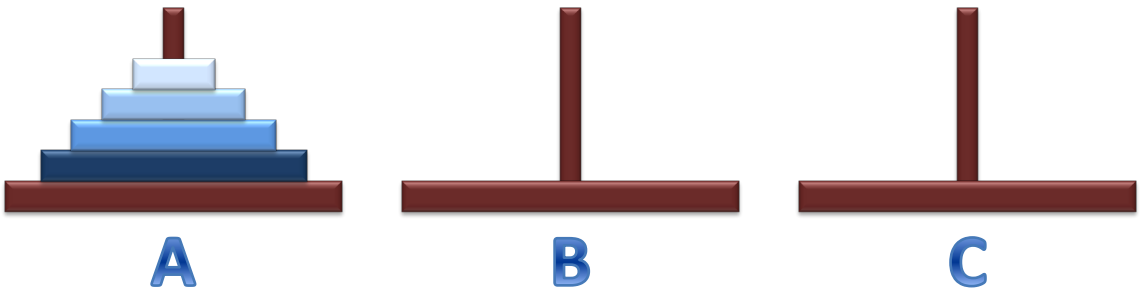


Recursive Computations



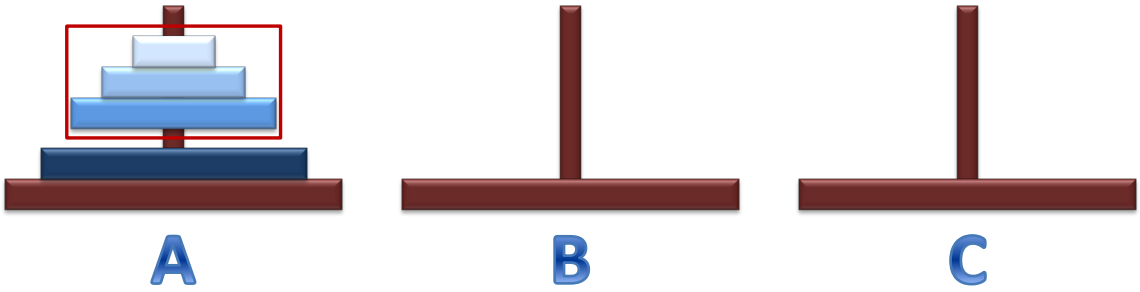

```
1.  int fact(int n) {
2.      if (n <= 1)
3.              return 1;
4.      int f = fact(n - 1);
5.      return f * n;
6.  }

7.  int main() {
8.      int a = 3;
9.      int b = fact(a);
10. }
```


Tower of Hanoi



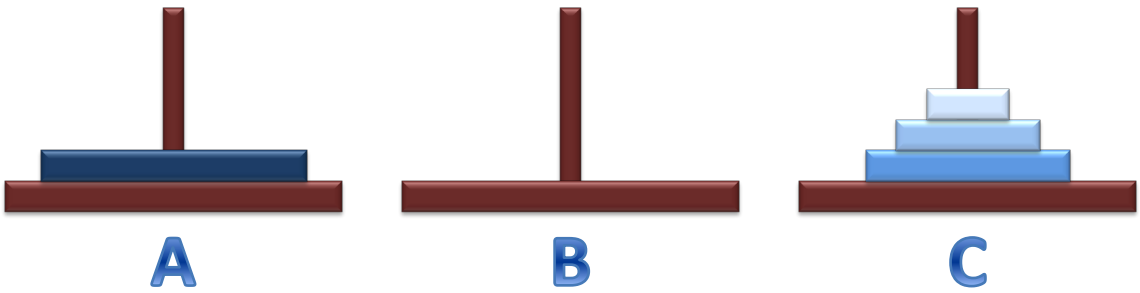
- Move the blue discs from A to B
  - Move one disc at a time
  - Never put a larger disc on a smaller one
- *Originally designed by Eduard Lucas (1883) for 8 discs*

Tower of Hanoi



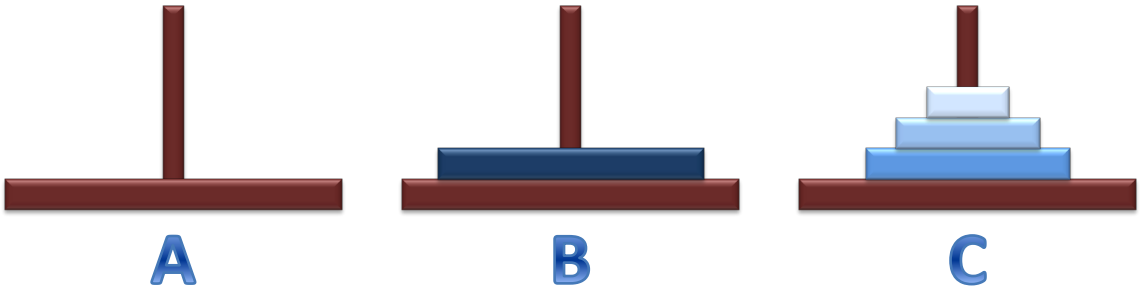
- Recursively,
  - Move the smaller n-1 discs from A to C
  - Move the largest disc from A to B
  - Move the smaller n-1 discs from C to B

Tower of Hanoi



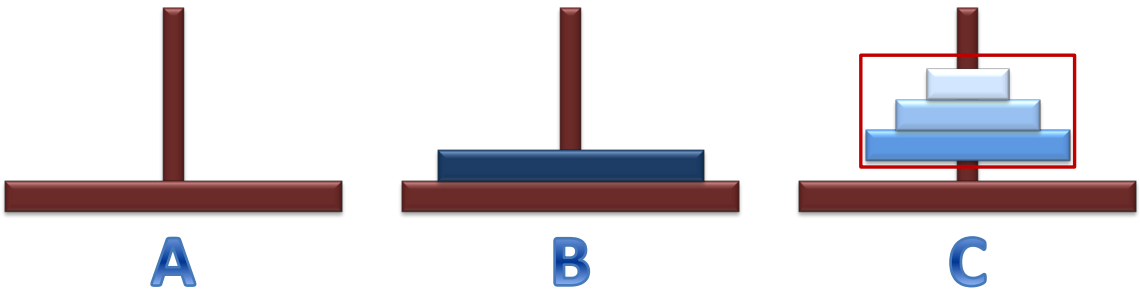
- Recursively,
  - Move the smaller n-1 discs from A to C
  - Move the largest disc from A to B
  - Move the smaller n-1 discs from C to B

Tower of Hanoi



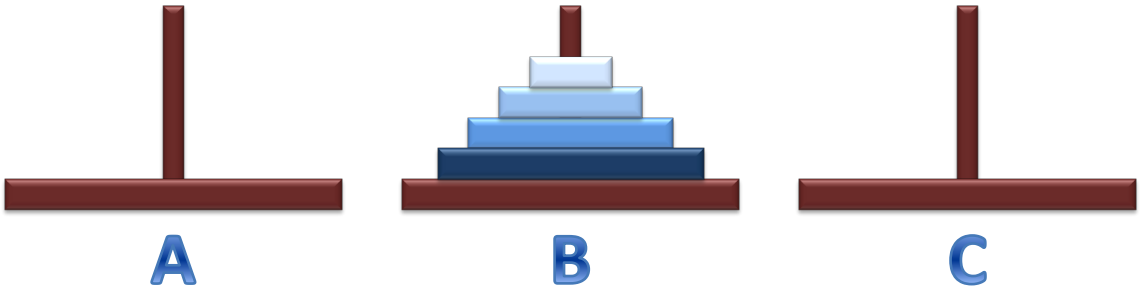
- Recursively,
  - Move the smaller n-1 discs from A to C
  - Move the largest disc from A to B
  - Move the smaller n-1 discs from C to B

Tower of Hanoi



- Recursively,
  - Move the smaller n-1 discs from A to C
  - Move the largest disc from A to B
  - Move the smaller n-1 discs from C to B

Tower of Hanoi



- Recursively,
  - Move the smaller n-1 discs from A to C
  - Move the largest disc from A to B
  - Move the smaller n-1 discs from C to B

Tower of Brahma

- God initially placed 64 golden discs
- Ordered a group of priests to move them according to the rules described before
- When they finish, the tower will crumble and the world will end

- Q: When the world will end?

Calculating Number of Moves

- $T(n)$ = number of moves required to move n discs

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

$\Rightarrow U(n) = 2\ T(n-1) + 2 = 2\ U(n-1)$

$\Rightarrow U(n) = 2^n$

$\Rightarrow T(n) = 2^n - 1$

The priests are still working!

- $2^{64}-1$ is about 18 quintillions

- If they move a disc in one microsecond(!)
  Still 5000 centries to go!

Recursive Computation on Lists

- Finding the smallest number in a list

- **min_list**([a_1, a_2, …, a_n]) =
  min(a_1, **min_list**([a_2, a_3, …, a_n]))
- **min_list**([a_1]) = a_1

Recursive Computation on Lists

- Finding the sum of the numbers of a list

- **sum_list**([a_1, a_2, …, a_n]) =
  a_1 + **sum_list**([a_2, a_3, …, a_n])
- **sum_list**([ ]) = 0

Recursive Computation on Lists

- Sort a list of numbers in ascending order

- Find the smallest number in the list
- Swap the first element with the smallest
- Sort the rest of the array

- *This method is called **selection sort***