Recursion

Recursion

- Recursion is the name given for expressing anything in terms of itself.
- Recursive function is a function which calls itself until a particular condition is met.

The factorial function

- Given a positive integer n, factorial is defined as the product of all integers between n and 1.

  i.e factorial of 4 is 4*3*2*1=24
- Hence we have the formula

n!=1	if n==0
n!=n*(n-1)*(n-2) … *1	if(n>0)

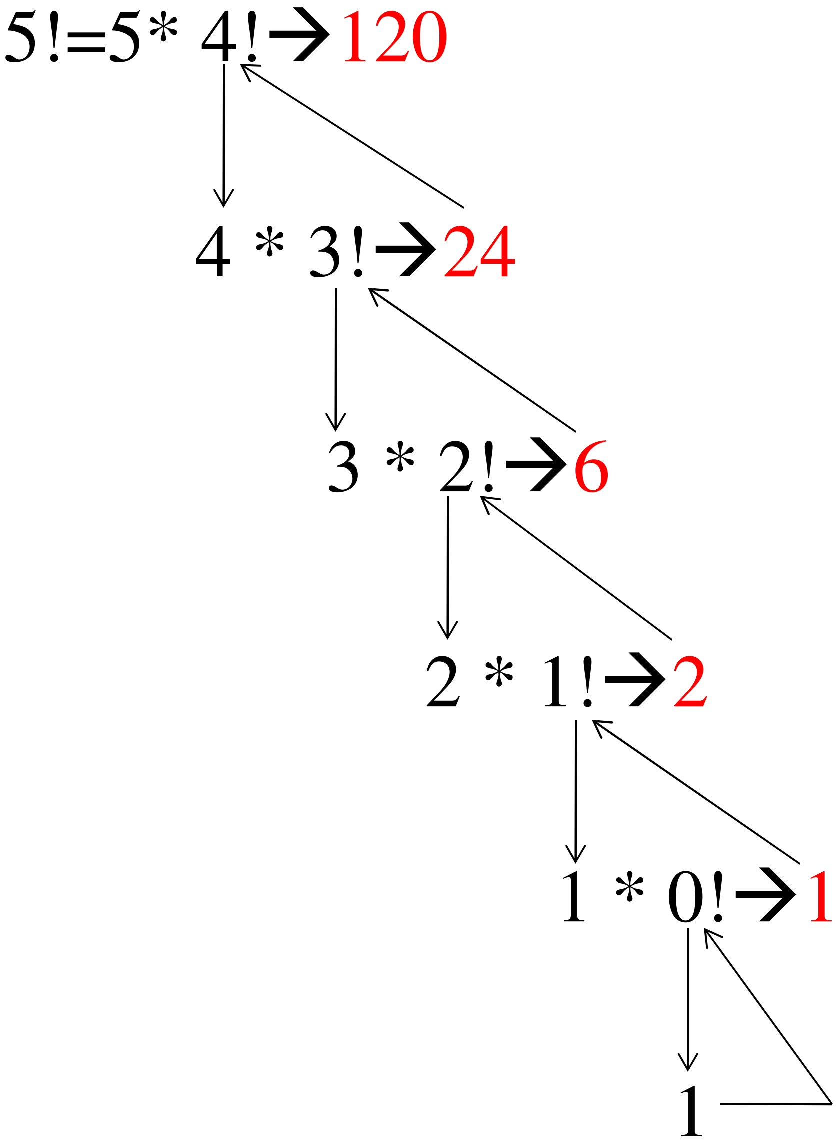
- n!=n*(n-1)!

n!=n*(n-1)*(n-2)! ….

= n*(n-1)*(n-2)*… *0! = n*(n-1)*(n-2)* …*1

- Hence this can be achieved by having a function which calls itself until 0 is reached. This is recursive function for factorial.

Ex:



5!=5* 4!→120

4 * 3!→24

3 * 2!→6

2 * 1!→2

1 * 0!→1

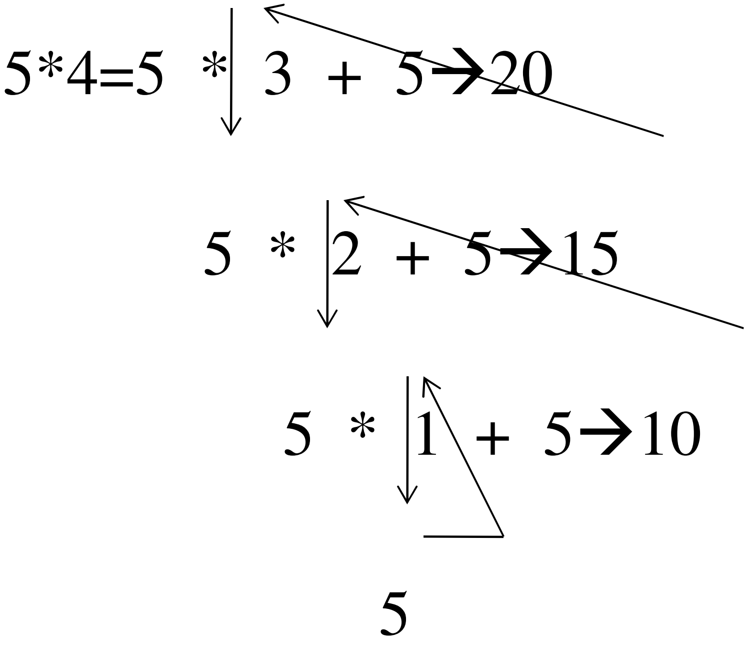
1

Multiplication of natural numbers

- Another example of recursive function.
- The product a*b, where a and b are positive integers is defined as a added to itself b times, which is a iterative definition.
- Recursive definition:

a*b=a if b==1

a*b=a*(b-1)+a if b>1



5*4=5 * 3 + 5→20

5 * 2 + 5→15

5 * 1 + 5→10

5

Fibonacci sequence

0,1,1,2,3,5,8,…..

- Each element is the sum of two preceding elements.
- Fibonacci of a number is nothing but the value at that position in sequence.

  i.e fib(0)==0

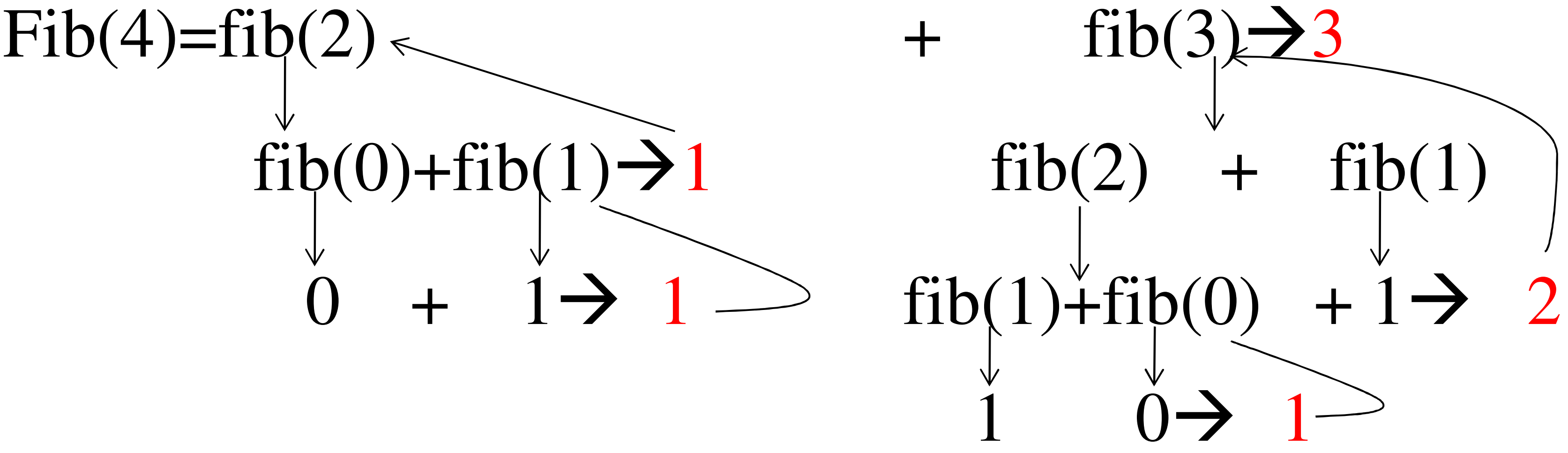
  fib(1)==1

  fib(2)==1

  fib(3)==2 and so on
- Fibonacci is defined in formula as

  fib(n)= n if n==0 or n==1

  fib(n)= fib(n-2) + fib(n-1) for n>=2



Fib(4)=fib(2) + fib(3)→3

fib(0)+fib(1)→1 fib(2) + fib(1)

0 + 1→ 1 fib(1)+fib(0) + 1→ 2

1 0→ 1

Binary search

- Binary search is an efficient method of search.

  1. element is compared with the middle element in the array. If the middle element is the element to be searched, search is successful.

  2. if element is less than the middle element, then searching is restricted to the first half.

  3. if element is greater than the middle element, then searching is restricted to the second half.

  4. this process is continued until the element is found or not found.

1	2	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8

Let the element to be searched is 2

Middle element is 5 and 2 is less than 5. hence first half is considered, which is

1	2	3	4
0	1	2	3

middle element is 2 and hence search is successful and element is found at position 1

1	2	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8

Let the element to be searched is 10

6	7	8	9
5	6	7	8

8	9
7	8

10 is not found.

Properties of recursive algorithms

- Recursive algorithm should terminate at some point, otherwise recursion will never end
- Hence recursive algorithm should have stopping condition to terminate (base case) along with recursive calls (general case).

  for ex: in factorial stopping condition is n!=1 if n==0

  In multiplication of 2 numbers, it is a*b=a if b==1

  In Fibonacci it is fib(0)=0 and fib(1)=1

` In binary search it is low > high

Factorial in C

```
int fact(int n)
{
    int x, y, res;
    if(n==0)
          return 1;
    else
    {
          x=n-1;
          y=fact(x);
          res=n*y;
          return res;
    }
}
```

Here y=fact(x), function gets called by itself each time with 1 less number than previous one until number gets zero.

Factorial in C

```
int fact(int n)
{
    //int x, y, res;
    if(n==0)
            return 1;
    else
    {       return n*fact(n-1);
/*          x=n-1;
            y=fact(x);
            res=n*y;
            return res; */
    }
}
```

Here y=fact(x), function gets called by itself each time with 1 less number than previous one until number gets zero.

Control flow in evaluating fact(4)

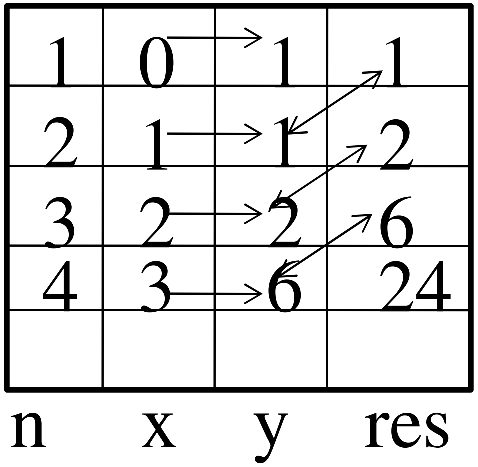
n	x	y	res
4			

n	x	y	res
4	3		
4			

n	x	y	res
3	2		
4	3		
4			

n	x	y	res
2	1		
3	2		
4	3		
4			

n	x	y	res
1	0		
2	1		
3	2		
4	3		
4			



n	x	y	res
1	0	1	1
2	1	1	2
3	2	2	6
4	3	6	24

Recursive program for multiplying 2 numbers

```
int mul(int m, int n) {
    int y;
    if(m==0 || n==0)
        return 0;
    if(n==1)
        return m;
    else{
        y=mul(m, n-1);
        return(y+m);
    }
}
```

Recursive program to find the nth fibonacci number

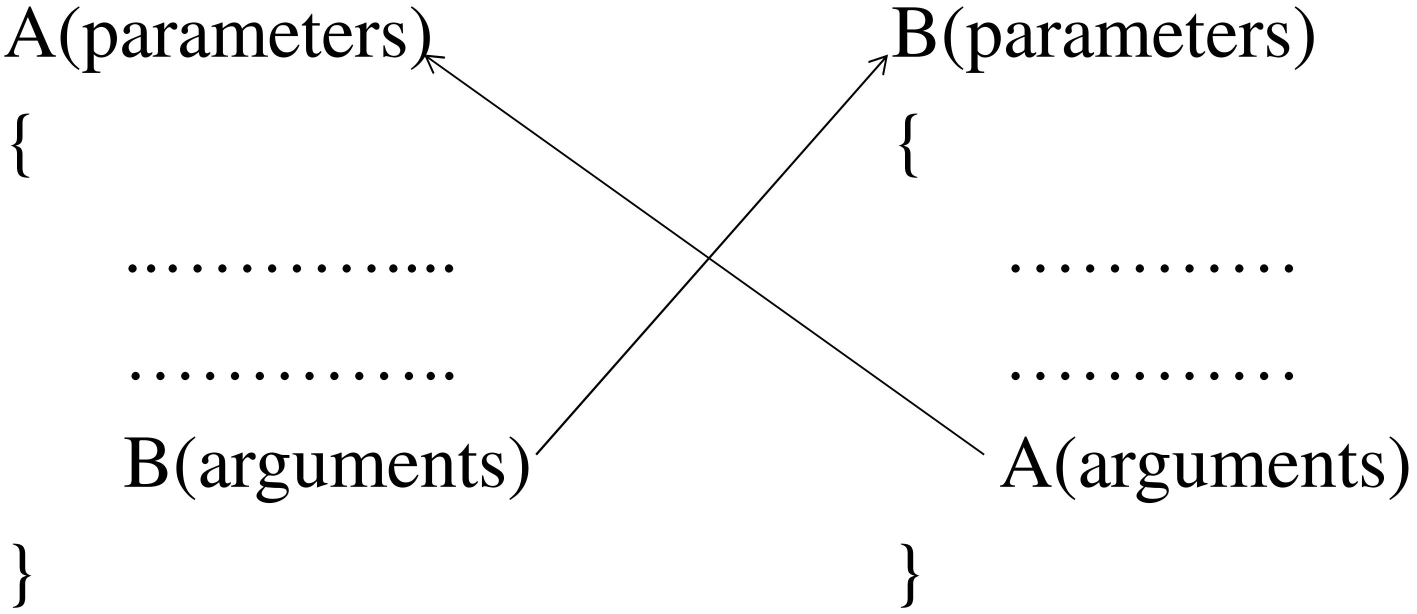
```
int fib(int n)
{
   if(n==0)
          return 0;
   if(n==1)
          return 1;
   else
          return (fib(n-1) + fib(n-2));

}
```

Recursive program to do a binary search

```
int binary(int item, int a[], int low, int high)
{
    int mid;
    if(low > high)
            return -1;
    mid=(low+high)/2;
    if(item==a[mid])
            return mid;
    else if(item<a[mid])
    {
            high=mid-1;
            return binary(item, a, low, high);
    }
    else
    {
            low=mid+1;
            return binary(item, a, low, high);
    }
}
```

Recursive chains

- Recursive function need not call itself directly. It can call itself indirectly as shown

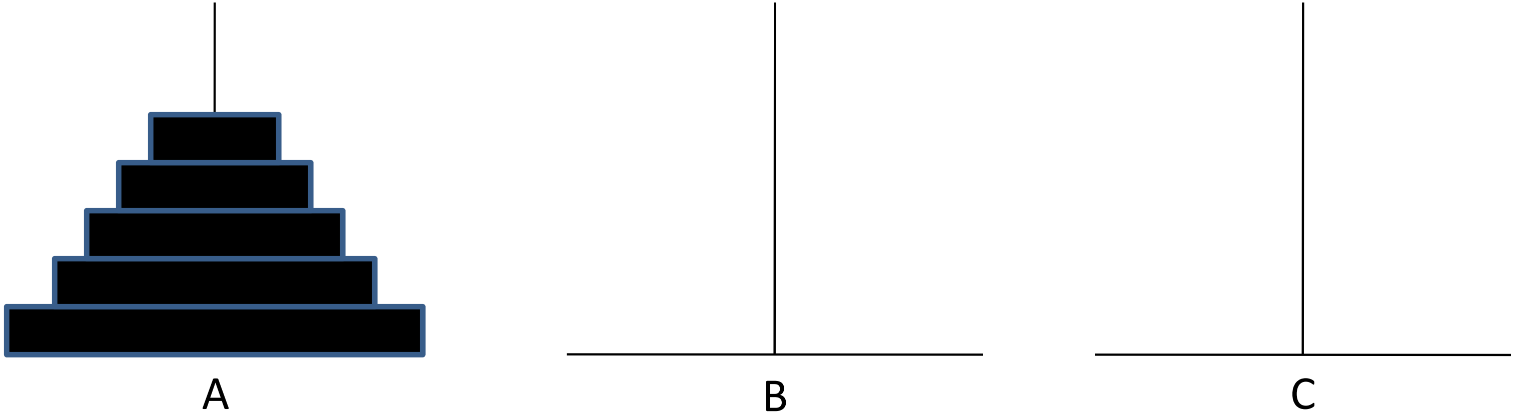


```
A(parameters)
{
      …………....
      …………..
      B(arguments)
}
```

```
B(parameters)
{
      …………
      …………
      A(arguments)
}
```

Towers of Hanoi problem

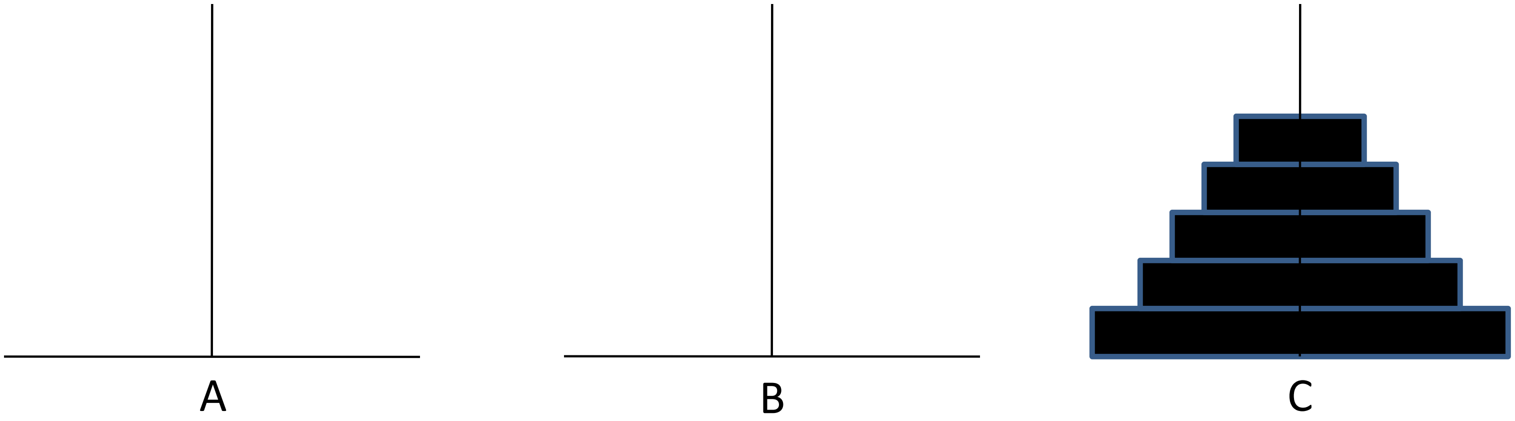
Initial setup



- There are 3 pegs A, B, and C and five disks of different diameters placed on peg A so that a larger disk is always below a smaller disk.
- The aim is to move five disks to peg C using peg B as auxiliary. Only the top disk on any peg may be moved to another peg, and a larger disk may never rest on a smaller one.

https://yongdanielliang.github.io/animation/web/TowerOfHanoi.html

After passing all the 5 disks to peg C:



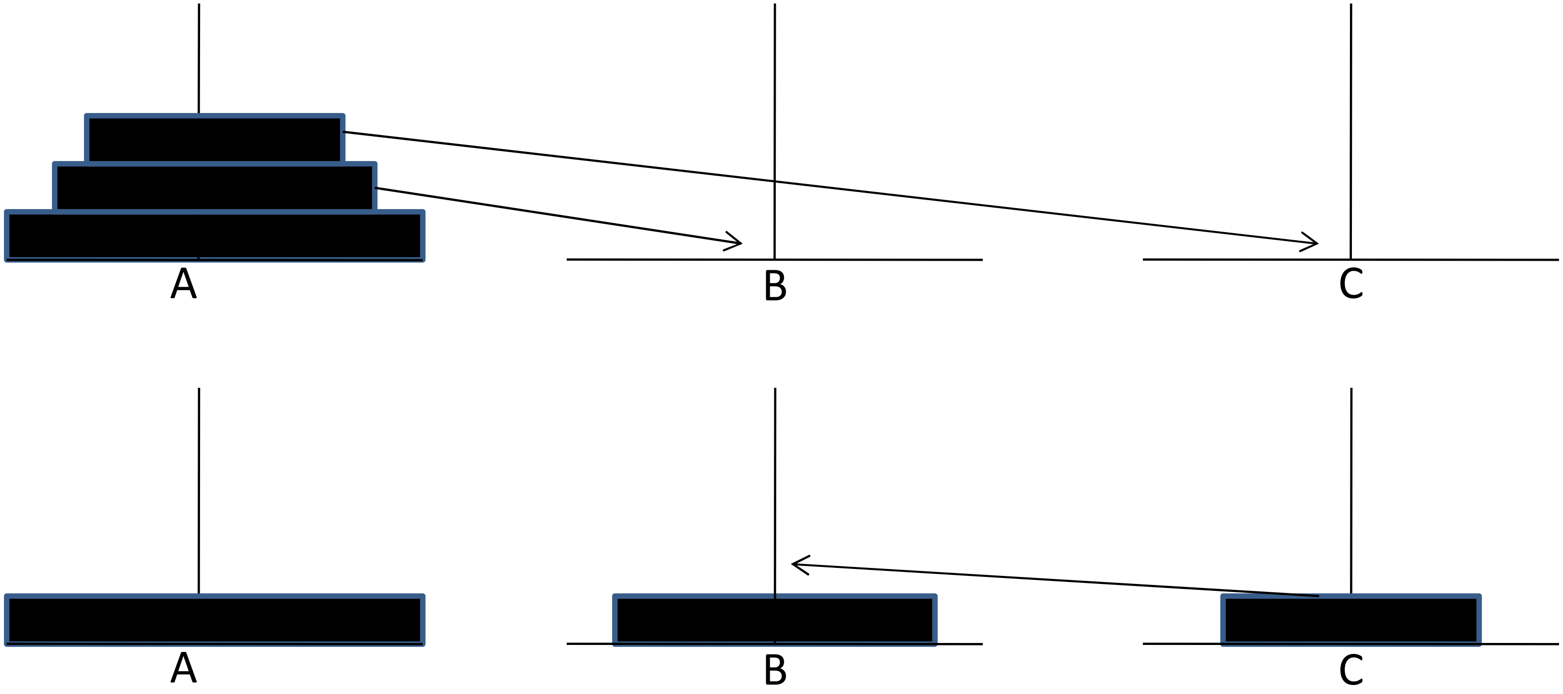
Lets consider the general case of n disks

To move n disks from A to C using B as auxiliary

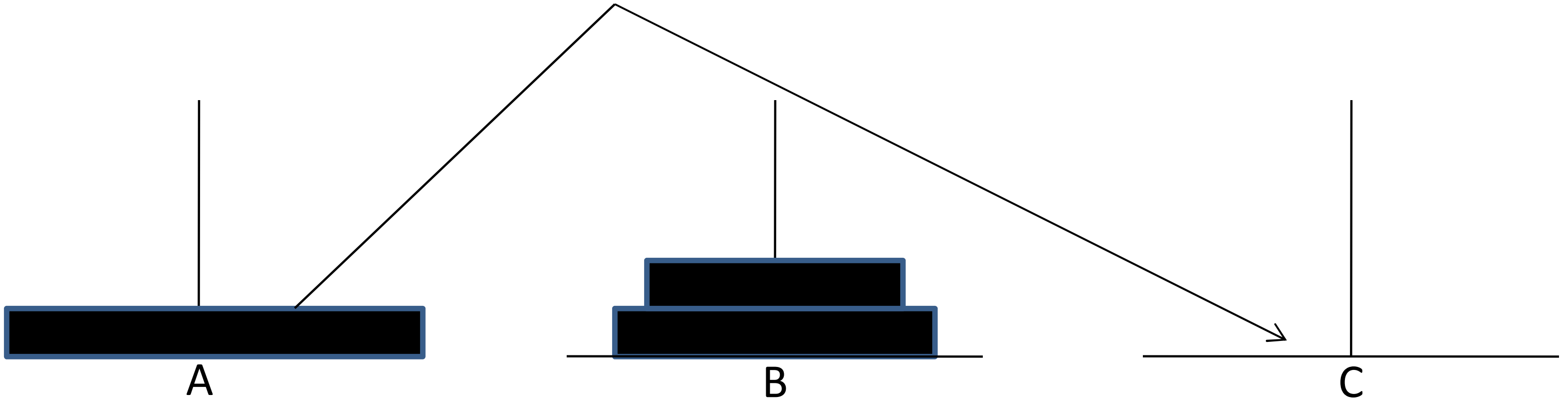
1. If n==1, move single disk from A to C.
2. Move the top n-1 disks from A to B using C as auxiliary.
3. Move the remaining disk from A to C.
4. Move the n-1 disks from B to C, using A as auxiliary.

- Here if n==1, step1 will produce a correct solution.
- If n==2, we know that we already have a solution for n-1, i.e., 1, so steps 2 and 4 can be performed.
- If n==3, we know that we have a solution for n-1, i.e., 2, so steps 2 and 4 can be performed.
- In this way we have solutions for 1,2,3…..up to any value.
- This clearly indicates the concept of recursion involved and hence this problem can be solved by recursion.

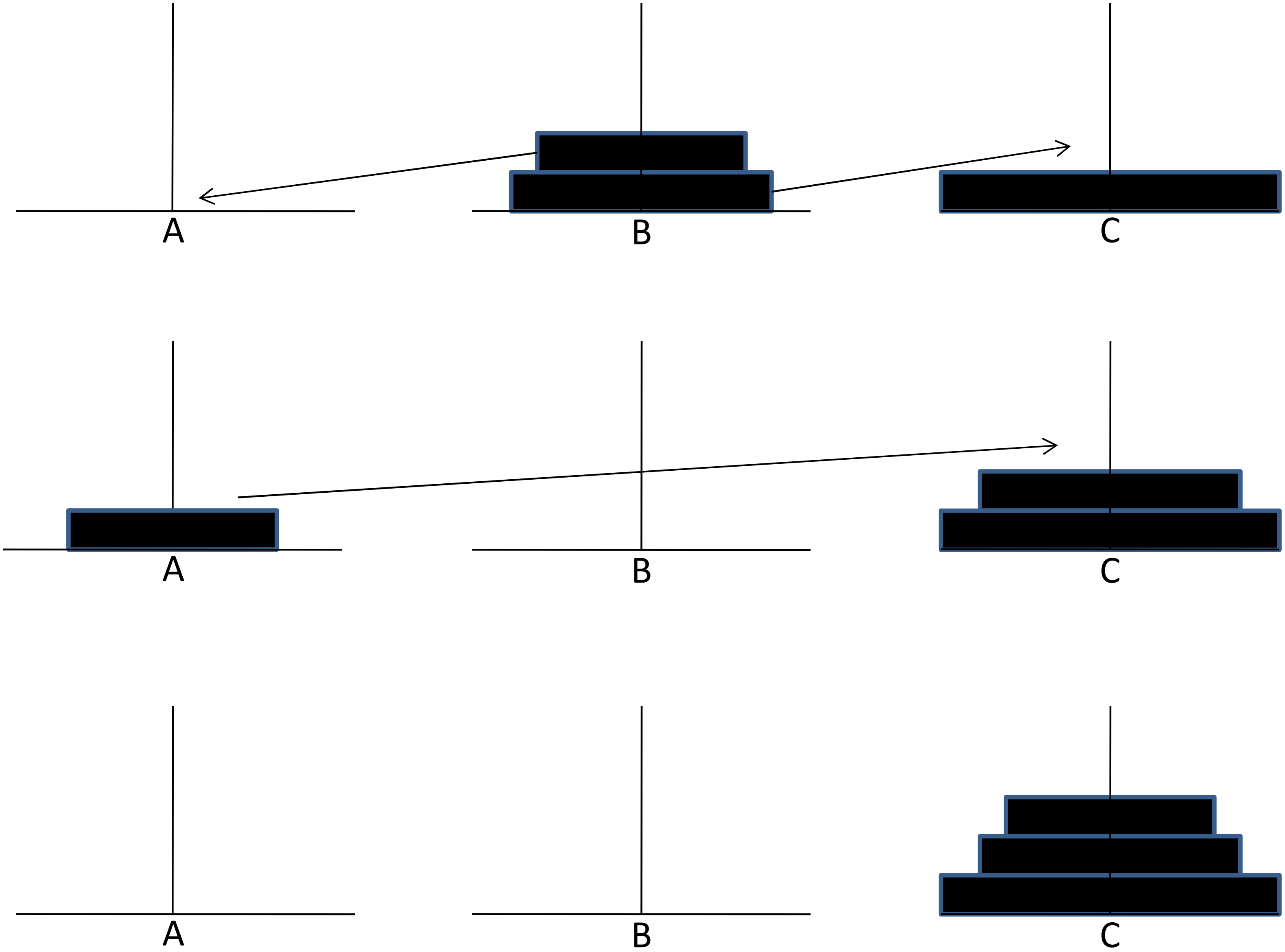
n==3. moving n-1 disks from A to B using C as auxiliary



Moving remaining 1 disk from A to C



Moving n-1 disks from B to C using A as auxiliary



C program for tower of hanoi problem

```
void tower (int n, char source, char temp, char destination) {
    if(n==1) {
        cout<<“move disk 1 from “<<source<<“ to “<<destination<<endl;
        return;
    }

    /*moving n-1 disks from A to B using C as auxiliary*/
    tower(n-1, source, destination, temp);

    cout<<“move disk “<<n<<“ from “<<source<<“ to
    “<<destination<<endl;

    /*moving n-1 disks from B to C using A as auxiliary*/
    tower(n-1, temp, source, destination);
}
```

Length of a string using recursion

```
int StrLen(char str[], int index)
{
   if (str[index] == '\0') return 0;
   return (1 + StrLen(str, index + 1));
}
```

```
int len;
len = Strlen("Thursday", 0);
```

Length of a string using recursion using static variable

```
int StrLen(char *str)
{
  static int length=0;
  if(*str != ‘\0’)
  {
    length++;
    StrLen(++str);
  }
  return length;
 }
```

To Check whether a given String is Palindrome or not using Recursion

```
int isPalindrome(char *inputString, int leftIndex, int rightIndex) {
        /* Recursion termination condition */
    if(leftIndex >= rightIndex) return 1;
    if(inputString[leftIndex] == inputString[rightIndex]){
        return isPalindrome(inputString, leftIndex + 1, rightIndex -
    1);
    }
    return 0;
}
```

```
int main(){
    char inputString[100];
    printf("Enter a string for palindrome check\n");
    scanf("%s", inputString);
    if(isPalindrome(inputString, 0, strlen(inputString) - 1))
              printf("%s is a Palindrome \n", inputString);
    else
        printf("%s is not a Palindrome \n", inputString);
    getch();
    return 0;
}
```

To Copy One String to another using Recursion

```
void copy(char str1[], char str2[], int index)
{
    str2[index] = str1[index];
    if (str1[index] == '\0') return;
    copy(str1, str2, index + 1);
}
```

already have str1 = " Today"   str2 is empty

copy(str1, str2, 0);

Advantages of Recursion

1. Clearer and simpler versions of algorithms can be created using recursion.
2. Recursive definition of a problem can be easily translated into a recursive function.
3. Lot of bookkeeping activities such as initialization etc required in iterative solution is avoided.

Disadvantages

1. When a function is called, the function saves formal parameters, local variables and return address and hence consumes a lot of memory.
2. Lot of time is spent in pushing and popping and hence consumes more time to compute result.

Iteration	Recursion
Uses loops	uses if-else and repetitive function calls
Counter controlled and body of loop terminates when the termination condition fails.	Terminates when base condition is reached.
Execution is faster and takes less space.	Consumes time and space because of push and pop.
Difficult to design for some problems.	Best suited for some problems and easy to design.