Example Algorithms

Algorithm Specification

```
Algorithm Max ( a, n )
// a is an array of size n
{
   result:=a[1];
   for  i:= 2 to n do
   {
      if(a[i]>result)then result:= a[i];
   }
   return result;
}
```

Image of a number

Iter	Result	num
Initialize	0	123
1	0*10+123%10=3	12
2	3*10+12%10=32	1
3	32*10+1%10=321	0

		0

		3

	3	2

3	2	1

Algorithm Specification

```
Algorithm Image (  num )
// returns the image of num
{
   result:=0;
   while(num ≠ 0)
   {
      result:=result*10+num%10;
      num:= num/10;
   }
   return result;
}
```

Algorithm Specification

```
Algorithm len (  num )
// returns the length of num
{
   count:=0;
   while(num ≠ 0)
   {
      num:= num/10;
      count:=count+1;
   }
   return result;
}
```

Algorithm Specification

```
Algorithm Sumd (  num )
// returns the image of num
{
   sum:=0;
   while(num ≠ 0)
   {
      sum:=sum+num%10;
      num:= num/10;
   }
   return sum;
}
```

Algorithm Specification

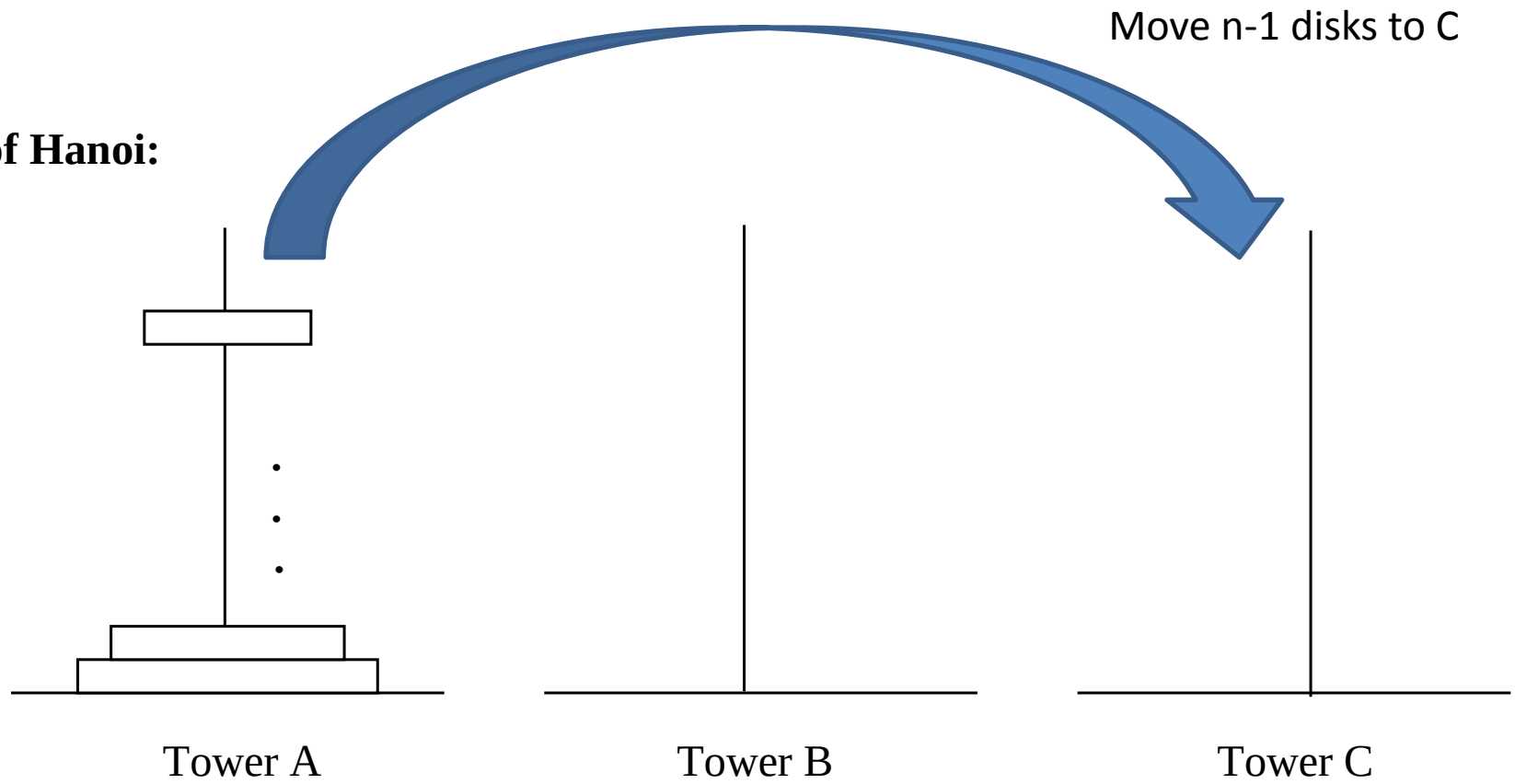
```
Algorithm SelectionSort ( a, n )
//Sort a[1:n] into nondecreasing order.
{
   for i :=1 to n do
   {
      j := i;
      for k := i+1 to n do
         if ( a[k] < a[j] )then j:=k;
      t:=a[i];a[i]:=a[j];a[j]:=t;
   }
 }
```

Algorithm Specification

- An algorithm is said to be recursive if the same algorithm is invoked in the body.
- An algorithm that calls itself is Direct Recursive.
- Algorithm ‘A’ is said to be Indirect Recursive if it calls another algorithm which in turns calls ‘A’.
- The Recursive mechanism, are externally powerful, but even more importantly, many times they can express an otherwise complex process very clearly.

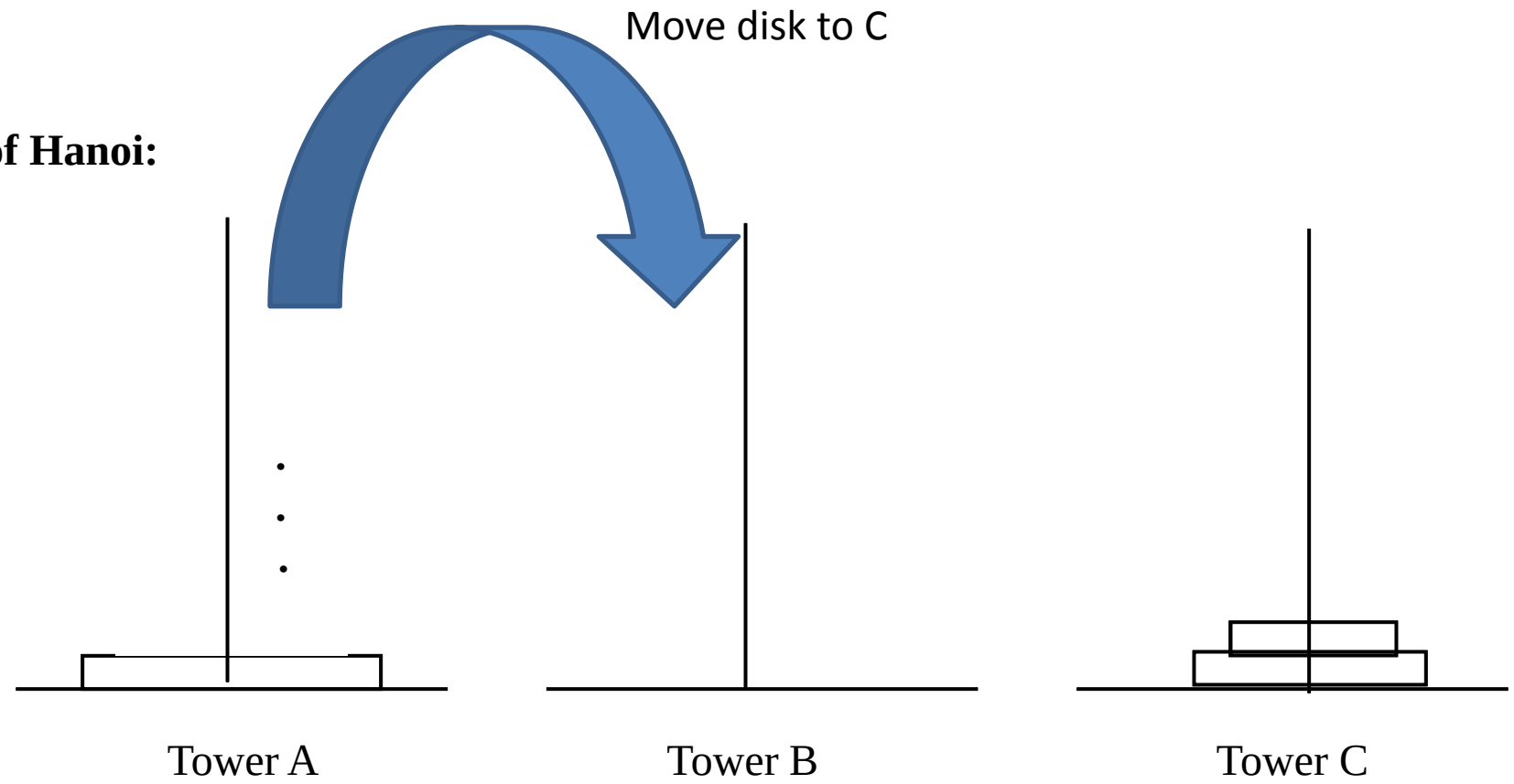
Algorithm Specification

Towers of Hanoi:



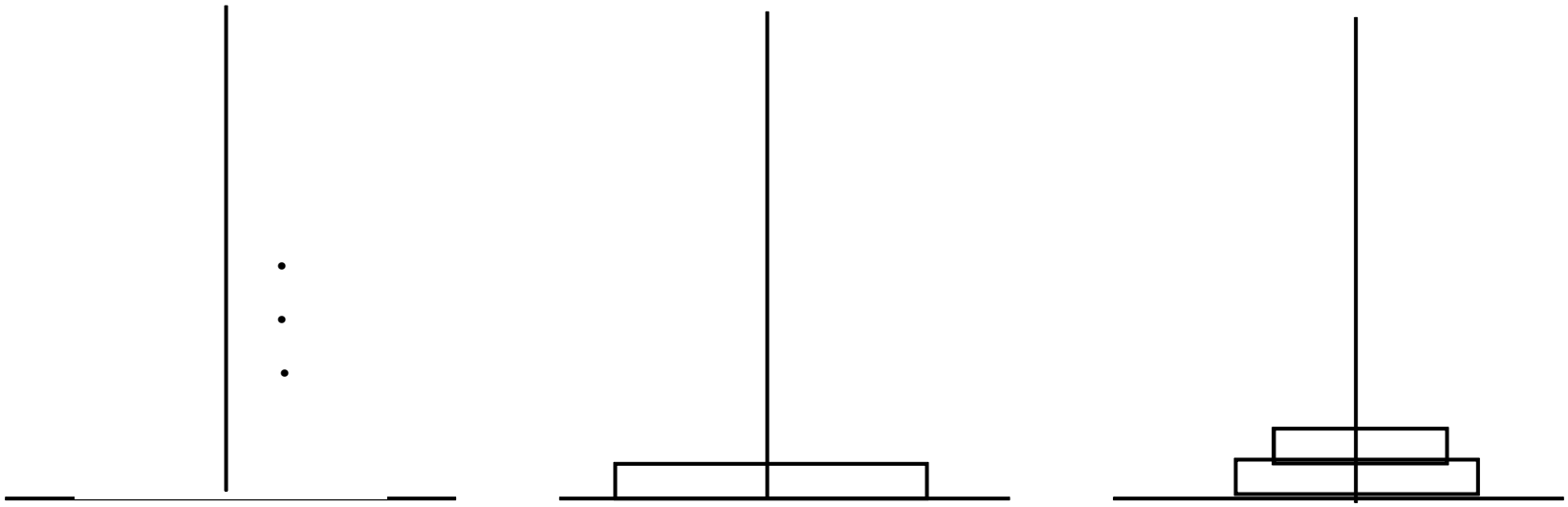
Algorithm Specification

Towers of Hanoi:



Algorithm Specification

Towers of Hanoi:



Algorithm Specification

```
Algorithm ToH ( n, x, y, z)
//Move top ‘n’ disks from tower x to
tower y.
{
   if( n ≥ 1 ) then
   {
      ToH ( n-1, x, z, y );
      write( "move from" x "to" y );
      ToH ( n-1, z, y, x );
   }
}
```

Algorithm Specification

```
if( n ≥ 1 ) then
  {
    ToH ( n-1, x, z, y );
    write( "move from" x "to" y );
    ToH ( n-1, z, y, x );
  }
```

ToH(3,1,2,3)

Algorithm Specification

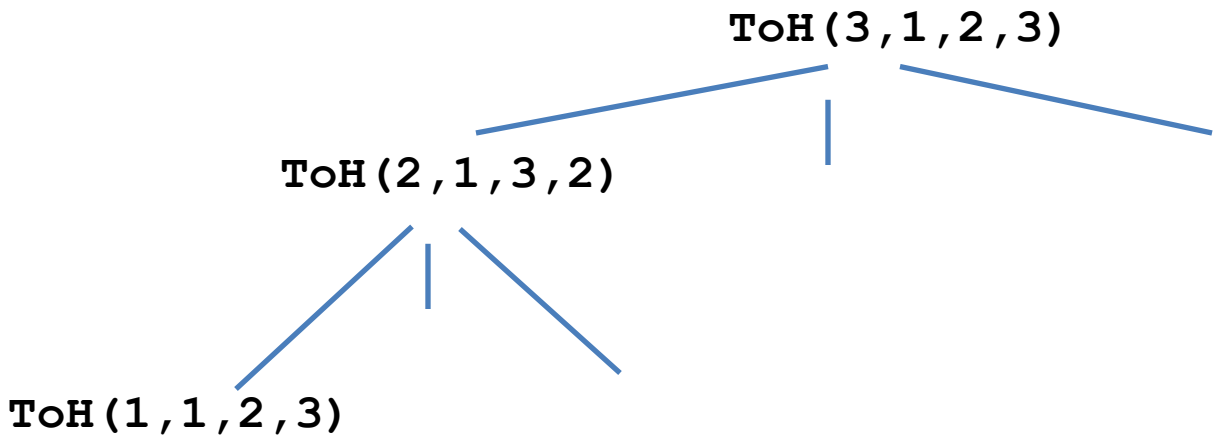
```
if( n ≥ 1 ) then
  {
    ToH ( n-1, x, z, y );
    write( "move from" x "to" y );
    ToH ( n-1, z, y, x );
  }
```

ToH(3,1,2,3)

ToH(2,1,3,2)

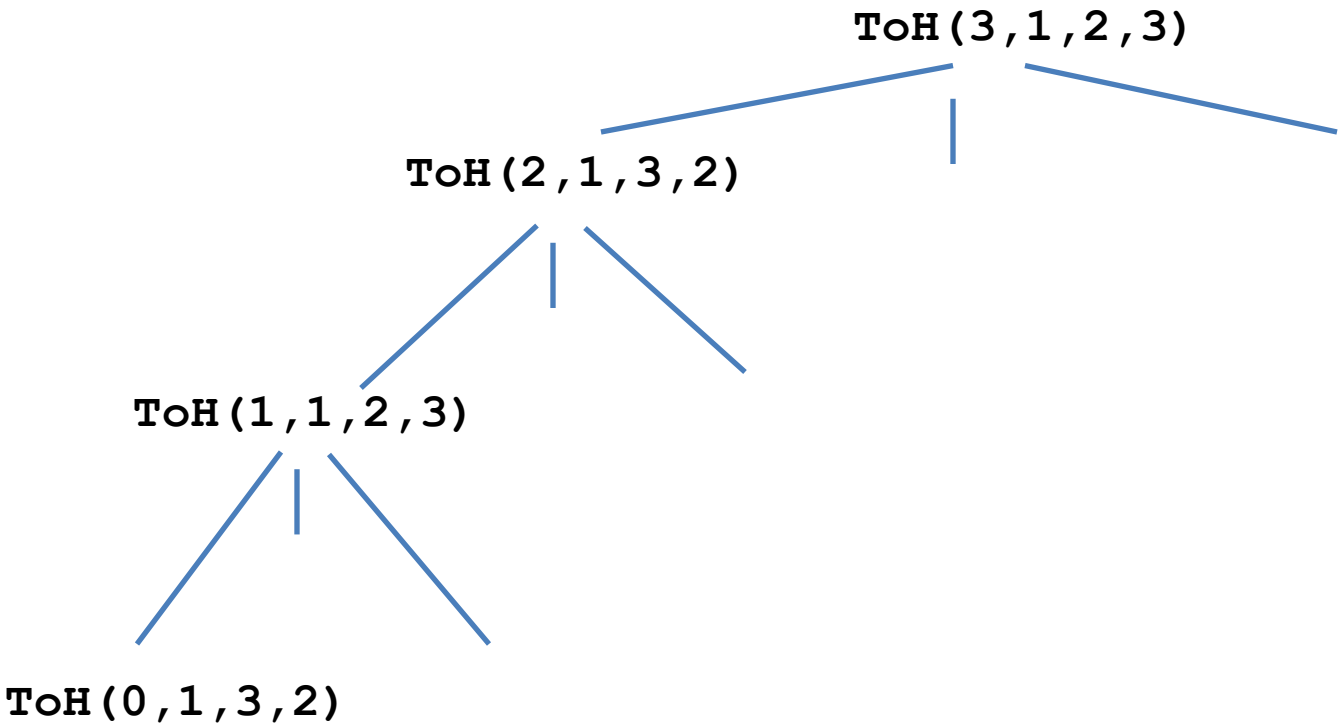
Algorithm Specification

```
if( n ≥ 1 ) then
  {
    ToH ( n-1, x, z, y );
    write( "move from" x "to" y );
    ToH ( n-1, z, y, x );
  }
```



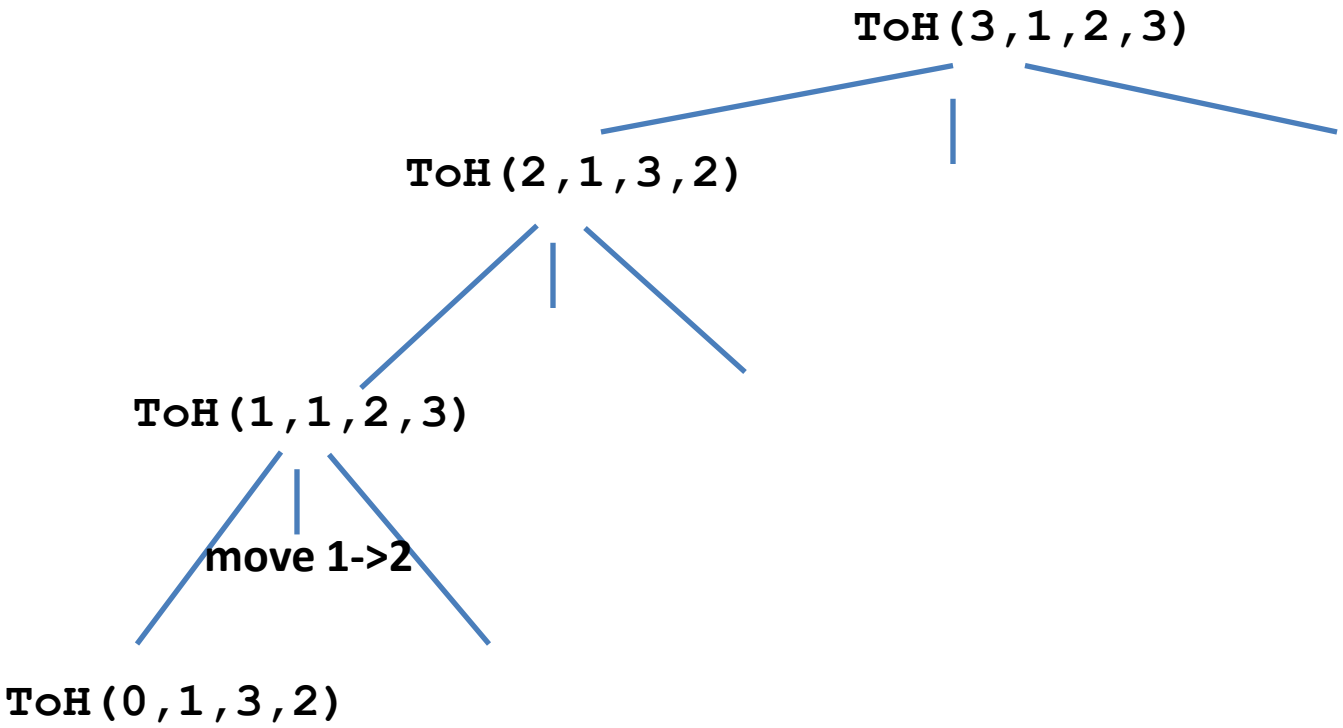
Algorithm Specification

```
if( n ≥ 1 ) then
  {
    ToH ( n-1, x, z, y );
    write( "move from" x "to" y );
    ToH ( n-1, z, y, x );
  }
```



Algorithm Specification

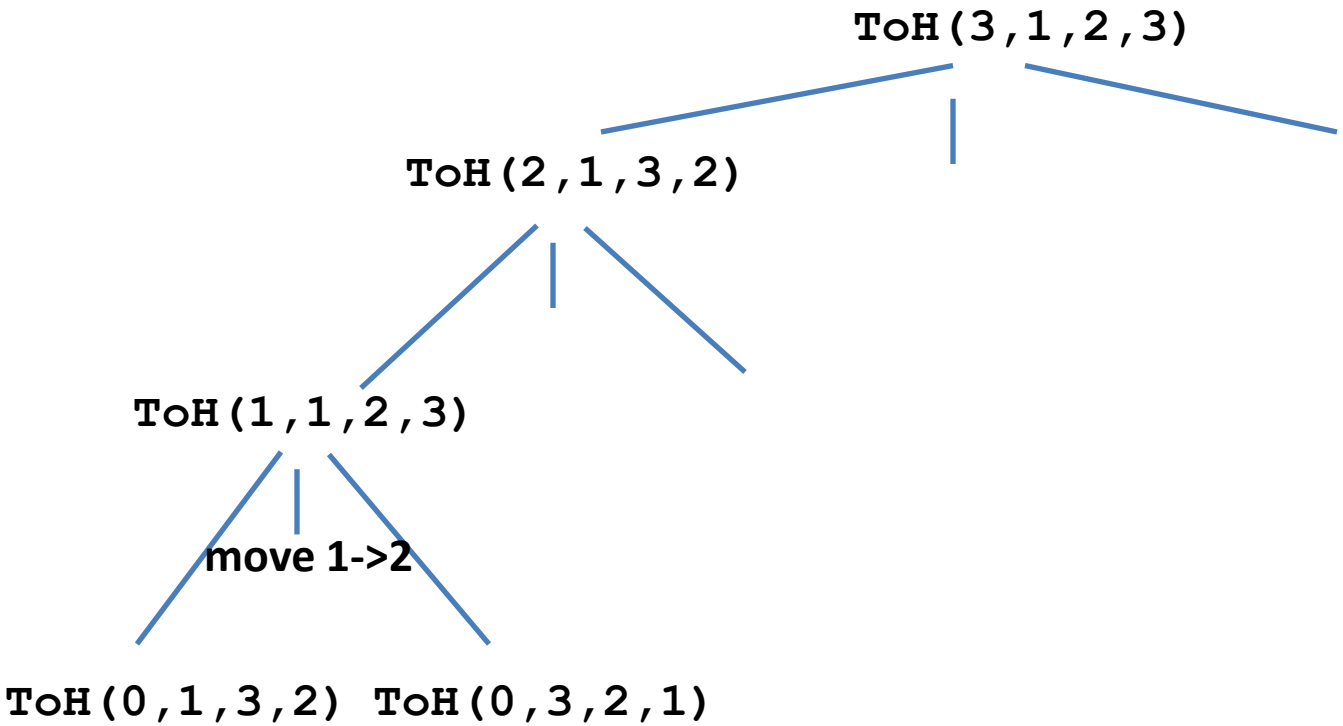
```
if( n ≥ 1 ) then
  {
    ToH ( n-1, x, z, y );
    write( "move from" x "to" y );
    ToH ( n-1, z, y, x );
  }
```




```
if( n ≥ 1 ) then
  {
    ToH ( n-1, x, z, y );
    write( "move from" x "to" y );
    ToH ( n-1, z, y, x );
  }
```

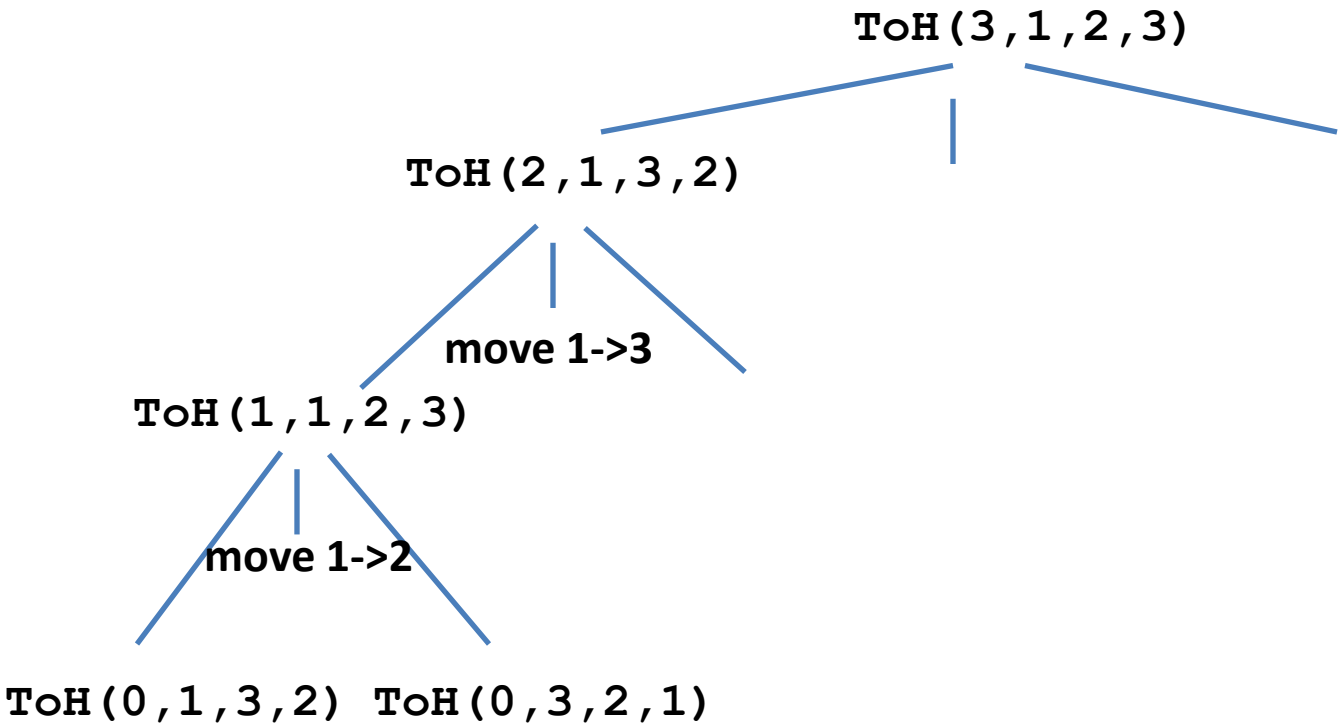
Algorithm Specification

```
if( n ≥ 1 ) then
  {
    ToH ( n-1, x, z, y );
    write( "move from" x "to" y );
    ToH ( n-1, z, y, x );
  }
```



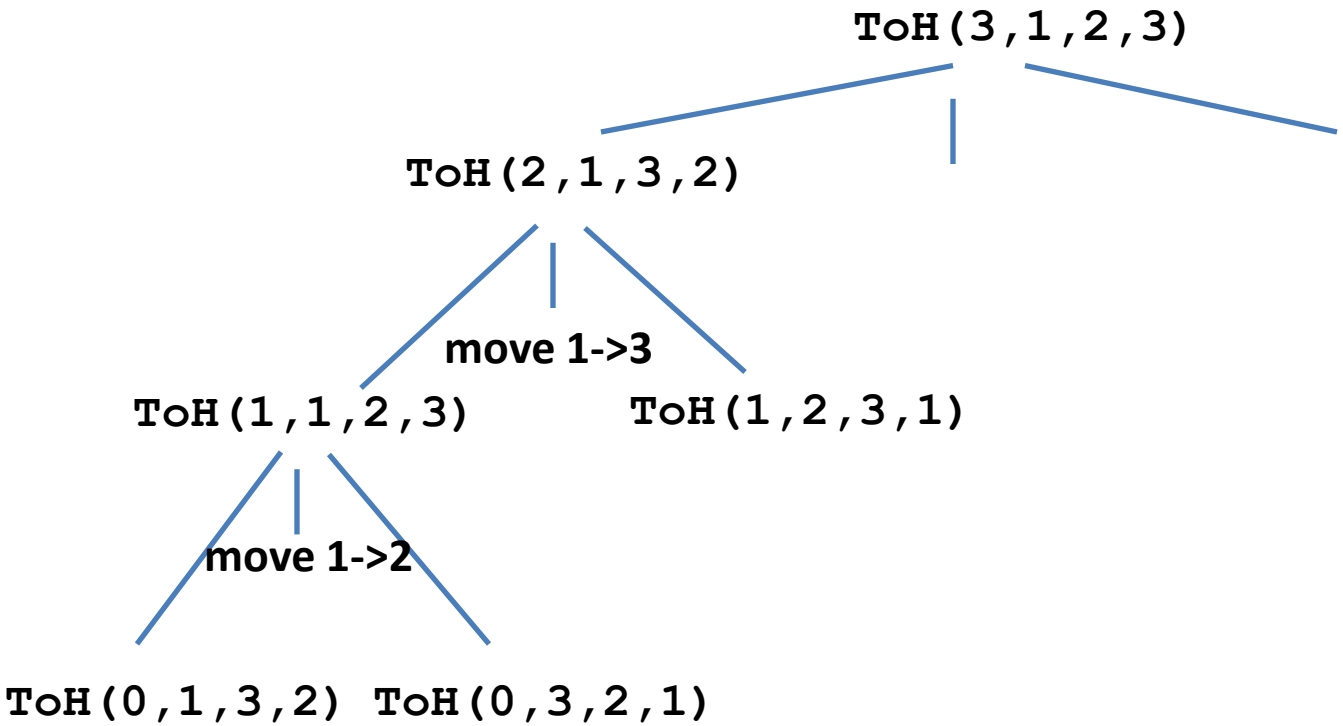
Algorithm Specification

```
if( n ≥ 1 ) then
  {
    ToH ( n-1, x, z, y );
    write( "move from" x "to" y );
    ToH ( n-1, z, y, x );
  }
```



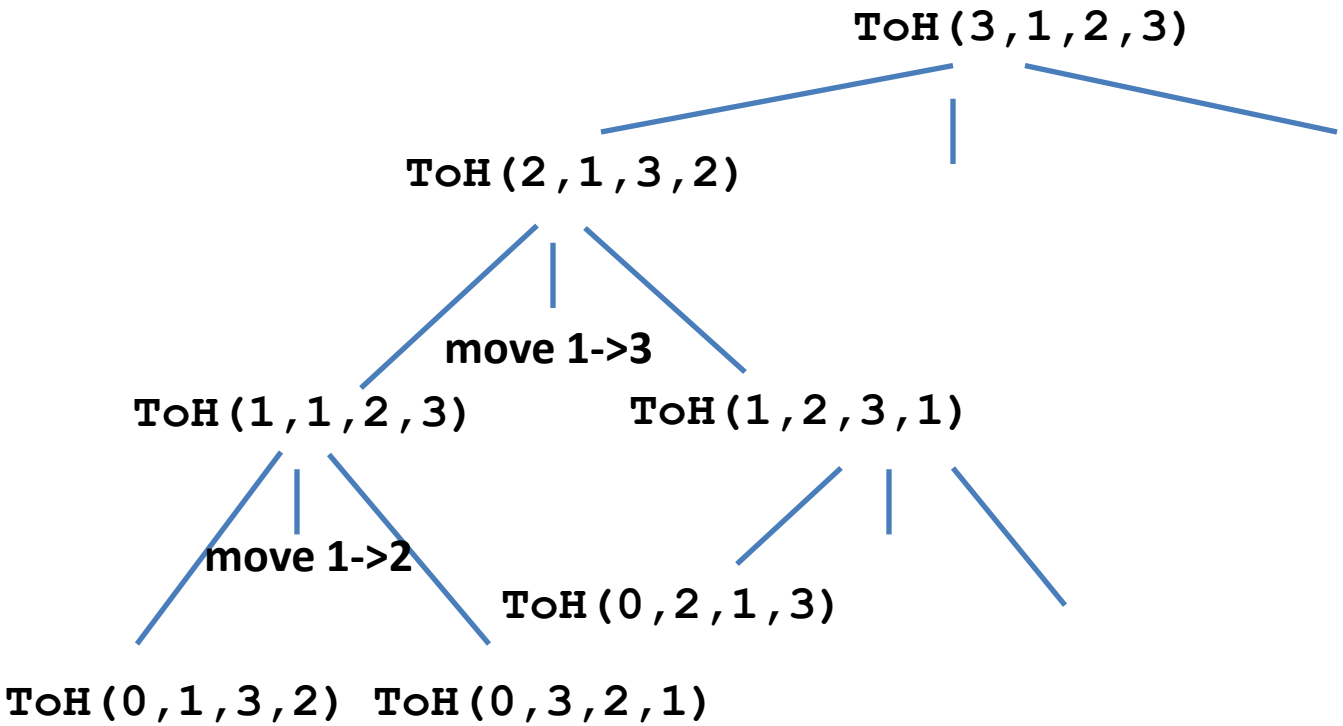
Algorithm Specification

```
if( n ≥ 1 ) then
  {
    ToH ( n-1, x, z, y );
    write( "move from" x "to" y );
    ToH ( n-1, z, y, x );
  }
```



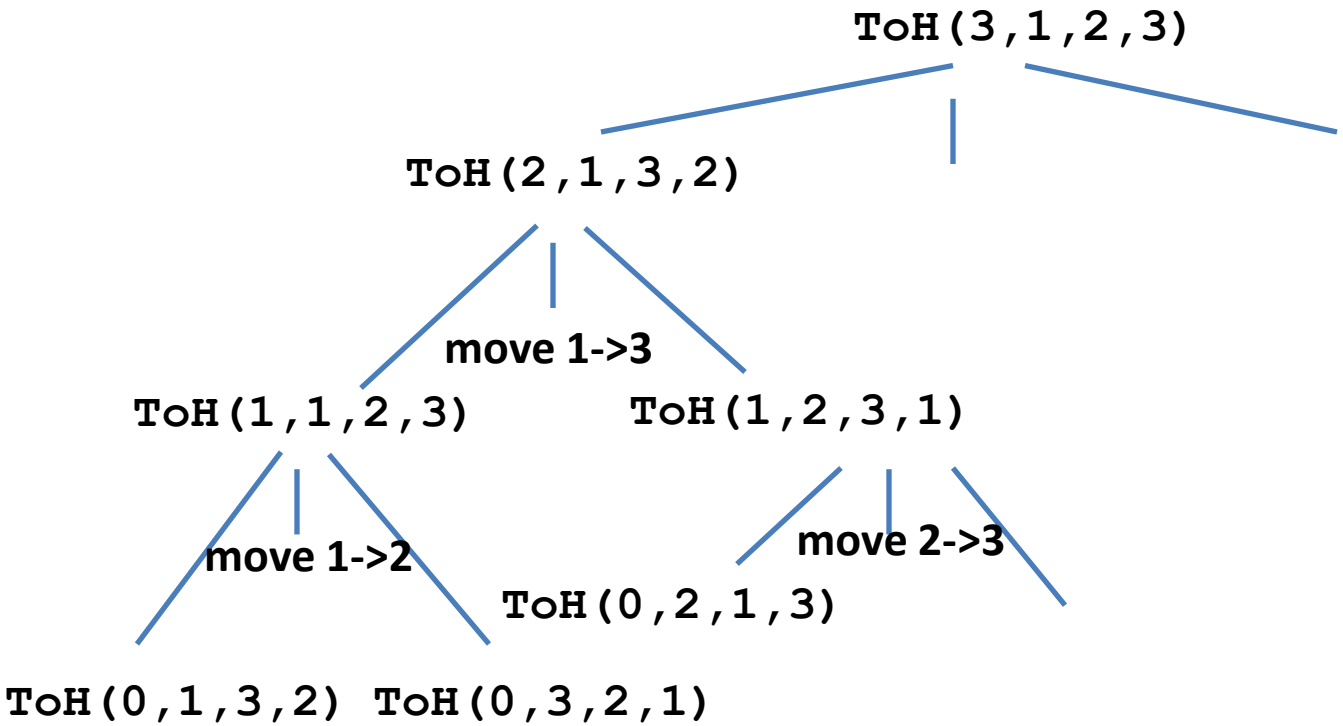
Algorithm Specification

```
if( n ≥ 1 ) then
  {
    ToH ( n-1, x, z, y );
    write( "move from" x "to" y );
    ToH ( n-1, z, y, x );
  }
```



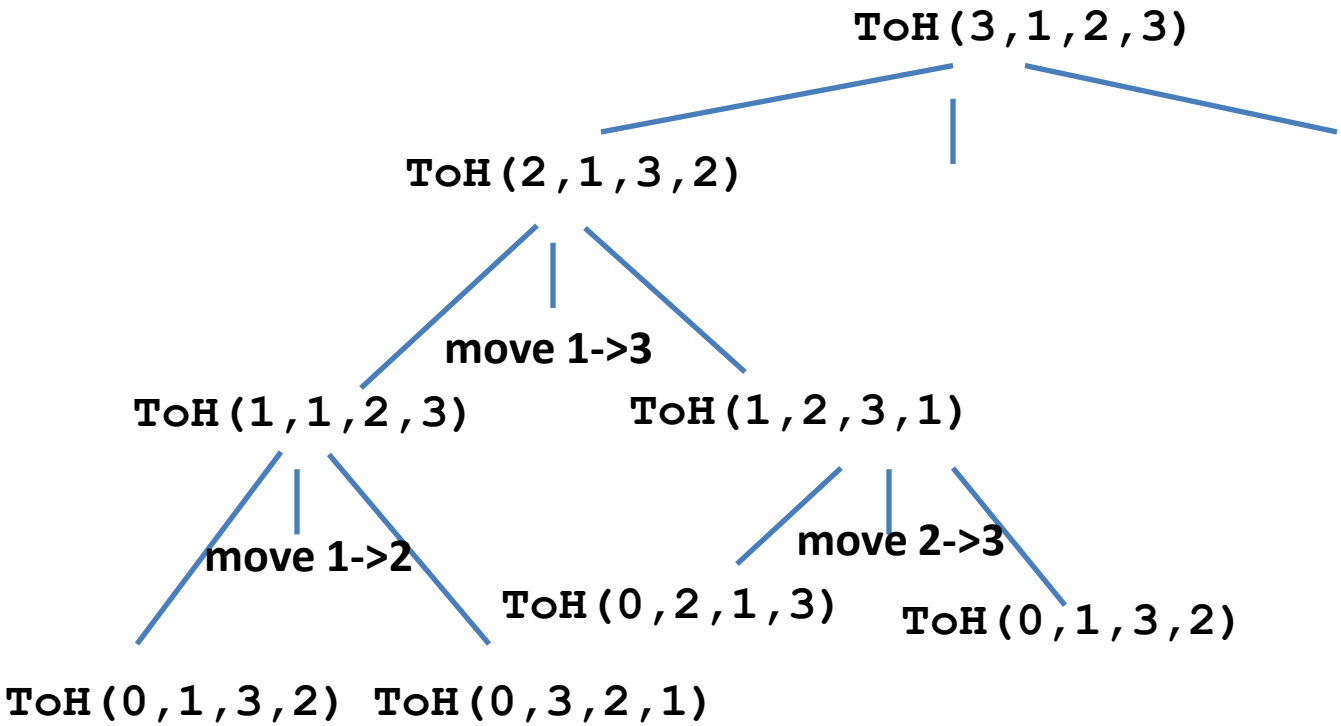
Algorithm Specification

```
if( n ≥ 1 ) then
  {
    ToH ( n-1, x, z, y );
    write( "move from" x "to" y );
    ToH ( n-1, z, y, x );
  }
```



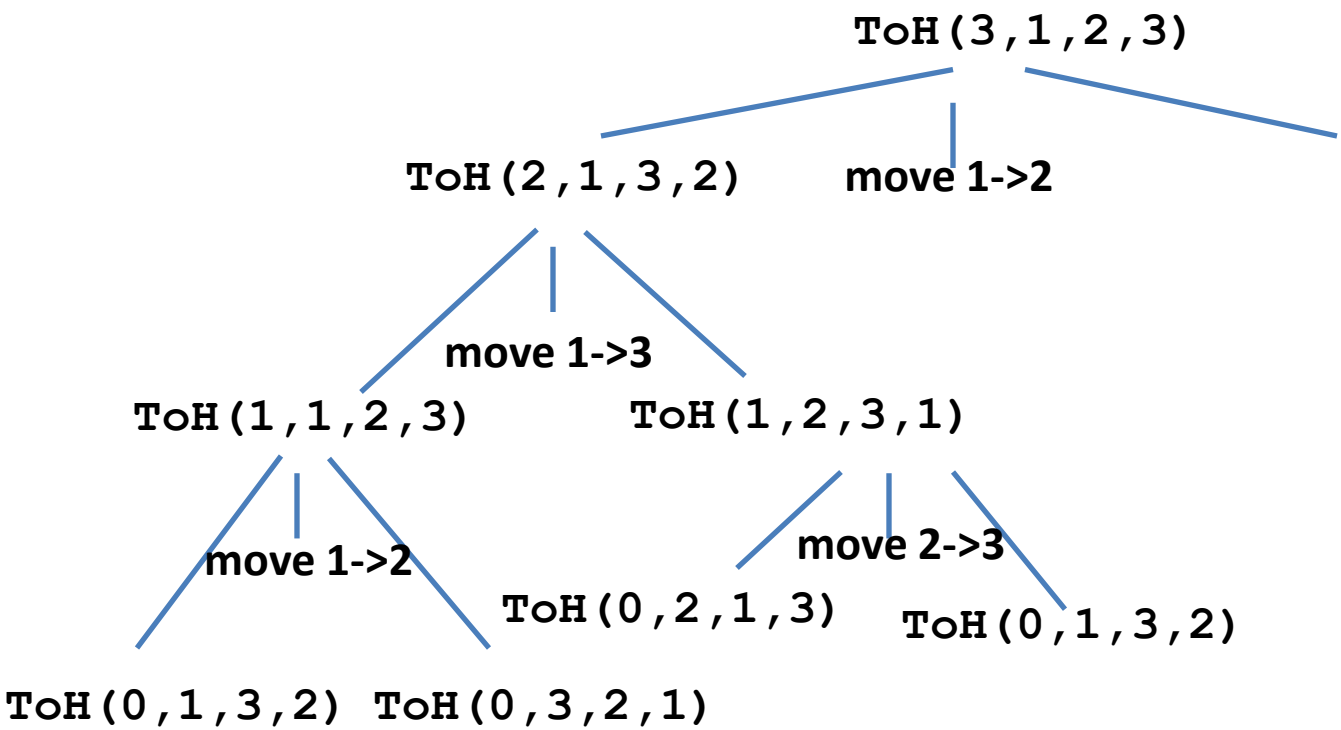
Algorithm Specification

```
if( n ≥ 1 ) then
  {
    ToH ( n-1, x, z, y );
    write( "move from" x "to" y );
    ToH ( n-1, z, y, x );
  }
```



Algorithm Specification

```
if( n ≥ 1 ) then
  {
    ToH ( n-1, x, z, y );
    write( "move from" x "to" y );
    ToH ( n-1, z, y, x );
  }
```



Permutation

- Given a set of elements $\{a, b, c\}$, the permutations are
- $(a, b, c), (a, c, b), (b, a, c), (b, c, a), (c, b, a),$ (c, a, b)
- Given a set of n elements, how many permutations are there?
- Consider the problem of writing the permutations of a given se of elements.

Permutation

- The solution to this problem can be recurively expressed as
- a followed by the permutations of $\{b, c\}$
- b followed by the permutations of $\{a, c\}$
- c followed by the permutations of $\{b, a\}$
- …
- The base case is Permutation of a single element is element itself.

Permutation

- Lets develop the algorithm.
- Assume that the array *l* contains the elements a, b, c

1	**2**	3
a	$\boldsymbol{b}$	c

- Let the algorithm be **`Permute(1,1,3)`**

Permutation

1	**2**	3
a	***b***	*c*

- **`Permute(l,1,3)`**
- **`Swap(l,1,1);Permute(l,2,3); Swap(l,1,1)`**
- **`Swap(l,1,2);Permute(l,2,3); Swap(l,1,2)`**
- **`Swap(l,1,3);Permute(l,2,3); Swap(l,1,3)`**
- …
- **`Permute(l,k,n) when k==n prints the whole array.`**

Permutation

1	**2**	3
a	***b***	*c*

- **Permute(1,2,3)**
- **Swap(1,2,2);Permute(1,3,3); Swap(1,2,2)**
- **abc**
- **Swap(1,2,3);Permute(1,3,3); Swap(1,2,3)**
- **acb**

Permutation

1	**2**	3
b	***a***	*c*

- **Permute(1,2,3)**
- **Swap(1,2,2);Permute(1,3,3); Swap(1,2,2)**
- **bac**
- **Swap(1,2,3);Permute(1,3,3); Swap(1,2,3)**
- **bca**

Permutation

1	**2**	3
c	***b***	*a*

- **Permute(1,2,3)**
- **Swap(1,2,2);Permute(1,3,3); Swap(1,2,2)**
- **cba**
- **Swap(1,2,3);Permute(1,3,3); Swap(1,2,3)**
- **cab**

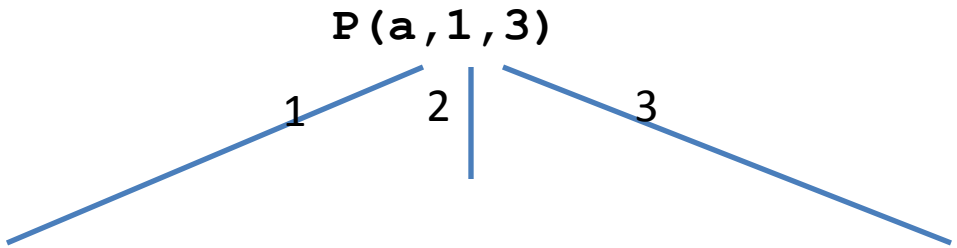
Permutation

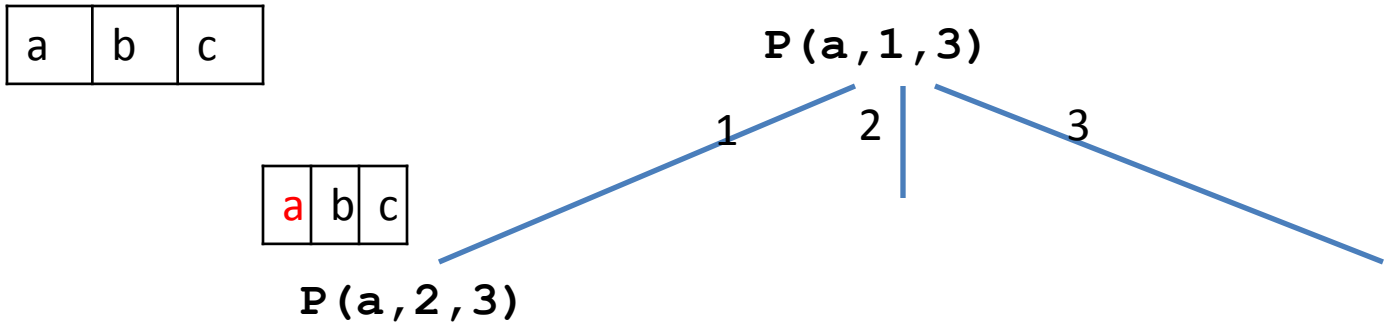
```
Algorithm Perm(a,k,n)
//Permute elements in the array a[k]…a[n]
{
   if(k=n) then write(a[1..n]);
   else
      for j:=k to n do
      {
         t:=a[j];a[j]:=a[k];a[k]:=t;
         Perm(a,k+1,n);
         t:=a[j];a[j]:=a[k];a[k]:=t;
      }
}
```

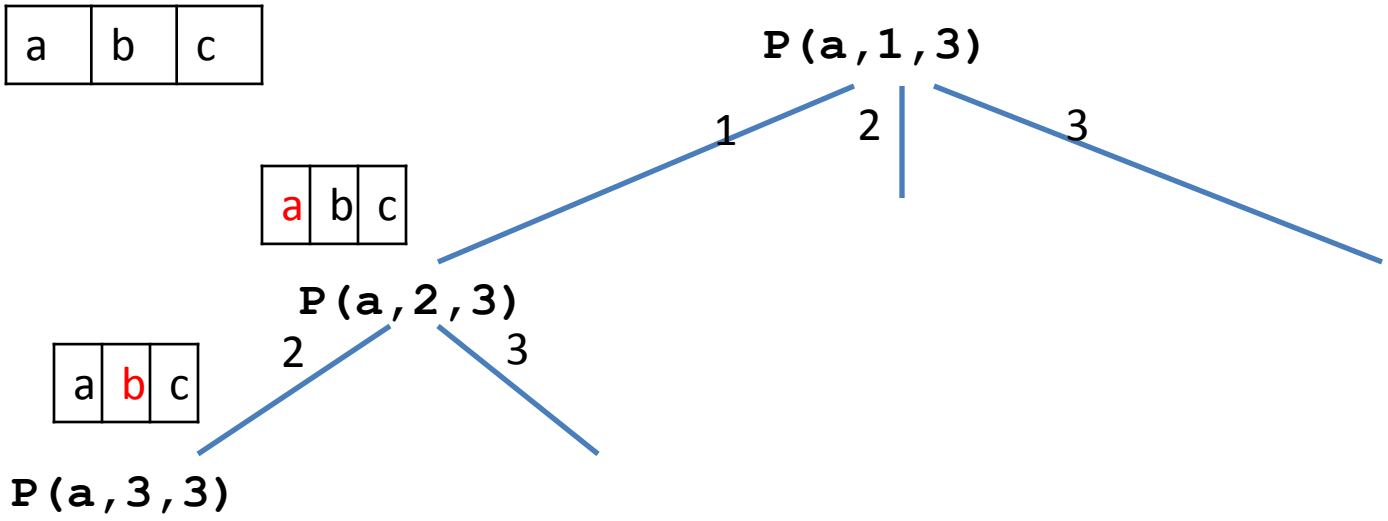
a	b	c

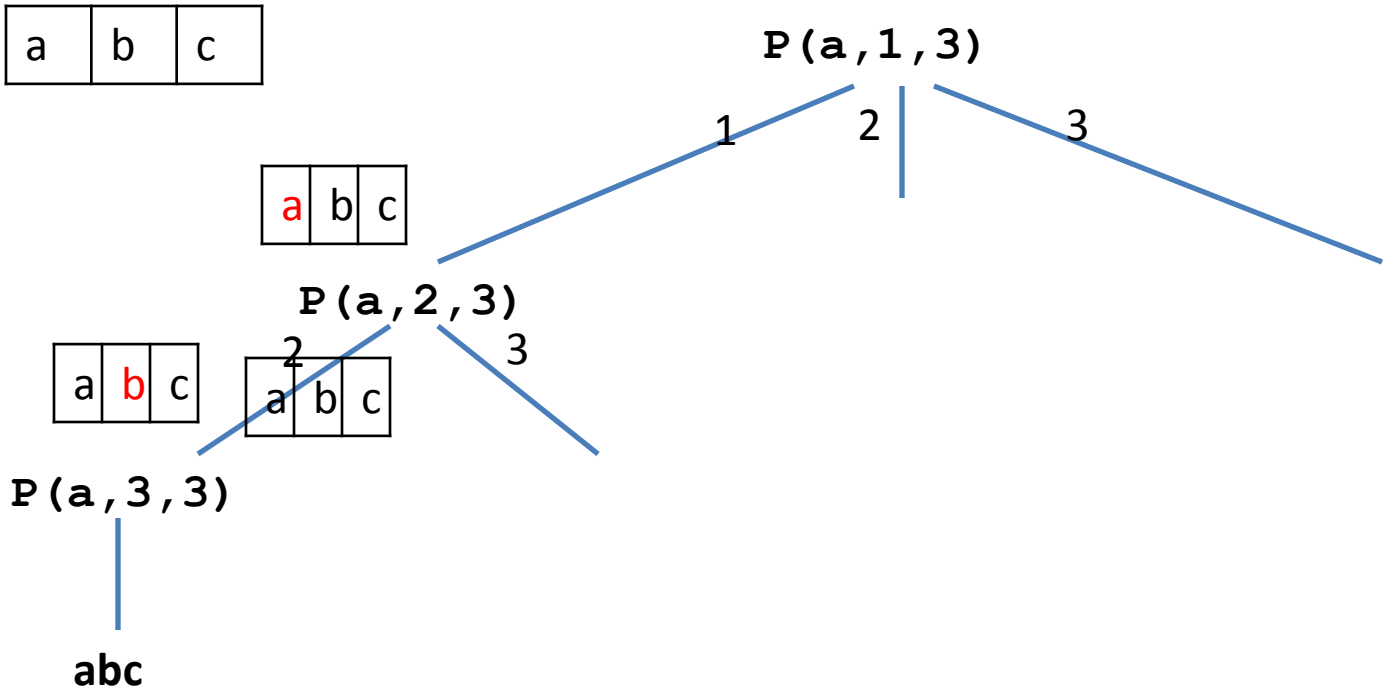
P(a,1,3)

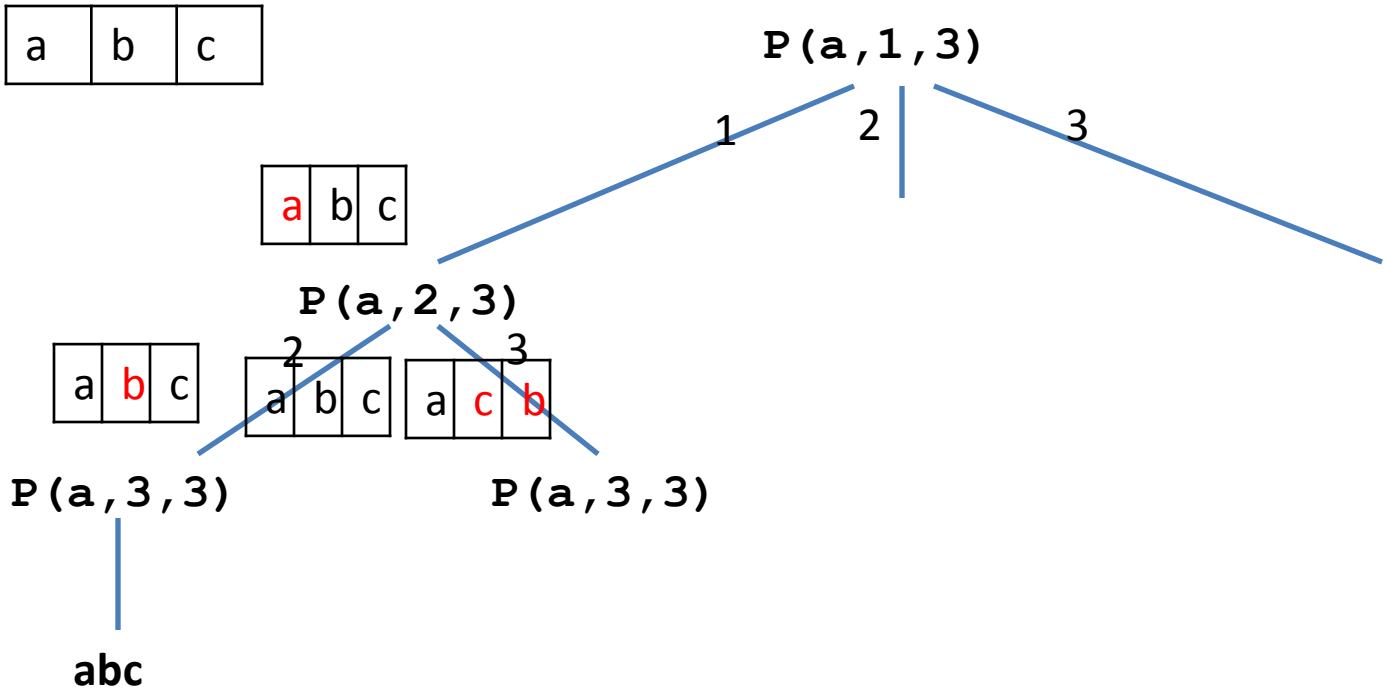
a	b	c

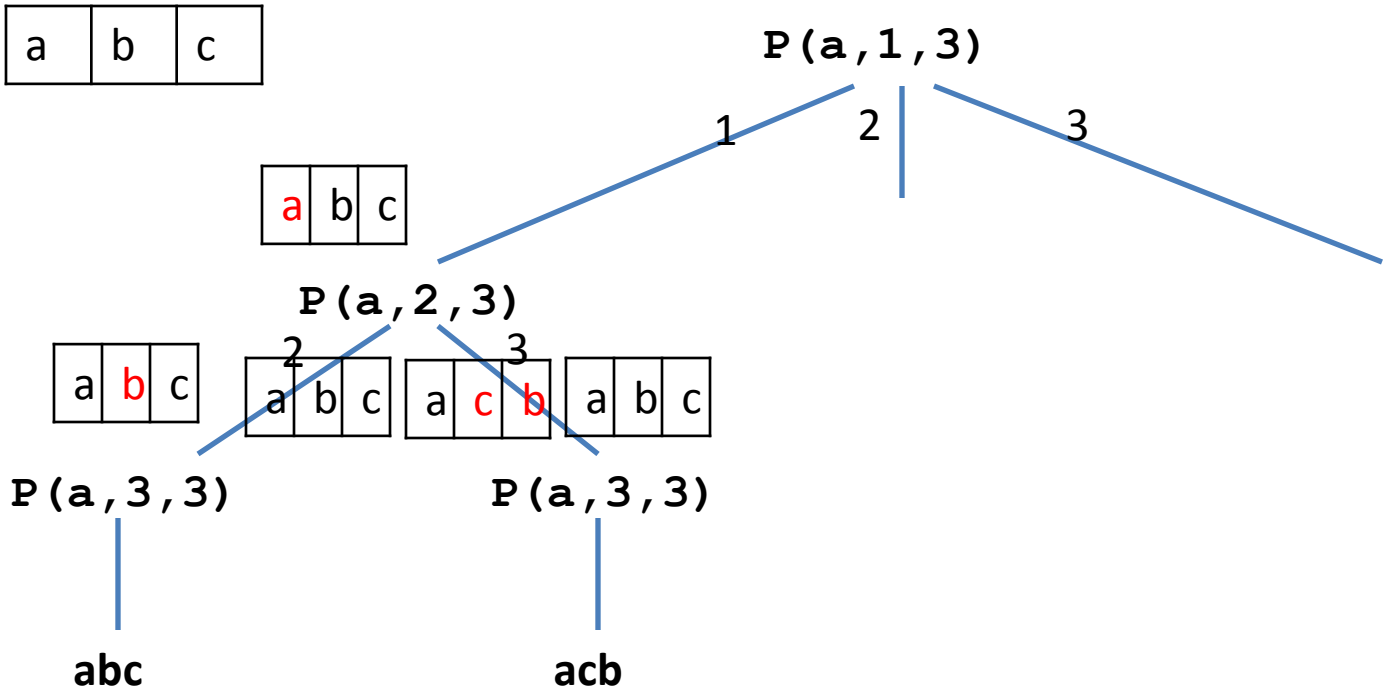


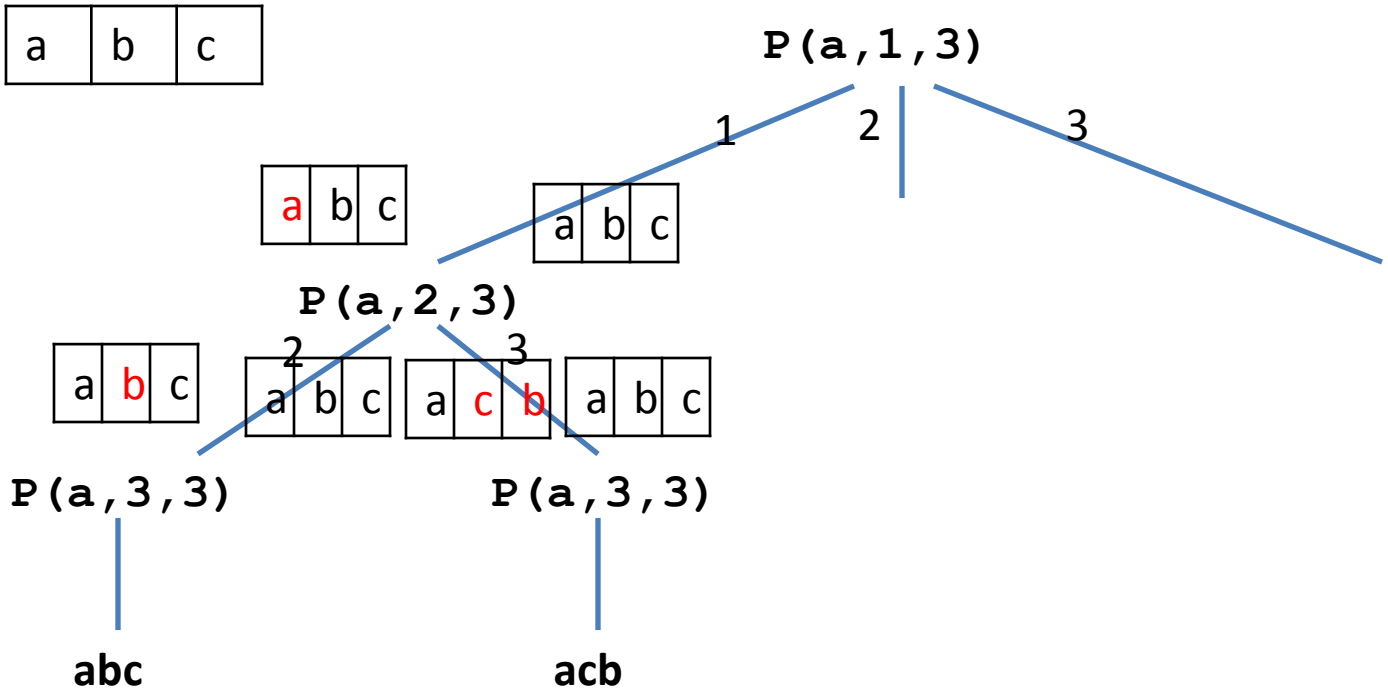

a b c
P(a,1,3)
1
2
3
a b c
P(a,2,3)

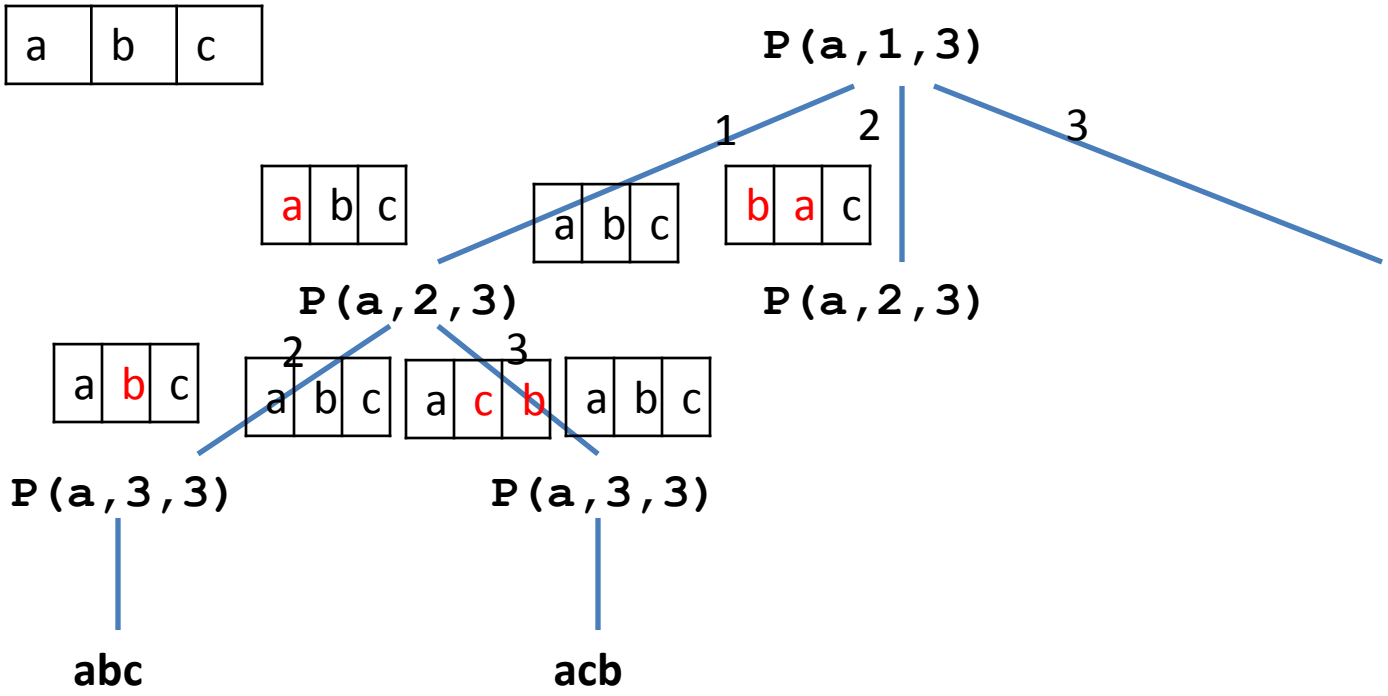

a b c
P(a,1,3)
1
2
3
a b c
P(a,2,3)
2
3
a b c
P(a,3,3)

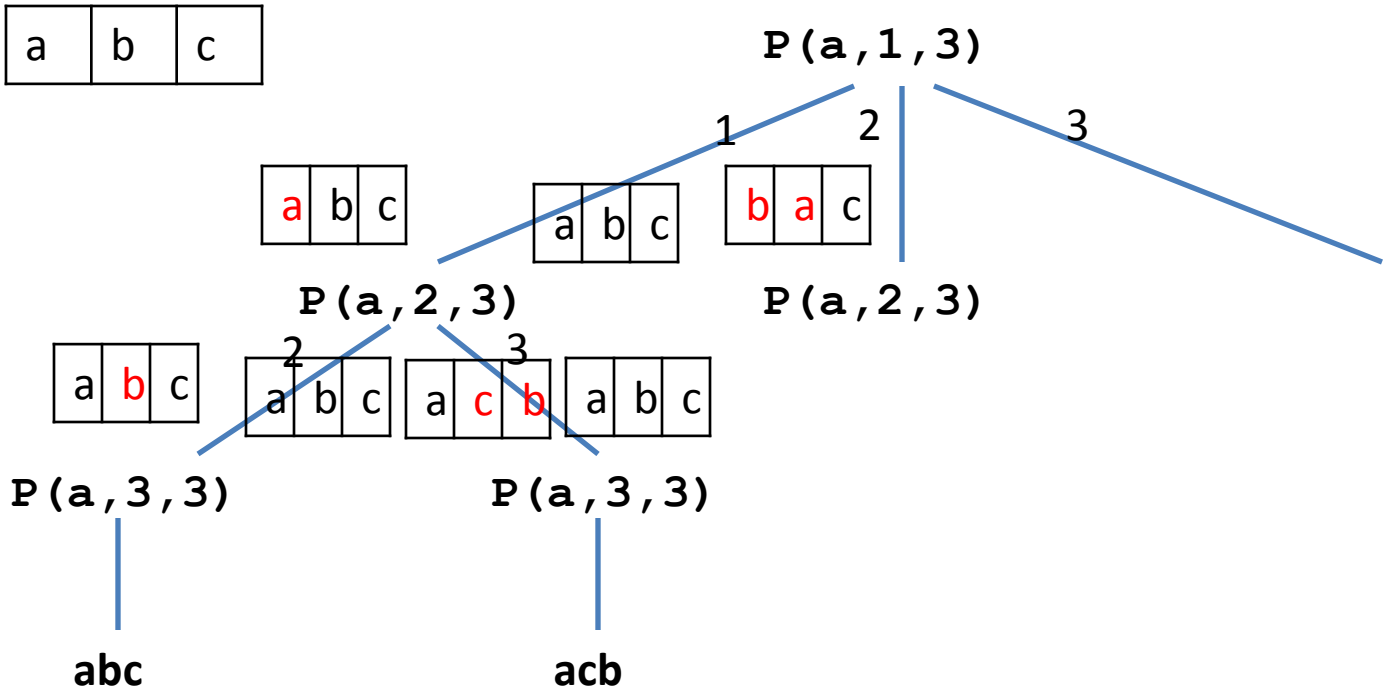

a b c
P(a,1,3)
1
2
3
a b c
P(a,2,3)
2
3
a b c
a b c
P(a,3,3)
abc


a b c
P(a,1,3)
1
2
3
a b c
P(a,2,3)
2
3
a b c
a b c
a c b
P(a,3,3)
P(a,3,3)
abc


a b c
P(a,1,3)
1
2
3
a b c
P(a,2,3)
2
3
a b c
a b c
a c b
a b c
P(a,3,3)
P(a,3,3)
abc
acb


a b c
P(a,1,3)
1
2
3
a b c
a b c
P(a,2,3)
2
3
a b c
a b c
a c b
a b c
P(a,3,3)
P(a,3,3)
abc
acb


a b c
P(a,1,3)
1
2
3
a b c
a b c
b a c
P(a,2,3)
P(a,2,3)
2
3
a b c
a b c
a c b
a b c
P(a,3,3)
P(a,3,3)
abc
acb


a b c
P(a,1,3)
1
2
3
a b c
a b c
b a c
P(a,2,3)
P(a,2,3)
2
3
a b c
a b c
a c b
a b c
P(a,3,3)
P(a,3,3)
abc
acb