

Vistula, IT Faculty, 2014

# Algorithms and Complexity

Dmitry A. Zaitsev

<http://daze.ho.ua>

## Lecture 8:

## Recursive algorithms

## Computing factorial in recursive manner

```
#include <stdio.h>
```

```
int fact( int n )
```

```
{
```

```
    int f;
```

```
    printf("call fact( %d )\n", n);
```

```
    if(n==0) f=1;
```

```
        else f=n*fact( n-1 );
```

```
    printf("return fact( %d ) = %d \n", n, f);
```

```
    return( f );
```

```
}
```

```
main(int argc, char * argv[])
```

```
{
```

```
    int n=atoi(argv[1]);
```

```
    int f=fact(n);
```

```
    printf("factorial( %d ) = %d\n", n, fact(n));
```

```
}
```

## Tracing recursion

>fact 6

call fact( 6 )

call fact( 5 )

call fact( 4 )

call fact( 3 )

call fact( 2 )

call fact( 1 )

call fact( 0 )

factorial( 6 ) = 720

return fact( 6 ) = 720

return fact( 5 ) = 120

return fact( 4 ) = 24

return fact( 3 ) = 6

return fact( 2 ) = 2

return fact( 1 ) = 1

return fact( 0 ) = 1

Complexity:  $O(n)$

## Computing Fibonacci numbers

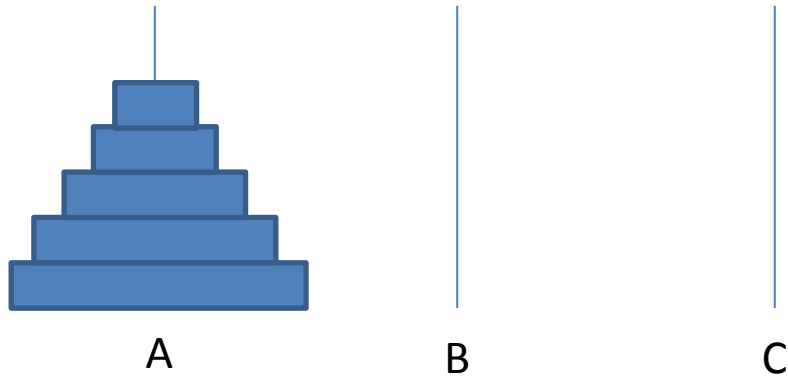
$F(0)=1, F(1)=1, F(n)=F(n-1)+F(n-2), n>1$

```
int fib( int n )  
{  
    if(n<2) return(1);  
    else return( fib(n-1) + fib(n-2) );  
}
```

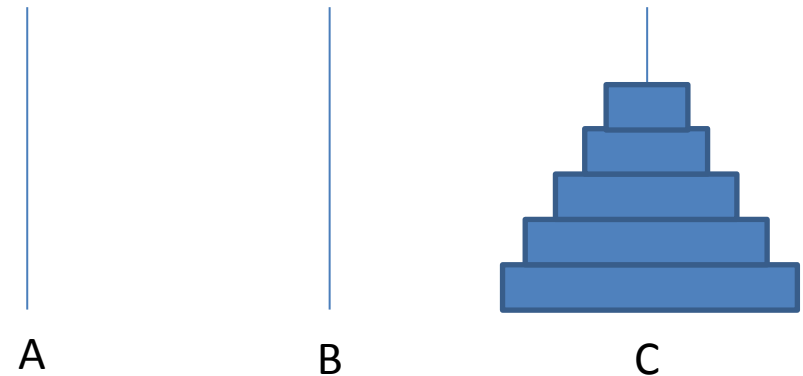
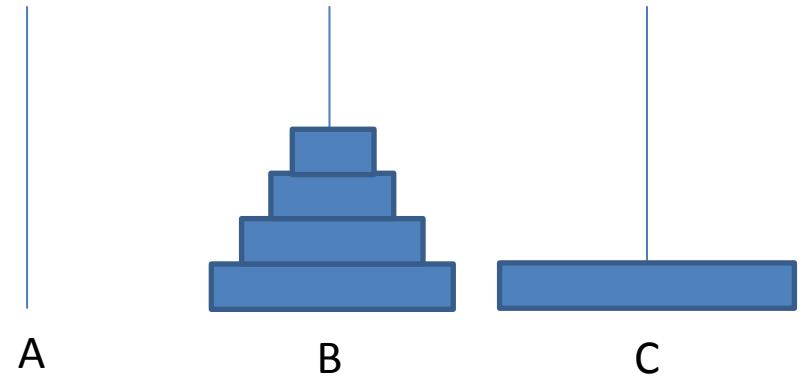
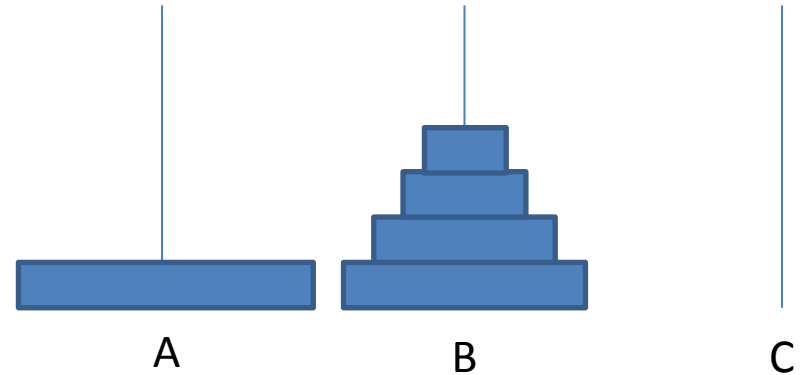
Complexity  $O(2^n)$

Is it inevitable?

# Hanoi tower



1. Move  $n-1$  from A to B using C
2. Move from A to C
3. Move  $n-1$  from B to C using A



# Hanoi tower program

```
#include <stdio.h>
#include <stdlib.h>

void disc_move( int n, char a, char b, char c )
{
    if(n>1)disc_move( n-1, a, c, b );
    printf("move disc from %c to %c\n", a, c );
    if(n>1)disc_move( n-1, b, a, c );
}

main(int argc, char * argv[])
{
    disc_move( atoi(argv[1]), 'A', 'B', 'C' );
}
```

# Tracing Hanoi tower program

hanoi 2

move disc from A to B  
move disc from A to C  
move disc from B to C

hanoi 3

move disc from A to C  
move disc from A to B  
move disc from C to B  
move disc from A to C  
move disc from B to A  
move disc from B to C  
move disc from A to C

hanoi 4

move disc from A to B  
move disc from A to C  
move disc from B to C  
move disc from A to B  
move disc from C to A  
move disc from C to B  
move disc from A to B  
move disc from A to C  
move disc from B to C  
move disc from B to A  
move disc from C to A  
move disc from B to C  
move disc from A to B  
move disc from A to C  
move disc from B to C

## Recursive algorithms for lists

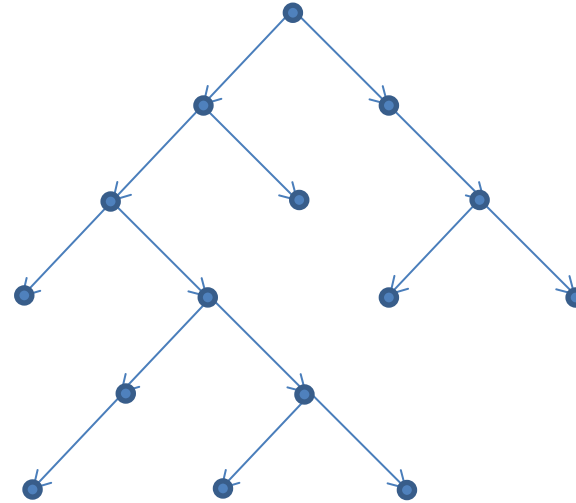
```
struct elist1 {  
    int cont;  
    struct elist1 * next;  
};
```

```
int lsum(struct elist1 * head)  
{  
    if(head==NULL) return(0);  
    else return( head->cont + lsum( head->next ) );  
}
```



# Recursive algorithms for binary trees

```
struct btree {
    struct btree * left;
    int cont;
    struct btree * right;
};
```



```
int btree_sum( struct btree *pt )
{
    if( pt==NULL ) return(0);
    else return( btree_sum(pt->left) + pt->cont + btree_sum(pt->right));
}
```