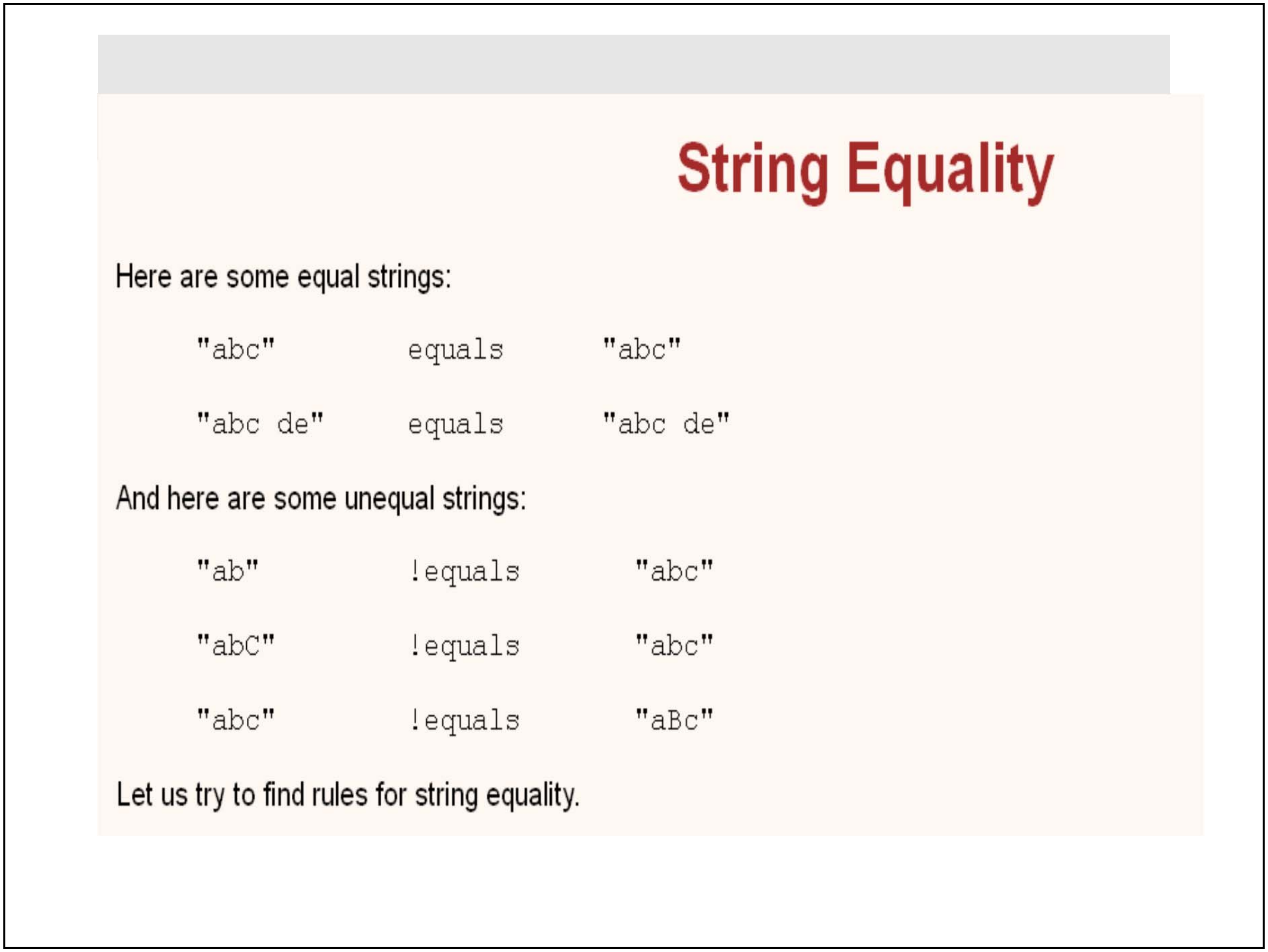
Recursion

- Recursion is a way of programming or coding a problem, in which a function calls itself one or more times in its body. Usually, it is returning the return value of this function call. If a function definition fulfills the condition of recursion, we call this function a recursive function.

  A recursive function has to terminate to be used in a program. A recursive function terminates, if with every recursive call the solution of the problem is downsized and moves towards a base case. A base case is a case, where the problem can be solved without further recursion. A recursion can lead to an infinite loop, if the base case is not met in the calls.



String Equality

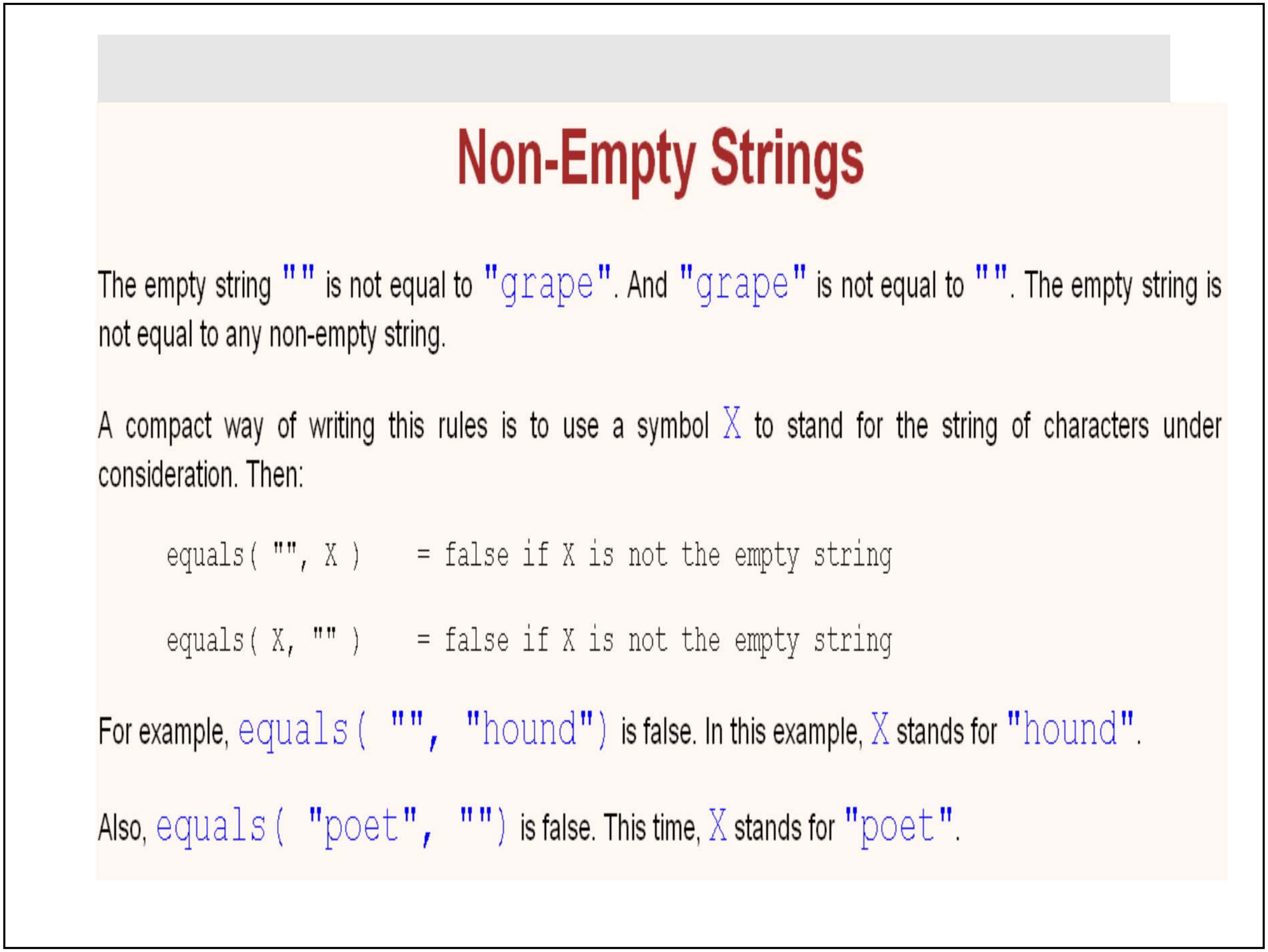
Here are some equal strings:

```
"abc"       equals      "abc"

"abc de"    equals      "abc de"
```

And here are some unequal strings:

```
"ab"        !equals      "abc"

"abC"       !equals      "abc"

"abc"       !equals      "aBc"
```

Let us try to find rules for string equality.



Non-Empty Strings

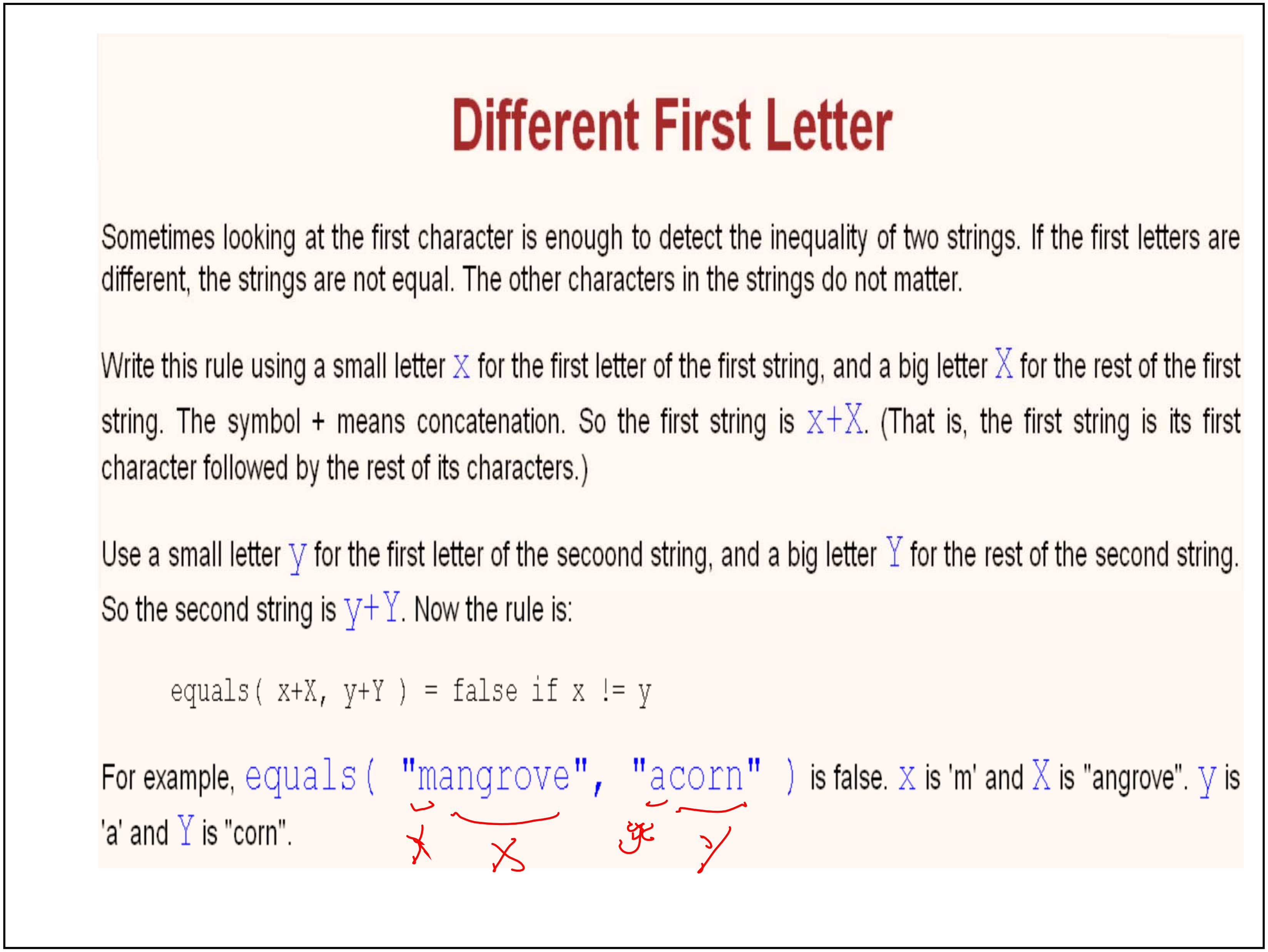
The empty string `""` is not equal to `"grape"`. And `"grape"` is not equal to `""`. The empty string is not equal to any non-empty string.

A compact way of writing this rules is to use a symbol `X` to stand for the string of characters under consideration. Then:

```
equals( "", X )    = false if X is not the empty string

equals( X, "" )    = false if X is not the empty string
```

For example, `equals( "", "hound")` is false. In this example, `X` stands for `"hound"`.

Also, `equals( "poet", "")` is false. This time, `X` stands for `"poet"`.



Different First Letter

Sometimes looking at the first character is enough to detect the inequality of two strings. If the first letters are different, the strings are not equal. The other characters in the strings do not matter.

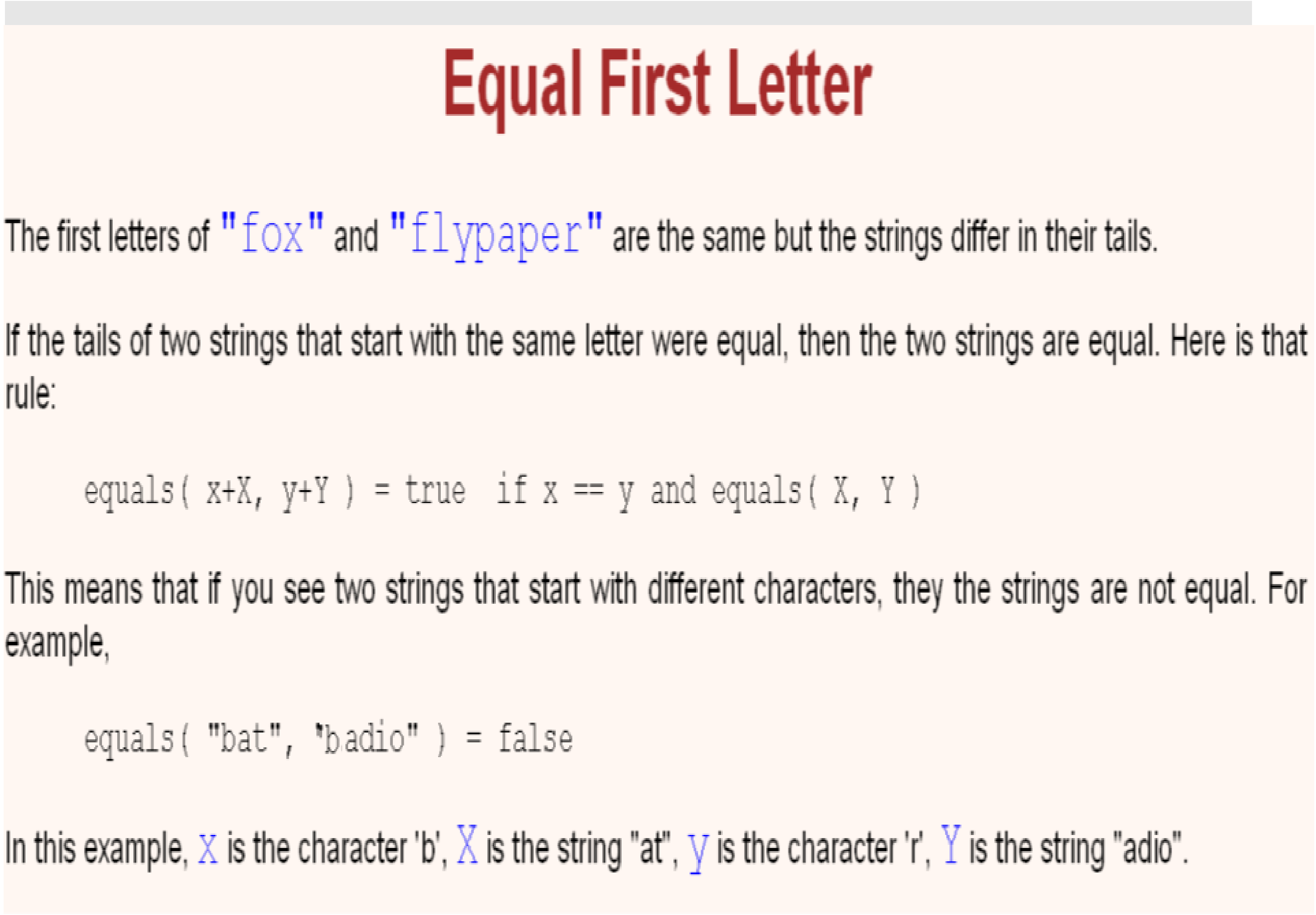
Write this rule using a small letter x for the first letter of the first string, and a big letter X for the rest of the first string. The symbol + means concatenation. So the first string is x+X. (That is, the first string is its first character followed by the rest of its characters.)

Use a small letter y for the first letter of the second string, and a big letter Y for the rest of the second string. So the second string is y+Y. Now the rule is:

```
    equals( x+X, y+Y ) = false if x != y
```

For example, `equals( "mangrove", "acorn" )` is false. x is 'm' and X is "angrove". y is 'a' and Y is "corn".

x X y Y



Equal First Letter

The first letters of "fox" and "flypaper" are the same but the strings differ in their tails.

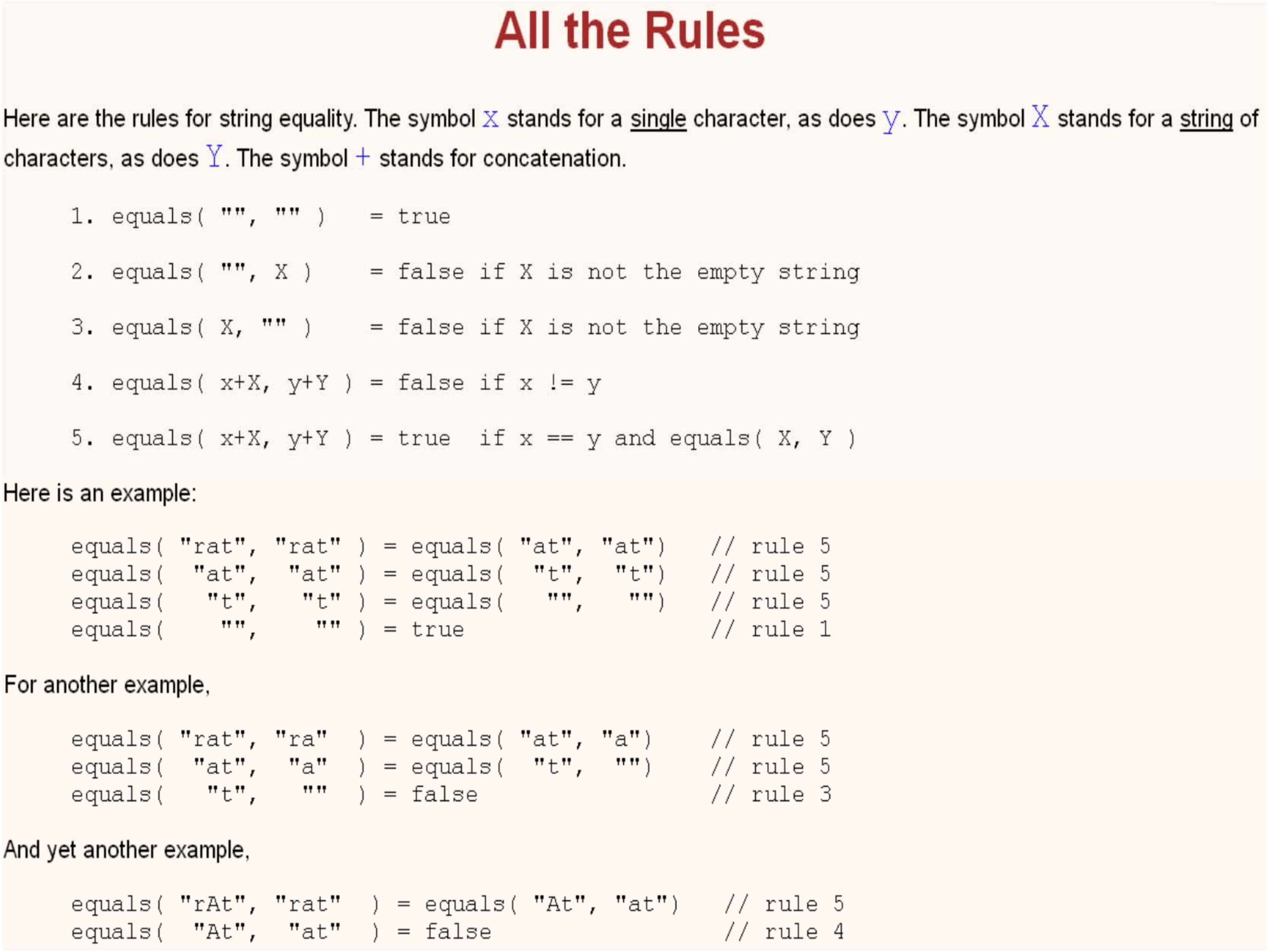
If the tails of two strings that start with the same letter were equal, then the two strings are equal. Here is that rule:

```
    equals( x+X, y+Y ) = true  if x == y and equals( X, Y )
```

This means that if you see two strings that start with different characters, they the strings are not equal. For example,

```
    equals( "bat", "badio" ) = false
```

In this example, x is the character 'b', X is the string "at", y is the character 'r', Y is the string "adio".



All the Rules

Here are the rules for string equality. The symbol x stands for a single character, as does y. The symbol X stands for a string of characters, as does Y. The symbol + stands for concatenation.

```
1. equals( "", "" )   = true

2. equals( "", X )    = false if X is not the empty string

3. equals( X, "" )    = false if X is not the empty string

4. equals( x+X, y+Y ) = false if x != y

5. equals( x+X, y+Y ) = true  if x == y and equals( X, Y )
```

Here is an example:

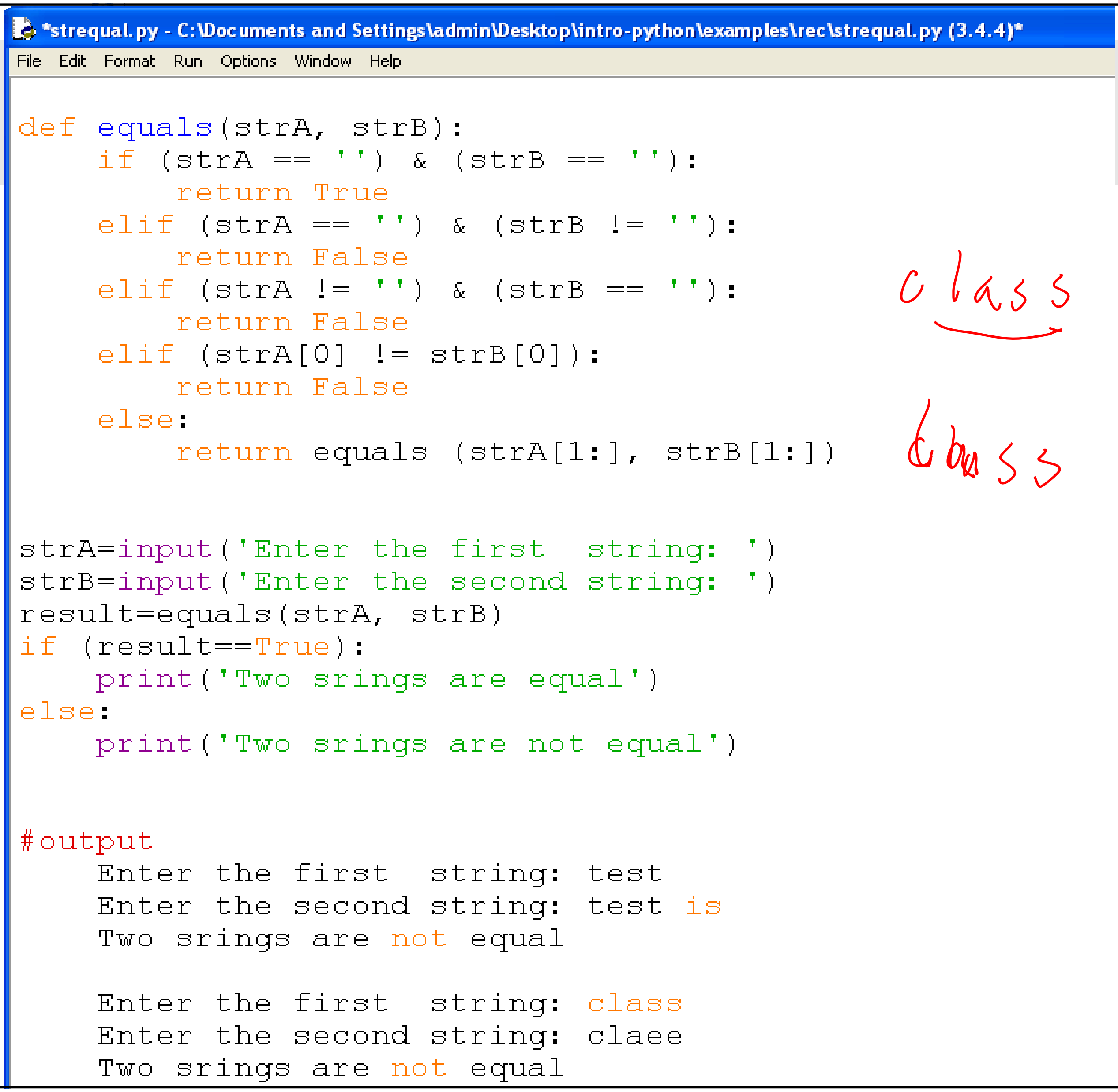
```
equals( "rat", "rat" ) = equals( "at", "at")   // rule 5
equals(  "at",  "at" ) = equals(  "t",  "t")   // rule 5
equals(   "t",   "t" ) = equals(   "",   "")   // rule 5
equals(    "",    "" ) = true                  // rule 1
```

For another example,

```
equals( "rat", "ra"  ) = equals( "at", "a")    // rule 5
equals(  "at",  "a"  ) = equals(  "t", "")     // rule 5
equals(   "t",   ""  ) = false                 // rule 3
```

And yet another example,

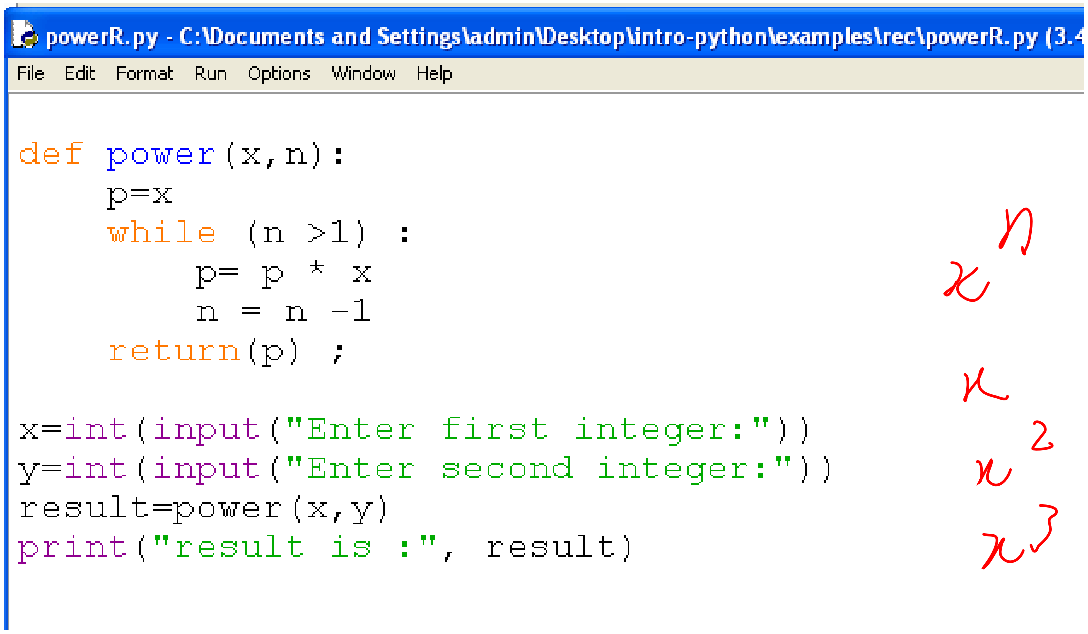
```
equals( "rAt", "rat"  ) = equals( "At", "at")   // rule 5
equals(  "At",  "at"  ) = false                 // rule 4
```


strequal.py - C:\Documents and Settings\admin\Desktop\intro-python\examples\rec\strequal.py (3.4.4)
File Edit Format Run Options Window Help

```
def equals(strA, strB):
    if (strA == '') & (strB == ''):
        return True
    elif (strA == '') & (strB != ''):
        return False
    elif (strA != '') & (strB == ''):
        return False
    elif (strA[0] != strB[0]):
        return False
    else:
        return equals (strA[1:], strB[1:])


strA=input('Enter the first  string: ')
strB=input('Enter the second string: ')
result=equals(strA, strB)
if (result==True):
    print('Two srings are equal')
else:
    print('Two srings are not equal')


#output
    Enter the first  string: test
    Enter the second string: test is
    Two srings are not equal

    Enter the first  string: class
    Enter the second string: claee
    Two srings are not equal
```

class   claee

lass   laee


powerR.py - C:\Documents and Settings\admin\Desktop\intro-python\examples\rec\powerR.py (3.4
File Edit Format Run Options Window Help

```
def power(x,n):
    p=x
    while (n >1) :
        p= p * x
        n = n -1
    return(p) ;

x=int(input("Enter first integer:"))
y=int(input("Enter second integer:"))
result=power(x,y)
print("result is :", result)
```

x^n

x

x^2

x^3

$O(n)$

```python
def power(x,n):
    p=1
    while (n >=1) :
        if((n%2)==1) : p=p*x
        x= x * x
        n = n //2
    return(p)

x=int(input("Enter first integer:"))
y=int(input("Enter second integer:"))
result=power(x,y)
print("result is :", result)
```

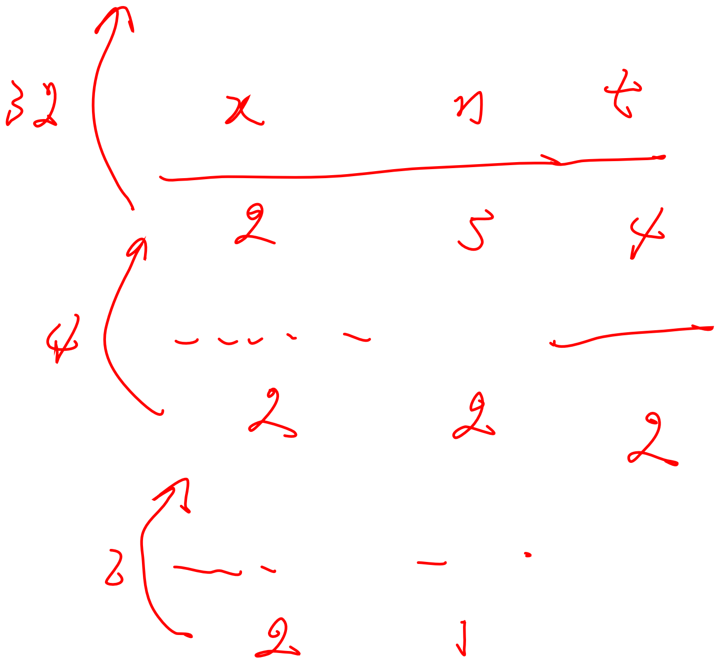
$$x^n = \begin{cases} x^{n/2} * x^{n/2} & \text{n زوجي} \\ x * x^{n/2} * x^{n/2} & \text{n فردي} \end{cases}$$

$x^1 = x$

```
def powerR(x, n):
    if(n==1):
        return(x) ;
    else:
        t=powerR(x, n//2) ;
        if((2*(n//2))==n):
            return (t*t) ;
        else:
            return(t*t*x) ;

x=int(input("Enter first integer:"))
y=int(input("Enter second integer:"))
result=powerR(x,y)
print("result is :", result)
```

$$x^n = \begin{cases} \underbrace{x^{n/2}}_{t} * \underbrace{x^{n/2}}_{t} & \text{زوج } n \\ x^{n/2} * x^{n/2} * x & \text{فرد } n \end{cases}$$

```
def powerR(x, n):
    if(n==1):
        return(x) ;
    else:
        t=powerR(x, n//2) ;
        if((2*(n//2))==n):
            return (t*t) ;
        else:
            return(t*t*x) ;
```



```
def powerR(x, n):
    if(n==1):
        return(x) ;
    else:
        t=powerR(x, n//2) ;
        if((2*(n//2))==n):
            return (t*t) ;
        else:
            return(t*t*x) ;
```

$T(n)$

$T(n/2)$

```
x=int(input("Enter first integer:"))
y=int(input("Enter second integer:"))
result=powerR(x,y)
print("result is :", result)
```

$T(1)=1$

$T(n)=T(n/2) + 1$ $k=1$

$T(n)=T(n/4)+1+1=T(n/4)+2$ $k=2$

$T(n)=T(n/8)+1+2=T(n/8)+3= T(n/2^3) + 3$ $k=3$

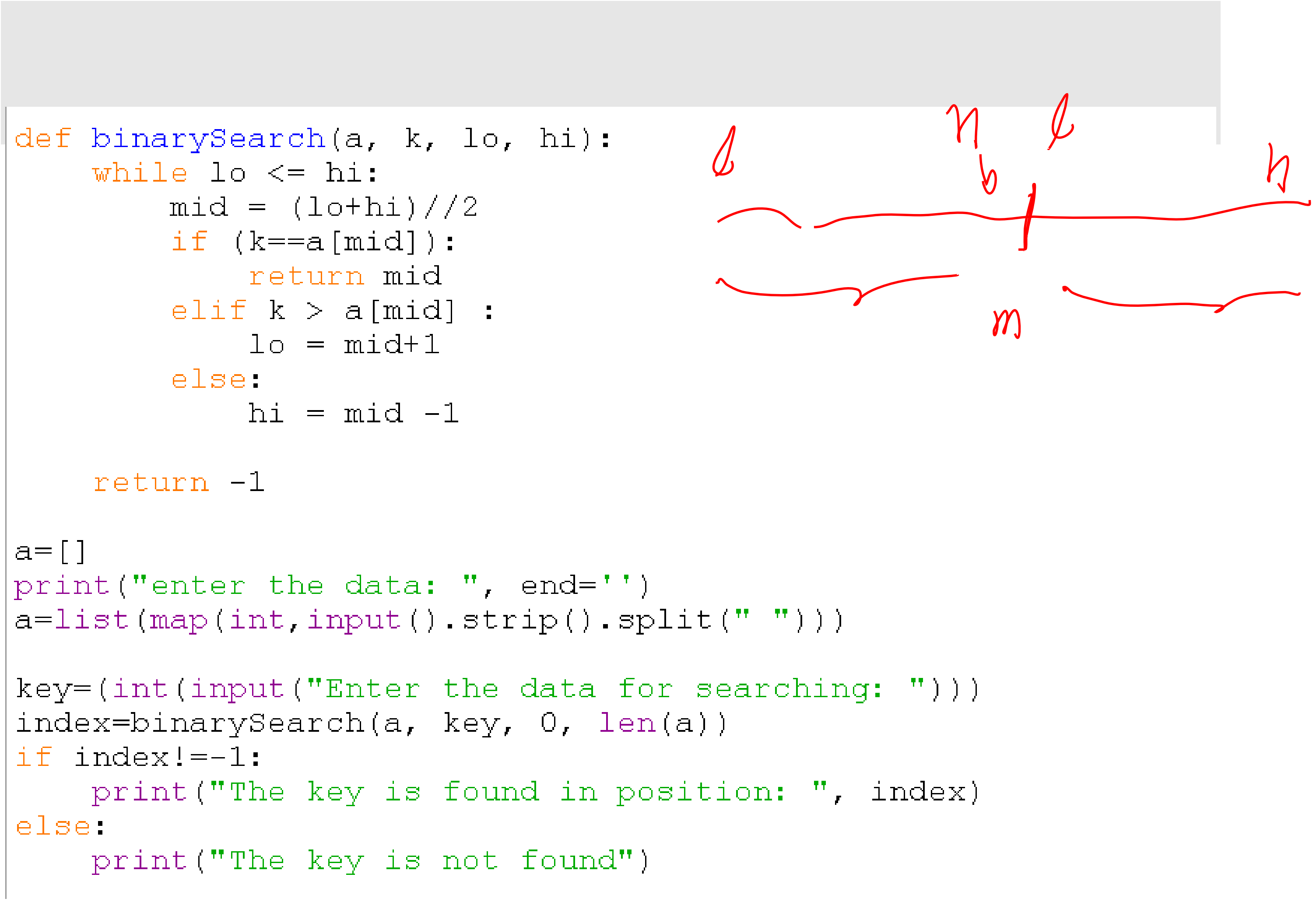
$T(n)=T(n/16)+1+3=T(n/16)+4= T(n/2^4) + 4$ $k=4$

……………………

$T(n)= T(n/2^k) + k$

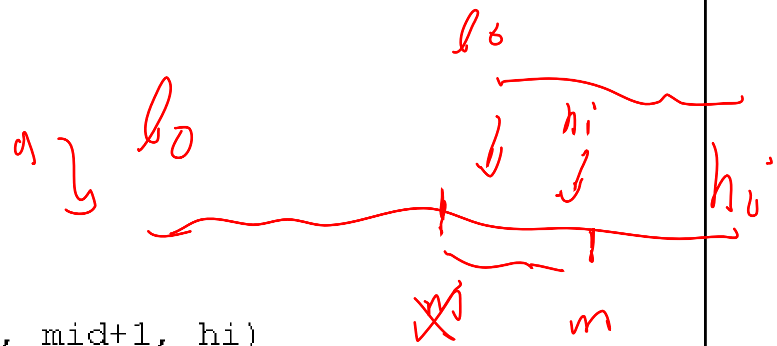
$n= 2^k$ $k=\log n$

$T(n)=T(1)+ \log n = 1 + \text{Log } n = O(\log n)$



```python
def binarySearch(a, k, lo, hi):
    while lo <= hi:
        mid = (lo+hi)//2
        if (k==a[mid]):
            return mid
        elif k > a[mid] :
            lo = mid+1
        else:
            hi = mid -1

    return -1

a=[]
print("enter the data: ", end='')
a=list(map(int,input().strip().split(" ")))

key=(int(input("Enter the data for searching: ")))
index=binarySearch(a, key, 0, len(a))
if index!=-1:
    print("The key is found in position: ", index)
else:
    print("The key is not found")
```

```python
def binarySearchR(a, k, lo, hi):
    if lo > hi:
        return -1
    else:
        mid = (lo+hi)//2
        if (k==a[mid]):
            return mid
        elif k > a[mid] :
            return binarySearchR(a, k, mid+1, hi)
        else:
            return binarySearchR(a, k, lo, mid-1)
    return

a=[]
print("enter the data: ", end='')
a=list(map(int,input().strip().split(" ")))

key=(int(input("Enter the data for searching: ")))
index=binarySearchR(a, key, 0, len(a))
if index!=-1:
    print("The key is found in position: ", index)
else:
    print("The key is not found")
```



T(n)

```
def binarySearchR(a, k, lo, hi):
    if lo > hi:
        return -1
    else:
        mid = (lo+hi)//2
        if (k==a[mid]):
            return mid
        elif k > a[mid] :
            return binarySearchR(a, k, mid+1, hi)
        else:
            return binarySearchR(a, k, lo, mid-1)
    return
```

1

$T(1)=1$

$T(n)=T(n/2) + 1$

$T(n)=T(n/4)+1+1=T(n/4)+2$

$T(n)=T(n/8)+1+2=T(n/8)+3= T(n/2^3) + 3$

$T(n)=T(n/16)+1+3=T(n/16)+4= T(n/2^4) + 4$

……………………

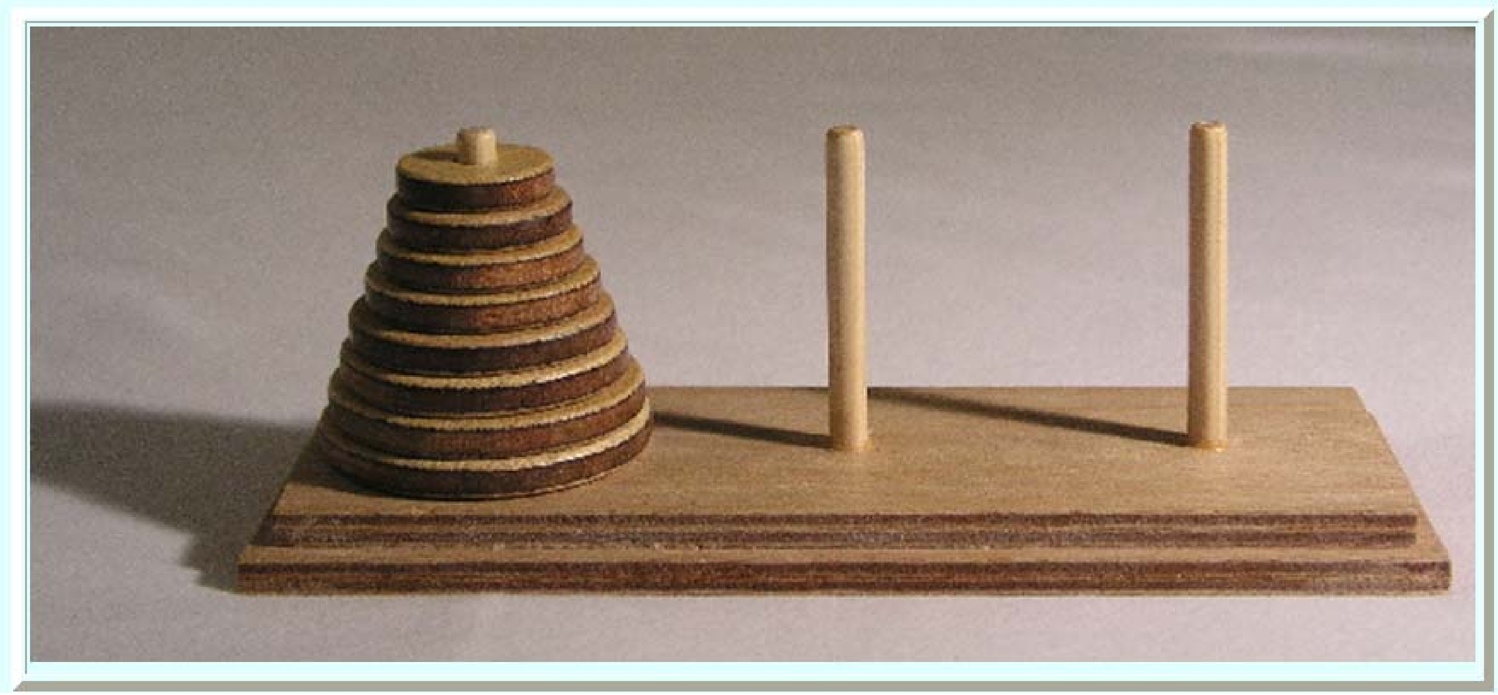
$T(n)= T(n/2^k) + k$

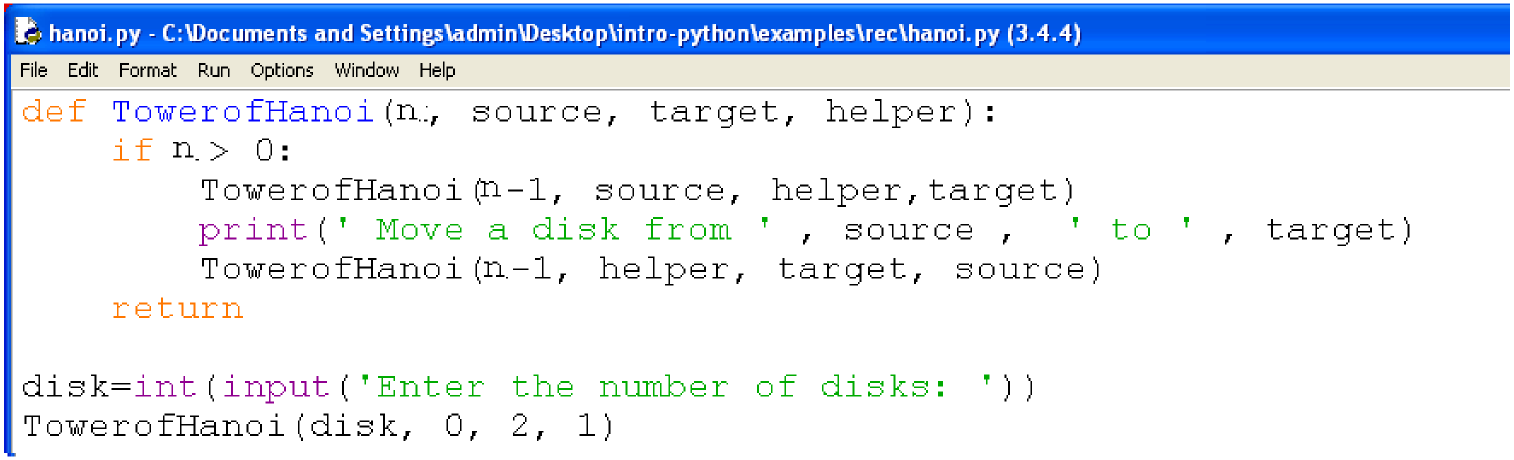
$n= 2^k$ $\quad$ $k=\log n$

$T(n)=T(1)+ \log n = 1 + \text{Log } n = O(\log n)$

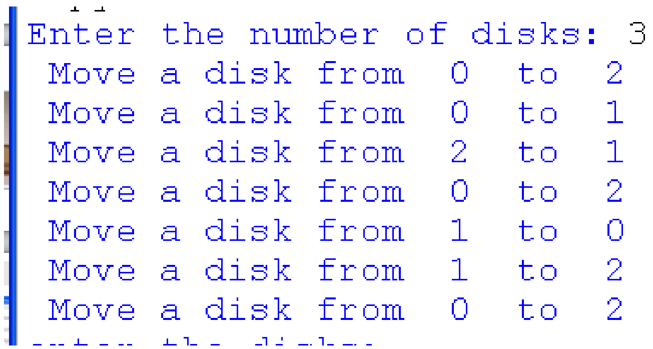
Tower of Hanoi

- The Tower of Hanoi puzzle was invented by the French mathematician Edouard Lucas in 1883. He was inspired by a legend that tells of a Hindu temple where the puzzle was presented to young priests. At the beginning of time, the priests were given three poles and a stack of 64 gold disks, each disk a little smaller than the one beneath it. Their assignment was to transfer all 64 disks from one of the three poles to another, with two important constraints. They could only move one disk at a time, and they could never place a larger disk on top of a smaller one. The priests worked very efficiently, day and night, moving one disk every second. When they finished their work, the legend said, the temple would crumble into dust and the world would vanish.
- Although the legend is interesting, you need not worry about the world ending any time soon. The number of moves required to correctly move a tower of 64 disks is $2^{64}-1$=18,446,744,073,709,551,615
- At a rate of one move per second, that is 584,942,417,355 years! Clearly there is more to this puzzle than meets the eye.

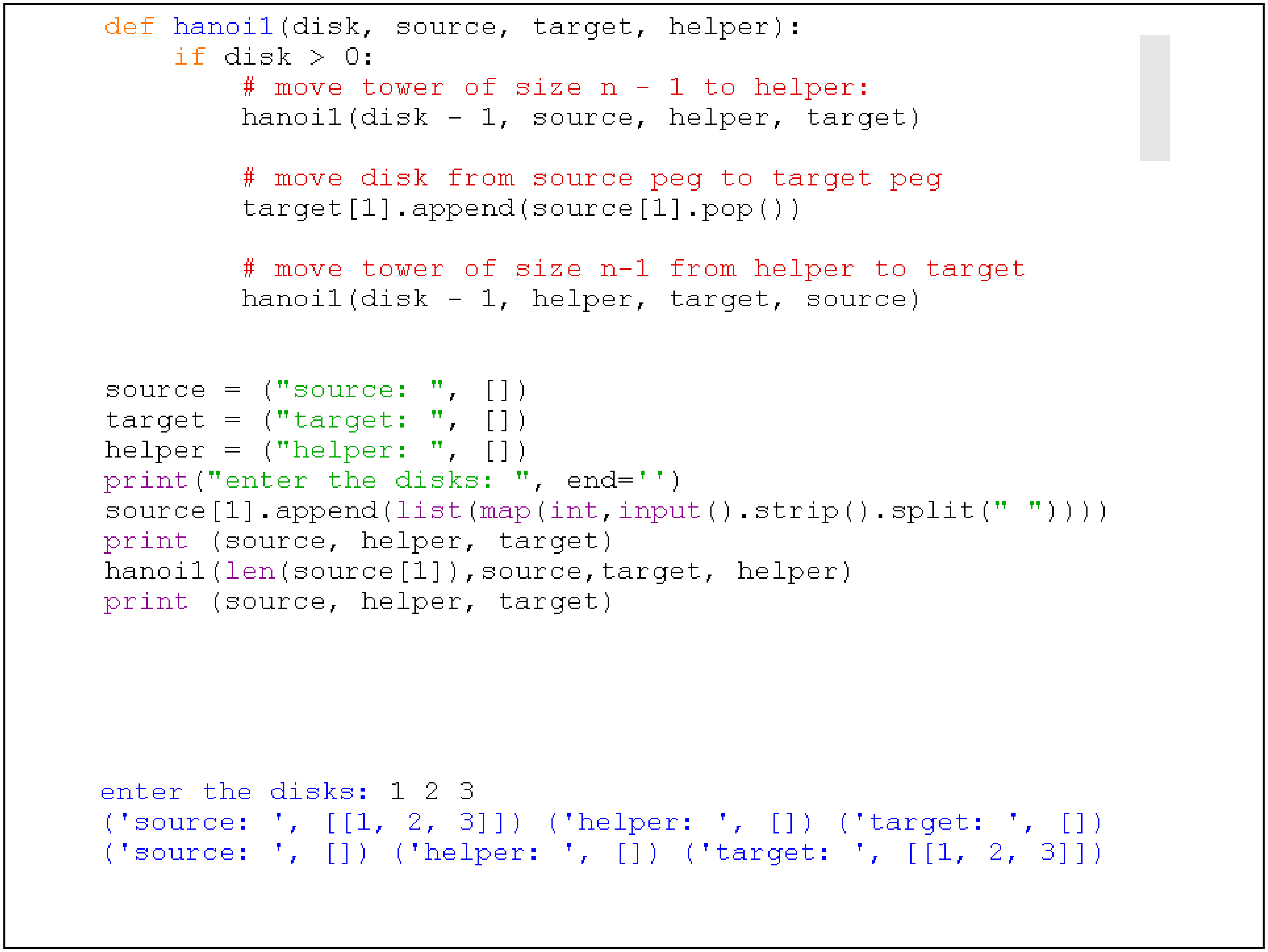



hanoi.py - C:\Documents and Settings\admin\Desktop\intro-python\examples\rec\hanoi.py (3.4.4)
File
Edit
Format
Run
Options
Window
Help

```
def TowerofHanoi(n, source, target, helper):
    if n > 0:
        TowerofHanoi(n-1, source, helper,target)
        print(' Move a disk from ' , source ,  ' to ' , target)
        TowerofHanoi(n-1, helper, target, source)
    return

disk=int(input('Enter the number of disks: '))
TowerofHanoi(disk, 0, 2, 1)
```

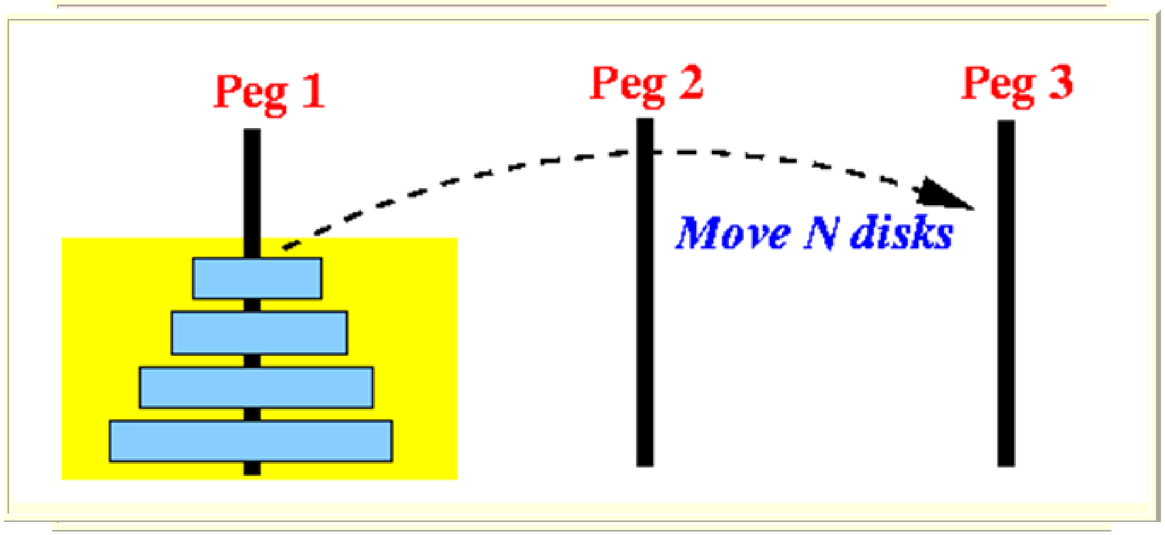


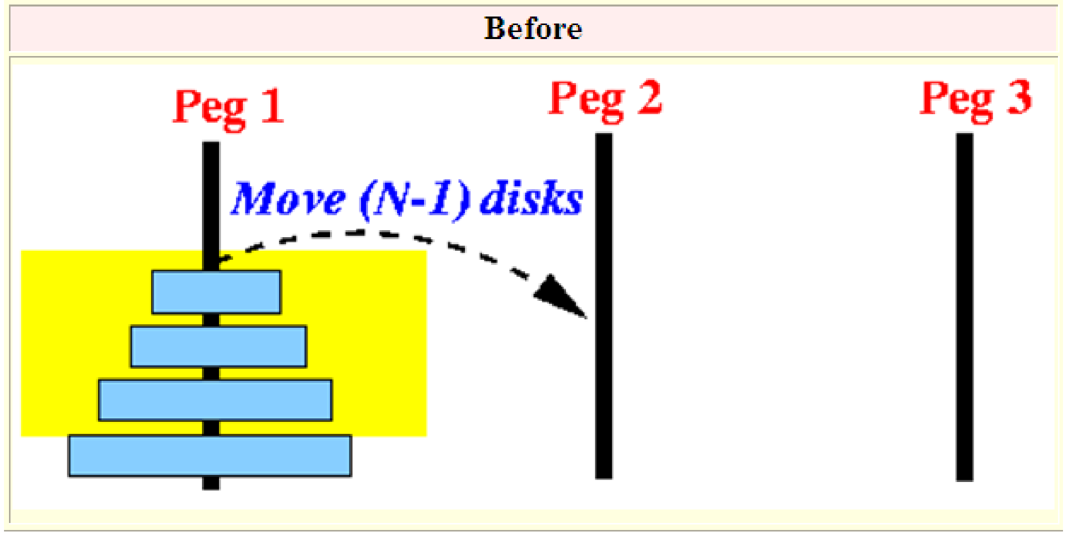
```
Enter the number of disks: 3
 Move a disk from  0  to  2
 Move a disk from  0  to  1
 Move a disk from  2  to  1
 Move a disk from  0  to  2
 Move a disk from  1  to  0
 Move a disk from  1  to  2
 Move a disk from  0  to  2
```

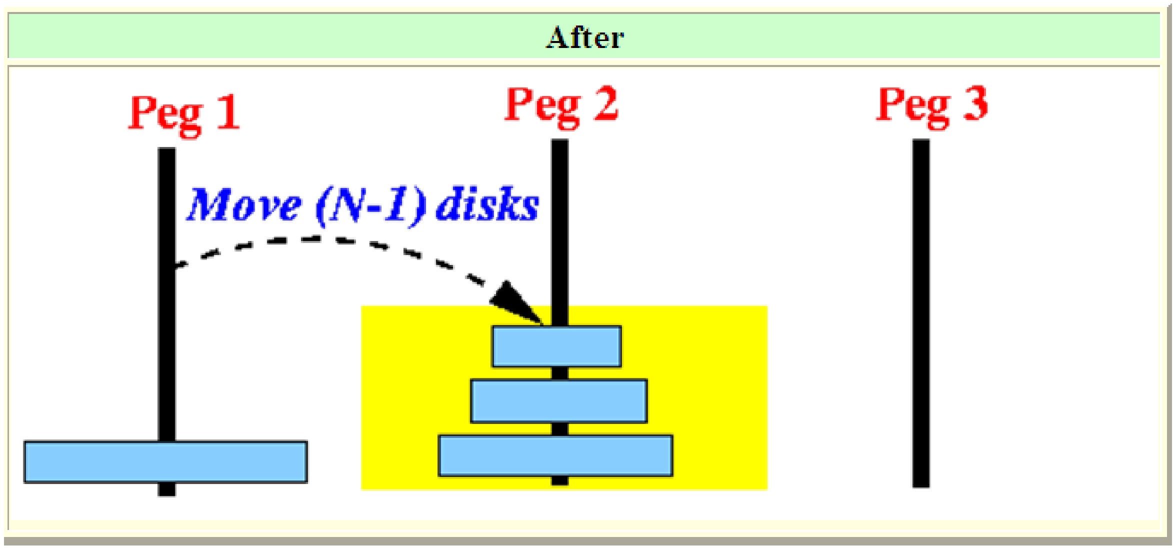


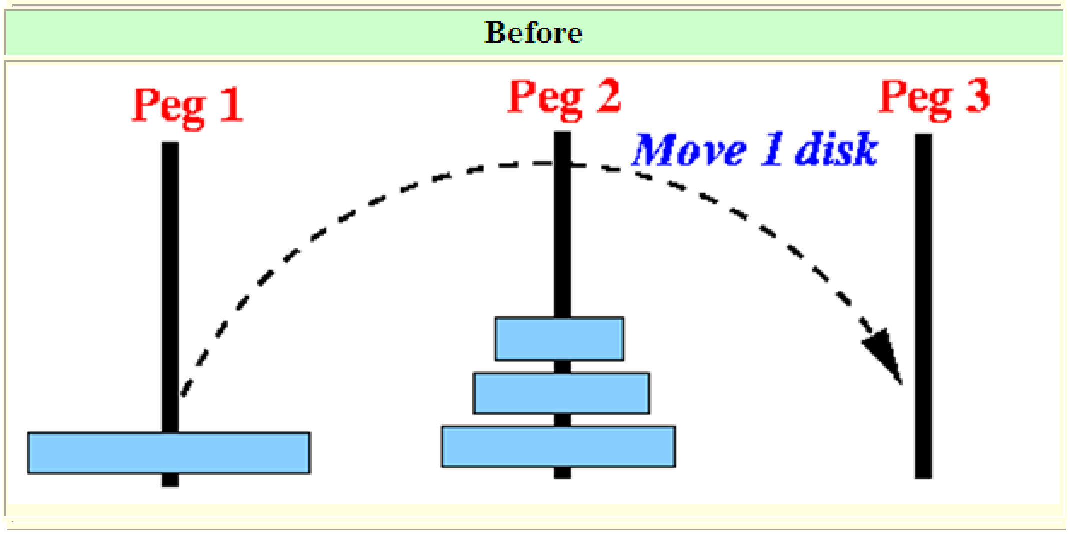
```
def hanoi1(disk, source, target, helper):
    if disk > 0:
        # move tower of size n - 1 to helper:
        hanoi1(disk - 1, source, helper, target)

        # move disk from source peg to target peg
        target[1].append(source[1].pop())

        # move tower of size n-1 from helper to target
        hanoi1(disk - 1, helper, target, source)


source = ("source: ", [])
target = ("target: ", [])
helper = ("helper: ", [])
print("enter the disks: ", end='')
source[1].append(list(map(int,input().strip().split(" "))))
print (source, helper, target)
hanoi1(len(source[1]),source,target, helper)
print (source, helper, target)
```

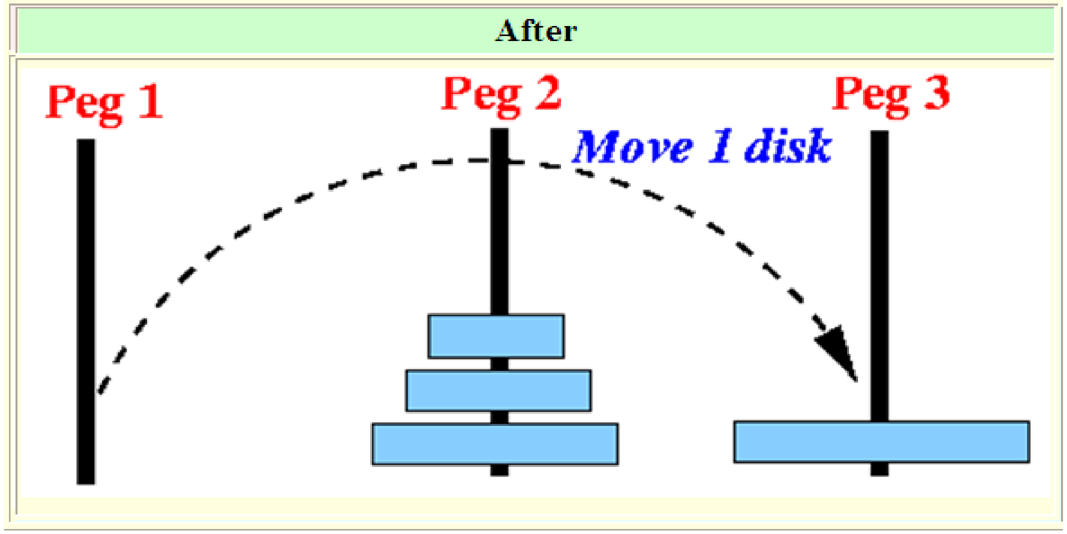
```
enter the disks: 1 2 3
('source: ', [[1, 2, 3]]) ('helper: ', []) ('target: ', [])
('source: ', []) ('helper: ', []) ('target: ', [[1, 2, 3]])
```

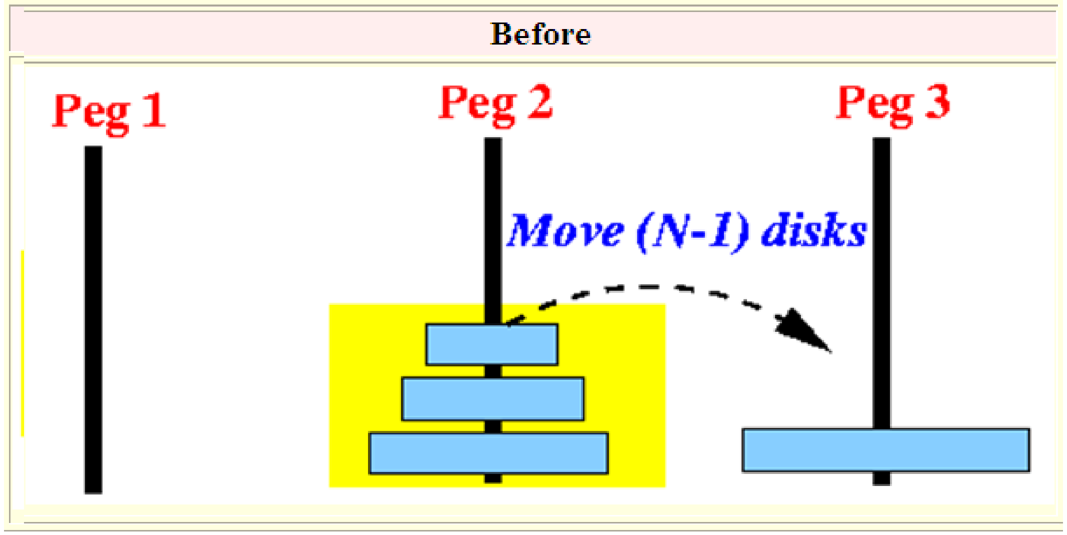

Peg 1
Peg 2
Peg 3
Move N disks

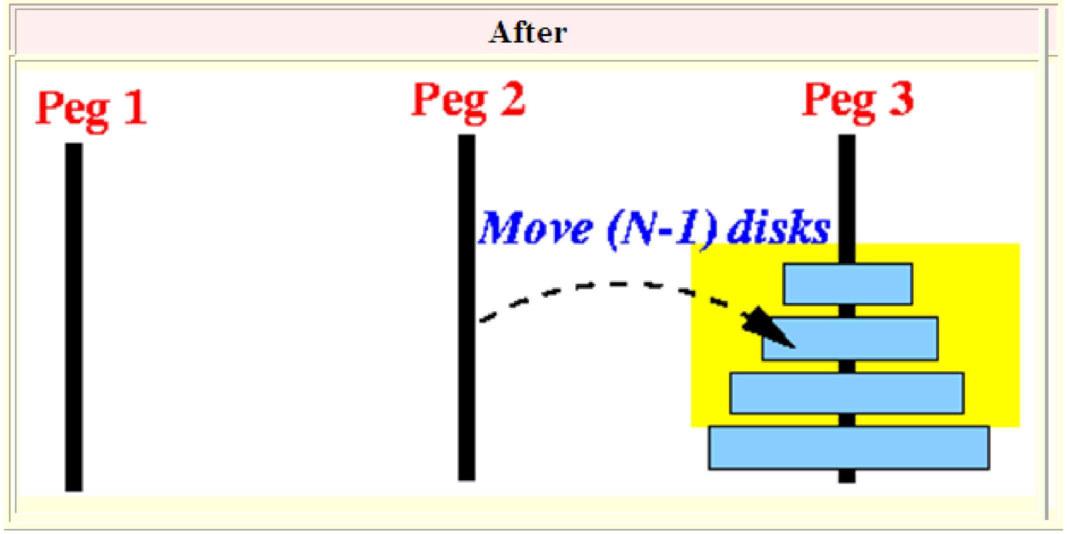

Before
Peg 1
Peg 2
Peg 3
Move (N-1) disks

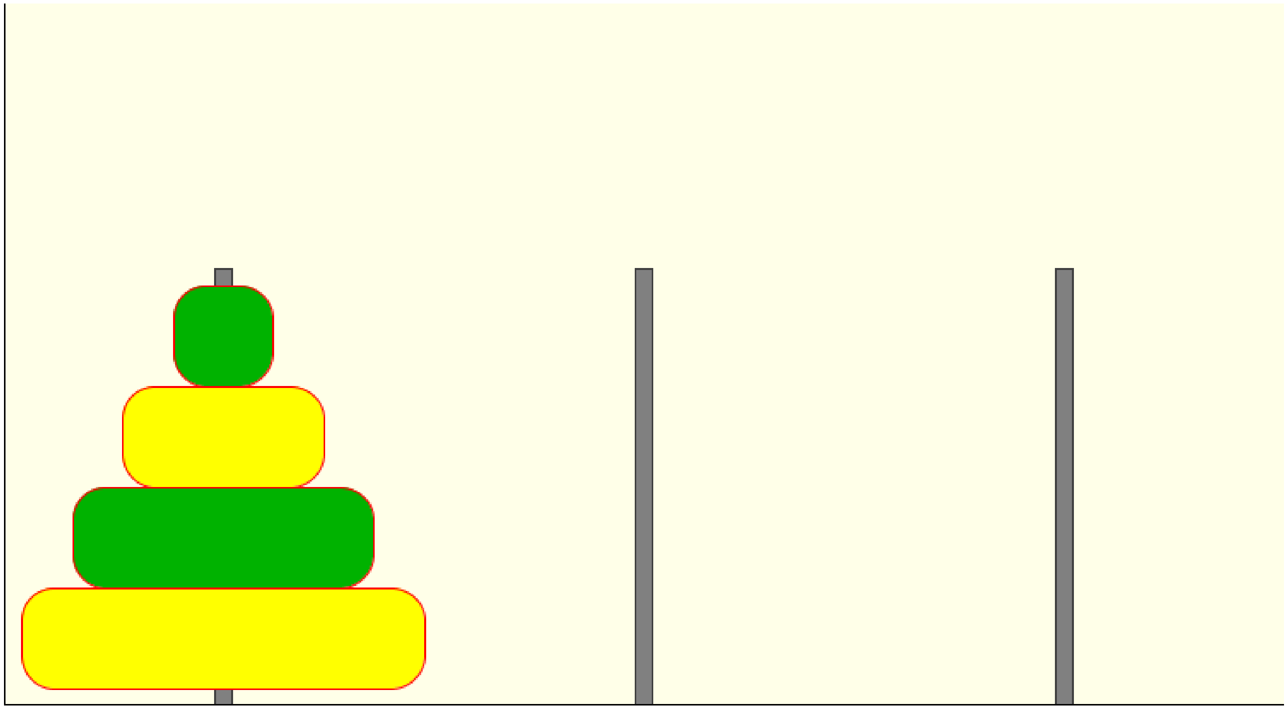

After
Peg 1
Peg 2
Peg 3
Move (N-1) disks

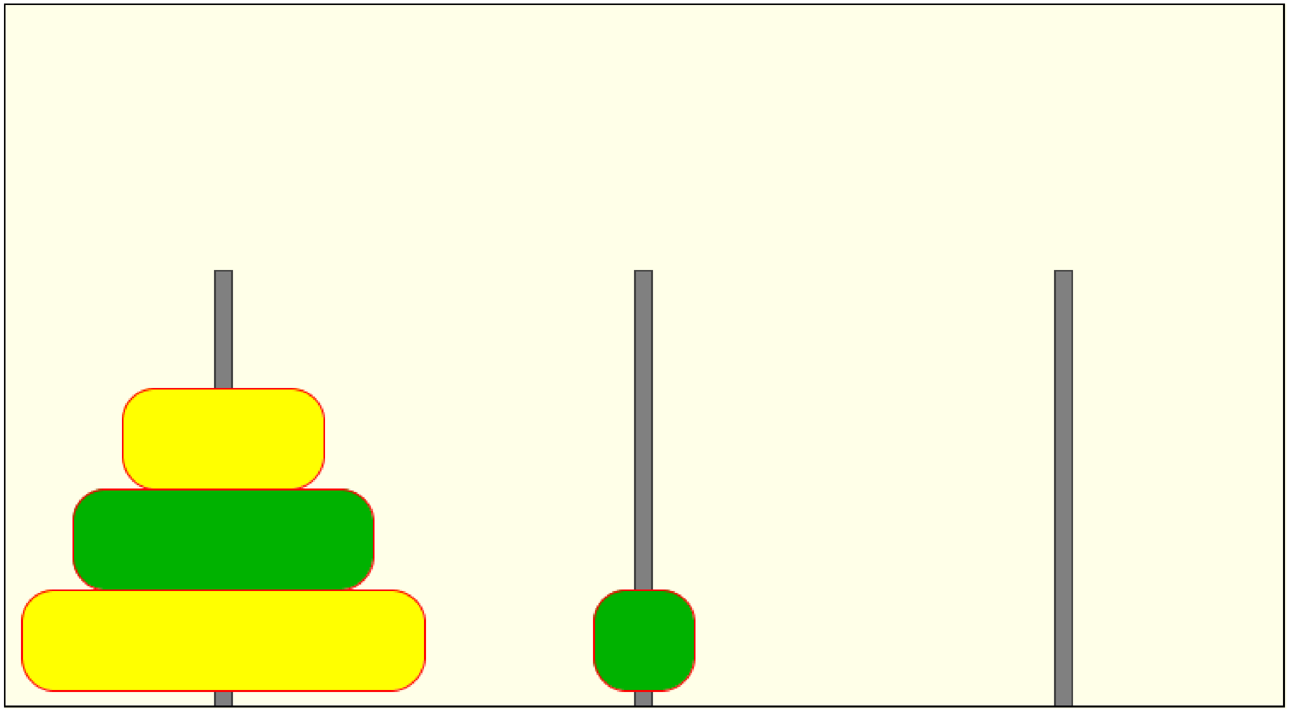

Before
Peg 1
Peg 2
Peg 3
Move 1 disk

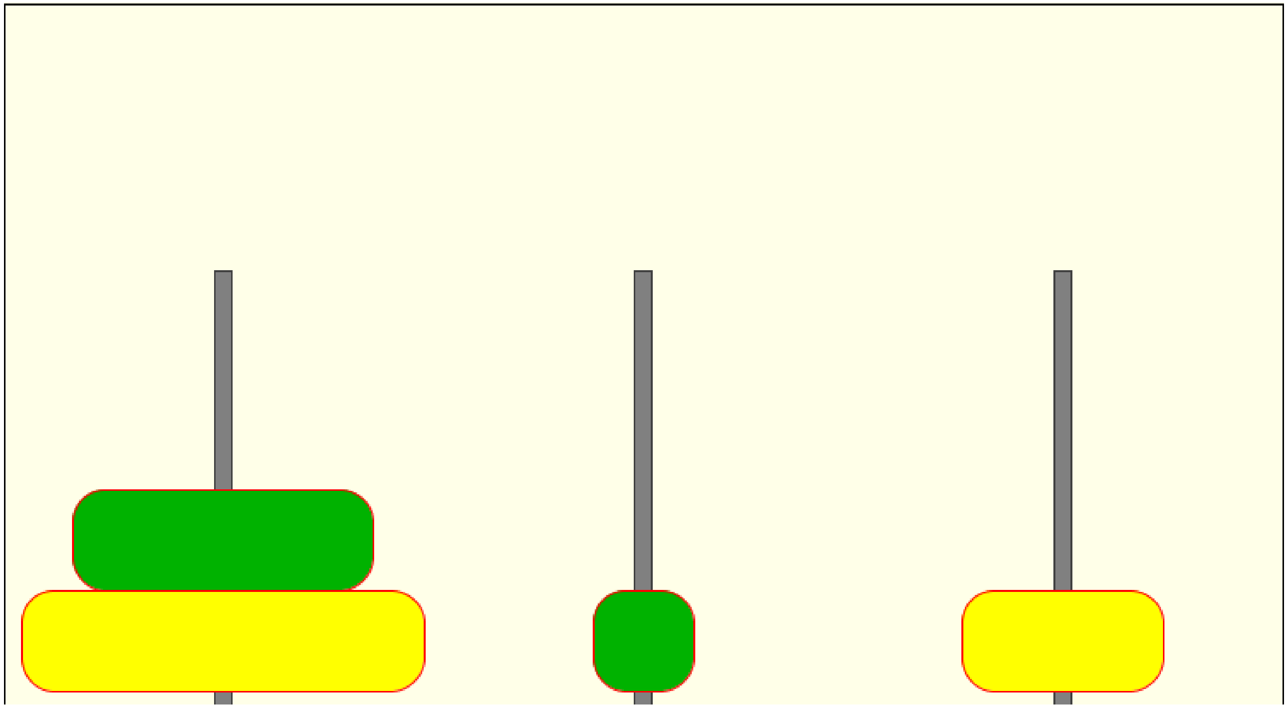

After
Peg 1
Peg 2
Peg 3
Move 1 disk

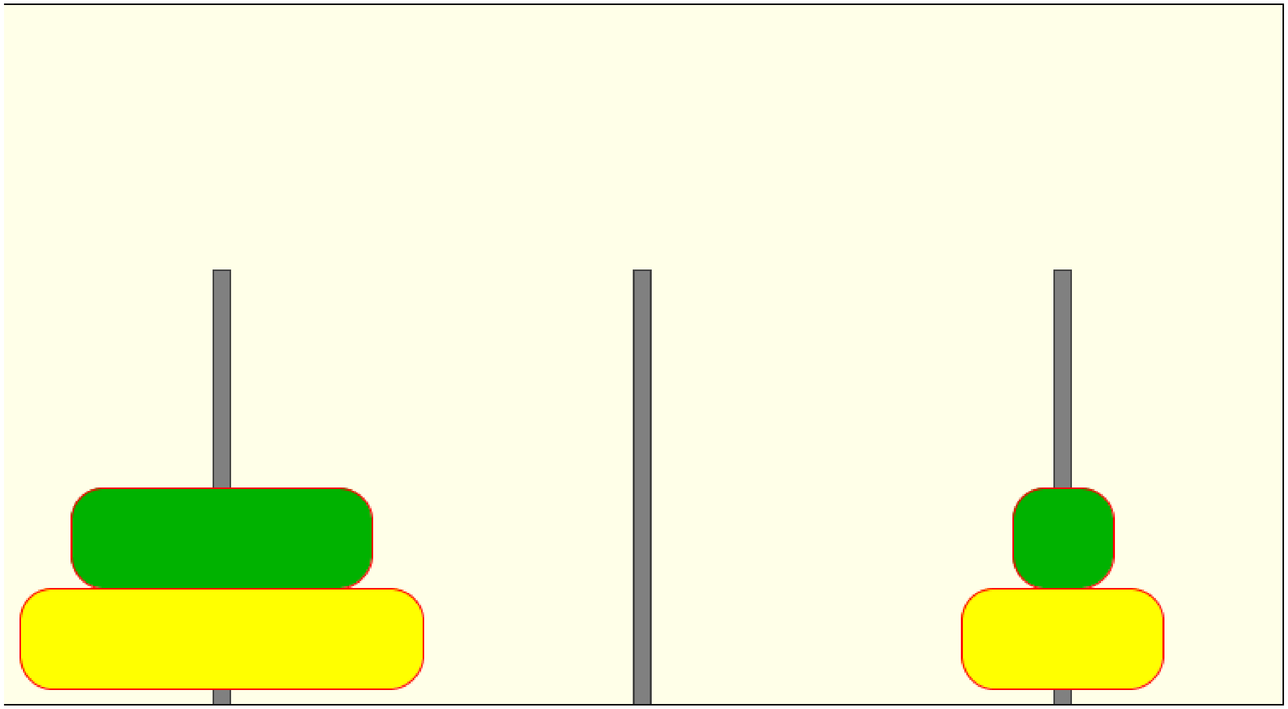

Before
Peg 1
Peg 2
Peg 3
Move (N-1) disks

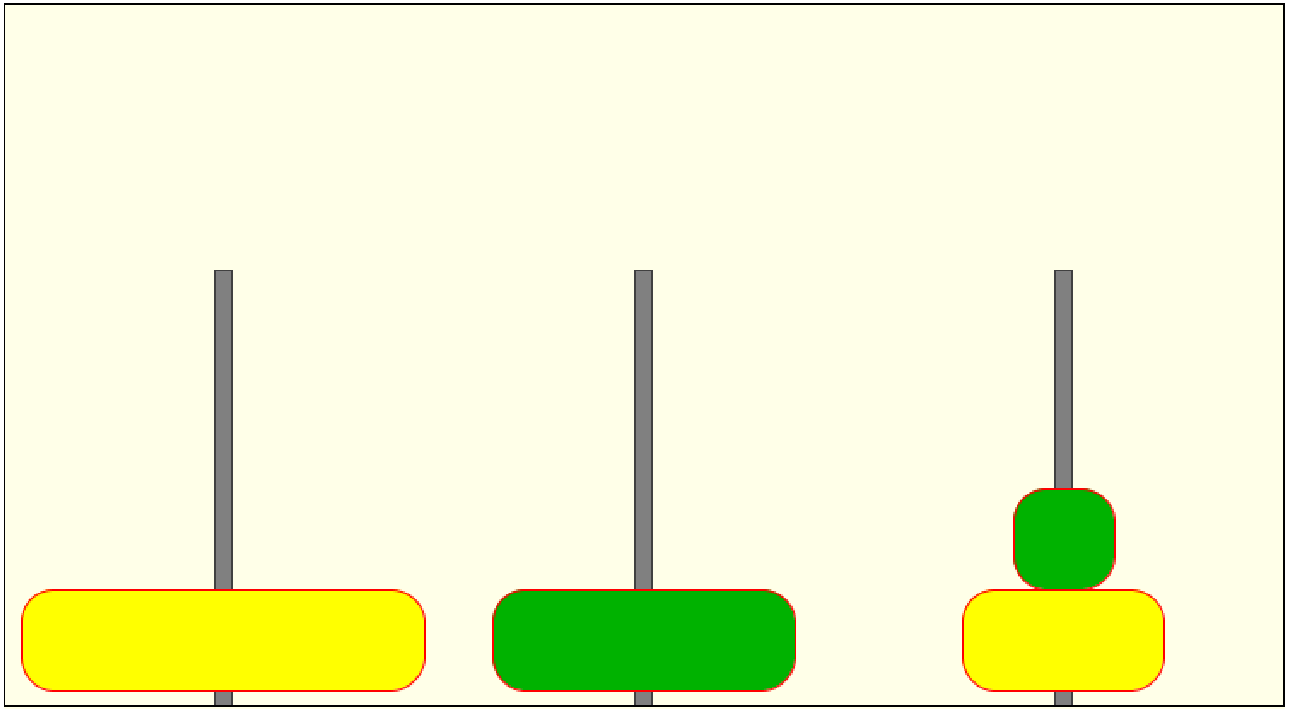

After
Peg 1
Peg 2
Peg 3
Move (N-1) disks

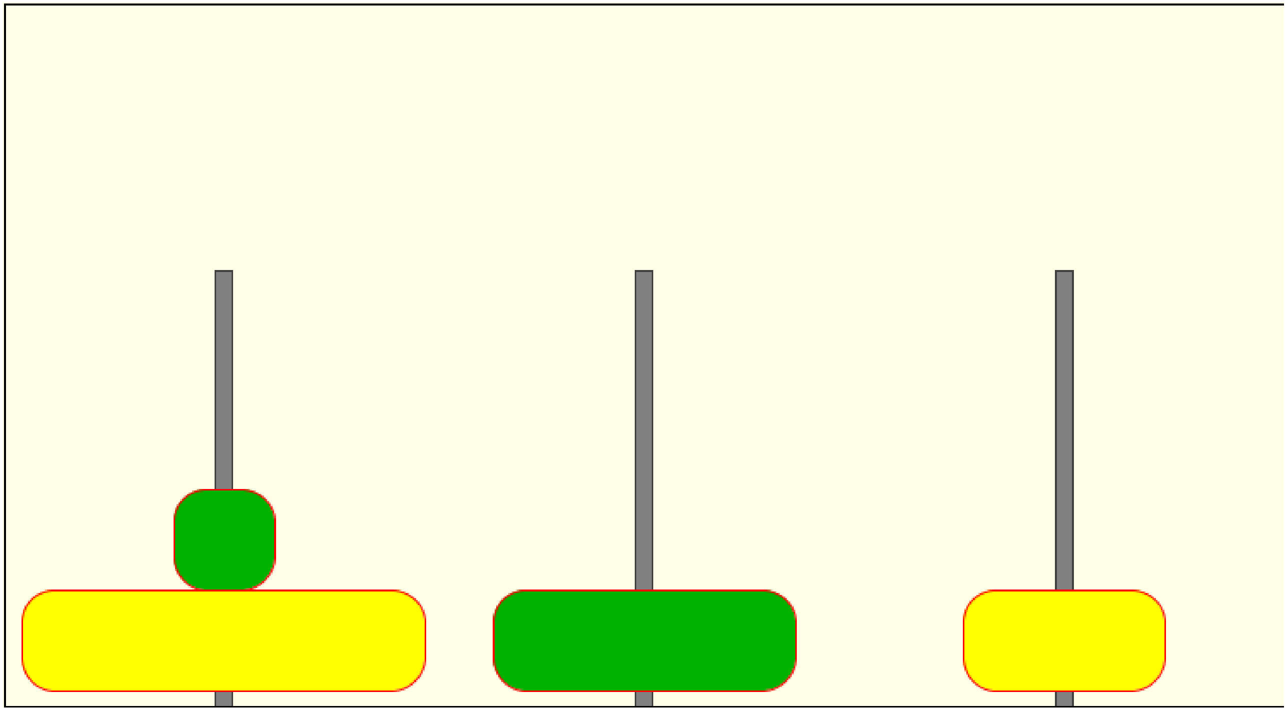


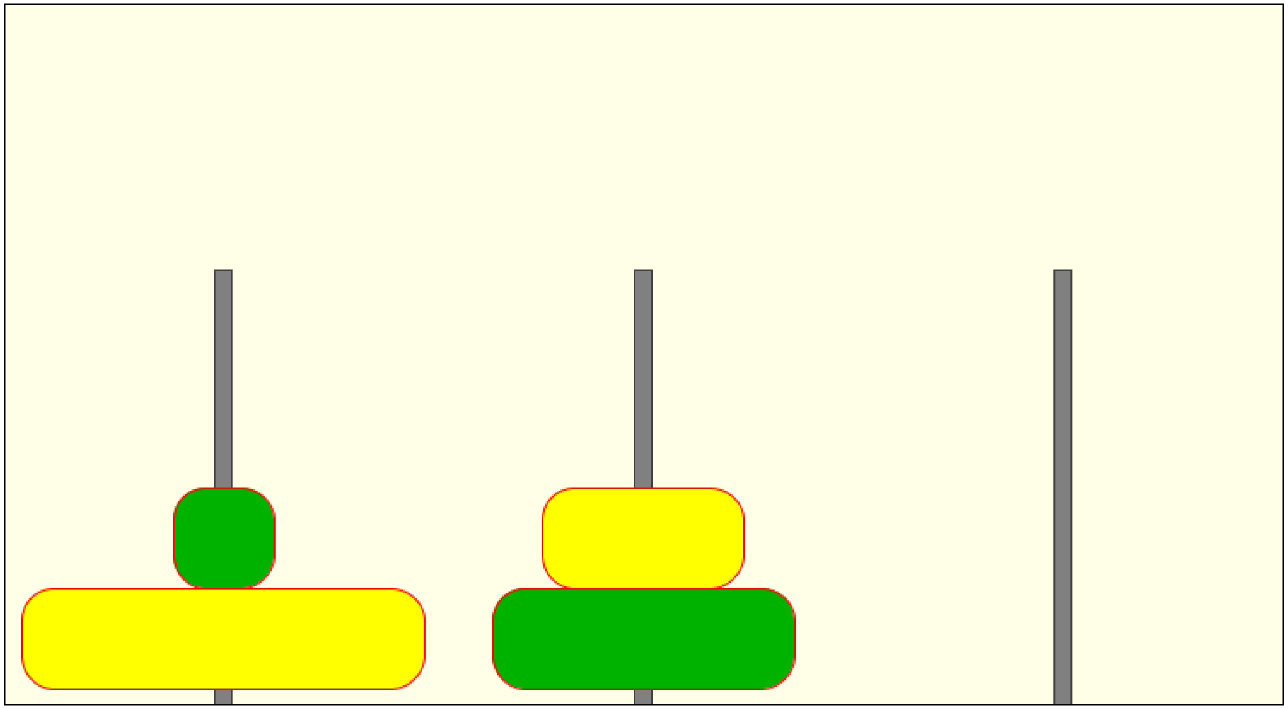


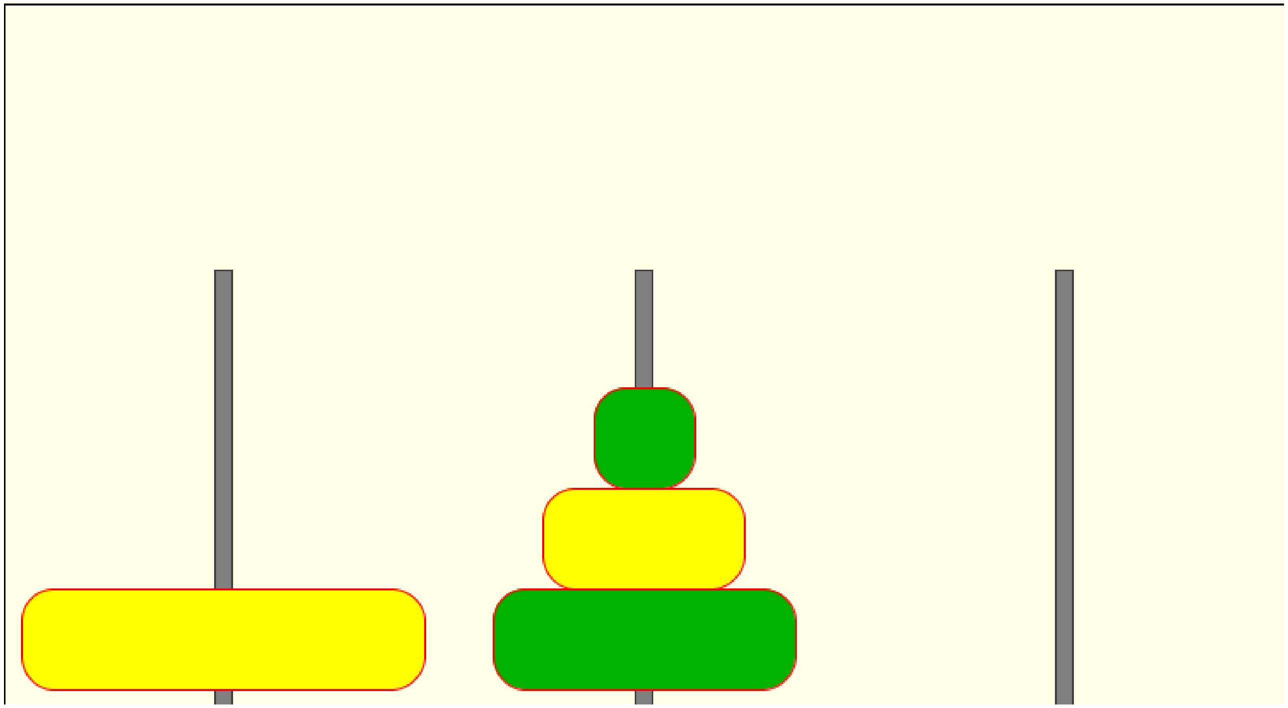


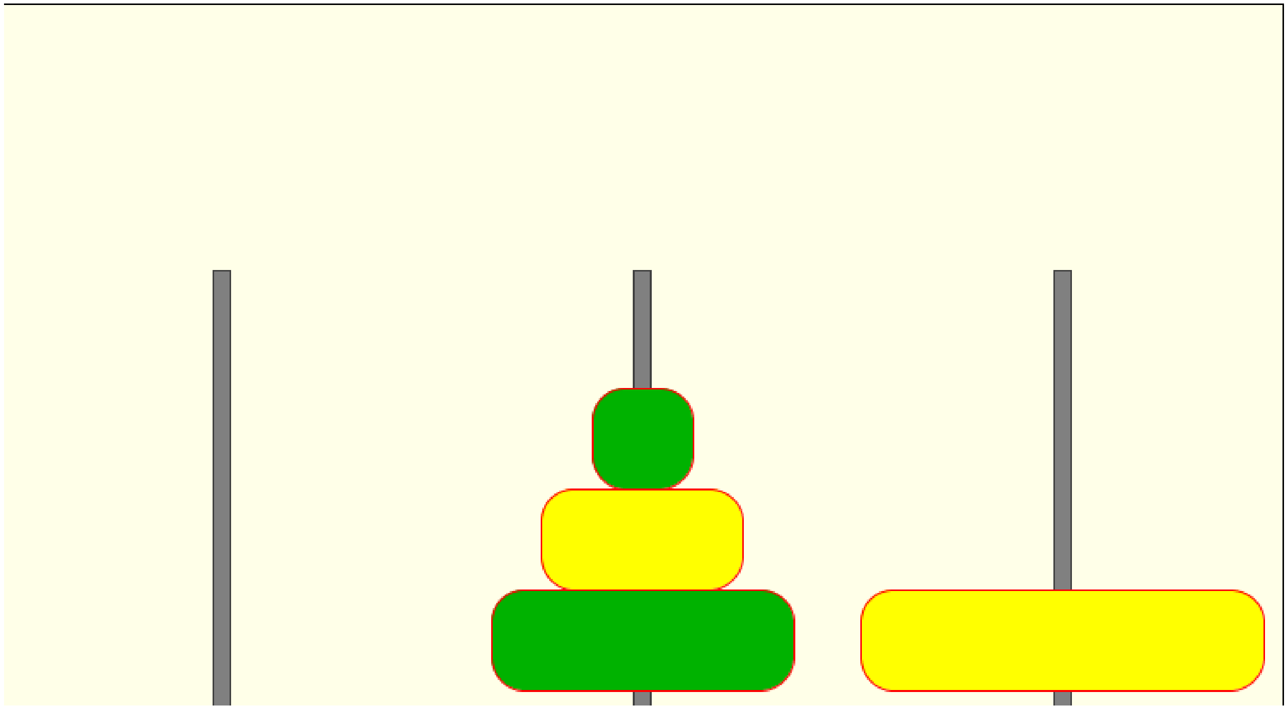


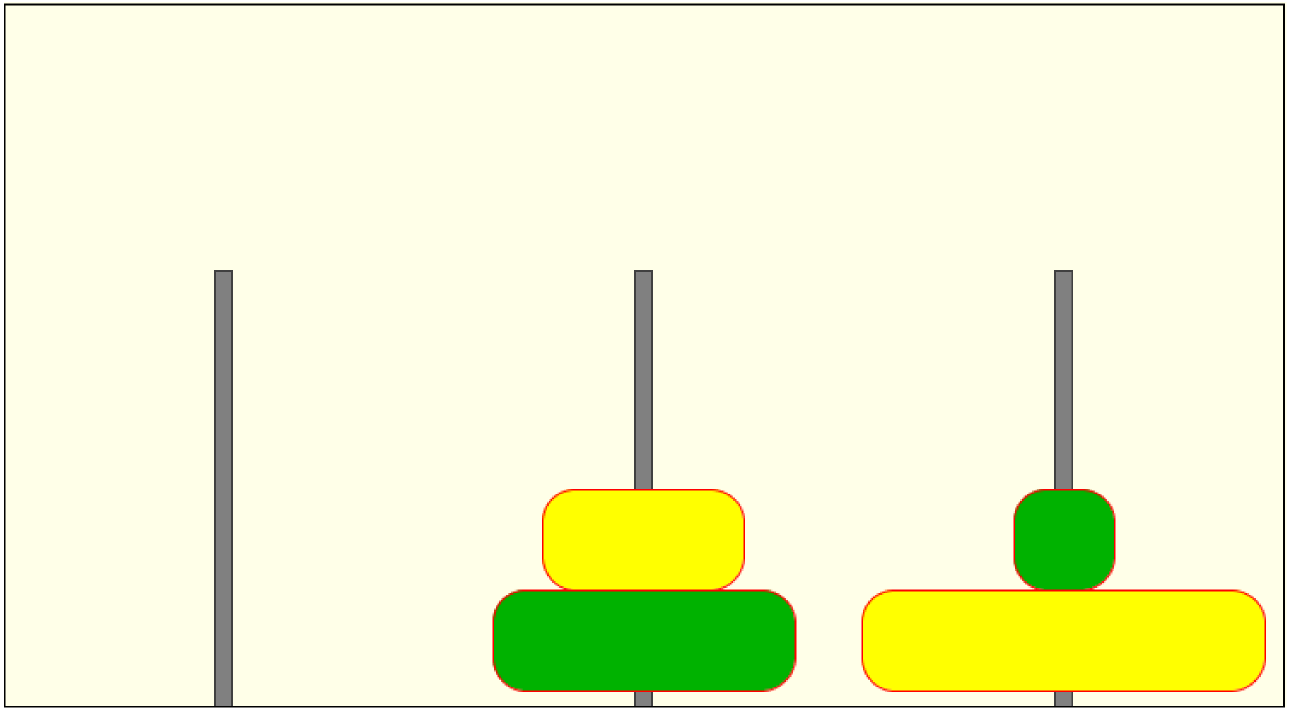


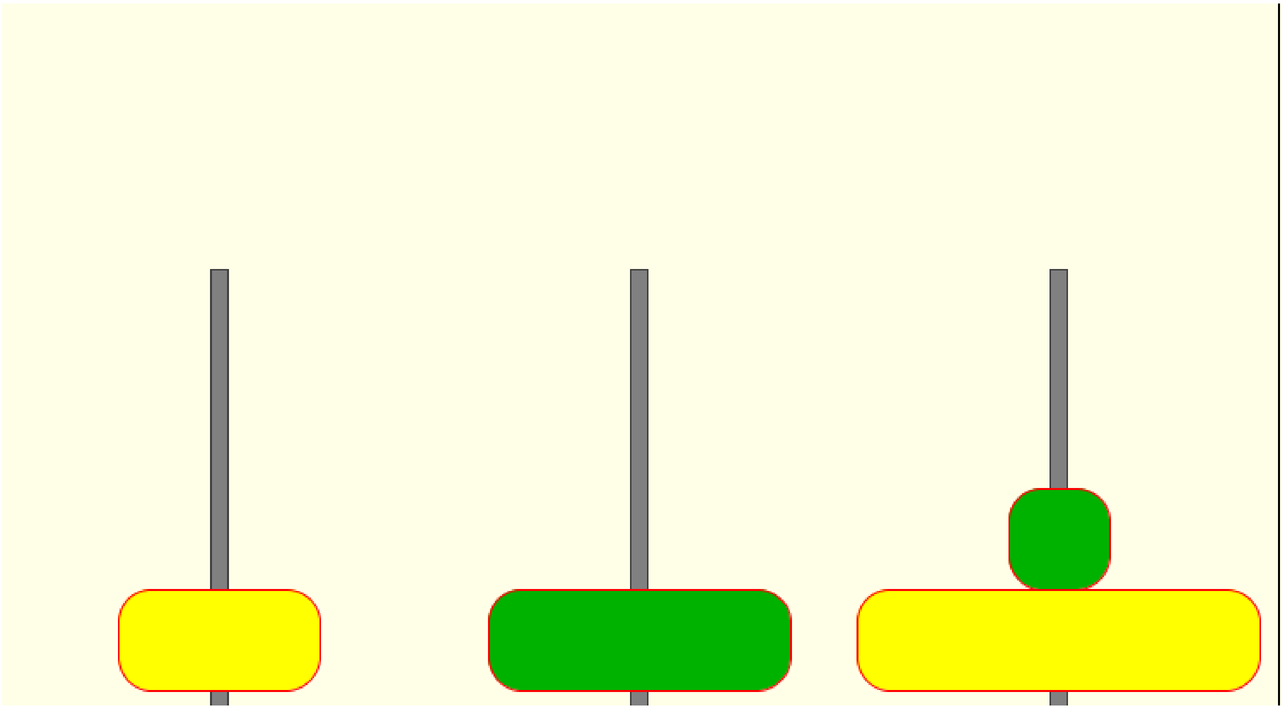


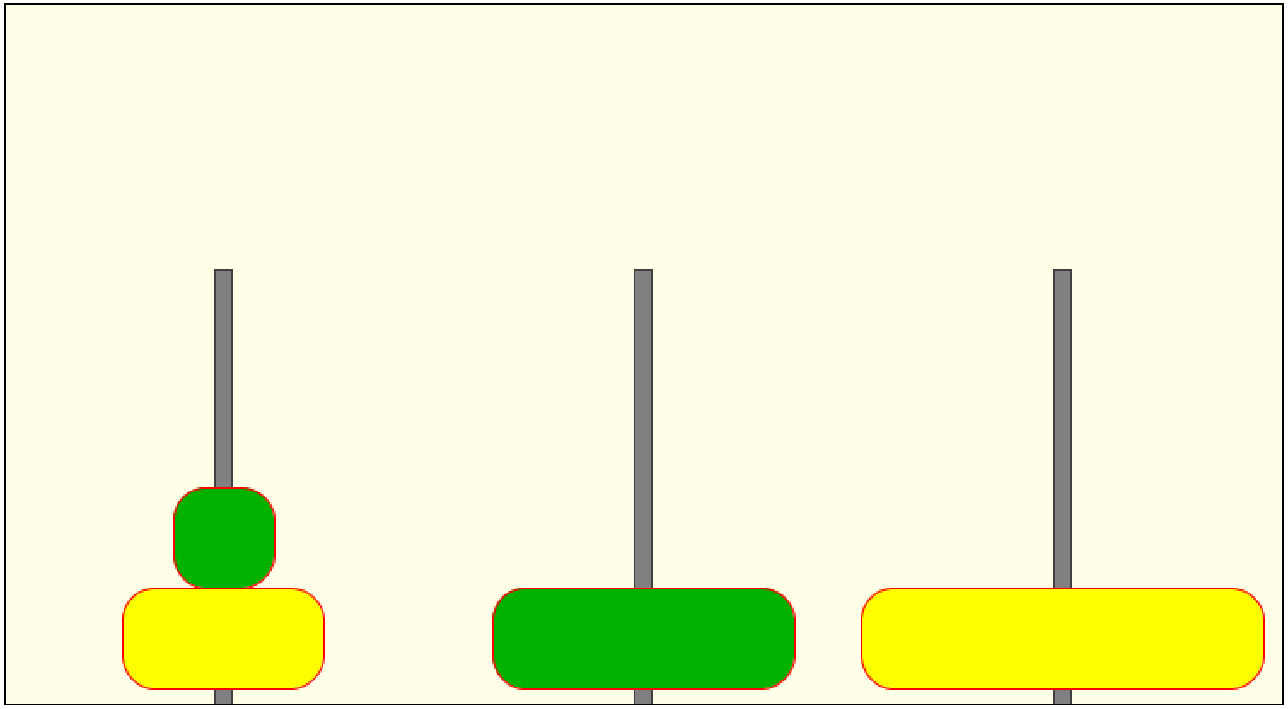


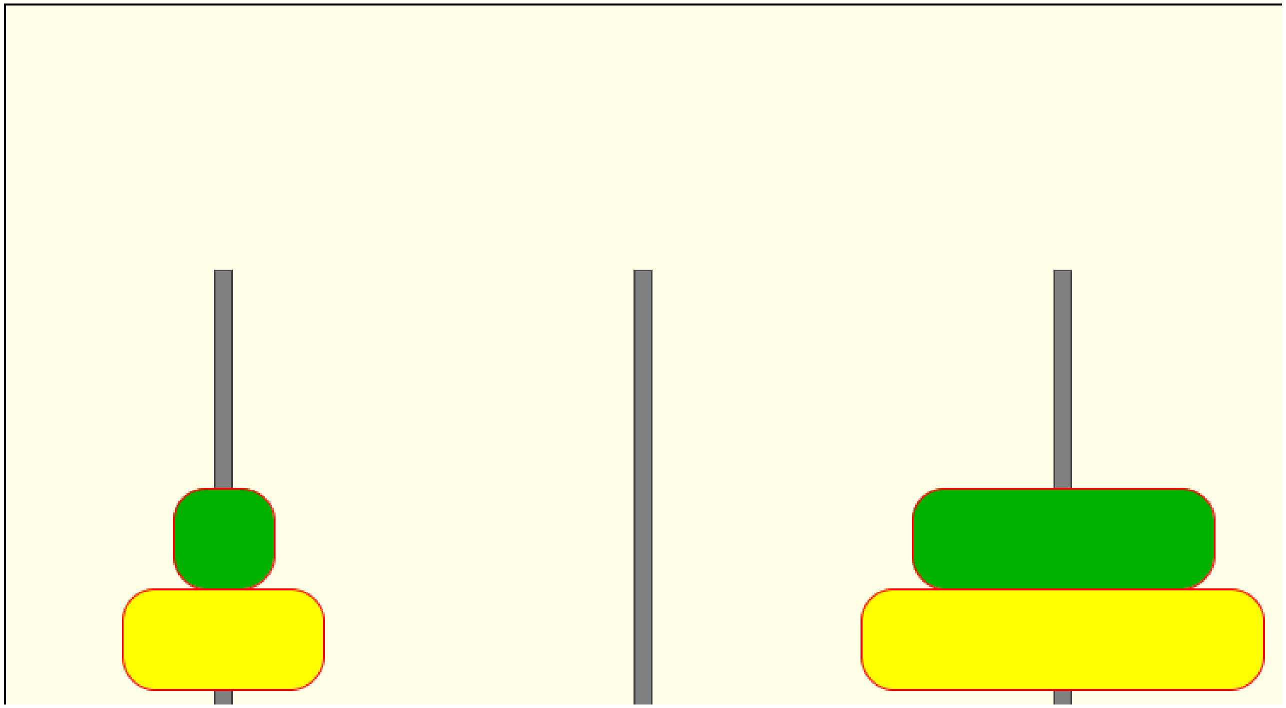


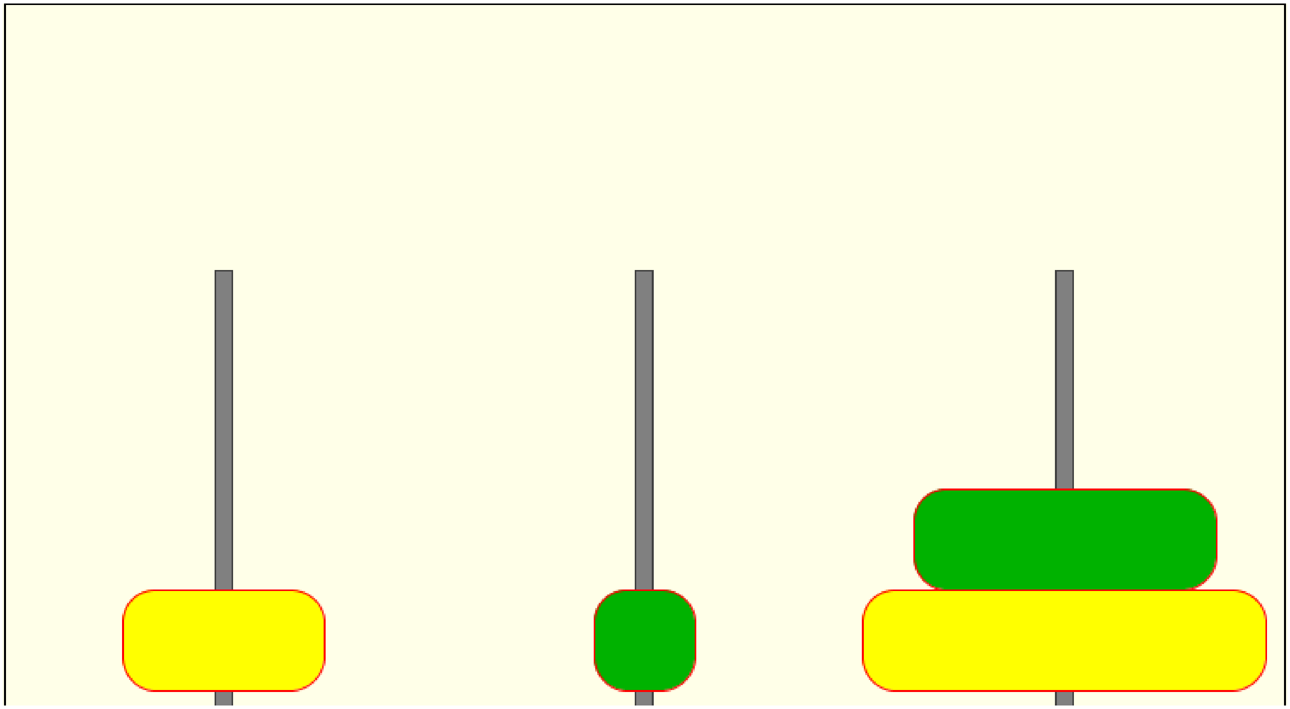


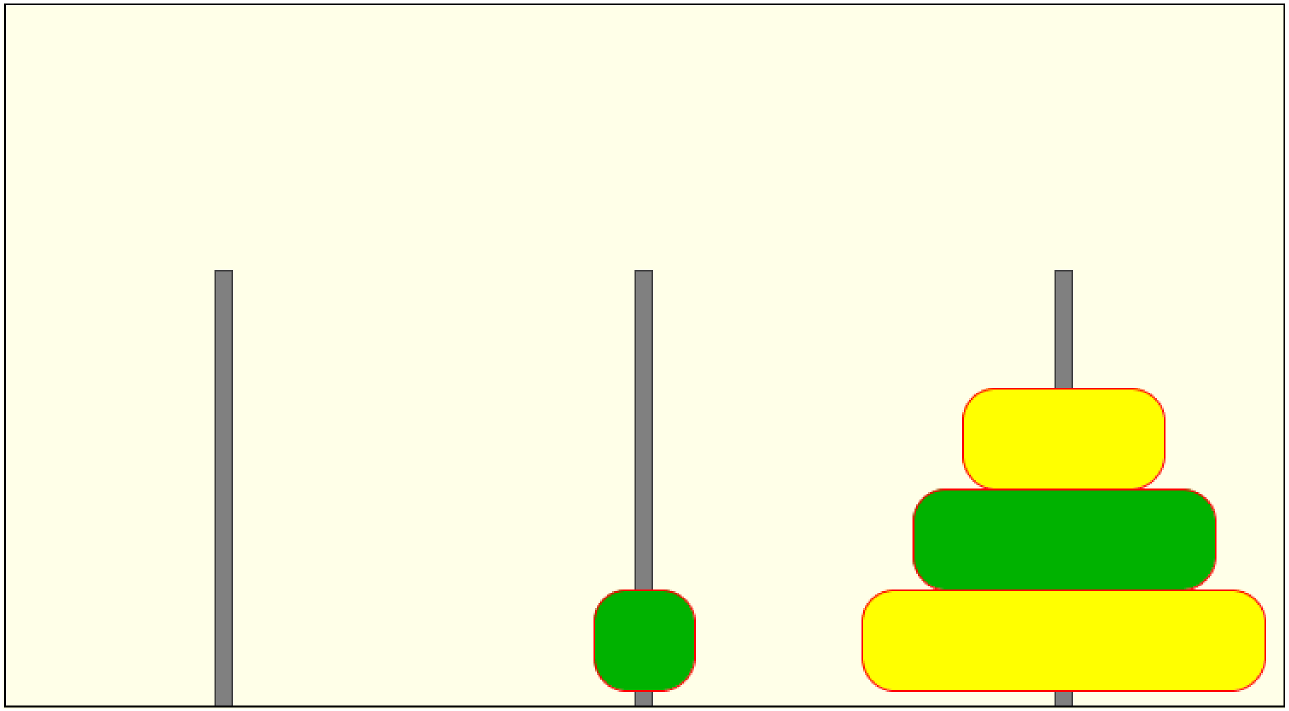


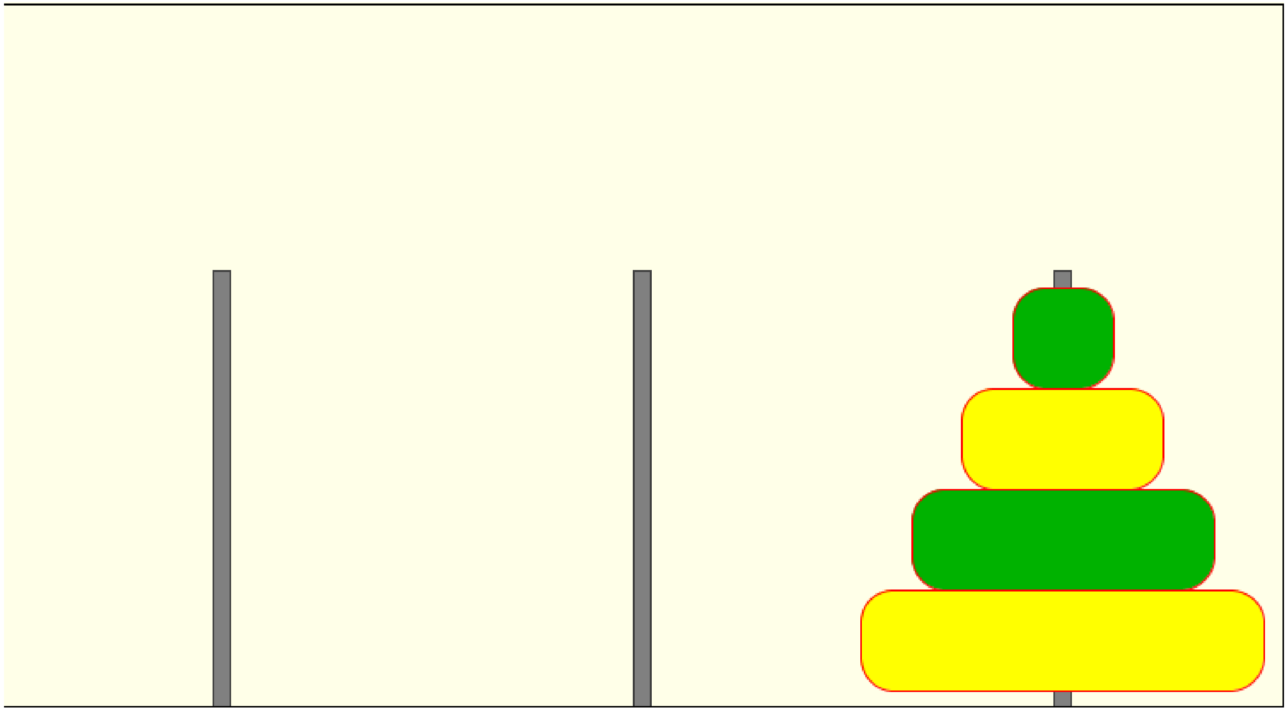


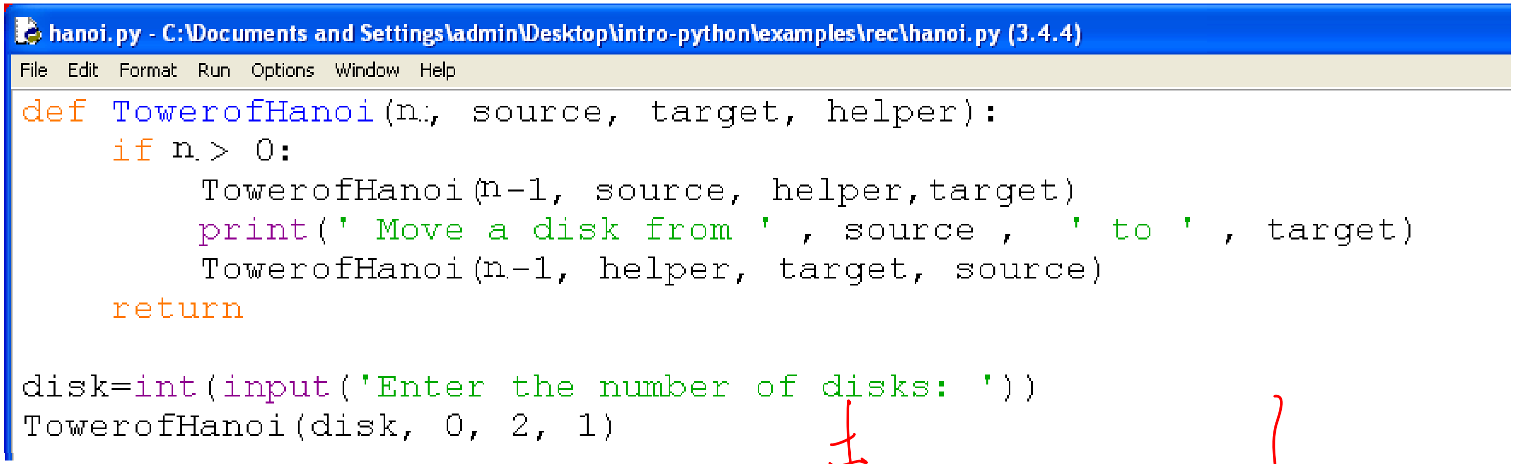






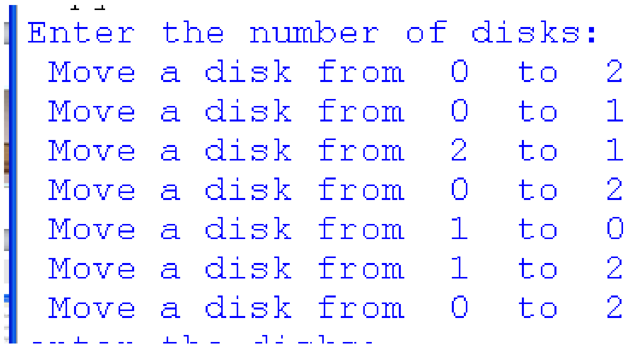




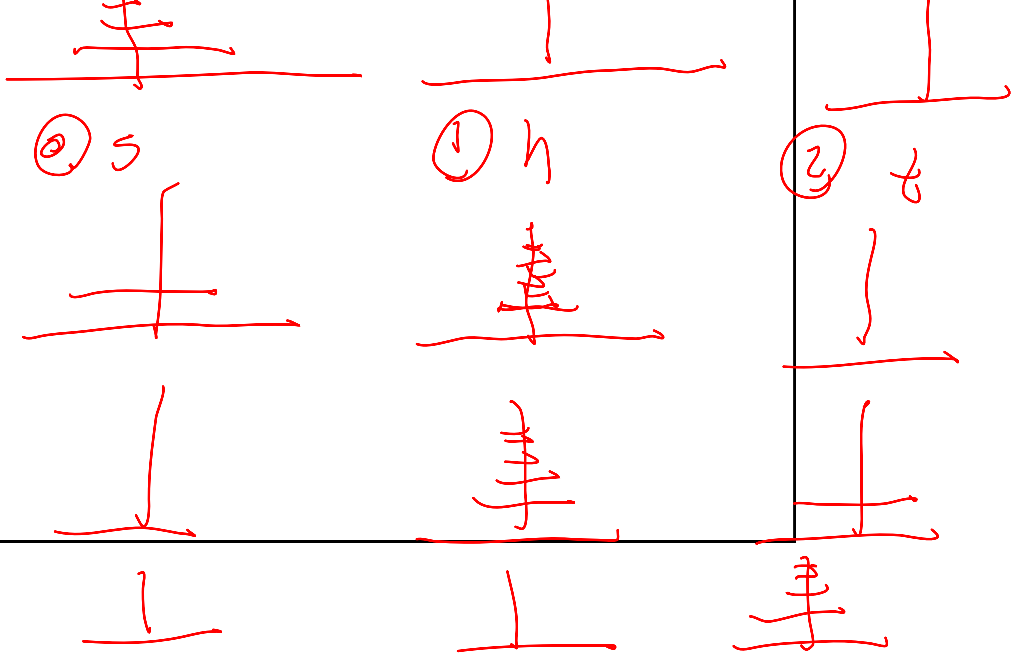


hanoi.py - C:\Documents and Settings\admin\Desktop\intro-python\examples\rec\hanoi.py (3.4.4)
File Edit Format Run Options Window Help

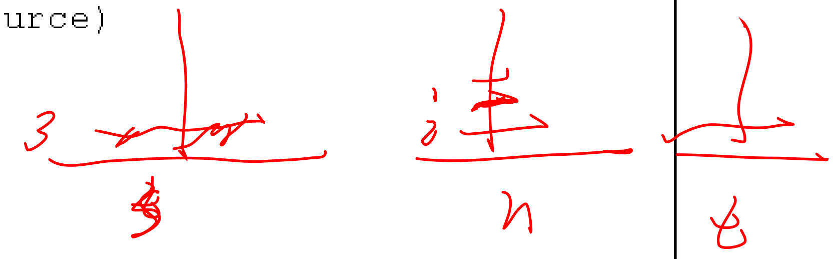
```
def TowerofHanoi(n, source, target, helper):
    if n > 0:
        TowerofHanoi(n-1, source, helper,target)
        print(' Move a disk from ' , source ,  ' to ' , target)
        TowerofHanoi(n-1, helper, target, source)
    return

disk=int(input('Enter the number of disks: '))
TowerofHanoi(disk, 0, 2, 1)
```



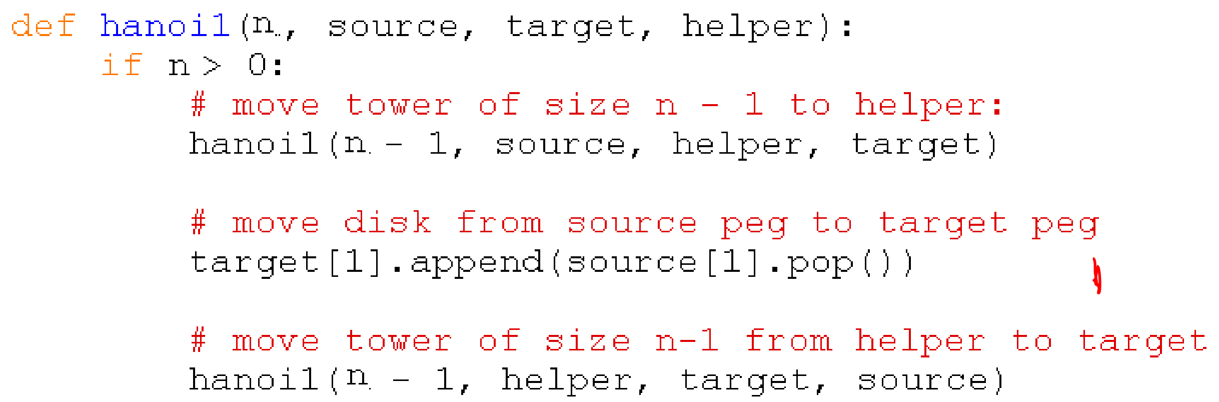
```
Enter the number of disks: 3
 Move a disk from  0  to  2
 Move a disk from  0  to  1
 Move a disk from  2  to  1
 Move a disk from  0  to  2
 Move a disk from  1  to  0
 Move a disk from  1  to  2
 Move a disk from  0  to  2
```



```
def hanoi1(n, source, target, helper):
    if n > 0:
        # move tower of size n - 1 to helper:
        hanoi1(n - 1, source, helper, target)

        # move disk from source peg to target peg
        target[1].append(source[1].pop())

        # move tower of size n-1 from helper to target
        hanoi1(n - 1, helper, target, source)

  source = ("source: ", [])
  target = ("target: ", [])
  helper = ("helper: ", [])
  print("enter the disks: ", end='')
  source[1].append(list(map(int,input().strip().split(" "))))
  print (source, helper, target)
  hanoi1(len(source[1]),source,target, helper)
  print (source, helper, target)
```



```
enter the disks: 1 2 3
('source: ', [[1, 2, 3]]) ('helper: ', []) ('target: ', [])
('source: ', []) ('helper: ', []) ('target: ', [[1, 2, 3]])
```



```
def hanoi1(n, source, target, helper):
    if n > 0:
        # move tower of size n - 1 to helper:
        hanoi1(n - 1, source, helper, target)

        # move disk from source peg to target peg
        target[1].append(source[1].pop())

        # move tower of size n-1 from helper to target
        hanoi1(n - 1, helper, target, source)
```

$T(n) \leq$

$T(0)=0$

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

$T(n)=2[2T(n-2)+1]+1=2^2T(n-2)+2+1$

$T(n)=2^2[2T(n-3)+1]+2+1=2^3T(n-3)+2^2+2+1$

$T(n)=2^3[2T(n-4)+1]+2^2+2+1=2^4T(n-4)+2^3+2^2+2+1$

…………………

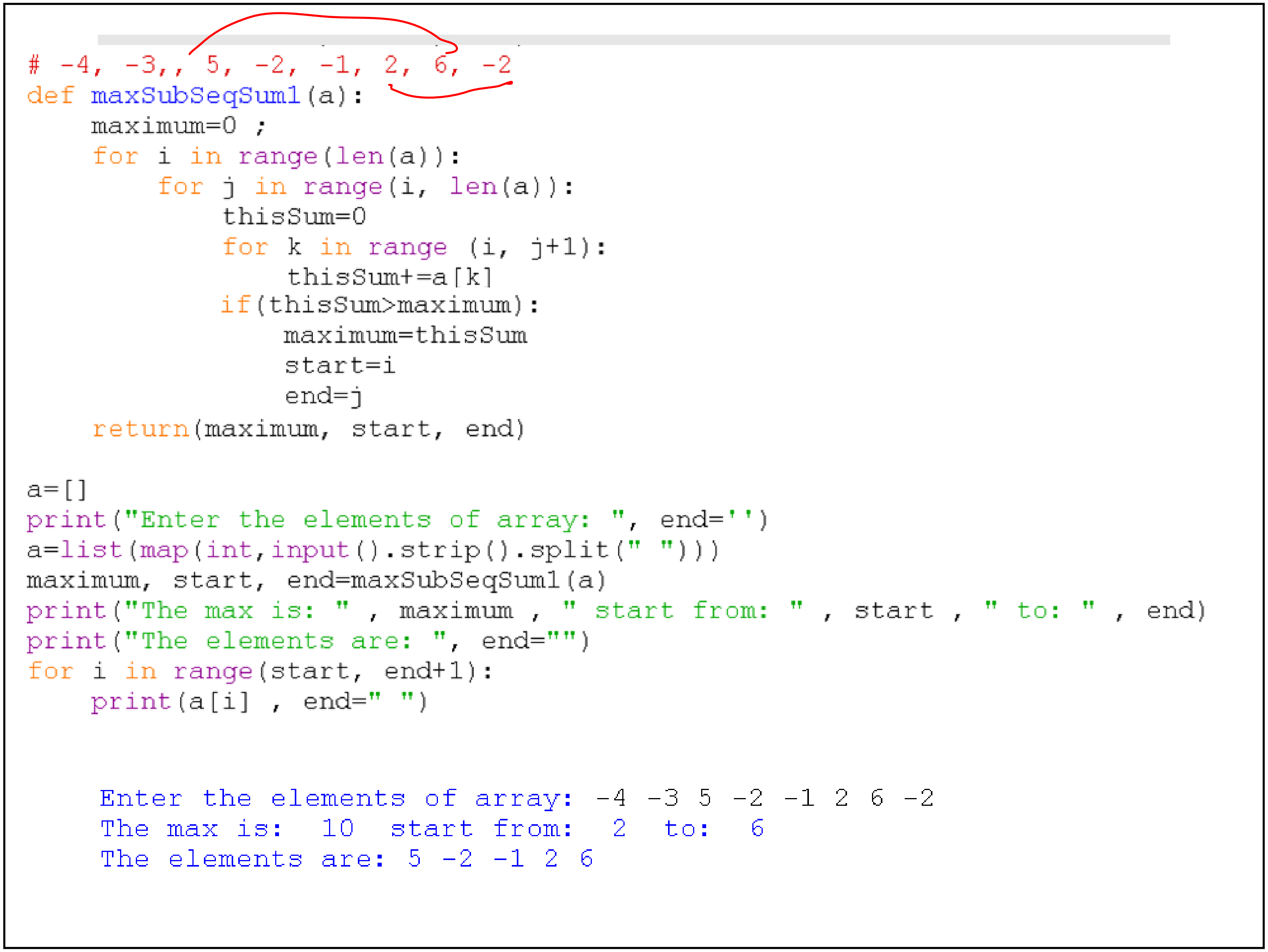
$T(n)=2^kT(n-k)+2^{k-1}+2^{k-2}+ \ldots .+ 2^2+2^1+2^0$

$k=n$

$T(n)=2^nT(0)+2^{n-1}+2^{n-2}+ \ldots .+ 2^2+2^1+2^0$

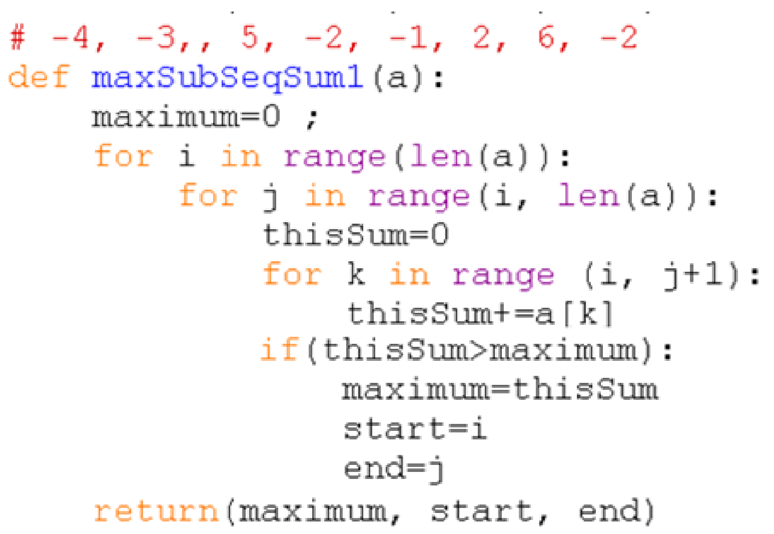
$T(n)=2^{n-1}+2^{n-2}+ \ldots .+ 2^2+2^1+2^0$

$T(n)=2^n-1=O(2^n)$



```
# -4, -3,, 5, -2, -1, 2, 6, -2
def maxSubSeqSum1(a):
    maximum=0 ;
    for i in range(len(a)):
        for j in range(i, len(a)):
            thisSum=0
            for k in range (i, j+1):
                thisSum+=a[k]
            if(thisSum>maximum):
                maximum=thisSum
                start=i
                end=j
    return(maximum, start, end)

a=[]
print("Enter the elements of array: ", end='')
a=list(map(int,input().strip().split(" ")))
maximum, start, end=maxSubSeqSum1(a)
print("The max is: " , maximum , " start from: " , start , " to: " , end)
print("The elements are: ", end="")
for i in range(start, end+1):
    print(a[i] , end=" ")
```

```
Enter the elements of array: -4 -3 5 -2 -1 2 6 -2
The max is:  10  start from:  2  to:  6
The elements are: 5 -2 -1 2 6
```



```
# -4, -3,, 5, -2, -1, 2, 6, -2
def maxSubSeqSum1(a):
    maximum=0 ;
    for i in range(len(a)):
        for j in range(i, len(a)):
            thisSum=0
            for k in range (i, j+1):
                thisSum+=a[k]
            if(thisSum>maximum):
                maximum=thisSum
                start=i
                end=j
    return(maximum, start, end)
```

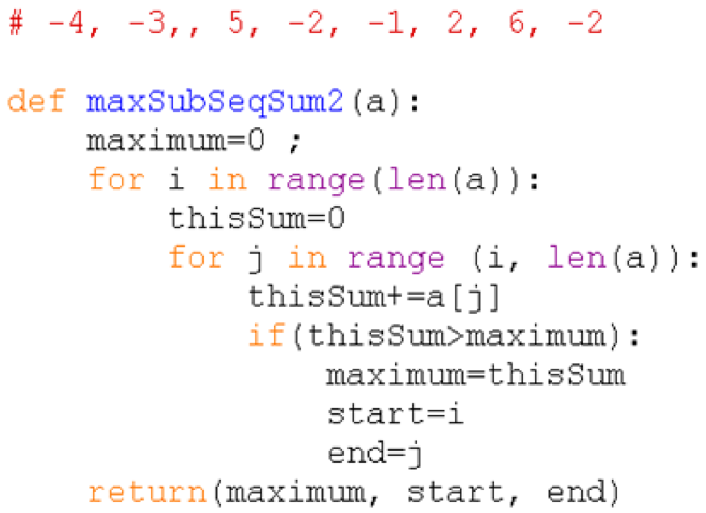
$O(n^3)$

M	t
0	-4
5	-7
	-2
	5
	-3
	2
	0

-4 -3 5 -2 -1 2 6 -2

i j j j j

j j j j



```
# -4, -3,, 5, -2, -1, 2, 6, -2

def maxSubSeqSum2(a):
    maximum=0 ;
    for i in range(len(a)):
        thisSum=0
        for j in range (i, len(a)):
            thisSum+=a[j]
            if(thisSum>maximum):
                maximum=thisSum
                start=i
                end=j
    return(maximum, start, end)
```

$O(n^2)$

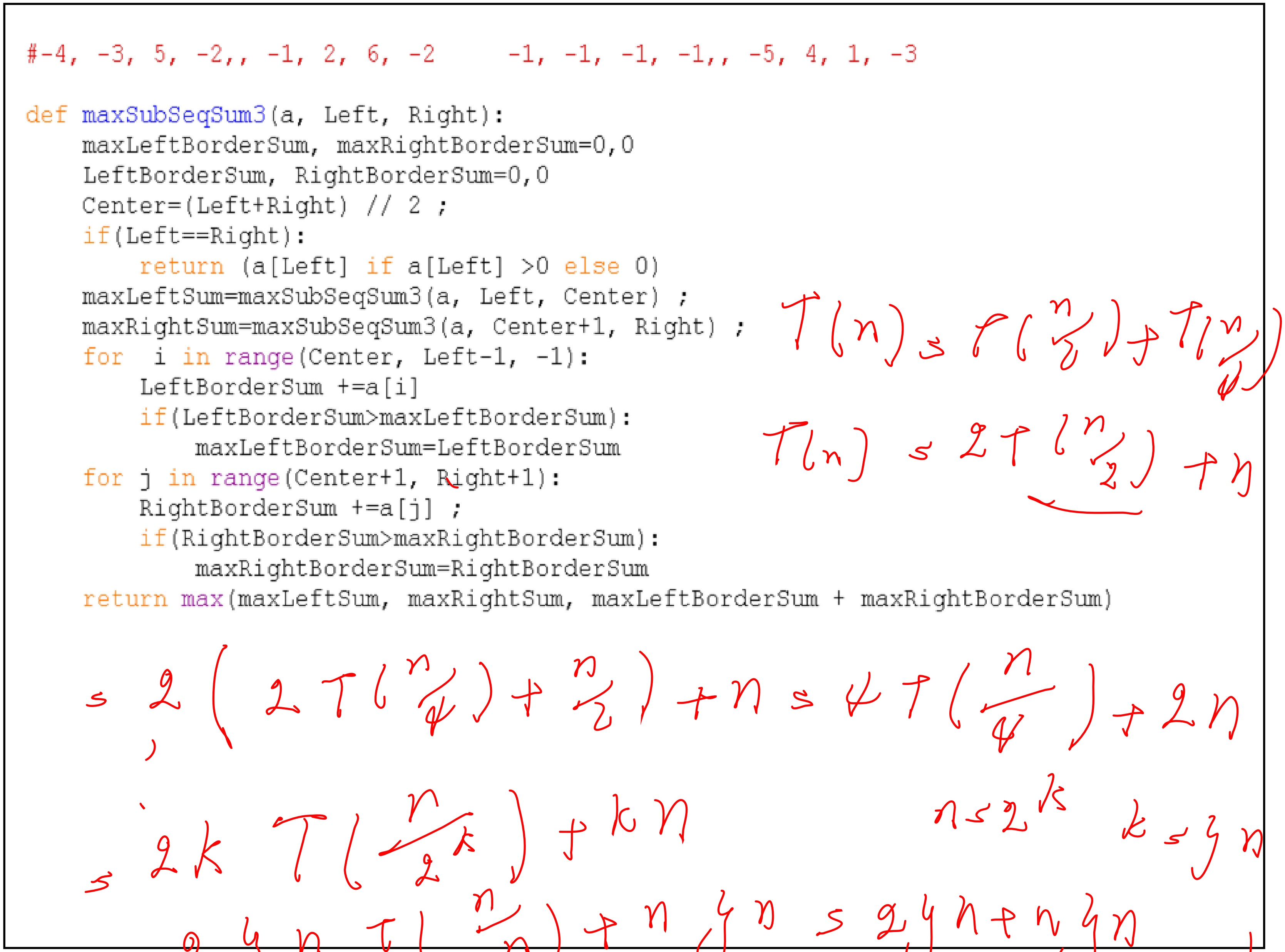
m	t
5	0
10	5
	3
	2
	4
	18

-4 -3 5 -2 -1 2 6 -2

i j j j j

s	e
2	6

```
#-4, -3, 5, -2,, -1, 2, 6, -2     -1, -1, -1, -1,, -5, 4, 1, -3

def maxSubSeqSum3(a, Left, Right):
    maxLeftBorderSum, maxRightBorderSum=0,0
    LeftBorderSum, RightBorderSum=0,0
    Center=(Left+Right) // 2 ;
    if(Left==Right):
        return (a[Left] if a[Left] >0 else 0)
    maxLeftSum=maxSubSeqSum3(a, Left, Center) ;
    maxRightSum=maxSubSeqSum3(a, Center+1, Right) ;
    for  i in range(Center, Left-1, -1):
        LeftBorderSum +=a[i]
        if(LeftBorderSum>maxLeftBorderSum):
            maxLeftBorderSum=LeftBorderSum
    for j in range(Center+1, Right+1):
        RightBorderSum +=a[j] ;
        if(RightBorderSum>maxRightBorderSum):
            maxRightBorderSum=RightBorderSum
    return max(maxLeftSum, maxRightSum, maxLeftBorderSum + maxRightBorderSum)


a=[]
print("Enter the elements of array: ", end='')
a=list(map(int,input().strip().split(" ")))
max1=maxSubSeqSum3(a, 0, len(a)-1)
print("The max is: " , max1 , " start from: " , start , " to: " , end)
print("The elements are: ", end="")
for i in range(start, end+1):
    print(a[i] , end=" ")
```



```
#-4, -3, 5, -2,, -1, 2, 6, -2     -1, -1, -1, -1,, -5, 4, 1, -3

def maxSubSeqSum3(a, Left, Right):
    maxLeftBorderSum, maxRightBorderSum=0,0
    LeftBorderSum, RightBorderSum=0,0
    Center=(Left+Right) // 2 ;
    if(Left==Right):
        return (a[Left] if a[Left] >0 else 0)
    maxLeftSum=maxSubSeqSum3(a, Left, Center) ;
    maxRightSum=maxSubSeqSum3(a, Center+1, Right) ;
    for  i in range(Center, Left-1, -1):
        LeftBorderSum +=a[i]
        if(LeftBorderSum>maxLeftBorderSum):
            maxLeftBorderSum=LeftBorderSum
    for j in range(Center+1, Right+1):
        RightBorderSum +=a[j] ;
        if(RightBorderSum>maxRightBorderSum):
            maxRightBorderSum=RightBorderSum
    return max(maxLeftSum, maxRightSum, maxLeftBorderSum + maxRightBorderSum)
```

$T(n) = T(\frac{n}{2}) + T(\frac{n}{2}) + n$

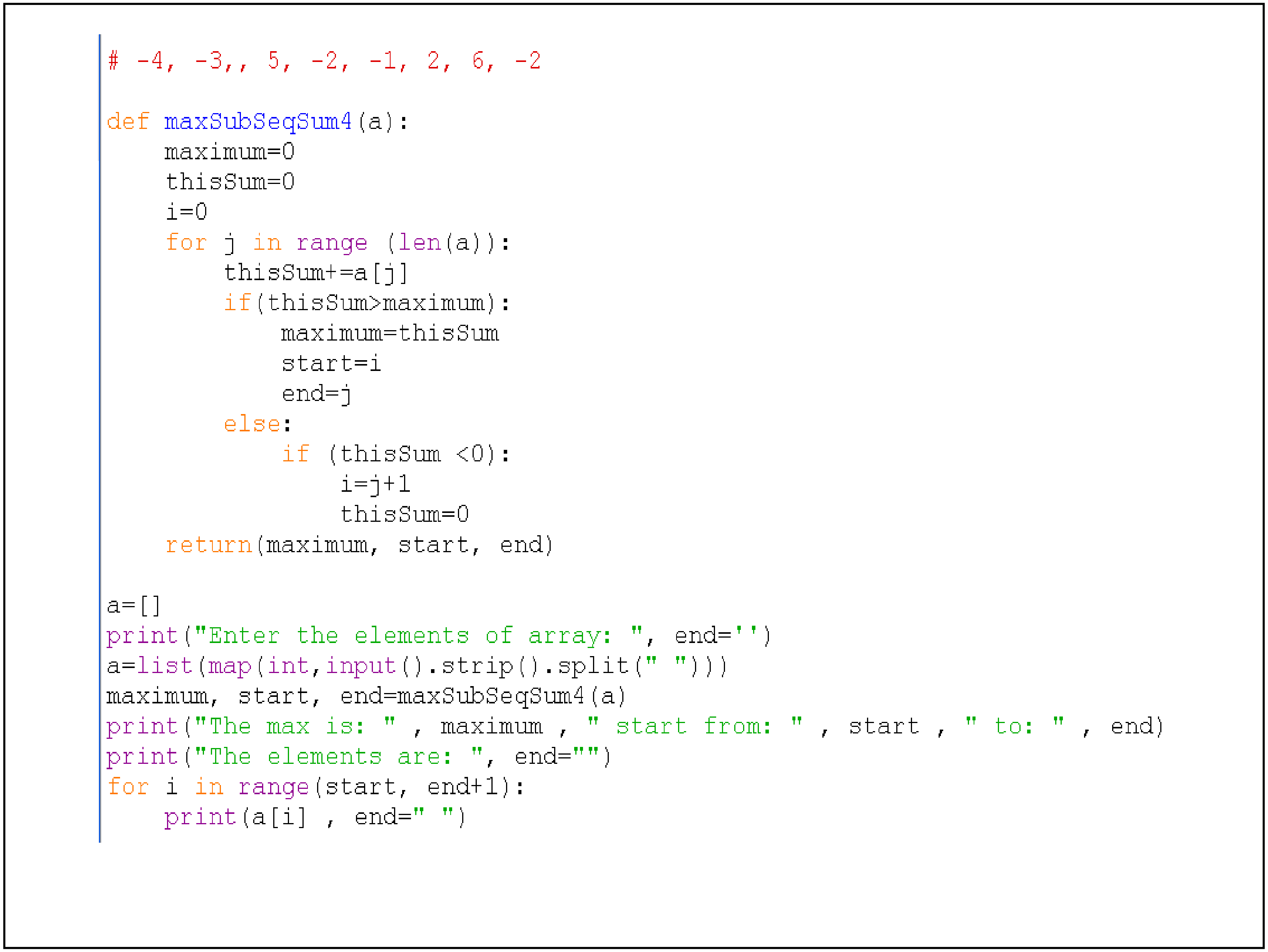
$T(n) = 2\underbrace{T(\frac{n}{2})} + n$

$= 2(2T(\frac{n}{4}) + \frac{n}{2}) + n = 4T(\frac{n}{4}) + 2n$ $k=2$

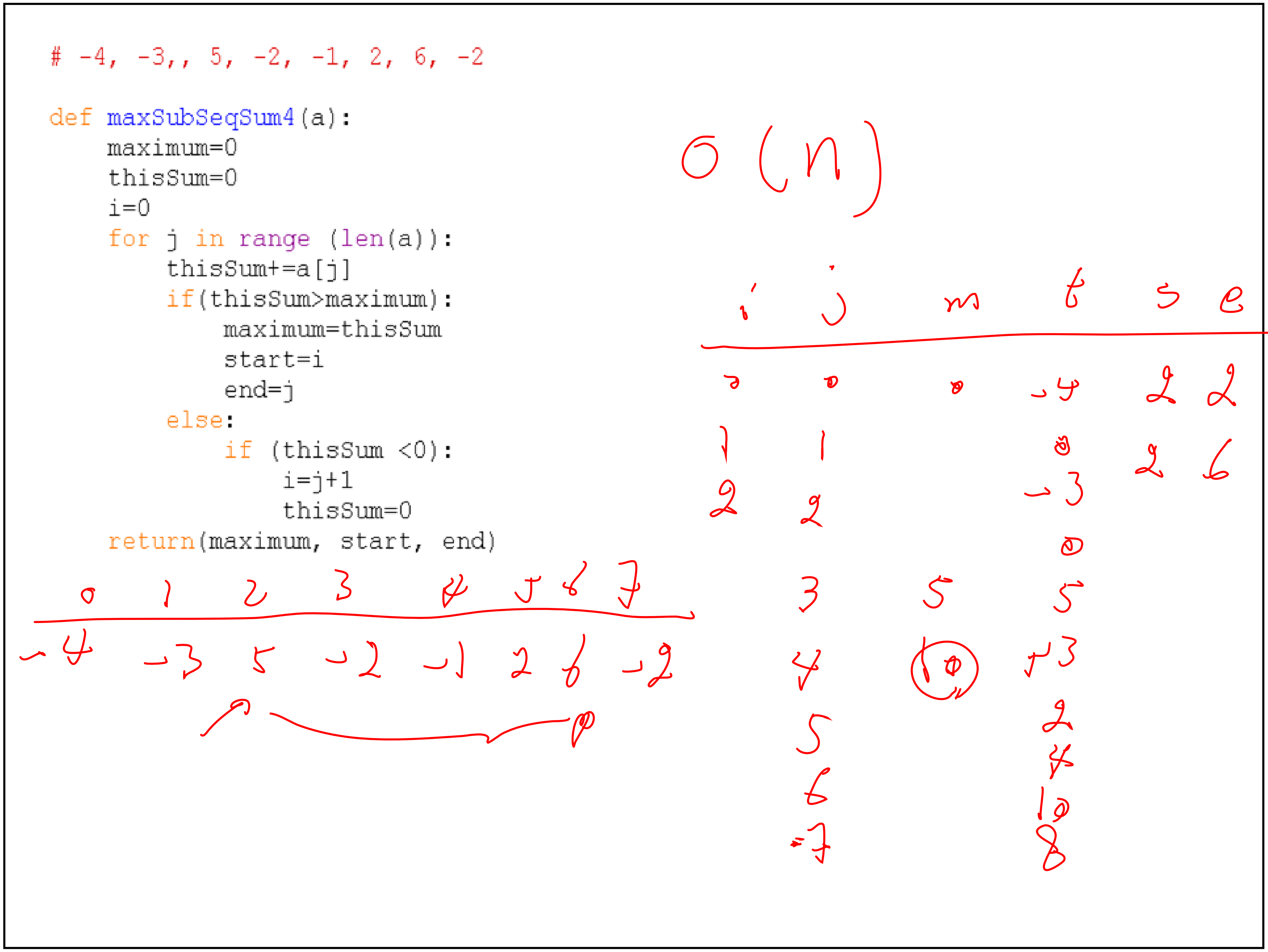
$= 2^k T(\frac{n}{2^k}) + kn$ $n = 2^k$ $k = \log n$

$= 2^{\log n} \underbrace{T(\frac{n}{n})} + n\log n = 2^{\log n} + n\log n$

$O(n\log n)$



```
# -4, -3,, 5, -2, -1, 2, 6, -2

def maxSubSeqSum4(a):
    maximum=0
    thisSum=0
    i=0
    for j in range (len(a)):
        thisSum+=a[j]
        if(thisSum>maximum):
            maximum=thisSum
            start=i
            end=j
        else:
            if (thisSum <0):
                i=j+1
                thisSum=0
    return(maximum, start, end)

a=[]
print("Enter the elements of array: ", end='')
a=list(map(int,input().strip().split(" ")))
maximum, start, end=maxSubSeqSum4(a)
print("The max is: " , maximum , " start from: " , start , " to: " , end)
print("The elements are: ", end="")
for i in range(start, end+1):
    print(a[i] , end=" ")
```



```
# -4, -3,, 5, -2, -1, 2, 6, -2

def maxSubSeqSum4(a):
    maximum=0
    thisSum=0
    i=0
    for j in range (len(a)):
        thisSum+=a[j]
        if(thisSum>maximum):
            maximum=thisSum
            start=i
            end=j
        else:
            if (thisSum <0):
                i=j+1
                thisSum=0
    return(maximum, start, end)
```

0	1	2	3	4	5	6	7
-4	-3	5	-2	-1	2	6	-2

$2 \underbrace{\qquad\qquad}$ 6

$O(n)$

i	j	m	t	s	e
0	0	0	-4	2	2
1	1		0	2	6
2	2		-3		
			0		
	3	5	5		
	4	10	3		
	5		2		
			4		
	6		10		
	7		8		

End of Chapter