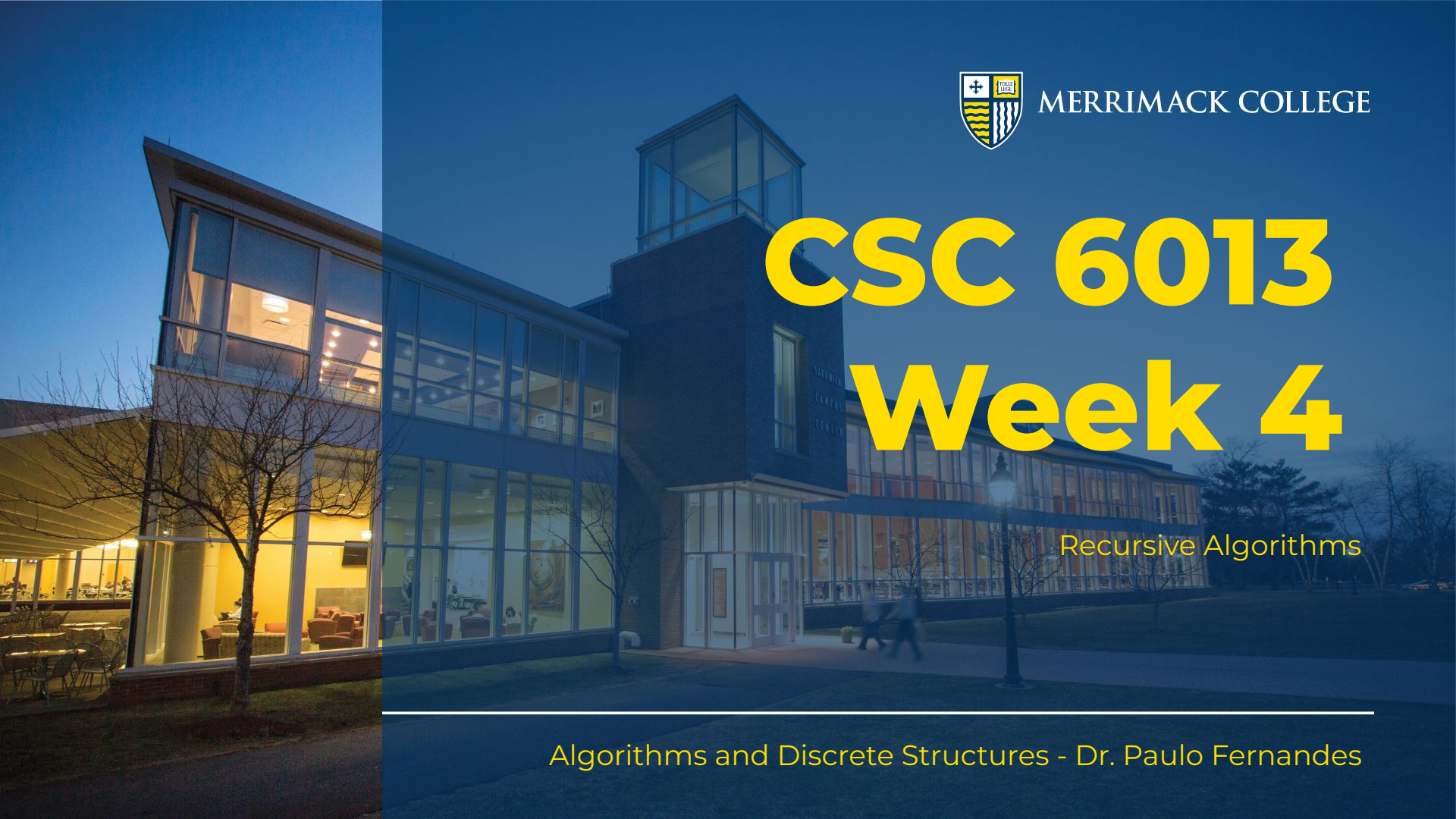


MERRIMACK COLLEGE

CSC 6013 Week 4

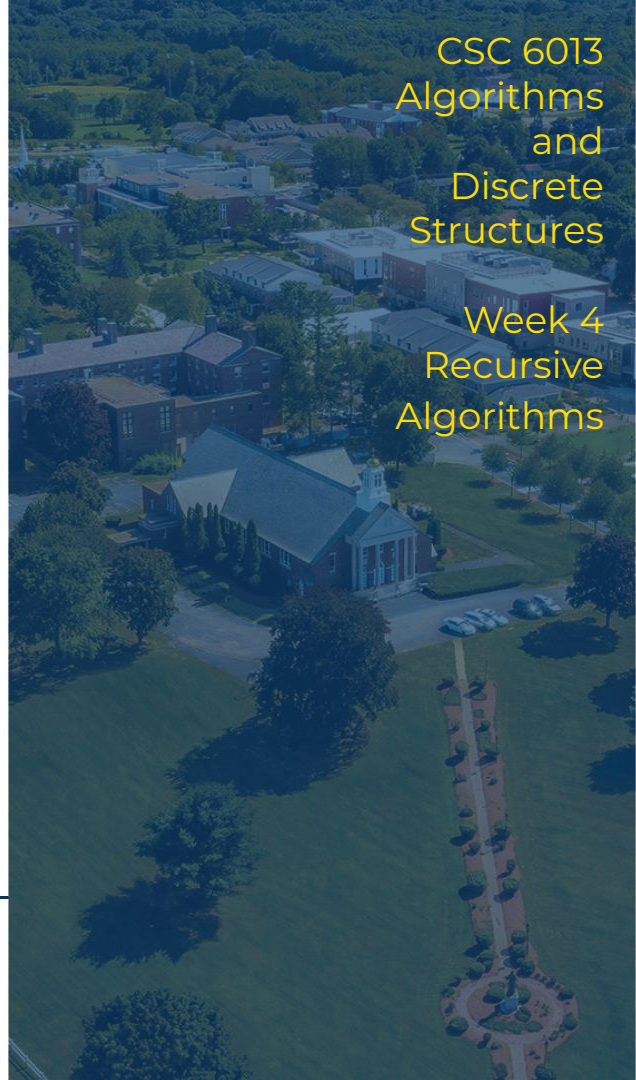
Recursive Algorithms

Algorithms and Discrete Structures - Dr. Paulo Fernandes

Presentation Agenda

Week 4

- What is recursion?
  a. basic concepts of recursion
  b. recursive equation and stopping condition
- Recursive Examples
  a. depth first search in graphs (DFS)
  b. towers of hanoi
  c. binary search
  d. maximum element in an array
- This Week's tasks



What is recursion?

A mathematical paradigm.

A philosophical tool.

We call recursive algorithms the class of algorithms that we use the same procedure repeatedly over the object of the algorithm until, for some reason, we stop repetitions and solve it in a different way.

However, the idea of recursion is much more widely applied, as in mathematics, and even philosophy. As such, we will introduce recursion with:

- **the concept of recursion;**
- recursive equation and stopping condition.

What is recursion?

Recursive definitions are not much useful:

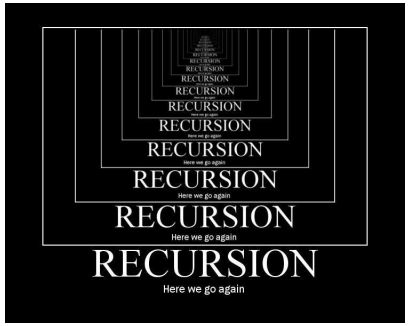
- ***Rules are rules.***

However, in many ways everything is kind of recursive:

- ***Bees are the insects that make honey;***
- ***Honey is produced by bees.***

Recursion can lead to paradoxes:

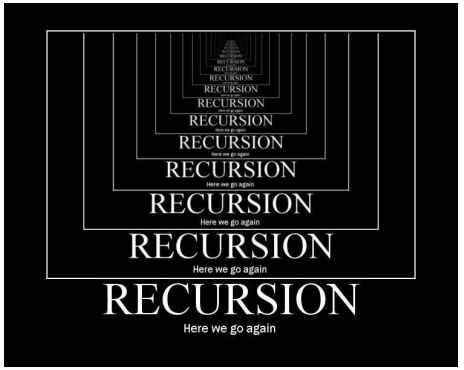
- The Russell's Paradox
  - ***You can have sets of sets;***
  - ***Sets can contain themselves;***
  - ***If we define a set X that is composed by all sets that do not contain themselves.***
  - ***Does X contain itself?***



Bertrand Russell
1872 - 1952 - 1970

Video: Russell's Paradox.

What is recursion in CS



Recursion is a smart thing…

In many ways, recursion has been with us all along.

- Natural numbers are defined recursively:
  - 1 is a natural number;
  - Any number plus one is a natural number.

- So, we can test if a number ***n*** is a natural number as this:
  - A given number ***n*** is a natural number if it is:
    - either ***1***, or
    - If ***n-1*** is a natural number.

We could test it with the code here in this slide, but **what is wrong with this code?**

```
1   def isNatural(n):
2       if (n == 1):
3           return True
4       else:
5           return isNatural(n-1)
```

WRONG

Wikipedia: Recursion.

What is recursion in CS

The previous code would never end if the number is not a Natural number

- (it never returns False)

```
1   def isNatural(n):
2       if (n == 1):
3           return True
4       else:
5           return isNatural(n-1)
```

WRONG

- So, we can test if a number ***n*** is a natural number as this:
  - A given number ***n*** is a natural number if it is:
    - either ***1***, or
    - If ***n-1*** is a natural number, but …
    - this test only applies to positive numbers.

We need a way to stop the recursion!

- We had that already to return ***True***;
- We need it also to return ***False***.

```
1   def isNatural(n):
2       if (n <= 0):
3           return False
4       elif (n == 1):
5           return True
6       else:
7           return isNatural(n-1)
```

CORRECT

Wikipedia: Recursion (computer science).

What is recursion in CS

```
1   def isNatural(n):
2       if (n <= 0):
3           return False
4       elif (n == 1):
5           return True
6       else:
7           return isNatural(n-1)
```

A recursive algorithm needs:

- one or more recursive equations;
- one or more stop conditions.

- A recursive equation, a relation, or set of relations that establishes the answer you are looking for to an instance n by calling itself to a different instance;
  - in this case:
    - the answer for ***n*** is the same as the answer for ***n-1***;
- A stopping condition that delivers the answer for a specific value or set of values;
  - in this case:
    - it returns ***True*** for ***n*** equal to ***1*** or
    - it returns ***False*** for ***n*** smaller than or equal to ***0***.

Video: What on Earth is Recursion?.

What is recursion?

A mathematical paradigm.

A philosophical tool.

We call recursive algorithms the class of algorithms that we use the same procedure repeatedly over the object of the algorithm until, for some reason, we stop repetitions and solve it in a different way.

However, the idea of recursion is much more widely applied, as in mathematics, and even philosophy. As such, we will introduce recursion with:

- the concept of recursion;
- **recursive equation and stopping condition.**

Classic Recursion

Computing the Factorial Recursively

```
1   def fact(n):
2       if (n == 1):
3           return 1
4       else:
5           return n * fact(n-1)
```

The recursive implementation is easy to understand because it is similar to the problem definition.

- We can define the factorial of a given natural number ***n*** as:
  - ***1*** for ***n*** equal to ***1***, or
  - ***n*** times the factorial of ***n-1***.
- The stopping condition is:
  - ***1!*** is equal to ***1***
- The recursive equation is:
  - ***n!*** is equal to ***n*** times ***(n-1)!***.

Is this code correct?

- It has the same problem as before, but assuming that the input (***n***) is a natural number it works.

Video: Learn recursion in 5 minutes!.

Classic Recursion

Computing the *n*-th term of the Fibonacci Sequence Recursively

```
1    def fibo(n):
2        if (n in [1, 2]):
3            return 1
4        else:
5            return fibo(n-1) + fibo(n-2)
```

The recursive implementation is easy to understand because it is similar to the problem definition.

- We can define the ***n***-th term of the Fibonacci sequence as:
  - ***1*** for the first two terms;
  - the sum of the (***n-1***)-th and (***n-2***)-th terms.

The Fibonacci Sequence

1,1,2,3,5,8,13,21,34,55,89,144,233,377…

1+1=2	13+21=34
1+2=3	21+34=55
2+3=5	34+55=89
3+5=8	55+89=144
5+8=13	89+144=233
8+13=21	144+233=377

- The stopping condition is:
  - Fibonacci sequence term ***n*** is equal to ***1*** for the ***n*** equal to ***1*** or ***2***;
- The recursive equation is:
  - Fibonacci sequence term ***n*** is equal to the sum of Fibonacci sequence term ***n-1*** and Fibonacci sequence term ***n-2***.

Video: Recursion Simply Explained.

Recursion or not?

Recursion is elegant, often efficient, but not always

Computing factorial with and without recursion

```
1   def fact(n):
2       if (n == 1):
3           return 1
4       else:
5           return fact(n-1)
```

```
1   def fact(n):
2       ans = 1
3       for i in range(2,n+1):
4           ans *= i
5       return ans
```

Computing Fibonacci with and without recursion

```
1   def fibo(n):
2       if (n in [1, 2]):
3           return 1
4       else:
5           return fibo(n-1) + fibo(n-2)
```

```
1   def fibo(n):
2       if (n in [1, 2]):
3           return 1
4       else:
5           nMinus2 = 1
6           nMinus1 = 1
7           for i in range(3, n+1):
8               nMinus2, nMinus1 = nMinus1, nMinus2+nMinus1
9       return nMinus1
```

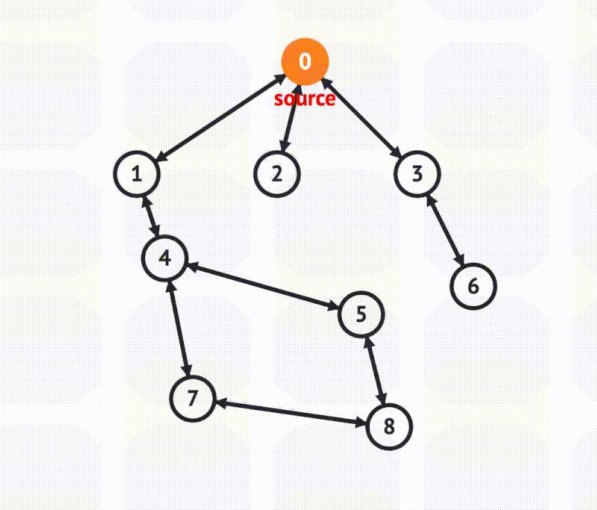

Video: Stepping Through Recursive Fibonacci.

Recursive Algorithms Examples

Algorithms Examples

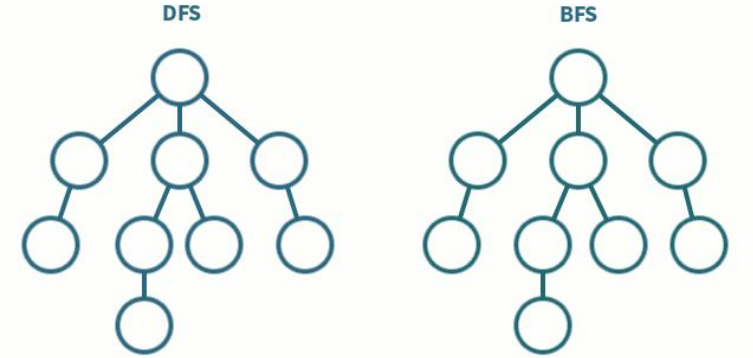
- **depth first search in graphs (DFS)**
- towers of hanoi
- binary search
- maximum element in an array

Example 1 - Depth First Search in Graphs



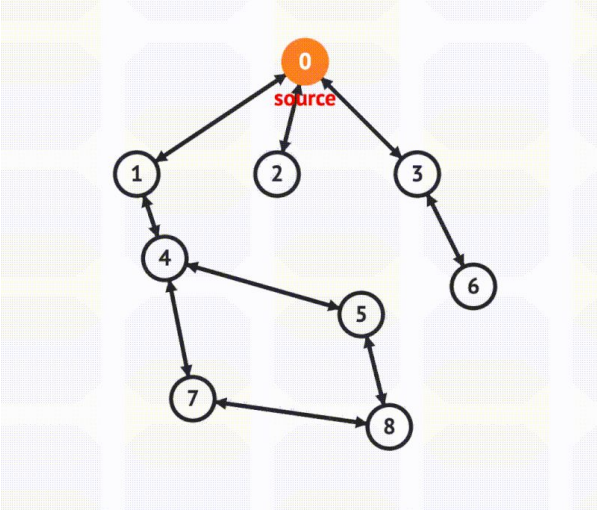
- Giving a graph defined by a ***V*** set of ***n*** Vertices and ***E*** a set of ***m*** Edges.
- Starting from the first vertex in ***V*** mark all vertices from ***0*** to ***n-1*** using Depth-First Search.

Depth First Search versus Breadth First Search



Check out the BFS in the previous week.

Example 1 - Depth First Search in Graphs



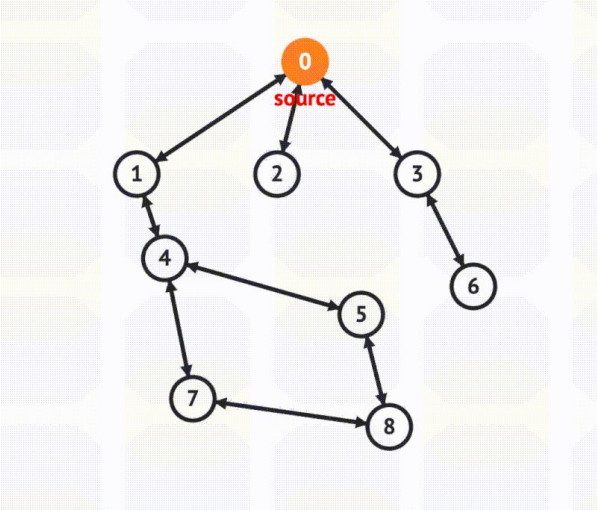
- Giving a graph defined by a ***V*** set of ***n*** Vertices and ***E*** a set of ***m*** Edges.
- Starting from the first vertex in ***V*** mark all vertices from ***0*** to ***n-1*** using Depth-First Search.

One recursive method (pre order) is:

- Starting with the first vertex ***v***:
  - visit ***v*** and
  - go to the unvisited neighbors one by one until there are no more neighbors, then return to the vertex you were before.

DFS is naturally recursive.

Example 1 - Depth First Search in Graphs

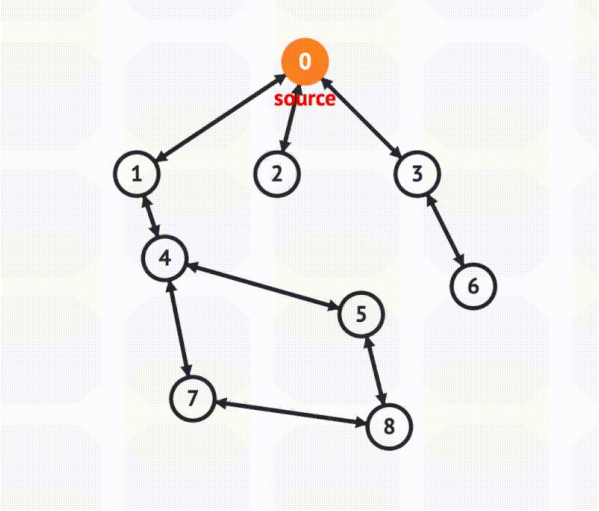


- The recursive implementation of this algorithm uses:
  - an array ***V*** with all vertices v_i holding information if they are visited or not;
  - an adjacency list of edges ***E*** with triplets $(v_i, v_j, 1)$
  - a counter ***count*** to keep the track of the nodes to visit.

There is another non recursive implementation to the DFS algorithm that uses a stack similarly to the BFS algorithm seen the previous week that uses a queue.

While DFS is naturally recursive, BFS is naturally iterative.

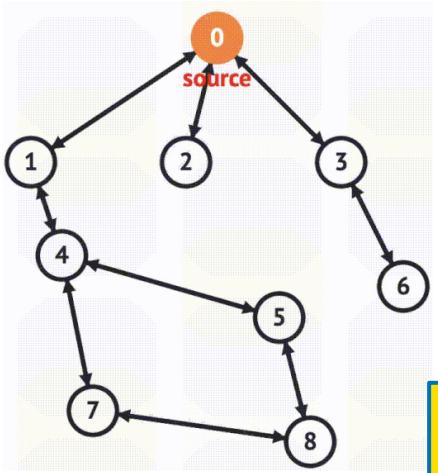
Example 1 - Depth First Search in Graphs




```python
def DFS(V, E):
    def __visit(i, count):
        V[i], count = count, count+1
        for e in E:
            if (e[0] == i) and (V[e[1]] == -1):
                count = __visit(e[1], count)
            elif (e[1] == i) and (V[e[0]] == -1):
                count = __visit(e[0], count)
        return count

    for i in range(len(V)):
        V[i] = -1
    count = 0
    for i in range (len(V)):
        if (V[i] == -1):
            count = __visit(i, count)

V = [0]*9
E = [[0,1,1], [0,2,1], [0,3,1], [1, 4, 1], [3,6,1],
     [4,5,1], [4,7,1], [5,8,1], [7,8,1]]
DFS(V,E)
print(V)
```

undirected graph

Example 1 - Depth First Search in Graphs



[-1 -1 -1 -1 -1 -1 -1 -1 -1]

0 1 2 3 4 5 6 7 8

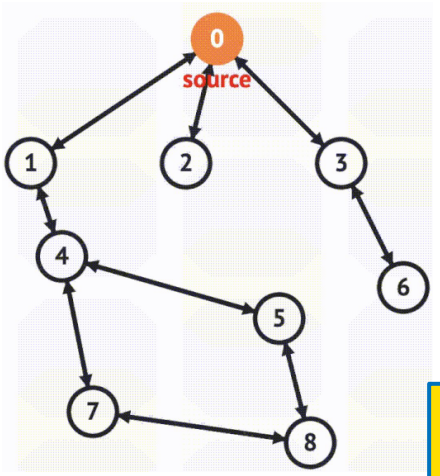
i = 0 count = 0

```python
def DFS(V, E):
    def __visit(i, count):
        V[i], count = count, count+1
        for e in E:
            if (e[0] == i) and (V[e[1]] == -1):
                count = __visit(e[1], count)
            elif (e[1] == i) and (V[e[0]] == -1):
                count = __visit(e[0], count)
        return count

    for i in range(len(V)):
        V[i] = -1
    count = 0
    for i in range (len(V)):
        if (V[i] == -1):
            count = __visit(i, count)

V = [0]*9
E = [[0,1,1], [0,2,1], [0,3,1], [1, 4, 1], [3,6,1],
     [4,5,1], [4,7,1], [5,8,1], [7,8,1]]
DFS(V,E)
print(V)
```

MERRIMACK COLLEGE

Step 1 of 27.

Example 1 - Depth First Search in Graphs



0

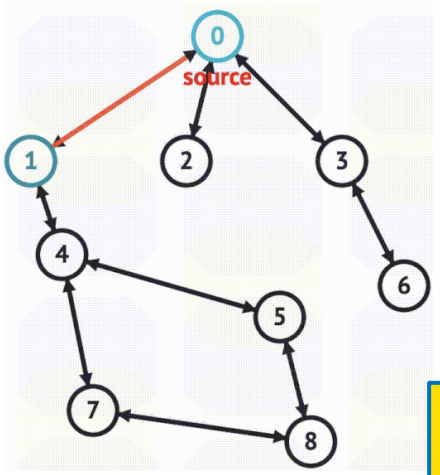
[0 -1 -1 -1 -1 -1 -1 -1 -1]

0 1 2 3 4 5 6 7 8

i = 0 count = 1

```python
def DFS(V, E):
    def __visit(i, count):
        V[i], count = count, count+1
        for e in E:
            if (e[0] == i) and (V[e[1]] == -1):
                count = __visit(e[1], count)
            elif (e[1] == i) and (V[e[0]] == -1):
                count = __visit(e[0], count)
        return count

    for i in range(len(V)):
        V[i] = -1
    count = 0
    for i in range (len(V)):
        if (V[i] == -1):
            count = __visit(i, count)

V = [0]*9
E = [[0,1,1], [0,2,1], [0,3,1], [1, 4, 1], [3,6,1],
     [4,5,1], [4,7,1], [5,8,1], [7,8,1]]
DFS(V,E)
print(V)
```

Example 1 - Depth First Search in Graphs



0

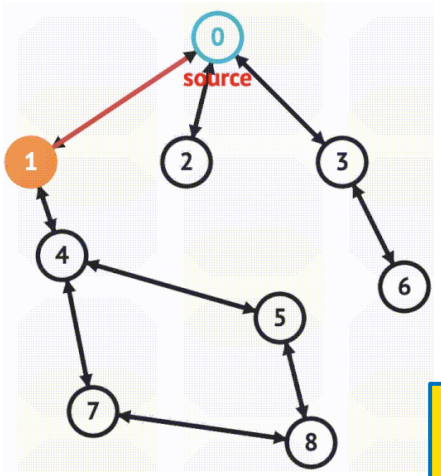
[0 -1 -1 -1 -1 -1 -1 -1 -1]

0 1 2 3 4 5 6 7 8

i = 0 count = 1

```python
def DFS(V, E):
    def __visit(i, count):
        V[i], count = count, count+1
        for e in E:
            if (e[0] == i) and (V[e[1]] == -1):
                count = __visit(e[1], count)
            elif (e[1] == i) and (V[e[0]] == -1):
                count = __visit(e[0], count)
        return count

    for i in range(len(V)):
        V[i] = -1
    count = 0
    for i in range (len(V)):
        if (V[i] == -1):
            count = __visit(i, count)

V = [0]
E = [[0,1,1], [0,2,1], [0,3,1], [1, 4, 1], [3,6,1],
     [4,5,1], [4,7,1], [5,8,1], [7,8,1]]
DFS(V,E)
print(V)
```

Example 1 - Depth First Search in Graphs



0
1

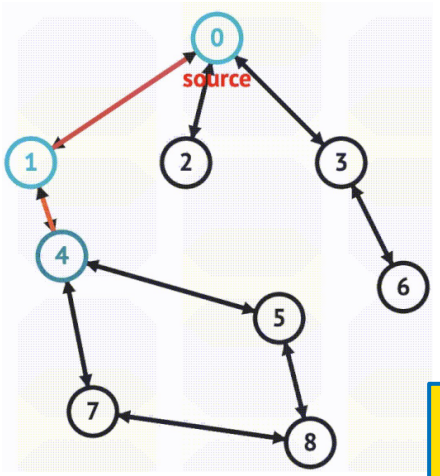
[0 1 -1 -1 -1 -1 -1 -1 -1]

0 1 2 3 4 5 6 7 8

i = 1 count = 2

```python
def DFS(V, E):
    def __visit(i, count):
        V[i], count = count, count+1
        for e in E:
            if (e[0] == i) and (V[e[1]] == -1):
                count = __visit(e[1], count)
            elif (e[1] == i) and (V[e[0]] == -1):
                count = __visit(e[0], count)
        return count

    for i in range(len(V)):
        V[i] = -1
    count = 0
    for i in range (len(V)):
        if (V[i] == -1):
            count = __visit(i, count)

V = [0]*9
E = [[0,1,1], [0,2,1], [0,3,1], [1, 4, 1], [3,6,1],
     [4,5,1], [4,7,1], [5,8,1], [7,8,1]]
DFS(V,E)
print(V)
```

Example 1 - Depth First Search in Graphs



0
1

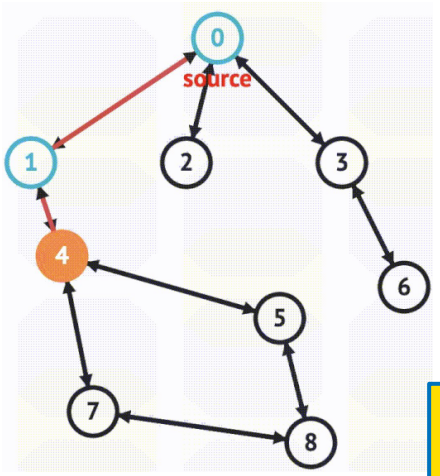
[0 1 -1 -1 -1 -1 -1 -1 -1]

0 1 2 3 4 5 6 7 8

i = 1 count = 2

```python
def DFS(V, E):
    def __visit(i, count):
        V[i], count = count, count+1
        for e in E:
            if (e[0] == i) and (V[e[1]] == -1):
                count = __visit(e[1], count)
            elif (e[1] == i) and (V[e[0]] == -1):
                count = __visit(e[0], count)
        return count

    for i in range(len(V)):
        V[i] = -1
    count = 0
    for i in range (len(V)):
        if (V[i] == -1):
            count = __visit(i, count)

V = [0]*9
E = [[0,1,1], [0,2,1], [0,3,1], [1, 4, 1], [3,6,1],
     [4,5,1], [4,7,1], [5,8,1], [7,8,1]]
DFS(V,E)
print(V)
```

Example 1 - Depth First Search in Graphs



0
1
4

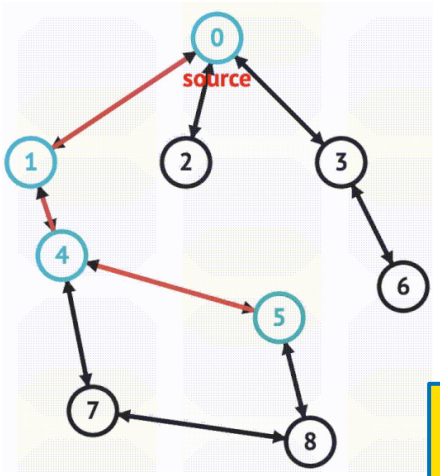
[0 1 -1 -1 2 -1 -1 -1 -1]

0 1 2 3 4 5 6 7 8

i = 4 count = 3

```python
def DFS(V, E):
    def __visit(i, count):
        V[i], count = count, count+1
        for e in E:
            if (e[0] == i) and (V[e[1]] == -1):
                count = __visit(e[1], count)
            elif (e[1] == i) and (V[e[0]] == -1):
                count = __visit(e[0], count)
        return count

    for i in range(len(V)):
        V[i] = -1
    count = 0
    for i in range (len(V)):
        if (V[i] == -1):
            count = __visit(i, count)

V = [0]*9
E = [[0,1,1], [0,2,1], [0,3,1], [1, 4, 1], [3,6,1],
     [4,5,1], [4,7,1], [5,8,1], [7,8,1]]
DFS(V,E)
print(V)
```

Example 1 - Depth First Search in Graphs



0
1
4

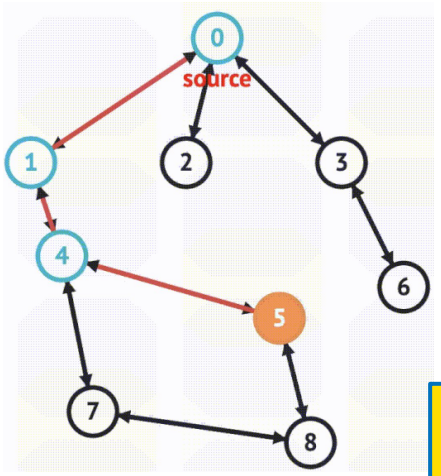
[0 1 -1 -1 2 -1 -1 -1 -1]

0 1 2 3 4 5 6 7 8

i = 4 count = 3

```python
def DFS(V, E):
    def __visit(i, count):
        V[i], count = count, count+1
        for e in E:
            if (e[0] == i) and (V[e[1]] == -1):
                count = __visit(e[1], count)
            elif (e[1] == i) and (V[e[0]] == -1):
                count = __visit(e[0], count)
        return count

    for i in range(len(V)):
        V[i] = -1
    count = 0
    for i in range (len(V)):
        if (V[i] == -1):
            count = __visit(i, count)

V = [0]*9
E = [[0,1,1], [0,2,1], [0,3,1], [1, 4, 1], [3,6,1],
     [4,5,1], [4,7,1], [5,8,1], [7,8,1]]
DFS(V,E)
print(V)
```

Example 1 - Depth First Search in Graphs



0
1
4
5

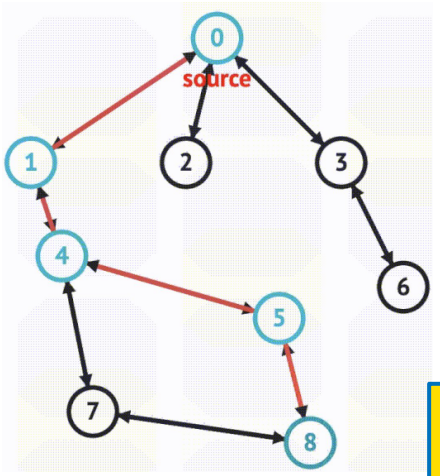
[0 1 -1 -1 2 3 -1 -1 -1]

0 1 2 3 4 5 6 7 8

i = 5 count = 4

```python
def DFS(V, E):
    def __visit(i, count):
        V[i], count = count, count+1
        for e in E:
            if (e[0] == i) and (V[e[1]] == -1):
                count = __visit(e[1], count)
            elif (e[1] == i) and (V[e[0]] == -1):
                count = __visit(e[0], count)
        return count

    for i in range(len(V)):
        V[i] = -1
    count = 0
    for i in range (len(V)):
        if (V[i] == -1):
            count = __visit(i, count)

V = [0]*9
E = [[0,1,1], [0,2,1], [0,3,1], [1, 4, 1], [3,6,1],
     [4,5,1], [4,7,1], [5,8,1], [7,8,1]]
DFS(V,E)
print(V)
```

Example 1 - Depth First Search in Graphs



0
1
4
5

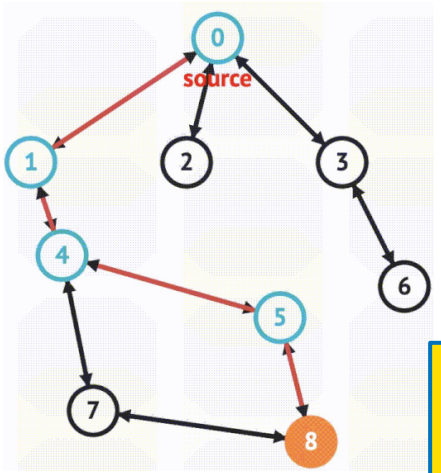
[0 1 -1 -1 2 3 -1 -1 -1]

0 1 2 3 4 5 6 7 8

i = 5 count = 4

```python
def DFS(V, E):
    def __visit(i, count):
        V[i], count = count, count+1
        for e in E:
            if (e[0] == i) and (V[e[1]] == -1):
                count = __visit(e[1], count)
            elif (e[1] == i) and (V[e[0]] == -1):
                count = __visit(e[0], count)
        return count

    for i in range(len(V)):
        V[i] = -1
    count = 0
    for i in range (len(V)):
        if (V[i] == -1):
            count = __visit(i, count)

V = [0]*9
E = [[0,1,1], [0,2,1], [0,3,1], [1, 4, 1], [3,6,1],
     [4,5,1], [4,7,1], [5,8,1], [7,8,1]]
DFS(V,E)
print(V)
```

Example 1 - Depth First Search in Graphs



0
1
4
5
8

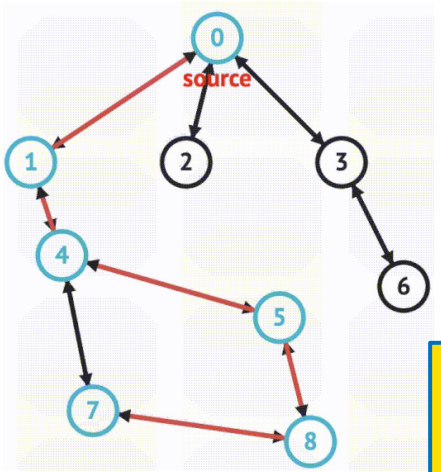
[0 1 -1 -1 2 3 -1 -1 4]

0 1 2 3 4 5 6 7 8

i = 8 count = 5

```python
def DFS(V, E):
    def __visit(i, count):
        V[i], count = count, count+1
        for e in E:
            if (e[0] == i) and (V[e[1]] == -1):
                count = __visit(e[1], count)
            elif (e[1] == i) and (V[e[0]] == -1):
                count = __visit(e[0], count)
        return count

    for i in range(len(V)):
        V[i] = -1
    count = 0
    for i in range (len(V)):
        if (V[i] == -1):
            count = __visit(i, count)

V = [0]*9
E = [[0,1,1], [0,2,1], [0,3,1], [1, 4, 1], [3,6,1],
     [4,5,1], [4,7,1], [5,8,1], [7,8,1]]
DFS(V,E)
print(V)
```

Example 1 - Depth First Search in Graphs



0
1
4
5
8

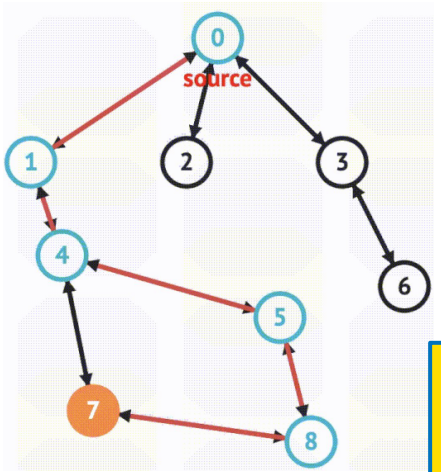
[0 1 -1 -1 2 3 -1 -1 4]

0 1 2 3 4 5 6 7 8

i = 8 count = 5

```
1   def DFS(V, E):
2       def __visit(i, count):
3           V[i], count = count, count+1
4           for e in E:
5               if (e[0] == i) and (V[e[1]] == -1):
6                   count = __visit(e[1], count)
7               elif (e[1] == i) and (V[e[0]] == -1):
8                   count = __visit(e[0], count)
9           return count
10
11      for i in range(len(V)):
12          V[i] = -1
13      count = 0
14      for i in range(len(V)):
15          if (V[i] == -1):
16              count = __visit(i, count)
17
18  V = [0]*9
19  E = [[0,1,1], [0,2,1], [0,3,1], [1, 4, 1], [3,6,1],
20       [4,5,1], [4,7,1], [5,8,1], [7,8,1]]
21  DFS(V,E)
22  print(V)
```

Example 1 - Depth First Search in Graphs



0
1
4
5
8
7

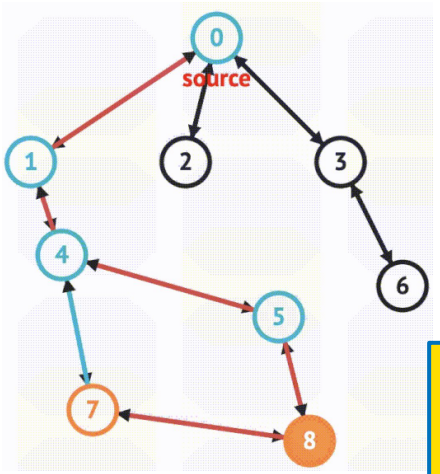
```python
def DFS(V, E):
    def __visit(i, count):
        V[i], count = count, count+1
        for e in E:
            if (e[0] == i) and (V[e[1]] == -1):
                count = __visit(e[1], count)
            elif (e[1] == i) and (V[e[0]] == -1):
                count = __visit(e[0], count)
        return count

    for i in range(len(V)):
        V[i] = -1
    count = 0
    for i in range (len(V)):
        if (V[i] == -1):
            count = __visit(i, count)

V = [0]*9
E = [[0,1,1], [0,2,1], [0,3,1], [1, 4, 1], [3,6,1],
     [4,5,1], [4,7,1], [5,8,1], [7,8,1]]
DFS(V,E)
print(V)
```

[0 1 -1 -1 2 3 -1 5 4]

0 1 2 3 4 5 6 7 8

i = 7 count = 6

Example 1 - Depth First Search in Graphs



0
1
4
5
8
7

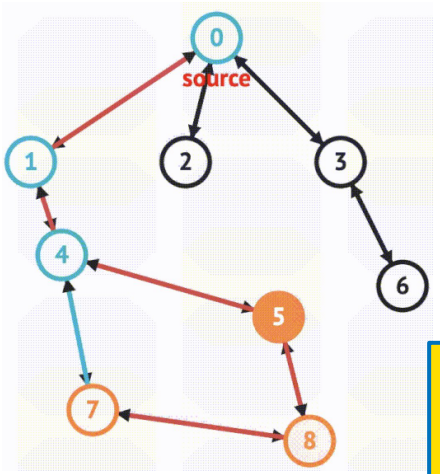
```
1   def DFS(V, E):
2       def __visit(i, count):
3           V[i], count = count, count+1
4           for e in E:
5               if (e[0] == i) and (V[e[1]] == -1):
6                   count = __visit(e[1], count)
7               elif (e[1] == i) and (V[e[0]] == -1):
8                   count = __visit(e[0], count)
9           return count
10
11      for i in range(len(V)):
12          V[i] = -1
13      count = 0
14      for i in range (len(V)):
15          if (V[i] == -1):
16              count = __visit(i, count)
17
18  V = [0]*9
19  E = [[0,1,1], [0,2,1], [0,3,1], [1, 4, 1], [3,6,1],
20       [4,5,1], [4,7,1], [5,8,1], [7,8,1]]
21  DFS(V,E)
22  print(V)
```

[0 1 -1 -1 2 3 -1 5 4]

0 1 2 3 4 5 6 7 8

i = 8 count = 6

Example 1 - Depth First Search in Graphs



0
1
4
5
8
7

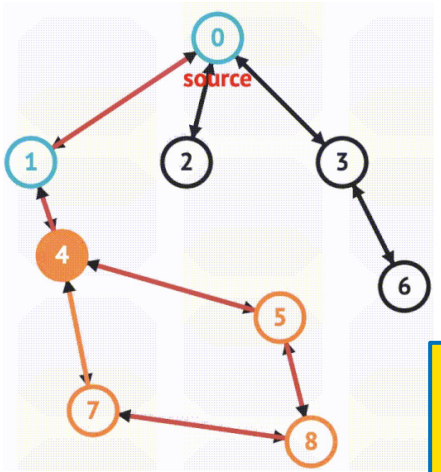
[0 1 -1 -1 2 3 -1 5 4]

0 1 2 3 4 5 6 7 8

i = 5 count = 6

```
1   def DFS(V, E):
2       def __visit(i, count):
3           V[i], count = count, count+1
4           for e in E:
5               if (e[0] == i) and (V[e[1]] == -1):
6                   count = __visit(e[1], count)
7               elif (e[1] == i) and (V[e[0]] == -1):
8                   count = __visit(e[0], count)
9           return count
10
11      for i in range(len(V)):
12          V[i] = -1
13      count = 0
14      for i in range (len(V)):
15          if (V[i] == -1):
16              count = __visit(i, count)
17
18  V = [0]*9
19  E = [[0,1,1], [0,2,1], [0,3,1], [1, 4, 1], [3,6,1],
20       [4,5,1], [4,7,1], [5,8,1], [7,8,1]]
21  DFS(V,E)
22  print(V)
```

Example 1 - Depth First Search in Graphs



0
1
4
5
8
7

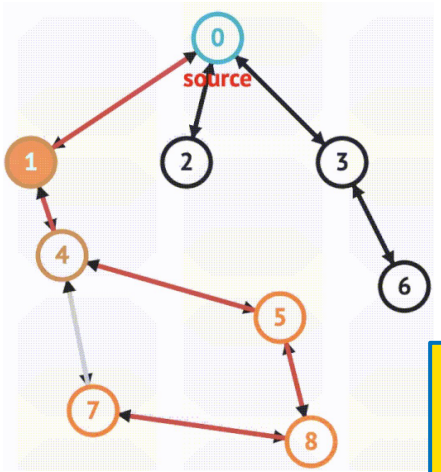
[0 1 -1 -1 2 3 -1 5 4]

0 1 2 3 4 5 6 7 8

i = 4 count = 6

```python
def DFS(V, E):
    def __visit(i, count):
        V[i], count = count, count+1
        for e in E:
            if (e[0] == i) and (V[e[1]] == -1):
                count = __visit(e[1], count)
            elif (e[1] == i) and (V[e[0]] == -1):
                count = __visit(e[0], count)
        return count

    for i in range(len(V)):
        V[i] = -1
    count = 0
    for i in range (len(V)):
        if (V[i] == -1):
            count = __visit(i, count)

V = [0]*9
E = [[0,1,1], [0,2,1], [0,3,1], [1, 4, 1], [3,6,1],
     [4,5,1], [4,7,1], [5,8,1], [7,8,1]]
DFS(V,E)
print(V)
```

Example 1 - Depth First Search in Graphs



0
1
4
5
8
7

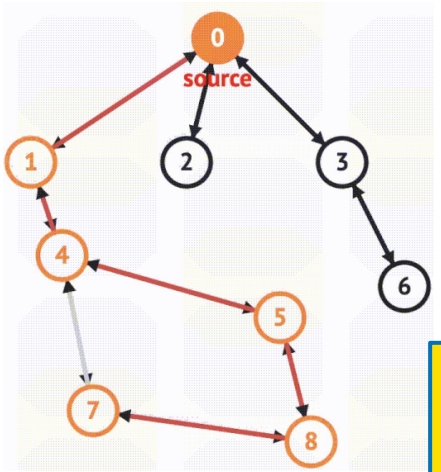
```python
def DFS(V, E):
    def __visit(i, count):
        V[i], count = count, count+1
        for e in E:
            if (e[0] == i) and (V[e[1]] == -1):
                count = __visit(e[1], count)
            elif (e[1] == i) and (V[e[0]] == -1):
                count = __visit(e[0], count)
        return count

    for i in range(len(V)):
        V[i] = -1
    count = 0
    for i in range (len(V)):
        if (V[i] == -1):
            count = __visit(i, count)

V = [0]*9
E = [[0,1,1], [0,2,1], [0,3,1], [1, 4, 1], [3,6,1],
     [4,5,1], [4,7,1], [5,8,1], [7,8,1]]
DFS(V,E)
print(V)
```

[0 1 -1 -1 2 3 -1 5 4]

0 1 2 3 4 5 6 7 8

i = 1 count = 6

Example 1 - Depth First Search in Graphs



0
1
4
5
8
7

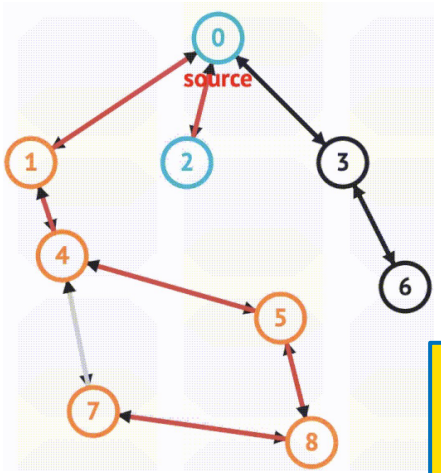
```python
def DFS(V, E):
    def __visit(i, count):
        V[i], count = count, count+1
        for e in E:
            if (e[0] == i) and (V[e[1]] == -1):
                count = __visit(e[1], count)
            elif (e[1] == i) and (V[e[0]] == -1):
                count = __visit(e[0], count)
        return count

    for i in range(len(V)):
        V[i] = -1
    count = 0
    for i in range (len(V)):
        if (V[i] == -1):
            count = __visit(i, count)

V = [0]*9
E = [[0,1,1], [0,2,1], [0,3,1], [1, 4, 1], [3,6,1],
     [4,5,1], [4,7,1], [5,8,1], [7,8,1]]
DFS(V,E)
print(V)
```

[0 1 -1 -1 2 3 -1 5 4]

0 1 2 3 4 5 6 7 8

i = 0 count = 6

Example 1 - Depth First Search in Graphs



0
1
4
5
8
7

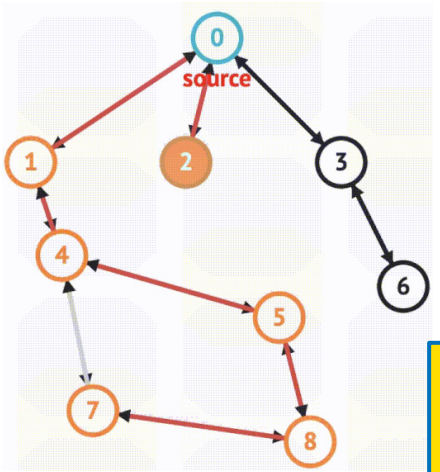
```python
def DFS(V, E):
    def __visit(i, count):
        V[i], count = count, count+1
        for e in E:
            if (e[0] == i) and (V[e[1]] == -1):
                count = __visit(e[1], count)
            elif (e[1] == i) and (V[e[0]] == -1):
                count = __visit(e[0], count)
        return count

    for i in range(len(V)):
        V[i] = -1
    count = 0
    for i in range (len(V)):
        if (V[i] == -1):
            count = __visit(i, count)

V = [0]*9
E = [[0,1,1], [0,2,1], [0,3,1], [1, 4, 1], [3,6,1],
     [4,5,1], [4,7,1], [5,8,1], [7,8,1]]
DFS(V,E)
print(V)
```

[0 1 -1 -1 2 3 -1 5 4]

0 1 2 3 4 5 6 7 8

i = 0 count = 6

Example 1 - Depth First Search in Graphs



0
1
4
5
8
7
2

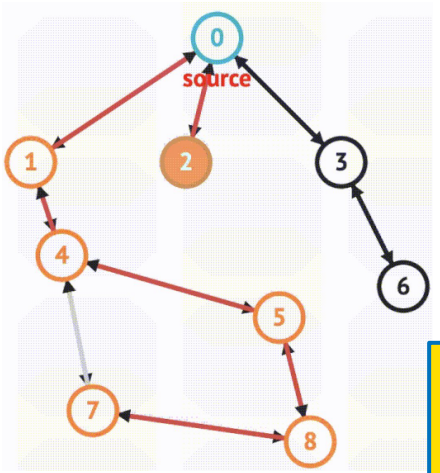
[0 1 6 -1 2 3 -1 5 4]

0 1 2 3 4 5 6 7 8

i = 2 count = 7

```python
def DFS(V, E):
    def __visit(i, count):
        V[i], count = count, count+1
        for e in E:
            if (e[0] == i) and (V[e[1]] == -1):
                count = __visit(e[1], count)
            elif (e[1] == i) and (V[e[0]] == -1):
                count = __visit(e[0], count)
        return count

    for i in range(len(V)):
        V[i] = -1
    count = 0
    for i in range (len(V)):
        if (V[i] == -1):
            count = __visit(i, count)

V = [0]*9
E = [[0,1,1], [0,2,1], [0,3,1], [1, 4, 1], [3,6,1],
     [4,5,1], [4,7,1], [5,8,1], [7,8,1]]
DFS(V,E)
print(V)
```

Example 1 - Depth First Search in Graphs



0
1
4
5
8
7
2

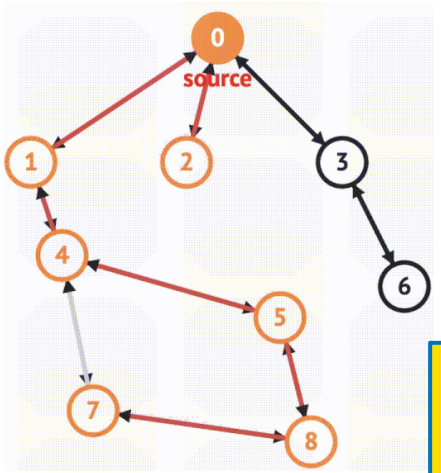
```python
def DFS(V, E):
    def __visit(i, count):
        V[i], count = count, count+1
        for e in E:
            if (e[0] == i) and (V[e[1]] == -1):
                count = __visit(e[1], count)
            elif (e[1] == i) and (V[e[0]] == -1):
                count = __visit(e[0], count)
        return count

    for i in range(len(V)):
        V[i] = -1
    count = 0
    for i in range (len(V)):
        if (V[i] == -1):
            count = __visit(i, count)

V = [0]*9
E = [[0,1,1], [0,2,1], [0,3,1], [1, 4, 1], [3,6,1],
     [4,5,1], [4,7,1], [5,8,1], [7,8,1]]
DFS(V,E)
print(V)
```

[0 1 6 -1 2 3 -1 5 4]

0 1 2 3 4 5 6 7 8

i = 0 count = 7

Example 1 - Depth First Search in Graphs



0
1
4
5
8
7
2

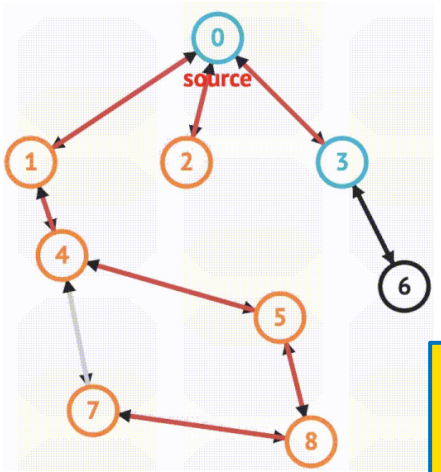
[0 1 6 -1 2 3 -1 5 4]

0 1 2 3 4 5 6 7 8

i = 0 count = 7

```python
def DFS(V, E):
    def __visit(i, count):
        V[i], count = count, count+1
        for e in E:
            if (e[0] == i) and (V[e[1]] == -1):
                count = __visit(e[1], count)
            elif (e[1] == i) and (V[e[0]] == -1):
                count = __visit(e[0], count)
        return count

    for i in range(len(V)):
        V[i] = -1
    count = 0
    for i in range (len(V)):
        if (V[i] == -1):
            count = __visit(i, count)

V = [0]*9
E = [[0,1,1], [0,2,1], [0,3,1], [1, 4, 1], [3,6,1],
     [4,5,1], [4,7,1], [5,8,1], [7,8,1]]
DFS(V,E)
print(V)
```

Example 1 - Depth First Search in Graphs



0
1
4
5
8
7
2

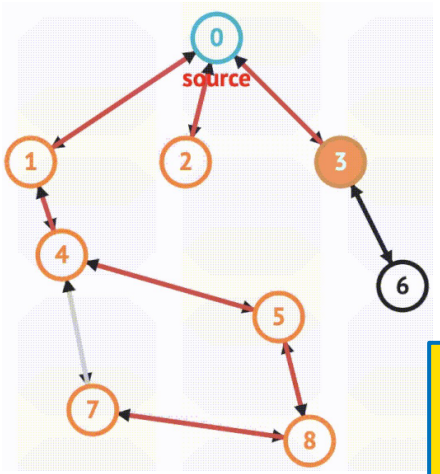
[0 1 6 -1 2 3 -1 5 4]

0 1 2 3 4 5 6 7 8

i = 0 count = 7

```python
def DFS(V, E):
    def __visit(i, count):
        V[i], count = count, count+1
        for e in E:
            if (e[0] == i) and (V[e[1]] == -1):
                count = __visit(e[1], count)
            elif (e[1] == i) and (V[e[0]] == -1):
                count = __visit(e[0], count)
        return count

    for i in range(len(V)):
        V[i] = -1
    count = 0
    for i in range(len(V)):
        if (V[i] == -1):
            count = __visit(i, count)

V = [0]*9
E = [[0,1,1], [0,2,1], [0,3,1], [1, 4, 1], [3,6,1],
     [4,5,1], [4,7,1], [5,8,1], [7,8,1]]
DFS(V,E)
print(V)
```

Example 1 - Depth First Search in Graphs



0
1
4
5
8
7
2
3

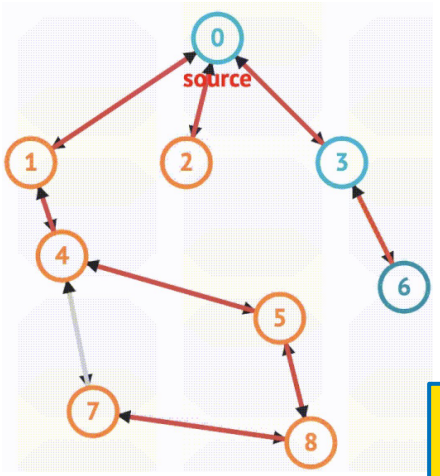
```python
def DFS(V, E):
    def __visit(i, count):
        V[i], count = count, count+1
        for e in E:
            if (e[0] == i) and (V[e[1]] == -1):
                count = __visit(e[1], count)
            elif (e[1] == i) and (V[e[0]] == -1):
                count = __visit(e[0], count)
        return count

    for i in range(len(V)):
        V[i] = -1
    count = 0
    for i in range (len(V)):
        if (V[i] == -1):
            count = __visit(i, count)

V = [0]*9
E = [[0,1,1], [0,2,1], [0,3,1], [1, 4, 1], [3,6,1],
     [4,5,1], [4,7,1], [5,8,1], [7,8,1]]
DFS(V,E)
print(V)
```

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

0 1 2 3 4 5 6 7 8

i = 3 count = 8

Example 1 - Depth First Search in Graphs



0
1
4
5
8
7
2
3

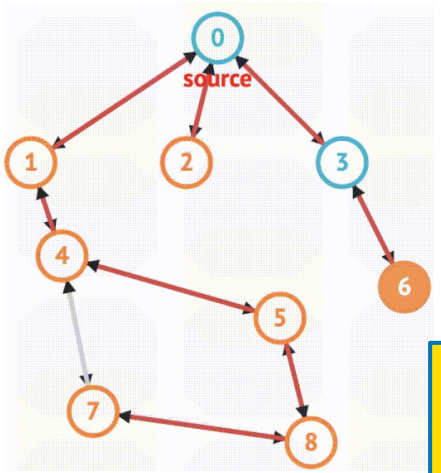
[0 1 6 7 2 3 -1 5 4]

0 1 2 3 4 5 6 7 8

i = 3 count = 8

```python
def DFS(V, E):
    def __visit(i, count):
        V[i], count = count, count+1
        for e in E:
            if (e[0] == i) and (V[e[1]] == -1):
                count = __visit(e[1], count)
            elif (e[1] == i) and (V[e[0]] == -1):
                count = __visit(e[0], count)
        return count

    for i in range(len(V)):
        V[i] = -1
    count = 0
    for i in range (len(V)):
        if (V[i] == -1):
            count = __visit(i, count)

V = [0]*9
E = [[0,1,1], [0,2,1], [0,3,1], [1, 4, 1], [3,6,1],
     [4,5,1], [4,7,1], [5,8,1], [7,8,1]]
DFS(V,E)
print(V)
```

Example 1 - Depth First Search in Graphs



0
1
4
5
8
7
2
3
6

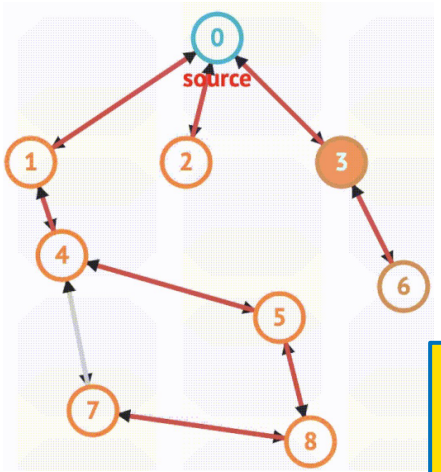
```python
def DFS(V, E):
    def __visit(i, count):
        V[i], count = count, count+1
        for e in E:
            if (e[0] == i) and (V[e[1]] == -1):
                count = __visit(e[1], count)
            elif (e[1] == i) and (V[e[0]] == -1):
                count = __visit(e[0], count)
        return count

    for i in range(len(V)):
        V[i] = -1
    count = 0
    for i in range (len(V)):
        if (V[i] == -1):
            count = __visit(i, count)

V = [0]*9
E = [[0,1,1], [0,2,1], [0,3,1], [1, 4, 1], [3,6,1],
     [4,5,1], [4,7,1], [5,8,1], [7,8,1]]
DFS(V,E)
print(V)
```

[0 1 6 7 2 3 8 5 4]

0 1 2 3 4 5 6 7 8

i = 6 count = 9

Example 1 - Depth First Search in Graphs



0
1
4
5
8
7
2
3
6

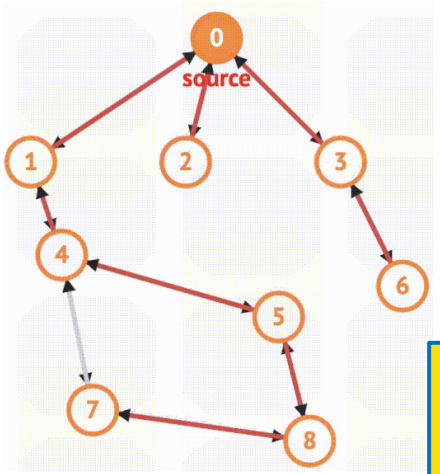
[0 1 6 7 2 3 8 5 4]

0 1 2 3 4 5 6 7 8

i = 3 count = 9

```python
def DFS(V, E):
    def __visit(i, count):
        V[i], count = count, count+1
        for e in E:
            if (e[0] == i) and (V[e[1]] == -1):
                count = __visit(e[1], count)
            elif (e[1] == i) and (V[e[0]] == -1):
                count = __visit(e[0], count)
        return count

    for i in range(len(V)):
        V[i] = -1
    count = 0
    for i in range (len(V)):
        if (V[i] == -1):
            count = __visit(i, count)

V = [0]*9
E = [[0,1,1], [0,2,1], [0,3,1], [1, 4, 1], [3,6,1],
     [4,5,1], [4,7,1], [5,8,1], [7,8,1]]
DFS(V,E)
print(V)
```

Example 1 - Depth First Search in Graphs



0
1
4
5
8
7
2
3
6

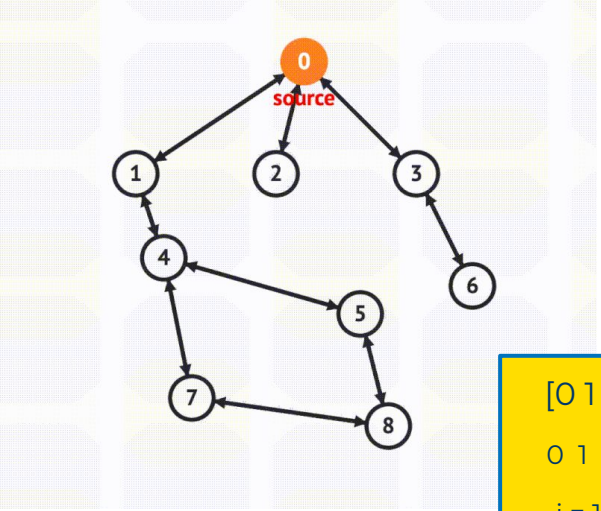
```python
def DFS(V, E):
    def __visit(i, count):
        V[i], count = count, count+1
        for e in E:
            if (e[0] == i) and (V[e[1]] == -1):
                count = __visit(e[1], count)
            elif (e[1] == i) and (V[e[0]] == -1):
                count = __visit(e[0], count)
        return count

    for i in range(len(V)):
        V[i] = -1
    count = 0
    for i in range (len(V)):
        if (V[i] == -1):
            count = __visit(i, count)

V = [0]*9
E = [[0,1,1], [0,2,1], [0,3,1], [1, 4, 1], [3,6,1],
     [4,5,1], [4,7,1], [5,8,1], [7,8,1]]
DFS(V,E)
print(V)
```

[0 1 6 7 2 3 8 5 4]

0 1 2 3 4 5 6 7 8

i = 0 count = 9

Example 1 - Depth First Search in Graphs



0
1
4
5
8
7
2
3
6

[0 1 6 7 2 3 8 5 4]

0 1 2 3 4 5 6 7 8

i = 1...8 count = 9

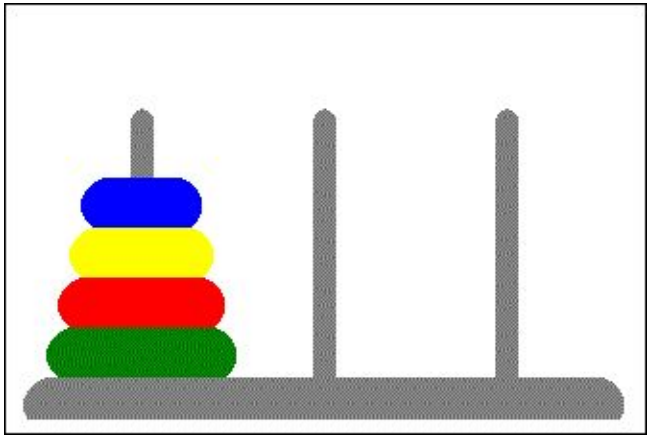
```python
def DFS(V, E):
    def __visit(i, count):
        V[i], count = count, count+1
        for e in E:
            if (e[0] == i) and (V[e[1]] == -1):
                count = __visit(e[1], count)
            elif (e[1] == i) and (V[e[0]] == -1):
                count = __visit(e[0], count)
        return count

    for i in range(len(V)):
        V[i] = -1
    count = 0
    for i in range (len(V)):
        if (V[i] == -1):
            count = __visit(i, count)

V = [0]*9
E = [[0,1,1], [0,2,1], [0,3,1], [1, 4, 1], [3,6,1],
     [4,5,1], [4,7,1], [5,8,1], [7,8,1]]
DFS(V,E)
print(V)
```

Recursive Algorithms Examples

Algorithms Examples

- depth first search in graphs (DFS)
- **towers of hanoi**
- binary search
- maximum element in an array

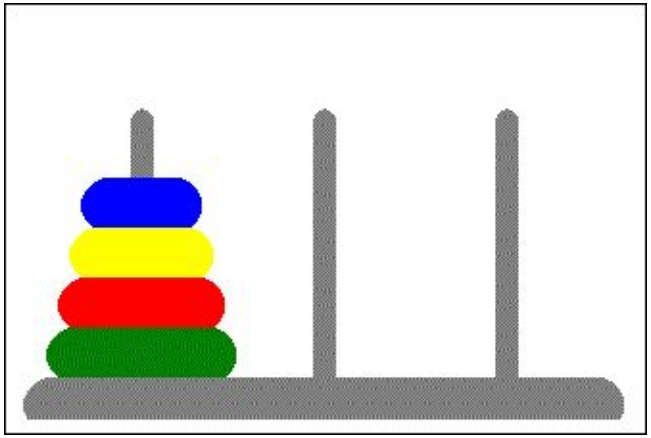
Example 2 - Towers of Hanoi

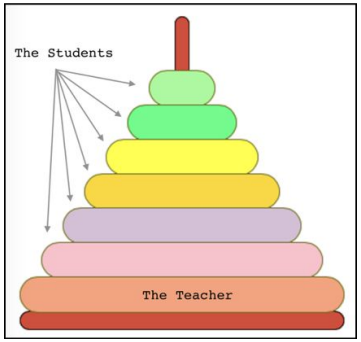


- All ***n*** disks start in the first pole
  - Each movement takes one single disk (the one on top) from one pole to another one;
  - A larger disk cannot be placed on top of a smaller one;
- Two kinds of problem
  - How to move efficiently (minimal number of movements) all disks to the last pole?
  - How many are the minimal movements for ***n*** disks

Wikipedia: Towers of Hanoi.

Example 2 - Towers of Hanoi



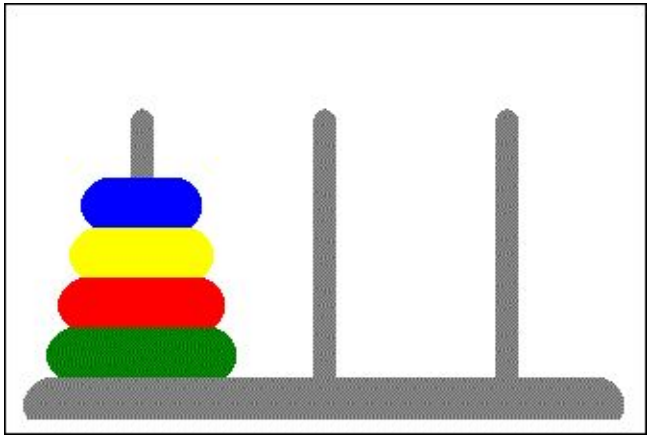


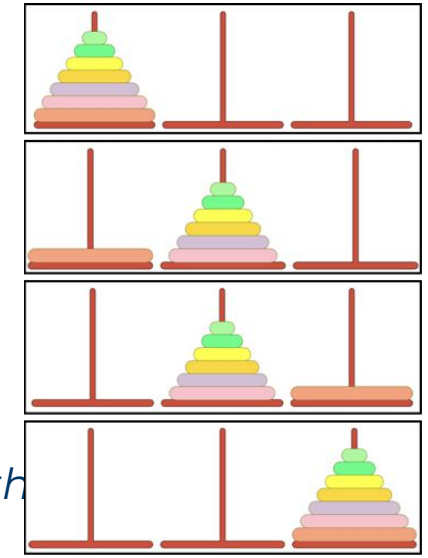
Let's see a recursive way to solve it:

- To move all disks from the initial pole to the last one corresponds to:
  - Take all disks at the top of the larger one (*The Teacher*) and move them (*The Students*) to the intermediary pole;
  - Move *The Teacher* to the final pole;
  - Take *The Students* from the intermediary pole back to the top of *The Teacher*.

Simulator: Towers of Hanoi.

Example 2 - Towers of Hanoi





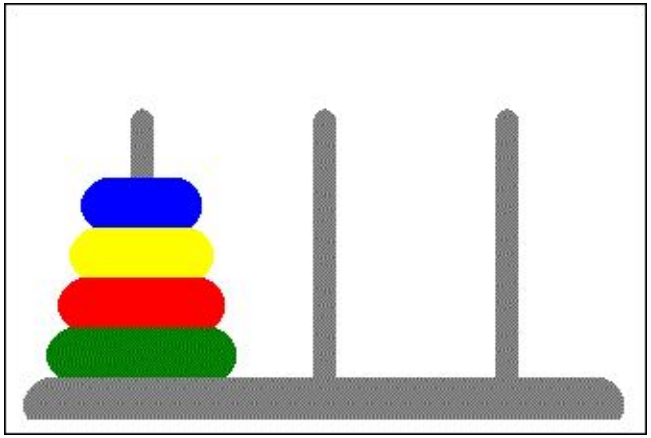
Let's see a recursive way to solve it:

- To move all disks from the initial pole to the last one corresponds to:
  - Take *The Students off th top of The Teacher* to the intermediary pole;
  - Move *The Teacher* to the final pole;
  - Take *The Students* from the intermediary pole back to the top of *The Teacher*.

To solve a ***n*** disk problem needs to solve two ***n-1*** disk problems plus 1 movement.

Video: How to solve the Towers of Hanoi.

Example 2 - Towers of Hanoi



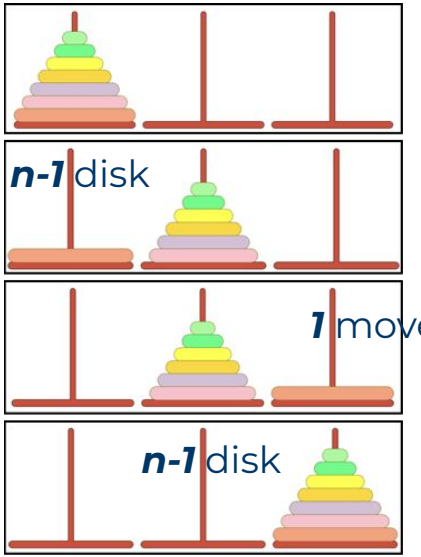
To solve a ***n*** disk problem needs to solve two ***n-1*** disk problems plus 1 movement.

The number of required movements, let's call it ***T(n)*** for ***n*** disks, can be expressed as:

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

The stop condition (trivial case) can be:

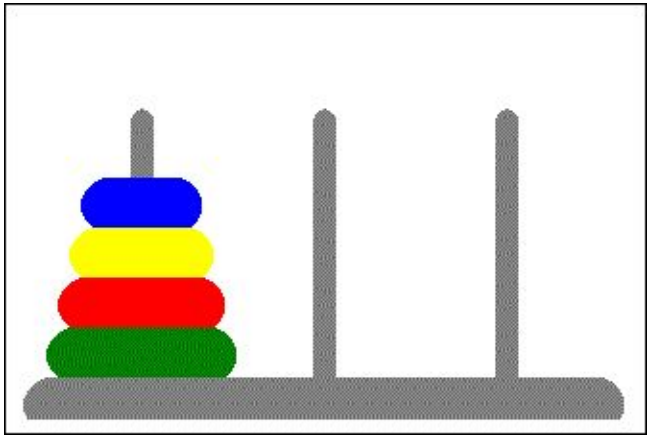
- ***T(0) = 0***



Stay in this page for now.
Don't go to the next ones yet.

Text: Towers of Hanoi (Khan Academy).

Example 2 - Towers of Hanoi



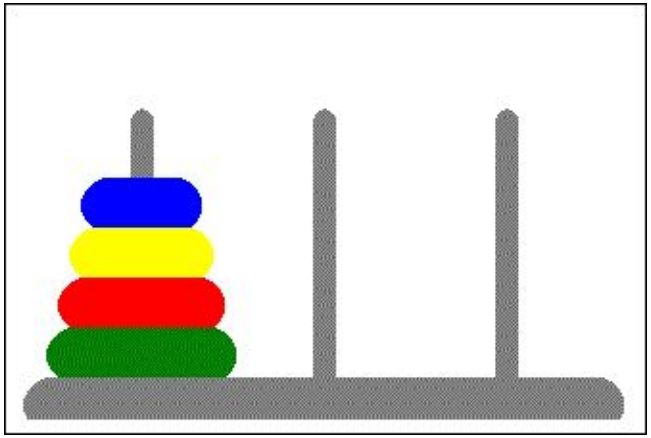
Considering:

- ***T(n) = 2 T(n-1) + 1***
- ***T(0) = 0***

- To solve an ***1*** disk problem requires just one movement:
  - ***T(1) = 1*** because it is ***2*** times ***T(0) + 1***
  - ***T(1) = 2 T(0) + 1***
- To solve a 2 disk problem requires 3 movements:
  - ***T(2) = 2 T(1) + 1 = 3***
- To solve a 3 disk problem requires 7 movements:
  - ***T(3) = 2 T(2) + 1 = 7***

Do you remember powers of 2?
$2^0 = 1, 2^1 = 2, 2^2 = 4, 2^3 = 8, 2^4 = 16$
Now minus 1...
$2^0-1 = \mathbf{0}, 2^1-1 = \mathbf{1}, 2^2-1 = \mathbf{3}, 2^3-1 = \mathbf{7}, 2^4-1 = \mathbf{15}$

Example 2 - Towers of Hanoi



```python
def hanoi(n):
    if (n == 0):
        return 0
    else:
        return 2 * hanoi(n-1) + 1

for i in range(10):
    print("with {} disks you need {} movements".format(i, hanoi(i)))
```

```
with 0 disks you need 0 movements
with 1 disks you need 1 movements
with 2 disks you need 3 movements
with 3 disks you need 7 movements
with 4 disks you need 15 movements
with 5 disks you need 31 movements
with 6 disks you need 63 movements
with 7 disks you need 127 movements
with 8 disks you need 255 movements
with 9 disks you need 511 movements
```

Considering:

- ***T(n) = 2 T(n-1) + 1***
- ***T(0) = 0***

Text: Towers of Hanoi, continued (Khan Academy).

Recursive Algorithms Examples

Algorithms Examples

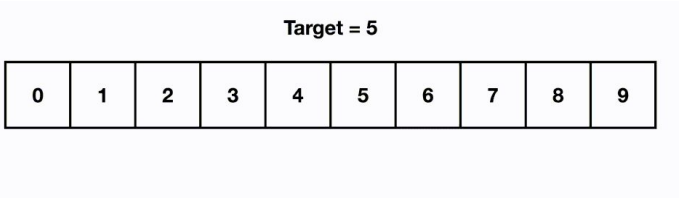
- depth first search in graphs (DFS)
- towers of hanoi
- **binary search**
- maximum element in an array

Example 3 - Binary Search

- Given a sorted array ***A*** with ***n*** elements
- Given a value ***k***
- If the value ***k*** is within the array ***A***:
  - Return the index ***i*** where the ***A[i] == k***; or return ***None***, otherwise.

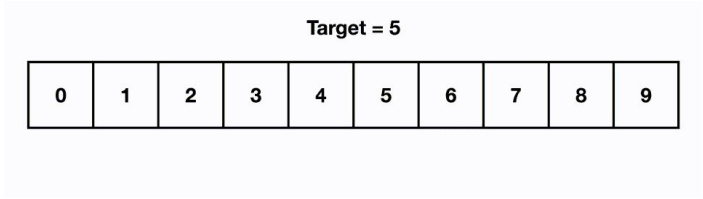
One recursive algorithm for binary search is:

- Giving a slice of the array (from ***start*** to ***end***), check the middle (***mid***):
  - If ***mid*** is equal to ***k***, returns the ***mid*** index;
  - If ***mid > k***, search from ***start*** to ***mid***;
  - If ***mid < k***, search from ***mid*** to ***end***;
- At the start check the whole array.



Wikipedia: Binary search.

Example 3 - Binary Search



- Giving the slice ***start*** to ***end***, check ***mid***:
  - If ***mid*** is equal to ***k***, returns the ***mid***;
  - If ***mid > k***, search from ***start*** to ***mid***;
  - If ***mid < k***, search from ***mid*** to ***end***;

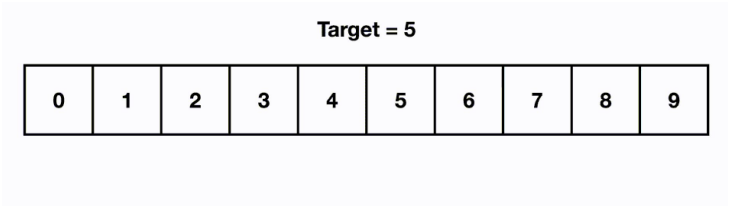
```python
def binSearch(A, start, end, k):
    mid = (end+start)//2
    if (start > end):
        return None
    elif (A[mid] == k):
        return mid
    elif (A[mid] > k):
        return binSearch(A, start, mid-1, k)
    else:
        return binSearch(A, mid+1, end, k)

A = list(range(10))
for i in [5, 2, 9, 4, 10]:
    print("{} is at index {}".format(i, binSearch(A, 0, len(A)-1, i)))
```

```
5 is at index 5
2 is at index 2
9 is at index 9
4 is at index 4
10 is at index None
```

- At the start check the whole array.

Video: Binary Search Algorithm (Recursive).

Example 3 - Binary Search



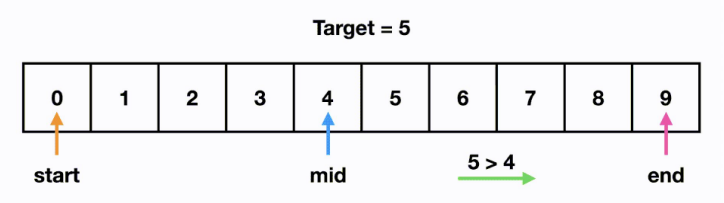
from 0 to 9 search 5

driver

1 of 10.

```python
def binSearch(A, start, end, k):
    mid = (end+start)//2
    if (start > end):
        return None
    elif (A[mid] == k):
        return mid
    elif (A[mid] > k):
        return binSearch(A, start, mid-1, k)
    else:
        return binSearch(A, mid+1, end, k)

A = list(range(10))
for i in [5, 2, 9, 4, 10]:
    print("{} is at index {}".format(i, binSearch(A, 0, len(A)-1, i)))
```

Example 3 - Binary Search



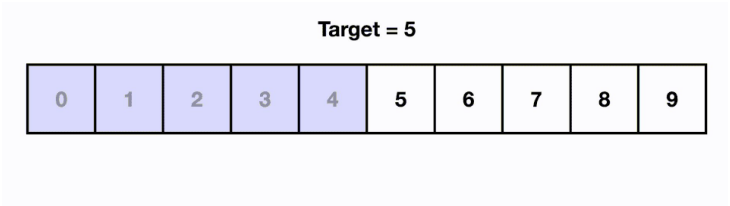
start = 0 - mid = 4 - end = 9

first call

```
1   def binSearch(A, start, end, k):
2       mid = (end+start)//2
3       if (start > end):
4           return None
5       elif (A[mid] == k):
6           return mid
7       elif (A[mid] > k):
8           return binSearch(A, start, mid-1, k)
9       else:
10          return binSearch(A, mid+1, end, k)
11
12  A = list(range(10))
13  for i in [5, 2, 9, 4, 10]:
14      print("{} is at index {}".format(i, binSearch(A, 0, len(A)-1, i)))
```

2 of 10.

Example 3 - Binary Search



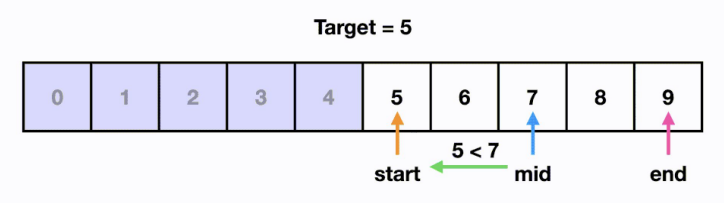
start = 0 - mid = 4 - end = 9

first call

3 of 10.

```python
def binSearch(A, start, end, k):
    mid = (end+start)//2
    if (start > end):
        return None
    elif (A[mid] == k):
        return mid
    elif (A[mid] > k):
        return binSearch(A, start, mid-1, k)
    else:
        return binSearch(A, mid+1, end, k)

A = list(range(10))
for i in [5, 2, 9, 4, 10]:
    print("{} is at index {}".format(i, binSearch(A, 0, len(A)-1, i)))
```

Example 3 - Binary Search



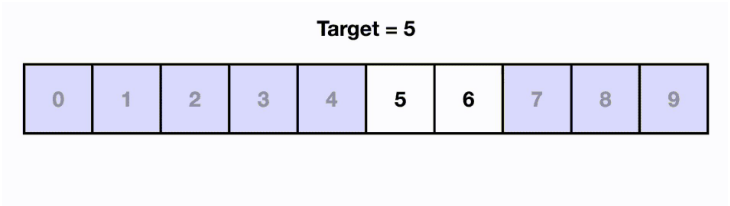
start = 5 - mid = 7 - end = 9

second call

```
1  def binSearch(A, start, end, k):
2      mid = (end+start)//2
3      if (start > end):
4          return None
5      elif (A[mid] == k):
6          return mid
7      elif (A[mid] > k):
8          return binSearch(A, start, mid-1, k)
9      else:
10         return binSearch(A, mid+1, end, k)
11
12 A = list(range(10))
13 for i in [5, 2, 9, 4, 10]:
14     print("{} is at index {}".format(i, binSearch(A, 0, len(A)-1, i)))
```

4 of 10.

Example 3 - Binary Search



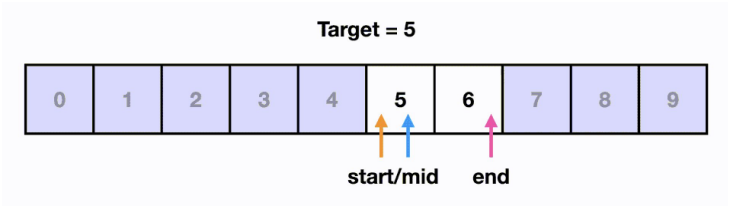
start = 5 - mid = 7 - end = 9

second call

5 of 10.

```python
def binSearch(A, start, end, k):
    mid = (end+start)//2
    if (start > end):
        return None
    elif (A[mid] == k):
        return mid
    elif (A[mid] > k):
        return binSearch(A, start, mid-1, k)
    else:
        return binSearch(A, mid+1, end, k)

A = list(range(10))
for i in [5, 2, 9, 4, 10]:
    print("{} is at index {}".format(i, binSearch(A, 0, len(A)-1, i)))
```

Example 3 - Binary Search



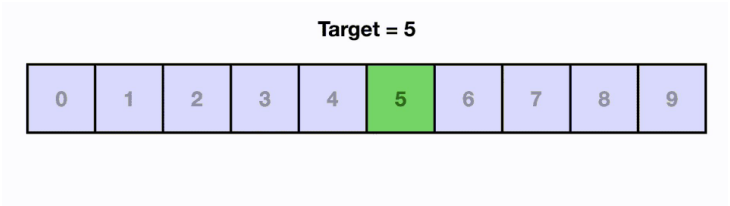
start = 5 - mid = 5 - end = 6

third call

```python
def binSearch(A, start, end, k):
    mid = (end+start)//2
    if (start > end):
        return None
    elif (A[mid] == k):
        return mid
    elif (A[mid] > k):
        return binSearch(A, start, mid-1, k)
    else:
        return binSearch(A, mid+1, end, k)

A = list(range(10))
for i in [5, 2, 9, 4, 10]:
    print("{} is at index {}".format(i, binSearch(A, 0, len(A)-1, i)))
```

6 of 10.

Example 3 - Binary Search



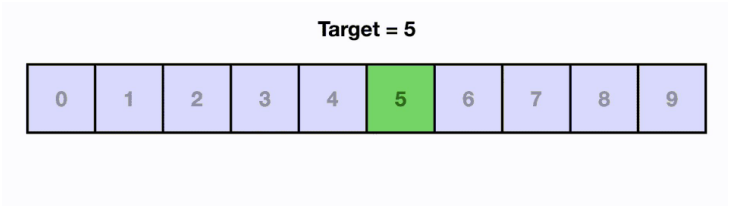
start = 5 - mid = 5 - end = 6

third call

```python
def binSearch(A, start, end, k):
    mid = (end+start)//2
    if (start > end):
        return None
    elif (A[mid] == k):
        return mid
    elif (A[mid] > k):
        return binSearch(A, start, mid-1, k)
    else:
        return binSearch(A, mid+1, end, k)

A = list(range(10))
for i in [5, 2, 9, 4, 10]:
    print("{} is at index {}".format(i, binSearch(A, 0, len(A)-1, i)))
```

7 of 10.

Example 3 - Binary Search



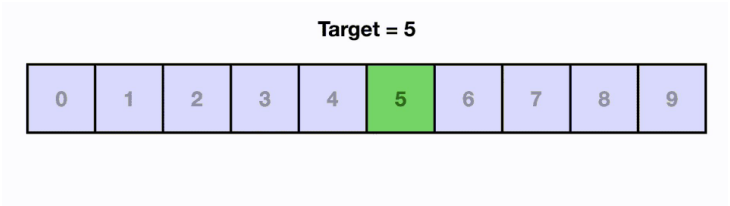
start = 5 - mid = 5 - end = 6

second call

8 of 10.

```python
def binSearch(A, start, end, k):
    mid = (end+start)//2
    if (start > end):
        return None
    elif (A[mid] == k):
        return mid
    elif (A[mid] > k):
        return binSearch(A, start, mid-1, k)
    else:
        return binSearch(A, mid+1, end, k)

A = list(range(10))
for i in [5, 2, 9, 4, 10]:
    print("{} is at index {}".format(i, binSearch(A, 0, len(A)-1, i)))
```

Example 3 - Binary Search



start = 0 - mid = 4 - end = 9

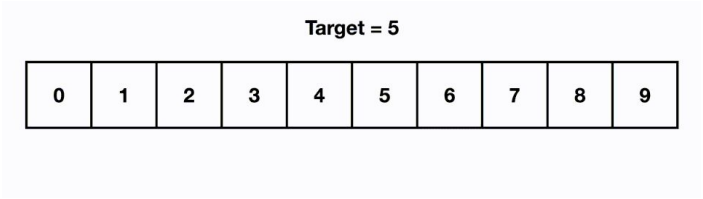
first call

9 of 10.

```python
def binSearch(A, start, end, k):
    mid = (end+start)//2
    if (start > end):
        return None
    elif (A[mid] == k):
        return mid
    elif (A[mid] > k):
        return binSearch(A, start, mid-1, k)
    else:
        return binSearch(A, mid+1, end, k)

A = list(range(10))
for i in [5, 2, 9, 4, 10]:
    print("{} is at index {}".format(i, binSearch(A, 0, len(A)-1, i)))
```

Example 3 - Binary Search

```
5 is at index 5
2 is at index 2
9 is at index 9
4 is at index 4
10 is at index None
```



from 0 to 9 search 5

driver

10 of 10.

```python
def binSearch(A, start, end, k):
    mid = (end+start)//2
    if (start > end):
        return None
    elif (A[mid] == k):
        return mid
    elif (A[mid] > k):
        return binSearch(A, start, mid-1, k)
    else:
        return binSearch(A, mid+1, end, k)

A = list(range(10))
for i in [5, 2, 9, 4, 10]:
    print("{} is at index {}".format(i, binSearch(A, 0, len(A)-1, i)))
```

Recursive Algorithms Examples

Algorithms Examples

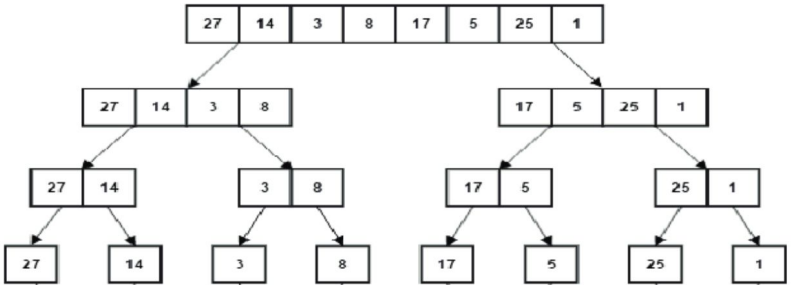
- depth first search in graphs (DFS)
- towers of hanoi
- binary search
- **maximum element in an array**

Example 4 - Maximum Element in an Array

- Given an unsorted array ***A*** with ***n*** elements
- Return the index of the largest element in this array

One recursive algorithm for finding the maximum element is:

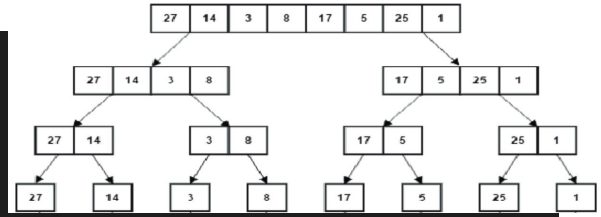
- Giving a slice of the array (from ***start*** to ***end***), the largest element is either the the largest among the first half or the larger of the second half of it;
- At the start check the whole array.



This problem is traditionally solved without recursion.

Example 4 - Maximum Element in an Array

- Giving a slice of the array (from ***start*** to ***end***), the largest element is either the the largest among the first half or the larger of the second half of it;
- At the start check the whole array.




```python
def Max(A, start, end):
    if (start == end):
        return end
    else:
        mid = (end+start)//2
        fst = Max(A, start, mid)
        lst = Max(A, mid+1, end)
        return fst if A[fst] > A[lst] else lst

from random import randint
A = [randint(0,100000) for _ in range(1000)]
i = Max(A, 0, len(A)-1)
print("The maximum number is", A[i], "at index", i)
```


The driver generates 1,000 elements from 0 to 99,999.

This Week's tasks

- In-class Exercise E#4
- Coding Project P#4
- Quiz Q#4

Tasks

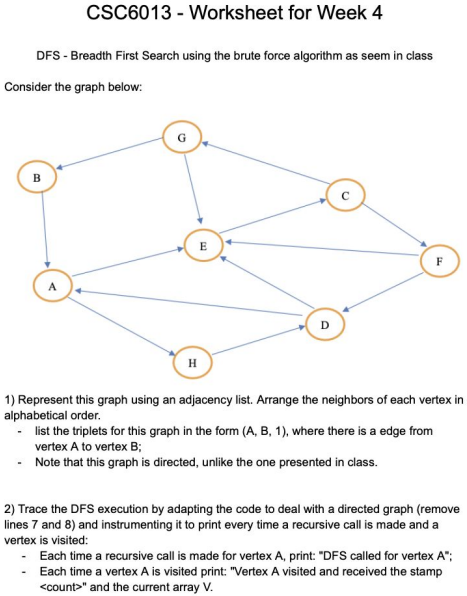
- Fill the worksheet as required.
- Adapt 2 Python programs:
  - selection sort;
  - bubble sort.
- Quiz #4 about this week topics.

In-class Exercise - E#4

Download the pdf (link here also) and perform the following tasks:

- (1) create the adjacency list for the graph presented;
- (2) Trace the algorithm printing every time a recursive call is made and every time a vertex is visited.

You have to submit a ***.pdf*** file with your answers.



CSC6013 - Worksheet for Week 4

DFS - Breadth First Search using the brute force algorithm as seem in class

Consider the graph below:

1) Represent this graph using an adjacency list. Arrange the neighbors of each vertex in alphabetical order.
   - list the triplets for this graph in the form (A, B, 1), where there is a edge from vertex A to vertex B;
   - Note that this graph is directed, unlike the one presented in class.

2) Trace the DFS execution by adapting the code to deal with a directed graph (remove lines 7 and 8) and instrumenting it to print every time a recursive call is made and a vertex is visited:
   - Each time a recursive call is made for vertex A, print: "DFS called for vertex A";
   - Each time a vertex A is visited print: "Vertex A visited and received the stamp <count>" and the current array V.

Feel free to type it or hand write it, but you have to submit a single ***.pdf*** with your answers.

This task counts towards the In-class Exercises grade and the deadline is This Friday.

Fourth Coding Project - P#4

1. **Adapt the Binary Search algorithm** saw in class to search a given element assuming that the array is not in ascending order, but instead it is sorted in descending order. Trace your algorithm execution printing:
    a. at each recursive call, print the subarray from start to end and show the mid element.
    b. test your algorithm for the arrays:
        i. A1 = [99, 67, 56, 51, 44, 39, 38, 23, 21, 17, 11, 2] searching for 44;
        ii. A1 = [99, 67, 56, 51, 44, 39, 38, 23, 21, 17, 11, 2] searching for 56;
        iii. A1 = [99, 67, 56, 51, 44, 39, 38, 23, 21, 17, 11, 2] searching for 42;
        iv. A2 = [9, 7, 6, 4, 2, 0, -1, -3, -5, -8, -9] searching for -1;
        v. A2 = [9, 7, 6, 4, 2, 0, -1, -3, -5, -8, -9] searching for -7.

To both programs you have to submit the code (***.py*** file) of your adapted algorithm and a ***.pdf*** with the test cases outputs.

Two algorithms to adapt, this is the first.

Fourth Coding Project - P#4

2. **Adapt the Maximum Element in an Array algorithm** saw in class to search for the minimum element in the array. Trace your algorithm execution printing:
    a. at the beginning of each recursive call the start and end values.
    b. at the returning of each recursive call the returned value (the minimum of the array slice).
    c. test your algorithm for the arrays:
        i. A3 = [44, 63, 77, 17, 20, 99, 84, 6, 39, 52]
        ii. A4 = [52, 84, 6, 39, 20, 77, 17, 99, 44, 63]
        iii. A5 = [6, 17, 20, 39, 44, 52, 63, 77, 84, 99]

Your task:

Go to Canvas, and submit your ***.py*** files and ***.pdf*** files within the deadline.

This assignment counts towards the Projects grade and the deadline is Next Monday.

Fourth Quiz - Q#4

- The fourth quiz in this course covers the topics of Week 4;
- The quiz will be available this Friday, and it is composed by 10 questions;
- The quiz should be taken on Canvas (Module 4), and it is not a timed quiz:
  - You can take as long as you want to answer it (a quiz taken in less than one hour is usually a too short time);
- The quiz is open book, open notes, and you can even use any language Interpreter to answer it;
- Yet, the quiz is evaluated and you are allowed to submit it only once.

Your task:

- Go to Canvas, answer the quiz and submit it within the deadline.

This quiz counts towards the Quizzes grade and the deadline is Next Monday.

Welcome to CSC 6013

- Do In-class Exercise E#4 until Friday;
- Do Quiz Q#4 (available Friday) until next Monday;
- Do Coding Project P#4 until next Monday.

Next Week - Complexity of Recursive Algorithms

MERRIMACK COLLEGE