Presentation Agenda

Week 6
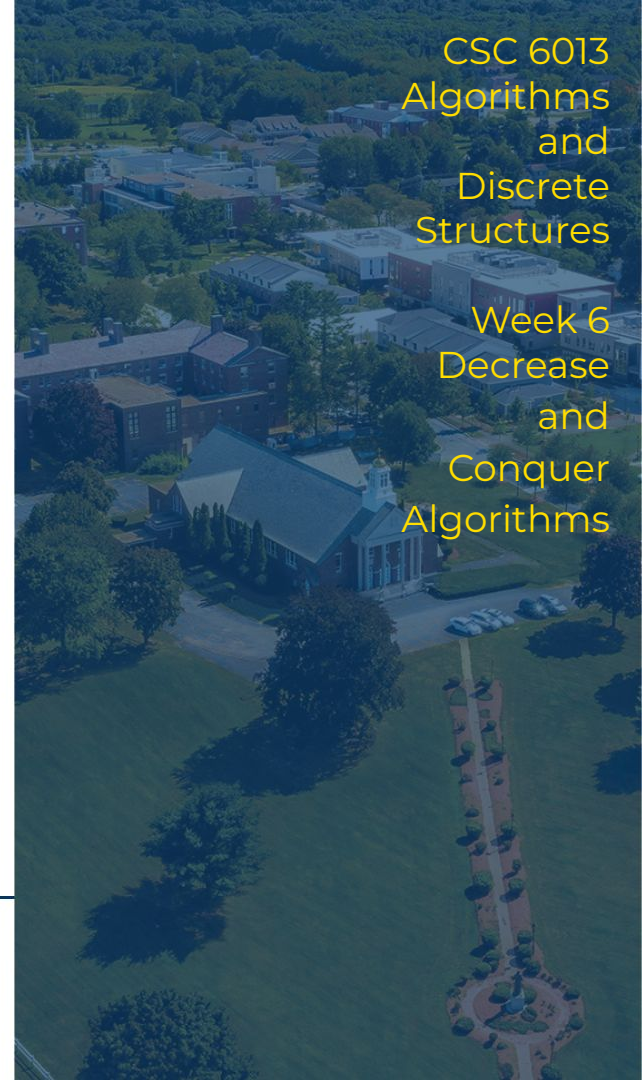- Decrease-and-Conquer
   a. kinds (const. amount, factor, and variable)
- Examples
   a. Insertion sort
   b. Topological sort
   c. Fake coin detection
   d. Russian peasants' multiplication
   e. Euclidean Greatest Common Divisor
   f. Lomuto partition
   g. K-th order statistic
- This Week's tasks

MERRIMACK COLLEGE

CSC 6013
Algorithms
and
Discrete
Structures

Week 6
Decrease
and
Conquer
Algorithms

Decrease-and-Conquer Algorithms

Each time your problem is a little smaller.

Some of the better known algorithms belong to the ...-and-conquer family.

- **Basic Definitions**
  - Decrease-and-Conquer
  - Divide-and-Conquer
  - Transform-and-Conquer

- Kinds of Decrease-and-Conquer
  - Decrease by a constant amount
  - Decrease by a constant factor
  - Decrease by a variable amount and factor

MERRIMACK COLLEGE

The ...-and-conquer family

Probably, one of the most known algorithmic techniques is the ***divide-and-conquer***. It has been talked about consistently (documented with this name) at least since the times of Julius Caesar, i.e., 2000 years ago!

However, there are other similar approaches to solve problems, and those are known as the ***...-and conquer family*** in the algorithm analysis context.

The more typical kind of algorithm of the family is the ***Decrease-and-Conquer*** kind, but let's talk about the family first, as many authors consider all problems in the family Divide-and-Conquer, and we do not.

MERRIMACK COLLEGE

Wikipedia: Divide-and-Conquer.
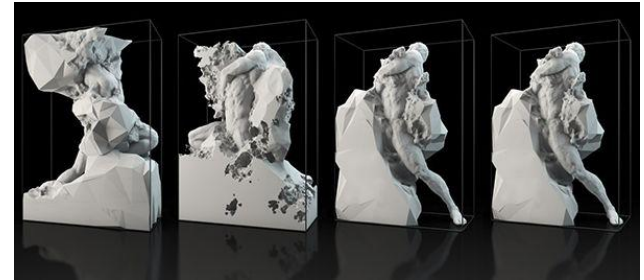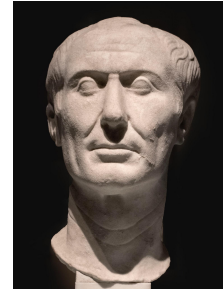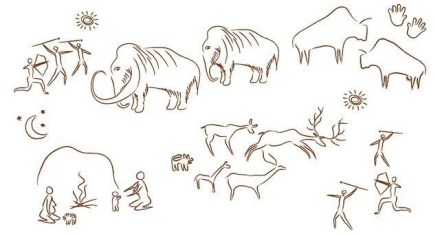
The …-and-conquer family
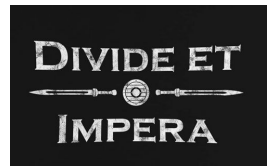
Julius Caesar became famous and rich because the conquest of Gaul from 58 to 50 bce.

The strategy, as written by Julius Caesar himself, was to divide-and-conquer (in Latin: **Divide et Impera**), i.e., not attacking Gaul entirely, but attacking and conquering a small tribe here and there until all tribes are vanquished, so he could claim the conquest of Gaul.

In such way, the …-and-conquer family was baptized by Julius Caesar reflecting his own algorithm to conquer a larger enemy with a smaller force.

Meet the ...-and-conquer family

While in military, politics, management, etc., the usage of divide-and-conquer became extremely popular, and effective, in algorithms we have a subdivision of the algorithm techniques:

- **Divide-and-conquer**
  - When the original problem is broken into complementary parts;
- **Decrease-and-conquer**
  - When at each time the problem becomes smaller, but not into complementary problems;
- **Transform-and-conquer**
  - When the problem not always becomes smaller.
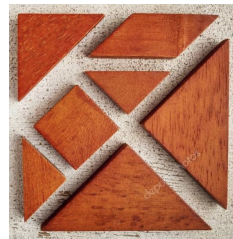
This division is acknowledged by great authors (as our textbook ones: Cormen, Leiserson, Rivest, and Stein), but it is not unanimous.

MERRIMACK COLLEGE

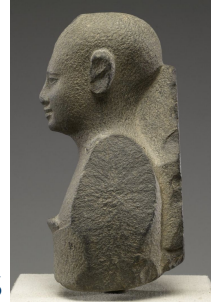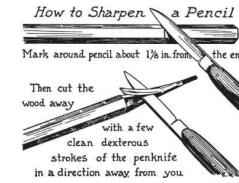# Decrease-and-Conquer Algorithms

Each time your problem is a little smaller.

Some of the better known algorithms belong to the ...-and-conquer family.

- Basic Definitions
  - Decrease-and-Conquer
  - Divide-and-Conquer
  - Transform-and-Conquer

- **Kinds of Decrease-and-Conquer**
  - Decrease by a constant amount
  - Decrease by a constant factor
  - Decrease by a variable amount and factor

Decrease-and-Conquer

Just like sculpting a block of stone or sharpening a pencil, the decrease-and-conquer algorithms are all about getting closer to the result by taking off bits of the problem until you get a problem that is already solved.

The recursive factorial seen previously was a decrease-and-conquer example:

```
def fact(n):
    if (n == 1):
        return 1
    else:
        return n * fact(n-1)
```

- If you want to know the factorial of **n**, and **n** is large, you can find out how much is the factorial of **n-1**;
- And you keep on reducing this problem to **n-2**, **n-3**, … until you get a problem that is not large anymore, for example the factorial of **1**, which has a trivial solution.

MERRIMACK COLLEGE

The Towers of Hanoi example also was resolved decreasing the problem **T(n) = 1 + 2T(n-1)** until **T(0) = 0**.

Decrease-and-Conquer

Despite the examples of recursive implementation of decrease-and-conquer algorithms, it does not need to be recursive.

Yet, we will define it frequently the algorithm recursively, even though it can also be implemented iteratively.

For example, see these two codes to compute the exponentiation n^k.

$O(k)$

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

Either way, the algorithm will be mathematically defined recursively:

- $T(n) = 1 + T(n-1)$
- $T(0) = 1$

$O(k)$

```
1    def power(n, k):
2        if (k == 0):
3            return 1
4        else:
5            return n * power(n, k-1)
```

MERRIMACK COLLEGE

Both implementations are decrease-and-conquer, one forward, the other backwards.
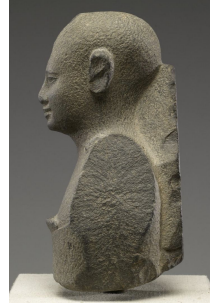
Decrease-and-Conquer

All Decrease-and-Conquer algorithms reduce the size of the problem repeatedly until you get a trivial problem to solve.

The kinds of Decrease-and-Conquer algorithms are defined according to how the problem size decreases:

- Algorithms that decreases at a constant amount;
- Algorithms that decreases at a constant factor;
- Algorithms that decreases at a variable amount and factor.

- The Towers of Hanoi always decrease 1 size at each step:
  - *T(n) = 2 T(n-1) + 1*
- The Binary Search always decrease to a half at each step:
  - *T(n) = T(n/2) + 2*
- The Euclid's algorithm for the GCD
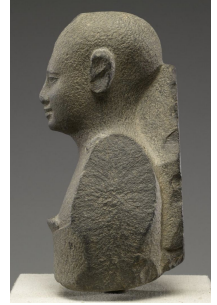
MERRIMACK COLLEGE

We will consider recursive implementations because it is more intuitive.

Divide-and-Conquer Algorithms Examples

The examples that will be seen are:
- Decrease by constant amount:
  - **Insertion sort**
  - Topological sort
- Decrease by constant factor:
  - Fake coin detection
  - Russian peasants' multiplication
- Decrease by variable amount and factor:
  - Euclidean GCD
  - Lomuto partition
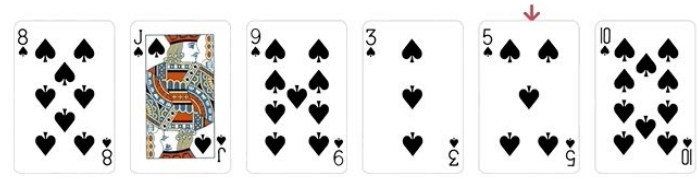 - K-th order statistic

Example 1 - Insertion Sort

- Sorting algorithm that decreases at a constant amount and is not recursive.
- Giving an array $A = [a_1 , a_2 , … a_n]$
- Deliver a permutation of A where $a_i <= a_j$ if $i < j$
- A decrease-and-conquer solution is:
  - Assume that the last element alone is sorted, then starting from the before last element:
    - Find its place in the last (already sorted) elements and insert it there causing a scootch over behavior;
    - Repeat the process to the element before that and keep on going until places the first element.

At each iteration there are $n-i$ unsorted elements. It decreases by a constant amount (1).

[8][J][9][3][5][10]
stays left of [10]
[8][J][9][3][5][10]
stays left of [5]
[8][J][9][3][5][10]
to the right of [5]
[8][J][3][5][9][10]
to the right of [10]
[8][3][5][9][10][J]
to the right of [5]
[3][5][8][9][10][J]

Video: Insertion sort in 2 minutes.

Example 1 - Insertion Sort

- Assume that the last element alone is sorted, then starting from the before last element:
    - Find its place in the last (already sorted) elements and insert it there causing a scootch over behavior;
    - Repeat the process to the element before that and keep on going until places the first element.

```python
1    def insertionSort(A):
2        for i in range(len(A)-2, -1, -1):      # from the before last to the first
3            for j in range(len(A)-1, i, -1):   # from the last to before i
4                if (A[i] > A[j]):              # if found A[i] place
5                    A.insert(j, A.pop(i))      # take it out and place at j
6                    break                      # go to the next to be placed
7
8    A = [8, 11, 9, 3, 5, 10]
9    insertionSort(A)
10   print(A)
```

```
[8, 11,  9,  3,  5, 10]
[8, 11,  3,  5,  9, 10]
[8,  3,  5,  9, 10, 11]
[3,  5,  8,  9, 10, 11]
```
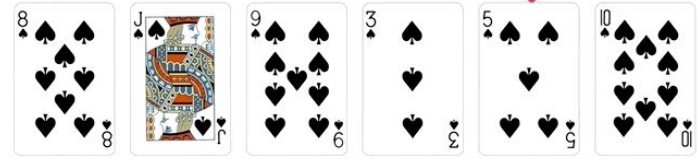
[8][J][9][3][5][10]
stays left of [10]
[8][J][9][3][5][10]
stays left of [5]
[8][J][9][3][5][10]
to the right of [5]
[8][J][3][5][9][10]
to the right of [10]
[8][3][5][9][10][J]
to the right of [5]
[3][5][8][9][10][J]

Example 1 - Insertion Sort
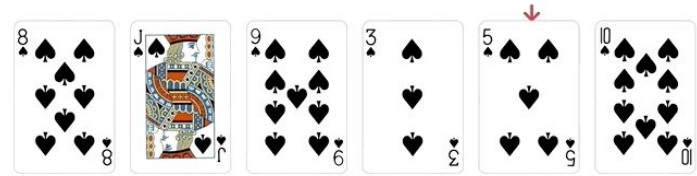
- Asymptotic Analysis
  - Line 2 is executed **n-1** times
  - Lines 3 and 4 are executed **((n-1)(n))/2** times
  - Lines 5 and 6 are executed **n-1** times

- Upper bounding **n-1** to **n** and ignoring the constant **1/2**

```python
1   def insertionSort(A):
2       for i in range(len(A)-2, -1, -1):    # from the before last to the first
3           for j in range(len(A)-1, i, -1): # from the last to before i
4               if (A[i] > A[j]):            # if found A[i] place
5                   A.insert(j, A.pop(i))    # take it out and place at j
6                   break                    # go to the next to be placed
7
8   A = [8, 11, 9, 3, 5, 10]
9   insertionSort(A)
10  print(A)
```

At each iteration there are **n-i** unsorted elements. It decreases by a constant amount (1).

[8][J][9][3][5][10]
stays left of [10]
[8][J][9][3][5][10]
stays left of [5]
[8][J][9][3][5][10]
to the right of [5]
[8][J][3][5][9][10]
to the right of [10]
[8][3][5][9][10][J]
to the right of [5]
[3][5][8][9][10][J]

MERRIMACK COLLEGE

$O(n^2)$ - Quadratic Algorithm

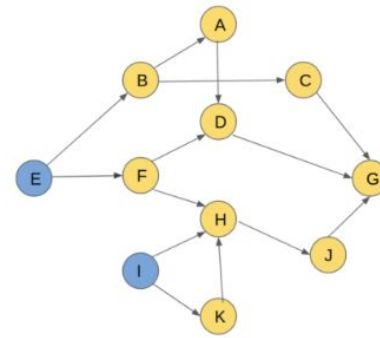# Divide-and-Conquer Algorithms Examples

The examples that will be seen are:
- Decrease by constant amount:
  - Insertion sort
  - **Topological sort**
- Decrease by constant factor:
  - Fake coin detection
  - Russian peasants' multiplication
- Decrease by variable amount and factor:
  - Euclidean GCD
  - Lomuto partition
  - K-th order statistic

Example 2 - Topological Sort

- Given a Directed Acyclic Graph - find an array **A** of vertices such as if a vertex **v** appears before vertex **w** in the array, there is no path from w to **v**:
  - Conversely, put the vertices in an order that no successor precedes its ancestor.
- Given a graph defined by a **V** set of **n** Vertices and **E** a set of **m** Edges, each one being a triplet **(v, w, 1)** representing an edge going from **v** towards **w**.
- Main idea:
  - find a vertex with no incoming edges, remove it and insert it into the array **A**;
  - Repeat the process until all vertices are removed and inserted.

At each call there are 1 less vertex to sort.
It decreases by a constant amount (1).

```
[ ]              [EBACFD]
[E]              [EBACFDI]
[EB]             [EBACFDIK]
[EBA]            [EBACFDIKH]
[EBAC]           [EBACFDIKHJ]
[EBACF]          [EBACFDIKHJG]
```

MERRIMACK COLLEGE

Wikipedia: Topological sorting.

Example 2 - Topological Sort

- Main idea:
  - Find a vertex with no incoming edges, remove it and insert it into the array **A**;
  - Repeat the process until all vertices are removed/inserted.
- Implementation:
  - For all vertices, check if there is an incoming edge from a vertex not yet in **A**, and if not so, include the vertex in **A** and consider the recursive call done;
    - If at the end no vertex was included in **A**, stop it (no DAG);
    - If at the end all vertices are in **A**, stop it!
    - If at the end not all vertices are in **A**, do another recursive call.
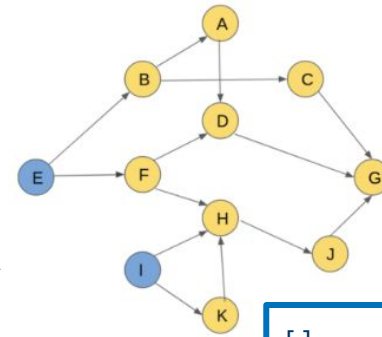
At each call there are 1 less vertex to sort.

It decreases by a constant amount (1).

[ ]
[E]
[EB]
[EBA]
[EBAC]
[EBACF]
[EBACFD]
[EBACFDI]
[EBACFDIK]
[EBACFDIKH]
[EBACFDIKHJ]
[EBACFDIKHJG]

MERRIMACK COLLEGE

Video: Topological Sort Visualized and Explained.

Example 2 - Topological Sort

- For all vertices, check if there is an incoming edge from a vertex not yet in **A**, and if not so, include the vertex in **A** and consider the recursive call done;
  - If at the end no vertex was included in **A**, stop it (no DAG);
  - If at the end all vertices are in **A**, stop it!
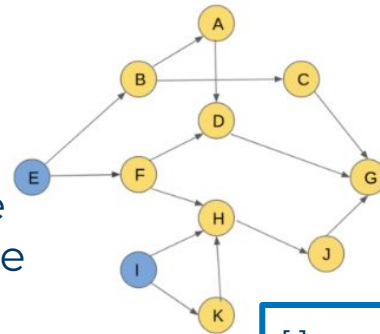 - If at the end not all vertices are in **A**, do another recursive call.

```
1   def topoSort(V, E, A):
2       size = len(A)
3       for v in range(len(V)):
4           if (v not in A):
5               for e in E:
6                   if (v == e[1]) and (e[0] not in A):
7                       break
8               else:
9                   A.append(v)
10                  break
11              continue
12      if (len(A) == size):
13          return False          # stop condition not DAG
14      elif (len(V) == len(A)):
15          return True           # stop condition all in!
16      else:
17          return topoSort(V,E,A) # do another recursive call
```

[ ]
[E]
[EB]
[EBA]
[EBAC]
[EBACF]
[EBACFD]
[EBACFDI]
[EBACFDIK]
[EBACFDIKH]
[EBACFDIKHJ]
[EBACFDIKHJG]

MERRIMACK COLLEGE

Example 2 - Topological Sort

- Step 1 of 13

```
V = ["A","B","C","D","E",
     "F","G","H","I","J","K"]
E = [[0,3,1],  #A --> D
     [1,0,1],  #B --> A
     [1,2,1],  #B --> C
     [2,6,1],  #C --> G
     [3,6,1],  #D --> G
     [4,1,1],  #E --> B
     [4,5,1],  #E --> F
     [5,3,1],  #F --> D
     [5,7,1],  #F --> H
     [7,9,1],  #H --> J
     [8,7,1],  #I --> H
     [8,10,1], #I --> K
     [9,6,1],  #J --> G
     [10,7,1]] #K --> H
```

RESULT :

```
A = []
if (topoSort(V,E,A)):
    print("Topological order found:")
    for a in A:
        print(V[a], end=" ")
    print()
else:
    print("The graph is not a DAG!")
```

```
1   def topoSort(V, E, A):
2       size = len(A)
3       for v in range(len(V)):
4           if (v not in A):
5               for e in E:
6                   if (v == e[1]) and (e[0] not in A):
7                       break
8                   else:
9                       A.append(v)
10                      break
11              continue
12      if (len(A) == size):
13          return False        # stop condition not DAG
14      elif (len(V) == len(A)):
15          return True         # stop condition all in!
16      else:
17          return topoSort(V,E,A) # do another recursive call
```
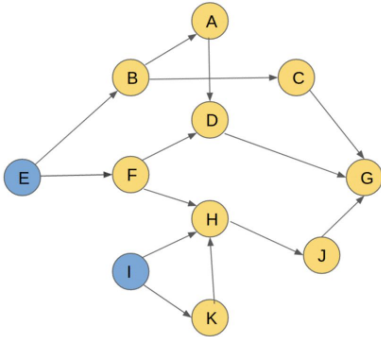
MERRIMACK COLLEGE

Example 2 - Topological Sort

```
V = ["A","B","C","D","E",
     "F","G","H","I","J","K"]
E = [[0,3,1],  #A --> D
     [1,0,1],  #B --> A
     [1,2,1],  #B --> C
     [2,6,1],  #C --> G
     [3,6,1],  #D --> G
     [4,1,1],  #E --> B
     [4,5,1],  #E --> F
     [5,3,1],  #F --> D
     [5,7,1],  #F --> H
     [7,9,1],  #H --> J
     [8,7,1],  #I --> H
     [8,10,1], #I --> K
     [9,6,1],  #J --> G
     [10,7,1]] #K --> H
```
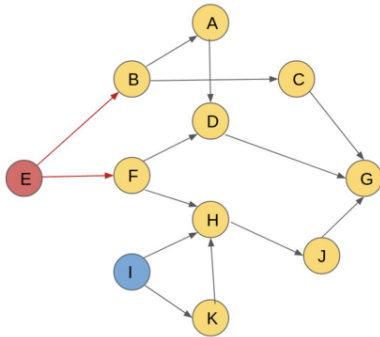
RESULT :
E

```
A = []
if (topoSort(V,E,A)):
    print("Topological order found:")
    for a in A:
        print(V[a], end=" ")
    print()
else:
    print("The graph is not a DAG!")
```

```
1    def topoSort(V, E, A):
2        size = len(A)
3        for v in range(len(V)):
4            if (v not in A):
5                for e in E:
6                    if (v == e[1]) and (e[0] not in A):
7                        break
8                    else:
9                        A.append(v)
10                       break
11               continue
12       if (len(A) == size):
13           return False         # stop condition not DAG
14       elif (len(V) == len(A)):
15           return True          # stop condition all in!
16       else:
17           return topoSort(V,E,A) # do another recursive call
```

MERRIMACK COLLEGE

Example 2 - Topological Sort

- Step 3 of 13


```
V = ["A","B","C","D","E",
     "F","G","H","I","J","K"]
E = [[0,3,1],  #A --> D
     [1,0,1],  #B --> A
     [1,2,1],  #B --> C
     [2,6,1],  #C --> G
     [3,6,1],  #D --> G
     [4,1,1],  #E --> B
     [4,5,1],  #E --> F
     [5,3,1],  #F --> D
     [5,7,1],  #F --> H
     [7,9,1],  #H --> J
     [8,7,1],  #I --> H
     [8,10,1], #I --> K
     [9,6,1],  #J --> G
     [10,7,1]] #K --> H
```
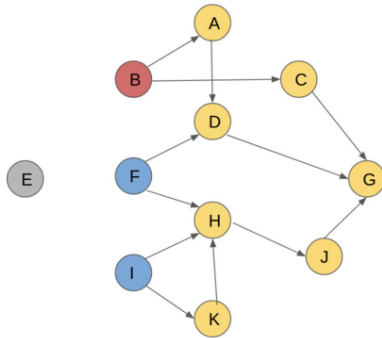
RESULT :
EB

```
A = []
if (topoSort(V,E,A)):
    print("Topological order found:")
    for a in A:
        print(V[a], end=" ")
    print()
else:
    print("The graph is not a DAG!")
```

```
1    def topoSort(V, E, A):
2        size = len(A)
3        for v in range(len(V)):
4            if (v not in A):
5                for e in E:
6                    if (v == e[1]) and (e[0] not in A):
7                        break
8                else:
9                    A.append(v)
10                   break
11               continue
12       if (len(A) == size):
13           return False          # stop condition not DAG
14       elif (len(V) == len(A)):
15           return True           # stop condition all in!
16       else:
17           return topoSort(V,E,A) # do another recursive call
```

Example 2 - Topological Sort

```
V = ["A","B","C","D","E",
     "F","G","H","I","J","K"]
E = [[0,3,1],  #A --> D
     [1,0,1],  #B --> A
     [1,2,1],  #B --> C
     [2,6,1],  #C --> G
     [3,6,1],  #D --> G
     [4,1,1],  #E --> B
     [4,5,1],  #E --> F
     [5,3,1],  #F --> D
     [5,7,1],  #F --> H
     [7,9,1],  #H --> J
     [8,7,1],  #I --> H
     [8,10,1], #I --> K
     [9,6,1],  #J --> G
     [10,7,1]] #K --> H
```
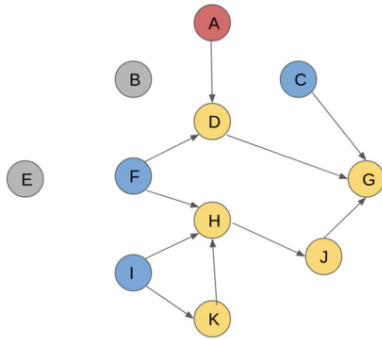
```
A = []
if (topoSort(V,E,A)):
    print("Topological order found:")
    for a in A:
        print(V[a], end=" ")
    print()
else:
    print("The graph is not a DAG!")
```

RESULT :
EBA

```
1    def topoSort(V, E, A):
2        size = len(A)
3        for v in range(len(V)):
4            if (v not in A):
5                for e in E:
6                    if (v == e[1]) and (e[0] not in A):
7                        break
8                else:
9                    A.append(v)
10                   break
11               continue
12       if (len(A) == size):
13           return False          # stop condition not DAG
14       elif (len(V) == len(A)):
15           return True           # stop condition all in!
16       else:
17           return topoSort(V,E,A) # do another recursive call
```

MERRIMACK COLLEGE

Example 2 - Topological Sort
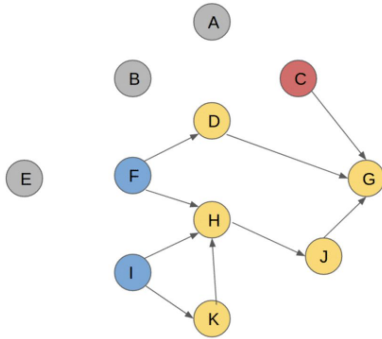
```
A = []
if (topoSort(V,E,A)):
    print("Topological order found:")
    for a in A:
        print(V[a], end=" ")
    print()
else:
    print("The graph is not a DAG!")
```

```
V = ["A","B","C","D","E",
     "F","G","H","I","J","K"]
E = [[0,3,1],  #A --> D
     [1,0,1],  #B --> A
     [1,2,1],  #B --> C
     [2,6,1],  #C --> G
     [3,6,1],  #D --> G
     [4,1,1],  #E --> B
     [4,5,1],  #E --> F
     [5,3,1],  #F --> D
     [5,7,1],  #F --> H
     [7,9,1],  #H --> J
     [8,7,1],  #I --> H
     [8,10,1], #I --> K
     [9,6,1],  #J --> G
     [10,7,1]] #K --> H
```

RESULT :
EBAC

```
1   def topoSort(V, E, A):
2       size = len(A)
3       for v in range(len(V)):
4           if (v not in A):
5               for e in E:
6                   if (v == e[1]) and (e[0] not in A):
7                       break
8               else:
9                   A.append(v)
10                  break
11              continue
12      if (len(A) == size):
13          return False          # stop condition not DAG
14      elif (len(V) == len(A)):
15          return True           # stop condition all in!
16      else:
17          return topoSort(V,E,A) # do another recursive call
```


MERRIMACK COLLEGE

Example 2 - Topological Sort

```
V = ["A","B","C","D","E",
     "F","G","H","I","J","K"]
E = [[0,3,1],  #A --> D
     [1,0,1],  #B --> A
     [1,2,1],  #B --> C
     [2,6,1],  #C --> G
     [3,6,1],  #D --> G
     [4,1,1],  #E --> B
     [4,5,1],  #E --> F
     [5,3,1],  #F --> D
     [5,7,1],  #F --> H
     [7,9,1],  #H --> J
     [8,7,1],  #I --> H
     [8,10,1], #I --> K
     [9,6,1],  #J --> G
     [10,7,1]] #K --> H
```
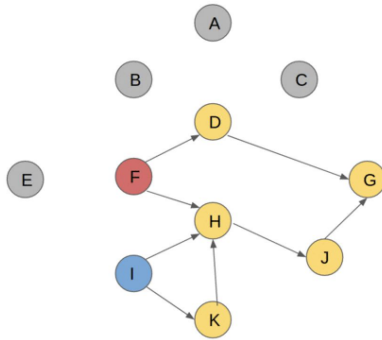
RESULT :
EBACF

```python
A = []
if (topoSort(V,E,A)):
    print("Topological order found:")
    for a in A:
        print(V[a], end=" ")
    print()
else:
    print("The graph is not a DAG!")
```

```python
1   def topoSort(V, E, A):
2       size = len(A)
3       for v in range(len(V)):
4           if (v not in A):
5               for e in E:
6                   if (v == e[1]) and (e[0] not in A):
7                       break
8                   else:
9                       A.append(v)
10                      break
11              continue
12      if (len(A) == size):
13          return False        # stop condition not DAG
14      elif (len(V) == len(A)):
15          return True         # stop condition all in!
16      else:
17          return topoSort(V,E,A) # do another recursive call
```

MERRIMACK COLLEGE

Example 2 - Topological Sort

```
V = ["A","B","C","D","E",
     "F","G","H","I","J","K"]
E = [[0,3,1],   #A --> D
     [1,0,1],   #B --> A
     [1,2,1],   #B --> C
     [2,6,1],   #C --> G
     [3,6,1],   #D --> G
     [4,1,1],   #E --> B
     [4,5,1],   #E --> F
     [5,3,1],   #F --> D
     [5,7,1],   #F --> H
     [7,9,1],   #H --> J
     [8,7,1],   #I --> H
     [8,10,1],  #I --> K
     [9,6,1],   #J --> G
     [10,7,1]]  #K --> H
```
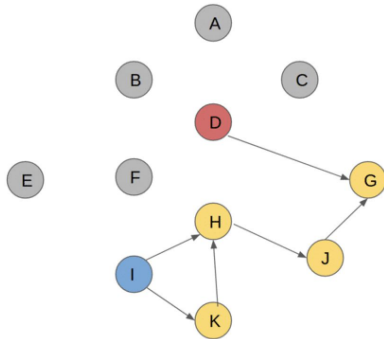
RESULT :
EBACFD

```
A = []
if (topoSort(V,E,A)):
    print("Topological order found:")
    for a in A:
        print(V[a], end=" ")
    print()
else:
    print("The graph is not a DAG!")
```

```python
1   def topoSort(V, E, A):
2       size = len(A)
3       for v in range(len(V)):
4           if (v not in A):
5               for e in E:
6                   if (v == e[1]) and (e[0] not in A):
7                       break
8                   else:
9                       A.append(v)
10                      break
11              continue
12      if (len(A) == size):
13          return False          # stop condition not DAG
14      elif (len(V) == len(A)):
15          return True           # stop condition all in!
16      else:
17          return topoSort(V,E,A) # do another recursive call
```

MERRIMACK COLLEGE

Example 2 - Topological Sort

- Step 8 of 13

```
V = ["A","B","C","D","E",
     "F","G","H","I","J","K"]
E = [[0,3,1],  #A --> D
     [1,0,1],  #B --> A
     [1,2,1],  #B --> C
     [2,6,1],  #C --> G
     [3,6,1],  #D --> G
     [4,1,1],  #E --> B
     [4,5,1],  #E --> F
     [5,3,1],  #F --> D
     [5,7,1],  #F --> H
     [7,9,1],  #H --> J
     [8,7,1],  #I --> H
     [8,10,1], #I --> K
     [9,6,1],  #J --> G
     [10,7,1]] #K --> H
```
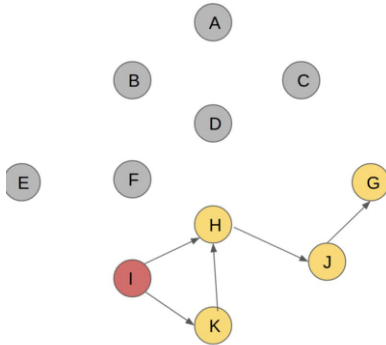
```
A = []
if (topoSort(V,E,A)):
    print("Topological order found:")
    for a in A:
        print(V[a], end=" ")
    print()
else:
    print("The graph is not a DAG!")
```

```
1   def topoSort(V, E, A):
2       size = len(A)
3       for v in range(len(V)):
4           if (v not in A):
5               for e in E:
6                   if (v == e[1]) and (e[0] not in A):
7                       break
8                   else:
9                       A.append(v)
10                      break
11              continue
12          if (len(A) == size):
13              return False        # stop condition not DAG
14          elif (len(V) == len(A)):
15              return True         # stop condition all in!
16          else:
17              return topoSort(V,E,A) # do another recursive call
```

RESULT :
EBACFDI

Example 2 - Topological Sort

```
V = ["A","B","C","D","E",
     "F","G","H","I","J","K"]
E = [[0,3,1],   #A --> D
     [1,0,1],   #B --> A
     [1,2,1],   #B --> C
     [2,6,1],   #C --> G
     [3,6,1],   #D --> G
     [4,1,1],   #E --> B
     [4,5,1],   #E --> F
     [5,3,1],   #F --> D
     [5,7,1],   #F --> H
     [7,9,1],   #H --> J
     [8,7,1],   #I --> H
     [8,10,1],  #I --> K
     [9,6,1],   #J --> G
     [10,7,1]]  #K --> H
```
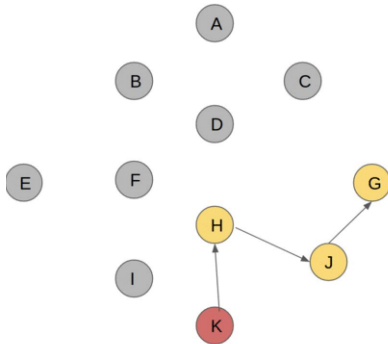
```
A = []
if (topoSort(V,E,A)):
    print("Topological order found:")
    for a in A:
        print(V[a], end=" ")
    print()
else:
    print("The graph is not a DAG!")
```

```python
1   def topoSort(V, E, A):
2       size = len(A)
3       for v in range(len(V)):
4           if (v not in A):
5               for e in E:
6                   if (v == e[1]) and (e[0] not in A):
7                       break
8               else:
9                   A.append(v)
10                  break
11              continue
12      if (len(A) == size):
13          return False          # stop condition not DAG
14      elif (len(V) == len(A)):
15          return True           # stop condition all in!
16      else:
17          return topoSort(V,E,A) # do another recursive call
```

RESULT :
EBACFDIK

MERRIMACK COLLEGE

Example 2 - Topological Sort

```
V = ["A","B","C","D","E",
     "F","G","H","I","J","K"]
E = [[0,3,1],   #A --> D
     [1,0,1],   #B --> A
     [1,2,1],   #B --> C
     [2,6,1],   #C --> G
     [3,6,1],   #D --> G
     [4,1,1],   #E --> B
     [4,5,1],   #E --> F
     [5,3,1],   #F --> D
     [5,7,1],   #F --> H
     [7,9,1],   #H --> J
     [8,7,1],   #I --> H
     [8,10,1],  #I --> K
     [9,6,1],   #J --> G
     [10,7,1]]  #K --> H
```
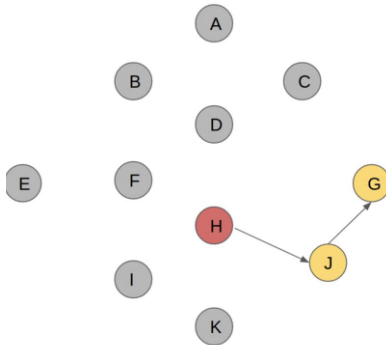
```
A = []
if (topoSort(V,E,A)):
    print("Topological order found:")
    for a in A:
        print(V[a], end=" ")
    print()
else:
    print("The graph is not a DAG!")
```

RESULT :
EBACFDIKH

```
1    def topoSort(V, E, A):
2        size = len(A)
3        for v in range(len(V)):
4            if (v not in A):
5                for e in E:
6                    if (v == e[1]) and (e[0] not in A):
7                        break
8                else:
9                    A.append(v)
10                   break
11               continue
12       if (len(A) == size):
13           return False            # stop condition not DAG
14       elif (len(V) == len(A)):
15           return True             # stop condition all in!
16       else:
17           return topoSort(V,E,A) # do another recursive call
```

MERRIMACK COLLEGE

Example 2 - Topological Sort

```
V = ["A","B","C","D","E",
     "F","G","H","I","J","K"]
E = [[0,3,1],  #A --> D
     [1,0,1],  #B --> A
     [1,2,1],  #B --> C
     [2,6,1],  #C --> G
     [3,6,1],  #D --> G
     [4,1,1],  #E --> B
     [4,5,1],  #E --> F
     [5,3,1],  #F --> D
     [5,7,1],  #F --> H
     [7,9,1],  #H --> J
     [8,7,1],  #I --> H
     [8,10,1], #I --> K
     [9,6,1],  #J --> G
     [10,7,1]] #K --> H
```

RESULT :
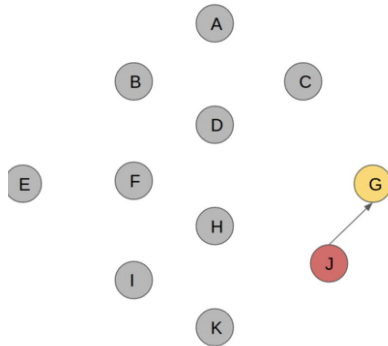EBACFDIKHJ

```
A = []
if (topoSort(V,E,A)):
    print("Topological order found:")
    for a in A:
        print(V[a], end=" ")
    print()
else:
    print("The graph is not a DAG!")
```

```
1   def topoSort(V, E, A):
2       size = len(A)
3       for v in range(len(V)):
4           if (v not in A):
5               for e in E:
6                   if (v == e[1]) and (e[0] not in A):
7                       break
8                   else:
9                       A.append(v)
10                      break
11              continue
12      if (len(A) == size):
13          return False          # stop condition not DAG
14      elif (len(V) == len(A)):
15          return True           # stop condition all in!
16      else:
17          return topoSort(V,E,A) # do another recursive call
```

MERRIMACK COLLEGE

Example 2 - Topological Sort

```
V = ["A","B","C","D","E",
     "F","G","H","I","J","K"]
E = [[0,3,1],  #A --> D
     [1,0,1],  #B --> A
     [1,2,1],  #B --> C
     [2,6,1],  #C --> G
     [3,6,1],  #D --> G
     [4,1,1],  #E --> B
     [4,5,1],  #E --> F
     [5,3,1],  #F --> D
     [5,7,1],  #F --> H
     [7,9,1],  #H --> J
     [8,7,1],  #I --> H
     [8,10,1], #I --> K
     [9,6,1],  #J --> G
     [10,7,1]] #K --> H
```
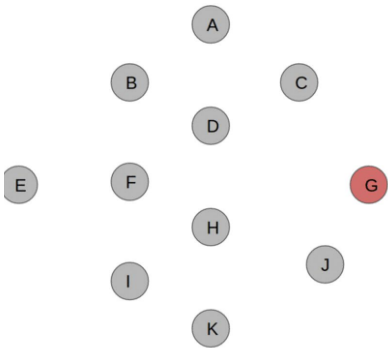
RESULT :
EBACFDIKHJG

```python
A = []
if (topoSort(V,E,A)):
    print("Topological order found:")
    for a in A:
        print(V[a], end=" ")
    print()
else:
    print("The graph is not a DAG!")
```

```python
1   def topoSort(V, E, A):
2       size = len(A)
3       for v in range(len(V)):
4           if (v not in A):
5               for e in E:
6                   if (v == e[1]) and (e[0] not in A):
7                       break
8                   else:
9                       A.append(v)
10                      break
11              continue
12      if (len(A) == size):
13          return False        # stop condition not DAG
14      elif (len(V) == len(A)):
15          return True         # stop condition all in!
16      else:
17          return topoSort(V,E,A) # do another recursive call
```


MERRIMACK COLLEGE

Example 2 - Topological Sort

```
A = []
if (topoSort(V,E,A)):
    print("Topological order found:")
    for a in A:
        print(V[a], end=" ")
    print()
else:
    print("The graph is not a DAG!")
```

```
V = ["A","B","C","D","E",
     "F","G","H","I","J","K"]
E = [[0,3,1],   #A --> D
     [1,0,1],   #B --> A
     [1,2,1],   #B --> C
     [2,6,1],   #C --> G
     [3,6,1],   #D --> G
     [4,1,1],   #E --> B
     [4,5,1],   #E --> F
     [5,3,1],   #F --> D
     [5,7,1],   #F --> H
     [7,9,1],   #H --> J
     [8,7,1],   #I --> H
     [8,10,1],  #I --> K
     [9,6,1],   #J --> G
     [10,7,1]]  #K --> H
```

```
Topological order found:
E B A C F D I K H J G
```

```
1   def topoSort(V, E, A):
2       size = len(A)
3       for v in range(len(V)):
4           if (v not in A):
5               for e in E:
6                   if (v == e[1]) and (e[0] not in A):
7                       break
8                   else:
9                       A.append(v)
10                      break
11              continue
12      if (len(A) == size):
13          return False          # stop condition not DAG
14      elif (len(V) == len(A)):
15 →        return True           # stop condition all in!
16      else:
17          return topoSort(V,E,A) # do another recursive call
```
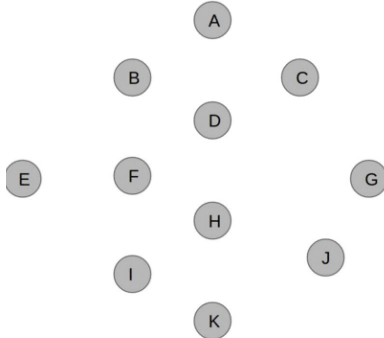
MERRIMACK COLLEGE

Example 2 - Topological Sort

- Each recursive call includes a vertex in **A**:
  - There are **n** vertices;
- In each recursive call all edges are tested (worst case):
  - There are **m** edges;
- The number of tasks (tests and inclusions) can be directed computed as such:
  - **n** times **m**.

```python
def topoSort(V, E, A):
    size = len(A)
    for v in range(len(V)):
        if (v not in A):
            for e in E:
                if (v == e[1]) and (e[0] not in A):
                    break
            else:
                A.append(v)
                break
            continue
    if (len(A) == size):
        return False          # stop condition not DAG
    elif (len(V) == len(A)):
        return True           # stop condition all in!
    else:
        return topoSort(V,E,A) # do another recursive call
```

```
[ ]
[E]
[EB]
[EBA]
[EBAC]
[EBACF]
[EBACFD]
[EBACFDI]
[EBACFDIK]
[EBACFDIKH]
[EBACFDIKHJ]
[EBACFDIKHJG]
```

MERRIMACK COLLEGE

O(nm) - Linear Algorithm

Divide-and-Conquer Algorithms Examples

The examples that will be seen are:
- Decrease by constant amount:
  - Insertion sort
  - Topological sort
- Decrease by constant factor:
  - **Fake coin detection**
  - Russian peasants' multiplication
- Decrease by variable amount and factor:
  - Euclidean GCD
  - Lomuto partition
  - K-th order statistic

Example 3 - Fake coin detection

- Given a pile of **n** coins, where one of them is a fake one that is lighter than the real ones.
- Having only a scale, how fast could you detect the fake one?

- Main idea:
  - Split the pile in two groups of exact same number of coins (one may be left out).
    - If the weight is equal, the faze is the one left out;
    - If one pile is lighter the coin is in there
      - Then repeat the process until you have only two coins to weight.

At each call there are half of the coins to analyze.
It decreases by a constant factor (1/2).

One of these coins is fake

MERRIMACK COLLEGE

A simple decision exercise.

Example 3 - Fake coin detection

- Split the pile in two groups of exact same number of coins (one may be left out).
  - If the weight is equal, the faze is the one left out;
  - If one pile is lighter the fake coin is in there.
    - Then repeat the process until you have only two coins to weight.

- This problem recursive relation and stop conditions are:
  - **T(n) = 1 + T(n/2)** (worst case)
  - **T(3) = T(2) = 1**

- What is the time complexity of this problem considering the weighting the fundamental unit of work?

One of these coins is fake

MERRIMACK COLLEGE

Solution by back-substitution or master method.

Example 3 - Fake coin detection

- This problem recursive relation and stop conditions are:
  - ***T(n) = 1 + T(n/2)*** (worst case)
  - ***T(3) = T(2) = 1***

- What is the time complexity of this problem considering the weighting the fundamental unit of work?
  - By back substitution:
    - The number of weighting is ***log_2 of n***
    - ***T(n) < log_2 n . 1 . T(2)***
  - By the master method:
    - ***T(n) = T(n/2) + 1***
    - ***a = 1, b = 2, f(n) = n^0***

One of these coins is fake
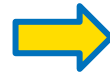
$$\text{if} \quad f(n) < n^{\log_b a} \quad \text{then} \quad T(n) = O(n^{\log_b a})$$
$$\text{if} \quad f(n) = n^{\log_b a} \quad \text{then} \quad T(n) = O(n^{\log_b a} \log n)$$
$$\text{if} \quad f(n) > n^{\log_b a} \quad \text{then} \quad T(n) = O(f(n))$$

O(log n) - Logarithmic Algorithm

Divide-and-Conquer Algorithms Examples

The examples that will be seen are:
- Decrease by constant amount:
  - Insertion sort
  - Topological sort
- Decrease by constant factor:
  - Fake coin detection
  - **Russian peasants' multiplication**
- Decrease by variable amount and factor:
  - Euclidean GCD
  - Lomuto partition
  - K-th order statistic

Example 4 - Russian Peasants Multiplication

- Assume that you want to multiply two numbers **n** and **m**, and you do not know multiplication facts very well. The only thing you master are sums and multiplications and divisions by 2.
- Assuming **p(n,m)** the product of **n** by **m**, the Russian Peasants method is based on these recursive relations:

$$p(n,m) = \begin{cases} p\left(\frac{n}{2}, 2m\right), & \text{If } n \text{ is even} \\ p\left(\frac{n-1}{2}, 2m\right) + m, & \text{If } n \text{ is odd} \\ m, & \text{If } n = 1 \end{cases}$$

At each call there are half of the larger operand to analyze.
It decreases by a constant factor (1/2).

Русские крестьяне умножение
Russkiye krest'yane umnozheniye

MERRIMACK COLLEGE

Video: Russian Peasant Multiplication.

Example 4 - Russian Peasants Multiplication

- Assuming **p(n,m)** the product of **n** by **m**, the Russian Peasants method is based on these recursive relations:

$$p\left(n, m\right) = \begin{cases} p\left(\frac{n}{2}, 2m\right), & \text{If } n \text{ is even} \\ p\left(\frac{n-1}{2}, 2m\right) + m, & \text{If } n \text{ is odd} \\ m, & \text{If } n = 1 \end{cases}$$

What is the time complexity of this problem considering the divisions and sums the fundamental units of work?

For example, the product **p(46,7)** can be computed as:

n	m	steps
46	**7**	46 is even
23	14	23 is odd (+14)
11	28	11 is odd (+28)
5	56	5 is odd (+56)
2	112	2 is even
1	224	224+14+28+56 = **322**

MERRIMACK COLLEGE

Text: How to Multiply Using the Russian Peasant Method.

Example 4 - Russian Peasants Multiplication

What is the time complexity of this problem considering the divisions and sums the fundamental units of work?
Possible recurrent relations:

$$p(n,m) = \begin{cases} p\left(\frac{n}{2}, 2m\right), & \text{If } n \text{ is even} \\ p\left(\frac{n-1}{2}, 2m\right) + m, & \text{If } n \text{ is odd} \\ m, & \text{If } n = 1 \end{cases}$$

$$T(n) = T\left(\frac{n}{2}\right) + 2 \quad \text{or} \quad T(n) = T\left(\frac{n-1}{2}\right) + 3$$

Upper bounding and stopping condition:

$$T(n) < T\left(\frac{n}{2}\right) + 3 \qquad T(1) = 0$$

$$\begin{array}{lll} \text{if} & f(n) < n^{\log_b a} & \text{then} \quad T(n) = O(n^{\log_b a}) \\ \text{if} & f(n) = n^{\log_b a} & \text{then} \quad T(n) = O(n^{\log_b a} \log n) \\ \text{if} & f(n) > n^{\log_b a} & \text{then} \quad T(n) = O(f(n)) \end{array}$$

- By back substitution:
  - The number of steps is **$\log_2$** of **n** and at each step there are 3 tasks (division, double, sum):
    - **$T(n) = \log_2 n \cdot 3 + T(1)$**
- By the master method:
  - **$T(n) = T(n/2) + 3$**
  - **$a = 1, b = 2, f(n) = n^0$**

MERRIMACK COLLEGE

O(log n) - Logarithmic Algorithm

Divide-and-Conquer Algorithms Examples

The examples that will be seen are:
- Decrease by constant amount:
  - Insertion sort
  - Topological sort
- Decrease by constant factor:
  - Fake coin detection
  - Russian peasants' multiplication
- Decrease by variable amount and factor:
  - **Euclidean GCD**
  - Lomuto partition
  - K-th order statistic

MERRIMACK COLLEGE

Example 5 - Euclidean GCD - Greatest Common Divisor

m	n	m%n
60	35	25
35	25	10
25	10	5
10	5	0
5	0	done!

- The GCD of two numbers **n** and **m** is the largest Integer **d** such that **n % d = 0** and **m % d = 0**.
- This algorithm is attributed to *Euclid of Megara*, Greek mathematician, author of the text *Elements*, who lived around **325 bce** (about **2,250** years ago).
- The recursive solution is given by the function GCD below:

60   35

```python
1   def GCD(m, n):
2       if (n == 0):
3           return m
4       else:
5           return GCD(n, m%n)
```

At each call there is a decrease to the remainder of the Integer division.
It decreases by a variable amount and factor.

MERRIMACK COLLEGE
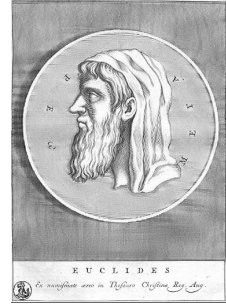
Wikipedia: Euclidean algorithm.

Example 5 - Euclidean GCD - Greatest Common Divisor

The Euclidean algorithm is one of the oldest ones we will see in the whole Program, and yet it is just a brilliant one with no better known solution.

Yet, it is not an historic proven fact that the algorithm was developed by Euclid, as it is likely that Euclid documented it from previous sources (as this happened with other concepts, notably to geometry).

The basic principle of the algorithm is that the GCD of two numbers does not change if the larger one is replaced by its difference with the smaller one.

1599

The GCD between **1599** and **650** is **13**.

1599
650
299
52
39
13

MERRIMACK COLLEGE

Example 5 - Euclidean GCD - Greatest Common Divisor
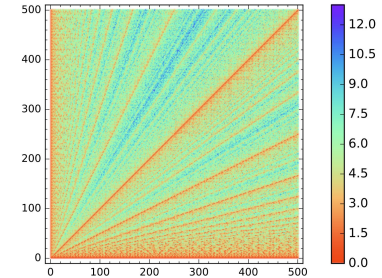
```python
1   def GCD(m, n):
2       if (n == 0):
3           return m
4       else:
5           return GCD(n, m%n)
```

What is the time complexity of this problem considering the remainder of Integer division the fundamental unit of work?

This is tricky one ... as the recurrence relations are:

- **$T(n,m) = 1 + T(m, r)$ where r is $m\%n$, thus $r < n$**
- **$T(m,0) = 0$**

There is not an easy way to estimate how much it drops, but in *1844* the French Mathematicien *Gabriel Lamé* managed to prove that it is **$O(h)$** where **h** is the number of digits in decimal representation of the smaller value between **n** and **m**. Thus, a loose upper bound could be **$O(log_{10} n)$** or **$O(log_{10} m)$**, which one is smaller.

MERRIMACK COLLEGE

$O(log\ n)$ - Logarithmic Algorithm

Divide-and-Conquer Algorithms Examples

The examples that will be seen are:
- Decrease by constant amount:
  - Insertion sort
  - Topological sort
- Decrease by constant factor:
  - Fake coin detection
  - Russian peasants' multiplication
- Decrease by variable amount and factor:
  - Euclidean GCD
  - **Lomuto partition**
  - K-th order statistic
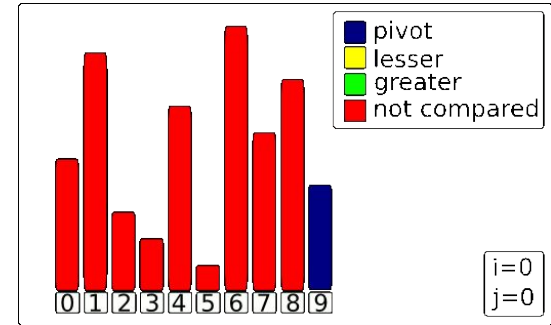
Example 6 - Lomuto Partition

Considering an unsorted array **A** with **n** elements, the Lomuto partition algorithm is supposed to, given a specific element **p** within the array **A**, put all elements smaller than **p** before it, and all elements greater than **p** after it.

The basic idea is to start with two indexes:

- **i** to the limit between the sorted elements smaller and greater than **p**; and
- **j** to the limit between the already sorted elements.

During execution the array would have three regions:

- From **0** to **i-1**: the sorted elements smaller than **p**;
- From **i** to **j-1**: the sorted elements greater than **p**;
- From **j** to **n-2**: the elements yet to sort.

At each iteration one more element is placed in its belonging partition.
It decreases by a constant amount (1).

MERRIMACK COLLEGE
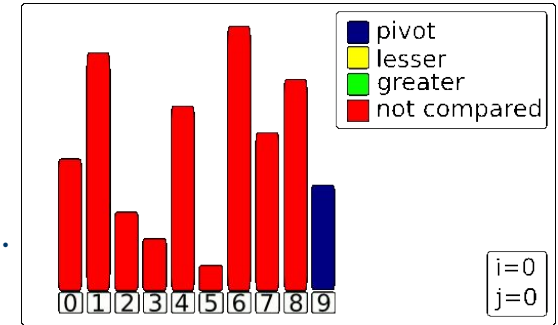
Text: Lomuto Partition Algorithm.

Example 6 - Lomuto Partition

The basic idea is to start with two indexes:
- ***i*** to the limit between the sorted elements smaller and greater than ***p***; and
- ***j*** to the limit between the already sorted elements.

During execution the array would have three regions:
- From ***0*** to ***i-1***: the sorted elements smaller than ***p***;
- From ***i*** to ***j-1***: the sorted elements greater than ***p***;
- From ***j*** to ***n-2***: the elements yet to sort.

Lomuto Partition is not the only algorithm to partition elements of an array, it is just the more popular one.

19	13	6	55	42	31	60	36	48	24
0	1	2	3	4	5	6	7	8	9

     i   ***j***   ***n-1***

MERRIMACK COLLEGE

Text: Nico Lomuto.

Example 6 - Lomuto Partition

During execution the array would have three regions:
- From **0** to **i-1**: the sorted elements smaller than **p**;
- From **i** to **j-1**: the sorted elements greater than **p**;
- From **j** to **n-2**: the elements yet to sort.

A function implementing Lomuto Partition could consider the last element the Pivot, and start deciding where each of the other elements (from the first to the last) will be before (smaller) or after (bigger) the Pivot.

The variable **i** serves as a limit between the smaller and bigger elements. The variable **j** goes to the first to the before last element.

```python
def lomuto(A, left, right):
    p = A[right]
    i = left
    for j in range(left, right):
        if A[j] < p:
            A[i], A[j] = A[j], A[i]
            i += 1
    A[i], A[right] = A[right], A[i]
    return i

A = [31, 55, 19, 13, 42, 6, 60, 36, 48, 24]
pvt = lomuto(A, 0, len(A)-1)
print("Lomuto with pivot at", pvt, ":", A[pvt])
print(A[:pvt])
print(A[pvt+1:])
```
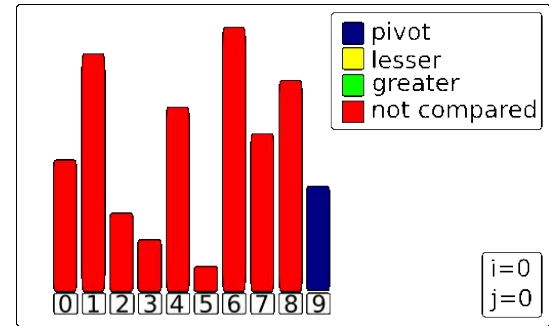
MERRIMACK COLLEGE

Video: Learn Lomuto Partition.

Example 6 - Lomuto Partition

During execution the array would have three regions:
- From **0** to **i-1**: the sorted elements smaller than **p**;
- From **i** to **j-1**: the sorted elements greater than **p**;
- From **j** to **n-2**: the elements yet to sort.

31	55	19	13	42	6	60	36	48	24
0	1	2	3	4	5	6	7	8	9

i j   *n-1*

i = 0   j = 0

```python
1   def lomuto(A, left, right):
2       p = A[right]
3       i = left
4       for j in range(left, right):
5           if A[j] < p:
6               A[i], A[j] = A[j], A[i]
7               i += 1
8       A[i], A[right] = A[right], A[i]
9       return i
10
11  A = [31, 55, 19, 13, 42, 6, 60, 36, 48, 24]
12  pvt = lomuto(A, 0, len(A)-1)
13  print("Lomuto with pivot at", pvt, ":", A[pvt])
14  print(A[:pvt])
15  print(A[pvt+1:])
```

MERRIMACK COLLEGE

Step 1 of 11.

Example 6 - Lomuto Partition

During execution the array would have three regions:
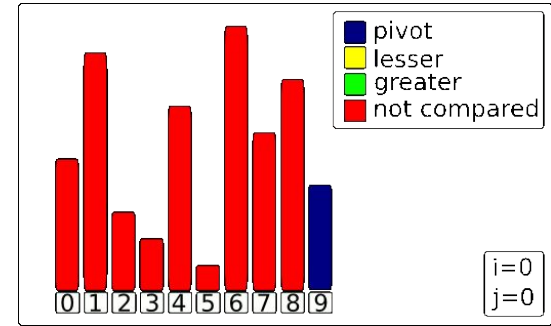- From **0** to **i-1**: the sorted elements smaller than **p**;
- From **i** to **j-1**: the sorted elements greater than **p**;
- From **j** to **n-2**: the elements yet to sort.

31	55	19	13	42	6	60	36	48	24
0	1	2	3	4	5	6	7	8	9

i   *j*   *n-1*

i = 0 j = 0 -> i = 0 j = 1

```python
1    def lomuto(A, left, right):
2        p = A[right]
3        i = left
4        for j in range(left, right):
5            if A[j] < p:
6                A[i], A[j] = A[j], A[i]
7                i += 1
8        A[i], A[right] = A[right], A[i]
9        return i
10
11   A = [31, 55, 19, 13, 42, 6, 60, 36, 48, 24]
12   pvt = lomuto(A, 0, len(A)-1)
13   print("Lomuto with pivot at", pvt, ":", A[pvt])
14   print(A[:pvt])
15   print(A[pvt+1:])
```
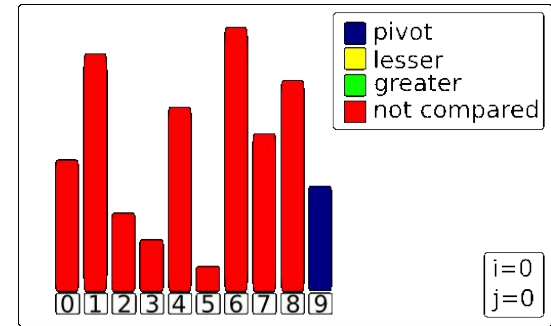
MERRIMACK COLLEGE

Step 2 of 11.

Example 6 - Lomuto Partition

During execution the array would have three regions:
- From **0** to **i-1**: the sorted elements smaller than **p**;
- From **i** to **j-1**: the sorted elements greater than **p**;
- From **j** to **n-2**: the elements yet to sort.

31	55	19	13	42	6	60	36	48	24
0	1	2	3	4	5	6	7	8	9

i   *j*   *n-1*

i = 0 j = 1   ->   i = 0 j = 2

```python
1   def lomuto(A, left, right):
2       p = A[right]
3       i = left
4       for j in range(left, right):
5           if A[j] < p:
6               A[i], A[j] = A[j], A[i]
7               i += 1
8       A[i], A[right] = A[right], A[i]
9       return i
10
11  A = [31, 55, 19, 13, 42, 6, 60, 36, 48, 24]
12  pvt = lomuto(A, 0, len(A)-1)
13  print("Lomuto with pivot at", pvt, ":", A[pvt])
14  print(A[:pvt])
15  print(A[pvt+1:])
```
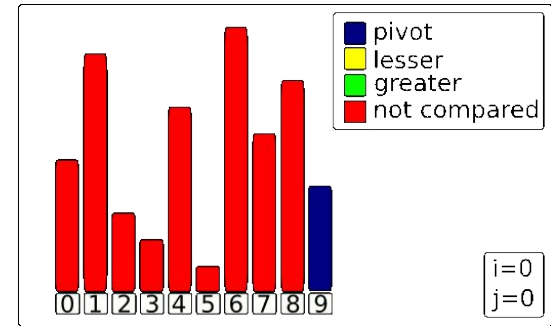
Example 6 - Lomuto Partition

During execution the array would have three regions:
- From **0** to **i-1**: the sorted elements smaller than **p**;
- From **i** to **j-1**: the sorted elements greater than **p**;
- From **j** to **n-2**: the elements yet to sort.

19	55	31	13	42	6	60	36	48	24
0	1	2	3	4	5	6	7	8	9

i   **j**   **n-1**

swap **A[i]** and **A[j]**

    i = 0 j = 2   ->   i = 1 j = 3

```python
1   def lomuto(A, left, right):
2       p = A[right]
3       i = left
4       for j in range(left, right):
5           if A[j] < p:
6               A[i], A[j] = A[j], A[i]
7               i += 1
8       A[i], A[right] = A[right], A[i]
9       return i
10
11  A = [31, 55, 19, 13, 42, 6, 60, 36, 48, 24]
12  pvt = lomuto(A, 0, len(A)-1)
13  print("Lomuto with pivot at", pvt, ":", A[pvt])
14  print(A[:pvt])
15  print(A[pvt+1:])
```
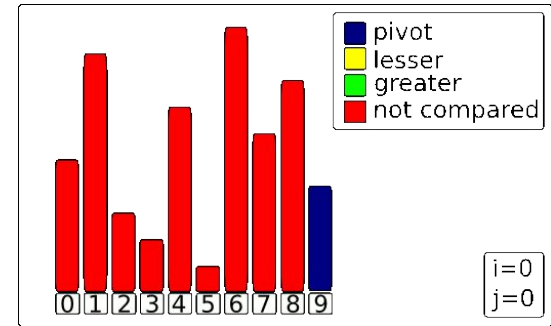
MERRIMACK COLLEGE

Step 4 of 11.

Example 6 - Lomuto Partition

During execution the array would have three regions:
- From **0** to **i-1**: the sorted elements smaller than **p**;
- From **i** to **j-1**: the sorted elements greater than **p**;
- From **j** to **n-2**: the elements yet to sort.

19	13	31	55	42	6	60	36	48	24
0	1	2	3	4	5	6	7	8	9
		i			*j*				*n-1*

swap **A[i]** and **A[j]**

$i = 1$ $j = 3$ -> $i = 2$ $j = 4$

```python
1   def lomuto(A, left, right):
2       p = A[right]
3       i = left
4       for j in range(left, right):
5           if A[j] < p:
6               A[i], A[j] = A[j], A[i]
7               i += 1
8       A[i], A[right] = A[right], A[i]
9       return i
10
11  A = [31, 55, 19, 13, 42, 6, 60, 36, 48, 24]
12  pvt = lomuto(A, 0, len(A)-1)
13  print("Lomuto with pivot at", pvt, ":", A[pvt])
14  print(A[:pvt])
15  print(A[pvt+1:])
```
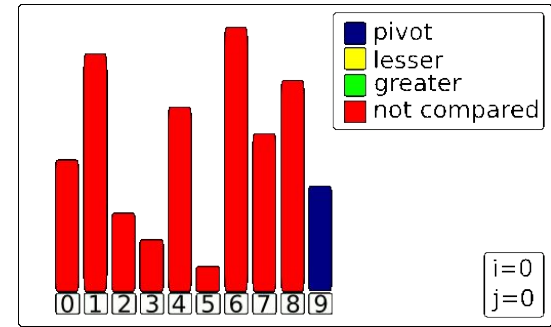
MERRIMACK COLLEGE

Step 5 of 11.

Example 6 - Lomuto Partition

During execution the array would have three regions:
- From **0** to **i-1**: the sorted elements smaller than **p**;
- From **i** to **j-1**: the sorted elements greater than **p**;
- From **j** to **n-2**: the elements yet to sort.

19	13	31	55	42	6	60	36	48	24
0	1	2	3	4	5	6	7	8	9

i j **n-1**

i = 2 j = 4 -> i = 2 j = 5

```python
1   def lomuto(A, left, right):
2       p = A[right]
3       i = left
4       for j in range(left, right):
5           if A[j] < p:
6               A[i], A[j] = A[j], A[i]
7               i += 1
8       A[i], A[right] = A[right], A[i]
9       return i
10
11  A = [31, 55, 19, 13, 42, 6, 60, 36, 48, 24]
12  pvt = lomuto(A, 0, len(A)-1)
13  print("Lomuto with pivot at", pvt, ":", A[pvt])
14  print(A[:pvt])
15  print(A[pvt+1:])
```
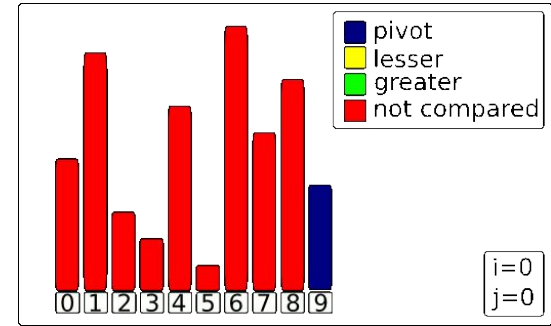
MERRIMACK COLLEGE

Step 6 of 11.

Example 6 - Lomuto Partition

During execution the array would have three regions:
- From **0** to **i-1**: the sorted elements smaller than **p**;
- From **i** to **j-1**: the sorted elements greater than **p**;
- From **j** to **n-2**: the elements yet to sort.

19	13	6	55	42	31	60	36	48	24
0	1	2	3	4	5	6	7	8	9

          i   ***j***   ***n-1***

swap **A[i]** and **A[j]**

  i = 2 j = 5   ->   i = 3 j = 6

```python
1   def lomuto(A, left, right):
2       p = A[right]
3       i = left
4       for j in range(left, right):
5           if A[j] < p:
6               A[i], A[j] = A[j], A[i]
7               i += 1
8       A[i], A[right] = A[right], A[i]
9       return i
10
11  A = [31, 55, 19, 13, 42, 6, 60, 36, 48, 24]
12  pvt = lomuto(A, 0, len(A)-1)
13  print("Lomuto with pivot at", pvt, ":", A[pvt])
14  print(A[:pvt])
15  print(A[pvt+1:])
```
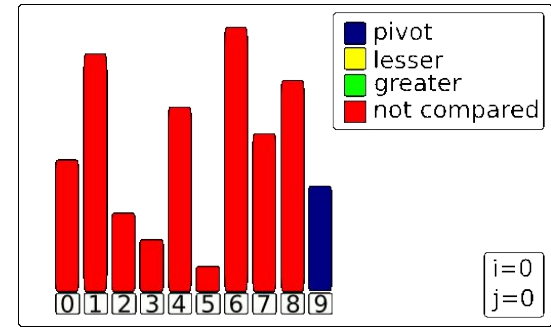
MERRIMACK COLLEGE

Step 7 of 11.

Example 6 - Lomuto Partition

During execution the array would have three regions:
- From **0** to **i-1**: the sorted elements smaller than **p**;
- From **i** to **j-1**: the sorted elements greater than **p**;
- From **j** to **n-2**: the elements yet to sort.

19	13	6	55	42	31	60	36	48	24
0	1	2	3	4	5	6	7	8	9

i j $n-1$

$i = 3$ $j = 6$ -> $i = 3$ $j = 7$

```python
1    def lomuto(A, left, right):
2        p = A[right]
3        i = left
4        for j in range(left, right):
5            if A[j] < p:
6                A[i], A[j] = A[j], A[i]
7                i += 1
8        A[i], A[right] = A[right], A[i]
9        return i
10
11   A = [31, 55, 19, 13, 42, 6, 60, 36, 48, 24]
12   pvt = lomuto(A, 0, len(A)-1)
13   print("Lomuto with pivot at", pvt, ":", A[pvt])
14   print(A[:pvt])
15   print(A[pvt+1:])
```
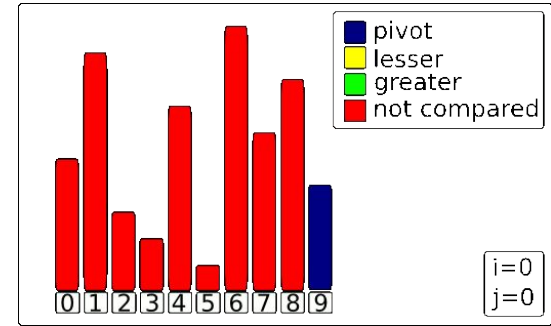
MERRIMACK COLLEGE

Step 8 of 11.

Example 6 - Lomuto Partition

During execution the array would have three regions:
- From **0** to **i-1**: the sorted elements smaller than **p**;
- From **i** to **j-1**: the sorted elements greater than **p**;
- From **j** to **n-2**: the elements yet to sort.

19	13	6	55	42	31	60	36	48	24
0	1	2	3	4	5	6	7	8	9

i   **j**   **n-1**

i = 3 j = 7   ->   i = 3 j = 8

```python
1   def lomuto(A, left, right):
2       p = A[right]
3       i = left
4       for j in range(left, right):
5           if A[j] < p:
6               A[i], A[j] = A[j], A[i]
7               i += 1
8       A[i], A[right] = A[right], A[i]
9       return i
10
11  A = [31, 55, 19, 13, 42, 6, 60, 36, 48, 24]
12  pvt = lomuto(A, 0, len(A)-1)
13  print("Lomuto with pivot at", pvt, ":", A[pvt])
14  print(A[:pvt])
15  print(A[pvt+1:])
```
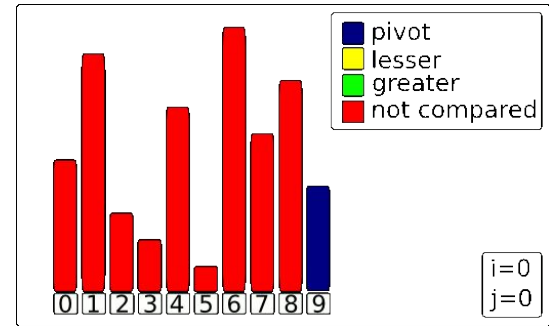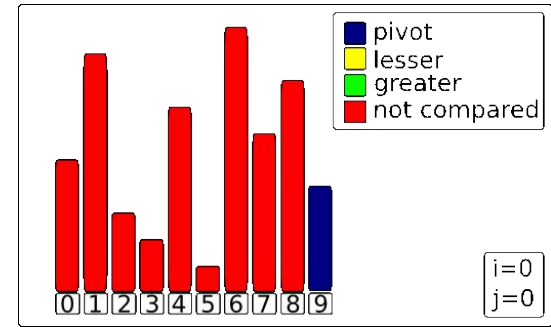
Step 9 of 11.

Example 6 - Lomuto Partition

During execution the array would have three regions:
- From **0** to **i-1**: the sorted elements smaller than **p**;
- From **i** to **j-1**: the sorted elements greater than **p**;
- From **j** to **n-2**: the elements yet to sort.

19	13	6	55	42	31	60	36	48	24
0	1	2	3	4	5	6	7	8	9

i   *n-1*
                                                   j

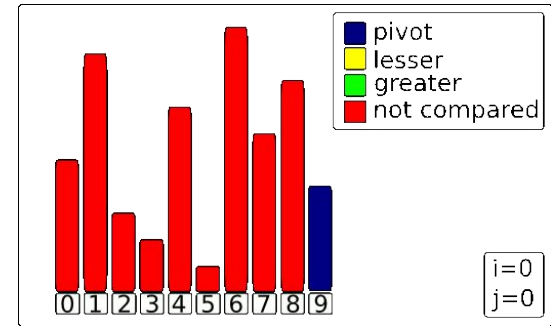**i = 3 j = 8 -> i = 3 j = 9**

```python
def lomuto(A, left, right):
    p = A[right]
    i = left
    for j in range(left, right):
        if A[j] < p:
            A[i], A[j] = A[j], A[i]
            i += 1
    A[i], A[right] = A[right], A[i]
    return i

A = [31, 55, 19, 13, 42, 6, 60, 36, 48, 24]
pvt = lomuto(A, 0, len(A)-1)
print("Lomuto with pivot at", pvt, ":", A[pvt])
print(A[:pvt])
print(A[pvt+1:])
```

Step 10 of 11.

Example 6 - Lomuto Partition

During execution the array would have three regions:
- From **0** to **i-1**: the sorted elements smaller than **p**;
- From **i** to **j-1**: the sorted elements greater than **p**;
- From **j** to **n-2**: the elements yet to sort.

19	13	6	24	42	31	60	36	48	55
0	1	2	3	4	5	6	7	8	9

i   **j n-1**

swap **A[i]** and the pivot

i = 3

```
Lomuto with pivot at 3 : 24
[19, 13, 6]
[42, 31, 60, 36, 48, 55]
```

```python
1   def lomuto(A, left, right):
2       p = A[right]
3       i = left
4       for j in range(left, right):
5           if A[j] < p:
6               A[i], A[j] = A[j], A[i]
7               i += 1
8       A[i], A[right] = A[right], A[i]
9       return i
10
11  A = [31, 55, 19, 13, 42, 6, 60, 36, 48, 24]
12  pvt = lomuto(A, 0, len(A)-1)
13  print("Lomuto with pivot at", pvt, ":", A[pvt])
14  print(A[:pvt])
15  print(A[pvt+1:])
```
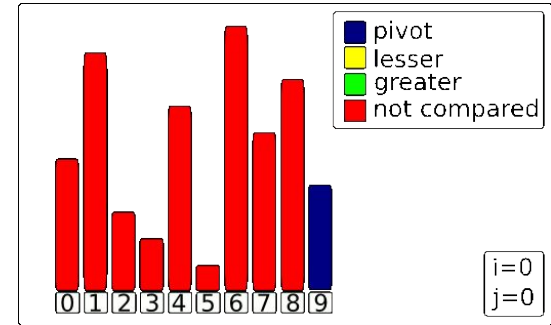
MERRIMACK COLLEGE

Step 11 of 11.

Example 6 - Lomuto Partition

What is the time complexity of Lomuto partition?

- Lines 4, 5, 6, and 7 happen **n-1** times (worst case)

Upper bounded by **n**, which is actually efficient considering what is done.


```python
1   def lomuto(A, left, right):
2       p = A[right]
3       i = left
4       for j in range(left, right):
5           if A[j] < p:
6               A[i], A[j] = A[j], A[i]
7               i += 1
8       A[i], A[right] = A[right], A[i]
9       return i
```

At each iteration one more element is placed in its belonging partition.
It decreases by a constant amount (1).

MERRIMACK COLLEGE

O(n) - Linear Algorithm

Divide-and-Conquer Algorithms Examples

The examples that will be seen are:
- Decrease by constant amount:
  - Insertion sort
  - Topological sort
- Decrease by constant factor:
  - Fake coin detection
  - Russian peasants' multiplication
- Decrease by variable amount and factor:
  - Euclidean GCD
  - Lomuto partition
  - **K-th order statistic**
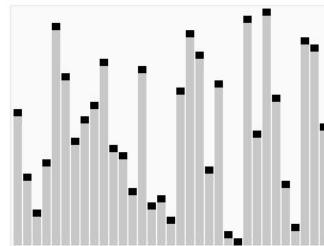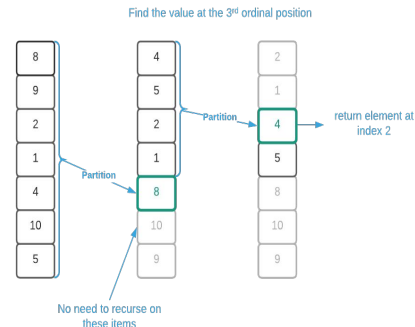
Example 7 - K-th order statistic

Given an unsorted array, how difficult is to find the smallest element?

- We seen that before, it is an **O(n)** problem.

What about about finding the **k**-th smallest element in an array?

- One brute force way is to sort the array and pick the **k**-th element, but it costs at least **O(n log n)**;

Given the large number of applications for this kind of search, an algorithm called QuickSelect finds the **k**-th smallest element of an array **A** of size **n** with possibly a smaller time complexity than the sort algorithms.

Find the value at the 3rd ordinal position

MERRIMACK COLLEGE

Example 7 - K-th order statistic

QuickSelect finds the **k**-th smallest element of an array **A** of size **n** with possibly a smaller time complexity than the sort algorithms.

The basic idea is to use recursively a partition algorithm (Lomuto for instance), until you pinpoint the **k**-th smallest element in the array.

This approach does not guarantee a better efficiency than sorting, but in general it is much faster.

Actually, the average complexity of QuickSelect is **O(n)** which is clearly better than the **O(n log n)** of the more efficient sorting algorithms, but the worst case can be as bad as **O(n²)**.
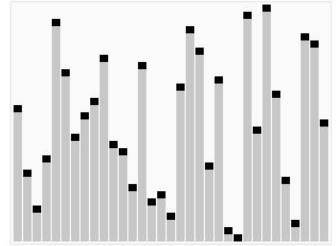
At each call there is a decrease to the number of elements in the partition to search. It decreases by a variable amount and factor.

MERRIMACK COLLEGE

This algorithm decreases the size of the problem to one partition, which can be advantageous or not.

Example 7 - K-th order statistic
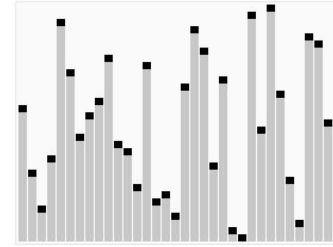
Let's say you have a array **A** of size **n** and you are looking for the **third** smaller element in this array.

You can call a Lomuto partition to it and it will give you (according to the pivot chosen) the **i**-th element of the array. For the example shown before it gives you the **fourth** smallest element.

31	55	19	13	42	6	60	36	48	24
0	1	2	3	4	5	6	7	8	9

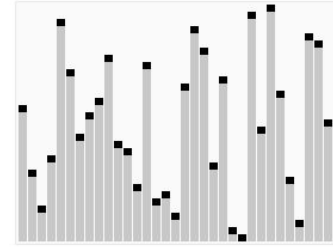| 19	13	6	24	42	31	60	36	48	55
0	1	2	3	4	5	6	7	8	9

If you are looking for the **third** one, you can call Lomuto again over the smaller than the pivot partition searching for the **third** element.

Searching the smaller than pivot partition is directed.

Example 7 - K-th order statistic

Let's say you have a array **A** of size **n** and you are looking for the **sixth** smaller element in this array.

You can call a Lomuto partition to it and it will give you (according to the pivot chosen) the **i**-th element of the array. For the example shown before it gives you the **fourth** smallest element.

31	55	19	13	42	6	60	36	48	24
0	1	2	3	4	5	6	7	8	9

19	13	6	24	42	31	60	36	48	55
0	1	2	3	4	5	6	7	8	9

If you are looking for the **sixth** one, you can call Lomuto again over the greater than the pivot partition searching for the **second** element (6-4).
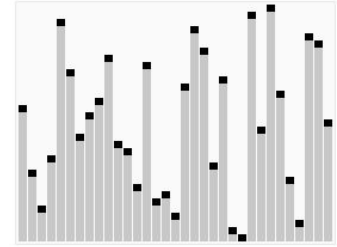
Searching the greater than pivot partition requires to adjust considering the position of the pivot.

MERRIMACK COLLEGE

Example 7 - K-th order statistic

Watch the video linked below that demonstrate step-by-step how to implement it.

Watch attently this video, including the last part where a Python implementation is presented in detail.

However, as in real life, things are not exactly as we would like to be. In this video, intently, I choose to present you an analogous problem: how to find the **k**-th largest element. So, you will need to adapt what you are seeing to what I described here. In fact, you will need to perform this adaptation (check the coding assignment of this week).

Find the value at the 3rd ordinal position

MERRIMACK COLLEGE

Video: <u>K-th Largest Element in an Array</u>.
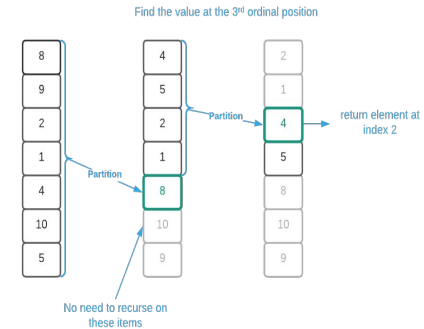
This Week's tasks

- In-class Exercise E#6
- Coding Project P#6
- Quiz Q#6

Tasks

- Fill the worksheet as required.

- Develop Python programs to find the k-th smallest element in an unsorted array.
- Quiz #6 about this week topics.

In-class Exercise - E#6

Download the pdf ([link here also](#)) and perform the following:

- (1) Trace the Russian Peasants Multiplication algorithm for the following products as shown in the live session.
  - 64 * 13   -   60 * 13   -   59 * 13
- (2) Trace the Lomuto partition with the array:
  - A = [100, 33, 22, 213, 65, 29, 153, 199, 47, 181, 85]

  In your trace, write down to each change in either *i* or *j*, stating: the values of *i* and *j*, swaps made, and elements divided into lesser than the pivot, greater than the pivot, and yet to compare.

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

CSC6013 - Worksheet for Week 6

Russian Peasants Multiplication

1. Trace the Russian Peasants Multiplication algorithm for the following products. Show each recursive call and the final result, as shown in the live session (table).

   a. 64 * 13
   b. 60 * 13
   c. 59 * 13

Lomuto partition

2. Trace the Lomuto partition with the array:

   a. A = [100, 33, 22, 213, 65, 29, 153, 199, 47, 181, 85]

   Using A[10] = 85 as pivot the final array will be:
   - A = [33, 22, 65, 29, 47, 85, 153, 199, 100, 181, 213]

   In your trace, write down to each change in either *i* or *j*, stating: the values of *i* and *j*, swaps made, and elements divided into lesser than the pivot, greater than the pivot, and yet to compare.

Sixth Coding Project - P#6

Develop one Python program to perform the Quick Select algorithm and for an array of **n** elements it should find the **k**-th smallest element of the array). Inspire yourself by the video example [K-th Largest Element in an Array](#), that needs to be adapted by yourself.

- You must code a function QuickSelect that receives an array and the element the user wants to find (**k**-th smallest);
- Then the main function of your program should generate a random array of 1000 elements to be searched, ask the user the value of **k**, call QuickSelect, and display the searched element found.

You have to submit the code (**.py** file) of your algorithm.

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

Sixth Quiz - Q#6

- The sixth quiz in this course covers the topics of Week 6;
- The quiz will be available this Friday, and it is composed by 10 questions;
- The quiz should be taken on Canvas (Module 6), 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.

MERRIMACK COLLEGE

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

- **Do In-class Exercise E#6 until Friday;**
- **Do Quiz Q#6 (available Friday) until next Monday;**
- **Do Coding Project P#6 until next Monday.**

Next Week - Divide-and-conquer algorithms

MERRIMACK COLLEGE