MERRIMACK COLLEGE

CSC 6013 Week 7

Divide-and-Conquer Algorithms

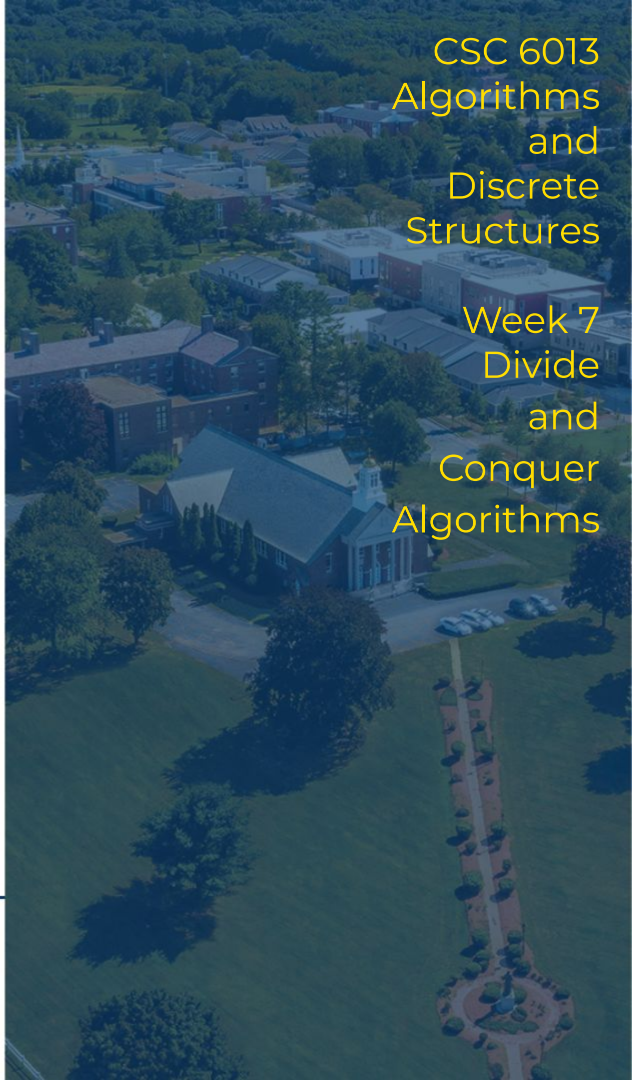
Algorithms and Discrete Structures - Dr. Paulo Fernandes

Presentation Agenda

Week 7

- Divide-and-Conquer
  a. Meet the family
  b. Basic Principles
  c. Recursion and Iteration
- Examples
  a. Array sum
  b. Mergesort
  c. Quicksort
  d. Height of Binary Tree
- This Week's tasks

MERRIMACK COLLEGE



Divide-and-Conquer Algorithms

Solve the problem by solving each of its parts.

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

Some of the more effective algorithms are divide-and-conquer.

- **Meet the family**
  - Decrease-and-Conquer
  - Divide-and-Conquer
  - Transform-and-Conquer
- Divide-and-Conquer
  - Basic Principles
  - Recursion and Iteration

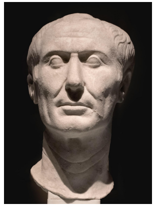
Meet the ...-and-conquer family

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:

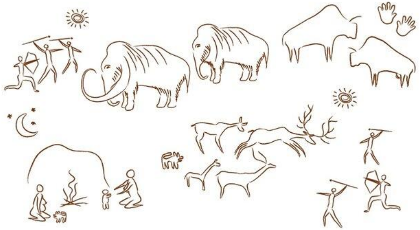
- **Divide-and-conquer**
  - When the original problem is broken into complementary parts;
- **Decrease-and-conquer**
  - When at each time the problem becomes smaller, but not into complementary problems;
- **Transform-and-conquer**
  - When the problem not always becomes smaller.

This division is acknowledged by great authors (as our textbook ones: Cormen, Leiserson, Rivest, and Stein), but it is not unanimous.

The …-and-conquer family

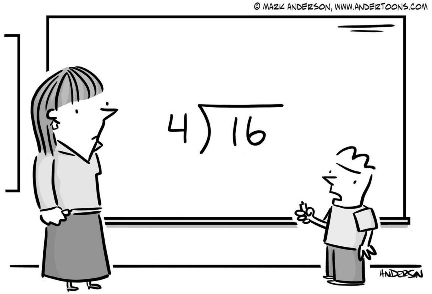


Probably, one of the most known algorithmic techniques is the ***divide-and-conquer***. It has been talked about consistently (documented with this name) at least since the times of Julius Caesar, i.e., 2000 years ago!



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

Divide-and-conquer is the big brother of the …-and-conquer family. Most of the more effective algorithms are divide-and-conquer ones.



"We've been dividing all week! When do we get to conquer?!"

Wikipedia: Divide-and-Conquer.

Divide-and-Conquer Algorithms

Solve the problem by solving each of its parts.

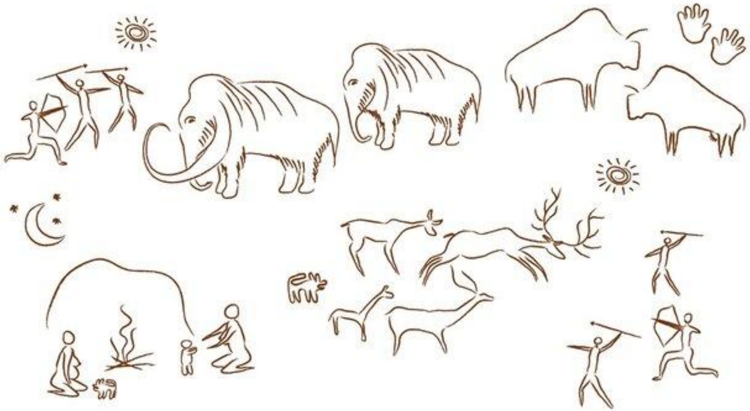
Some of the better known algorithms belong to the …-and-conquer family.

Some of the more effective algorithms are divide-and-conquer.

- Meet the family
  - Decrease-and-Conquer
  - Divide-and-Conquer
  - Transform-and-Conquer
- **Divide-and-Conquer**
  - Basic Principles
  - Recursion and Iteration

Divide-and-Conquer

Tackling a problem by dividing it into subproblems is so intuitive that sometimes we wonder if this technique can be said to be invented. Splitting our tasks into complementary subtasks has been done perhaps even before we learned how to speak.



If you have to carry a large lego statue, it does make sense to break it into piece to be able to carry it.

If you have a pile of boxes to move upstairs, it makes sense to not carry them all together.

Julius Caesar was very smart, but not the creator of Divide-and-Conquer.

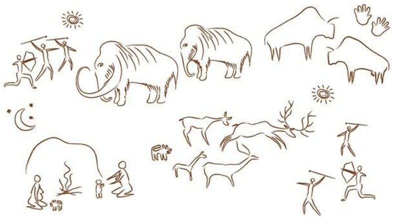
Divide-and-Conquer

While decrease-and-conquer can be implemented **recursively or iteratively**, the very large majority of divide-and-conquer algorithms are implemented **recursively** because it is very natural keep on breaking a problem into parts until it becomes trivial.

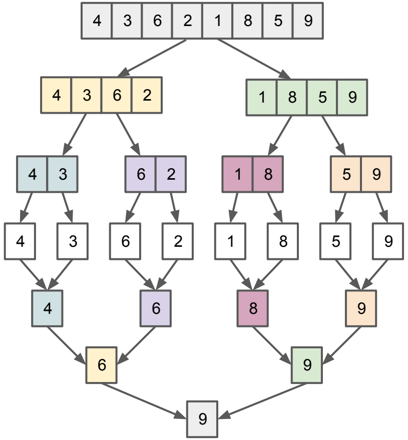
```python
def Max(A, start, end):
    if (start == end):
        return end
    else:
        mid = (end+start)//2
        fst = Max(A, start, mid)
        lst = Max(A, mid+1, end)
        return fst if A[fst] > A[lst] else lst

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)
```

Also it is much natural to break into halves, as we will see, it makes the algorithms simpler and more efficient.



With some effort we can always manage, often with large effort and no gain, to turn a divide-and-conquer implementation into a non-recursive (iterative) version.



MERRIMACK COLLEGE

Text: [Introduction to Divide and Conquer Algorithm](#).

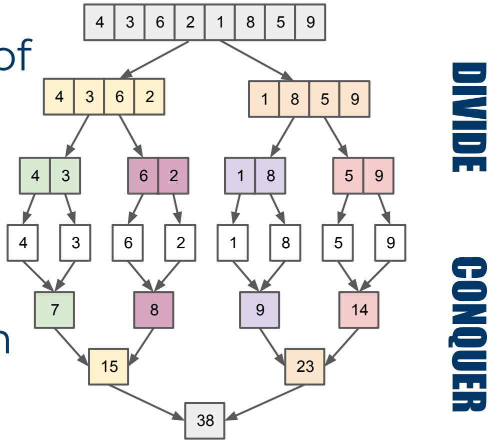
Divide-and-Conquer Algorithms Examples

The examples that will be seen are:

- **Array sum**
- Mergesort
- Quicksort
- Height of Binary Tree

Example 1 - Array sum

- How to sum all elements of and array ***A*** composed of ***n*** elements?
- You can go iteratively adding up all of them, sure, and this is not bad at all, but there is divide and conquer way to do it as well.



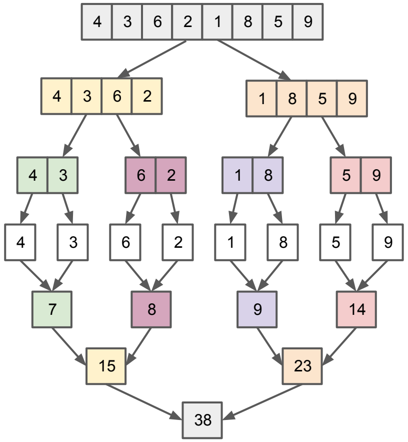
- What if we say we split the array in two and the sum is the sum of the two parts.
    - … and we keep on doing that until we have a single element where the obvious sum is the element itself.

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


Video: [Summation of a list of numbers using Divide and Conquer in Python]().

Example 1 - Array sum

The sum all elements of and array ***A*** composed of ***n*** elements is given by recursively add the two halves of the array, until we have a single element where the obvious sum is the element itself.




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

from random import shuffle
A = list(range(1000))
shuffle(A)
print("The sum is:", arraySum(A, 0, 999))
print("... and using Gauss it should be:", (999 * 1000)//2)
```

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

Master method:

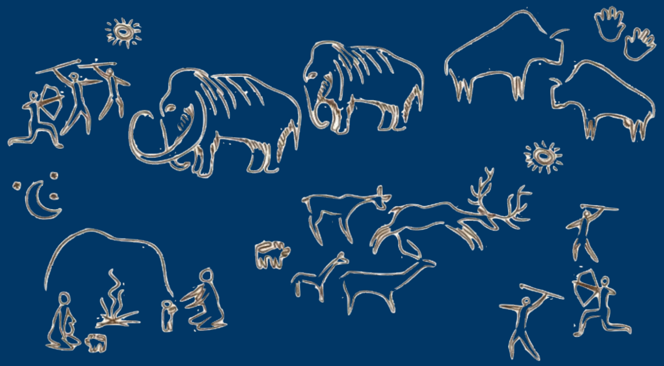
- ***a=2***, ***b=2***, ***f(n)=n⁰***, ***log₂2=1***

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

MERRIMACK COLLEGE

O(n) - Linear Algorithm

Divide-and-Conquer Algorithms Examples



The examples that will be seen are:

- Array sum
- **Height of Binary Tree**
- Mergesort
- Quicksort

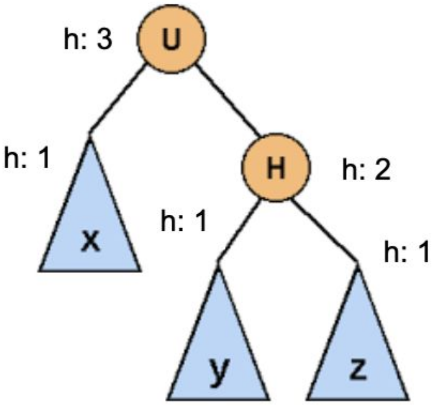
Example 2 - Height of Binary Tree

- Giving a tree with ***n*** nodes
- What is the height of the tree (height of the root)?

```
class Node:
    def __init__(self, d):
        self.data = d
        self.Left = None
        self.Right = None
```

The algorithm simply consider the height of a node as plus one of the tallest child subtree.

The height of a tree is equal to the maximum depth of this tree, since both are the distance between the root and the deepest leaf.




```
def heightTree(node):
    if (node == None):
        return -1
    else:
        left = heightTree(node.Left)
        right = heightTree(node.Right)
        if (left < right):
            return right + 1
        else:
            return left + 1
```


Video: Maximum Depth of a Binary Tree.

Example 2 - Height of Binary Tree

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

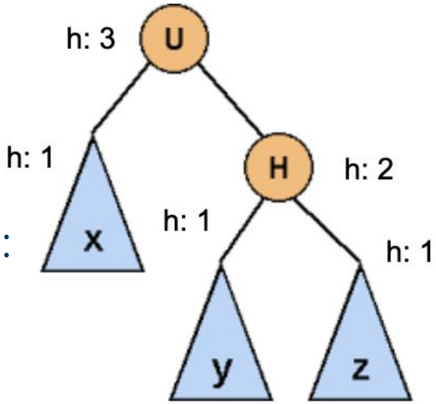
Worst case:

- ***T(n) = T(n-1) + 1***
- Back-substitution:
  - ***O(n)***

Best case:

- ***T(n) = 2T(n/2) + 1***
- Master method:
  - ***a=2, b=2, f(n)=1, $log_2 2=1$***

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



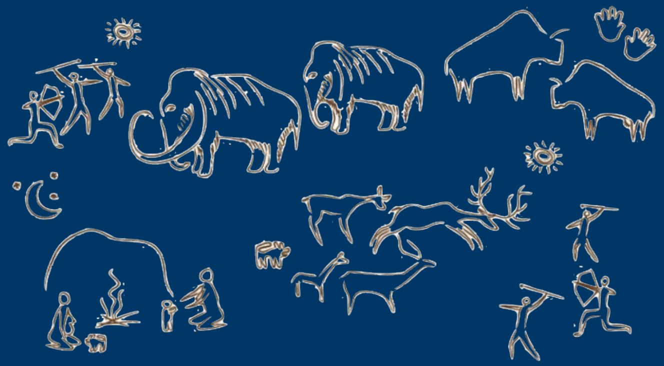

```python
class Node:
    def __init__(self, d):
        self.data = d
        self.Left = None
        self.Right = None
```

```python
def heightTree(node):
    if (node == None):
        return -1
    else:
        left = heightTree(node.Left)
        right = heightTree(node.Right)
        if (left < right):
            return right + 1
        else:
            return left + 1
```

MERRIMACK COLLEGE

O(n) - Linear Algorithm

Divide-and-Conquer Algorithms Examples



The examples that will be seen are:

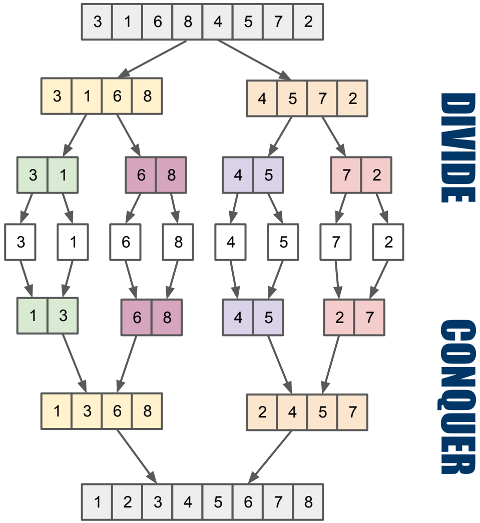
- Array sum
- Height of Binary Tree
- **Mergesort**
- Quicksort

Example 3 - Mergesort

- Giving an array $\boldsymbol{A = [\, a_1, a_2, \ldots\, a_n \,]}$
- Deliver a permutation of $\boldsymbol{A}$ where $\boldsymbol{a_i <= a_j}$ if $\boldsymbol{i < j}$

The Mergesort performs the sort:

- splitting the array in two halves and keep splitting until you end up with a single element, thus sorted;
  - this is the ***divide*** part;
- merging the sorted halves into a sorted array repeatedly, until the whole array is sorted;
  - this is the ***conquer*** part.



MergeSort:

3 1 6 8 4 5 7 2

method call

return value

merge

Mergesort is very time efficient.

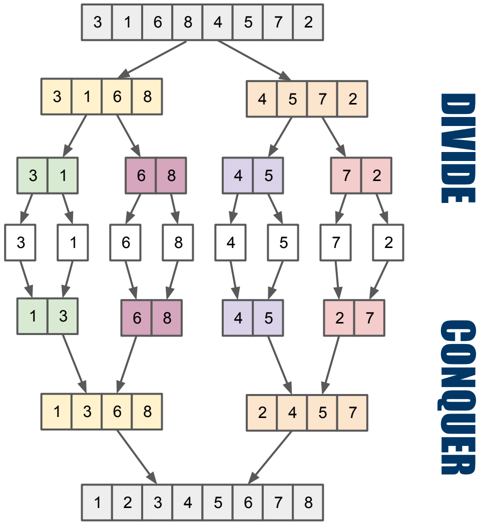
MERRIMACK COLLEGE

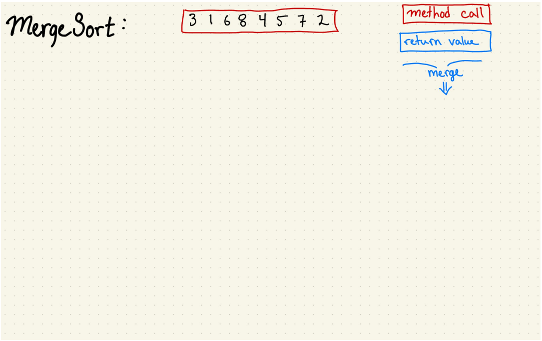
Example 3 - Mergesort

The Mergesort performs the sort:

- splitting the array in two halves and keep splitting until you end up with a single element, thus sorted;
    - this is the ***divide*** part.
- if it is not a single element:
    - split in two;
    - recursive call on left;
    - recursive call on right;
    - call the merge (conquer) part.

```python
def mergesort(A):
    if len(A) <= 1:
        return A
    else:
        mid = len(A) // 2
        left = mergesort(A[:mid])
        right = mergesort(A[mid:])
        return merge(left, right)
```





MERRIMACK COLLEGE

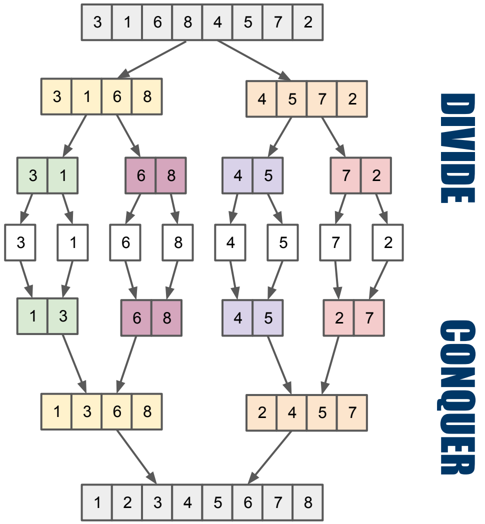
Video: Merge Sort In Python Explained.

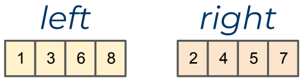
Example 3 - Mergesort

The Mergesort performs the sort:

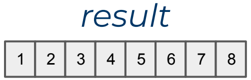
- merging the sorted halves into a sorted array repeatedly, until the whole array is sorted;
    - this is the ***conquer*** part.

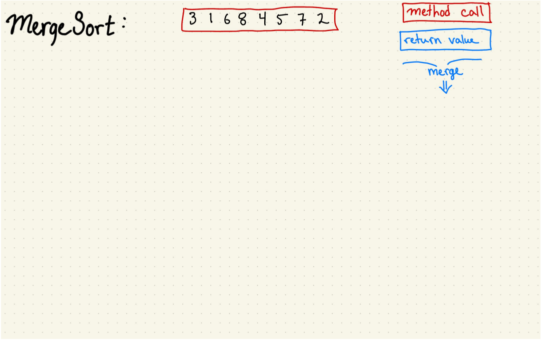
```python
def merge(left, right):
    result, i, j = [], 0 , 0
    while i < len(left) and j < len(right):
        if (left[i] <= right[j]):
            result.append(left[i])
            i += 1
        else:
            result.append(right[j])
            j += 1
    result += left[i:]
    result += right[j:]
    return result
```





fetch the smallest between the left and right heads.





Note that the input are two arrays, and a third one is returned (requires more memory).

Example 3 - Mergesort

The Mergesort performs the sort:

- merging the sorted halves into a sorted array repeatedly, until the whole array is sorted;
    - this is the ***conquer*** part.

```python
def merge(left, right):
    result, i, j = [], 0 , 0
    while i < len(left) and j < len(right):
        if (left[i] <= right[j]):
            result.append(left[i])
            i += 1
        else:
            result.append(right[j])
            j += 1
    result += left[i:]
    result += right[j:]
    return result
```

left

1	3	6	8

i = 0

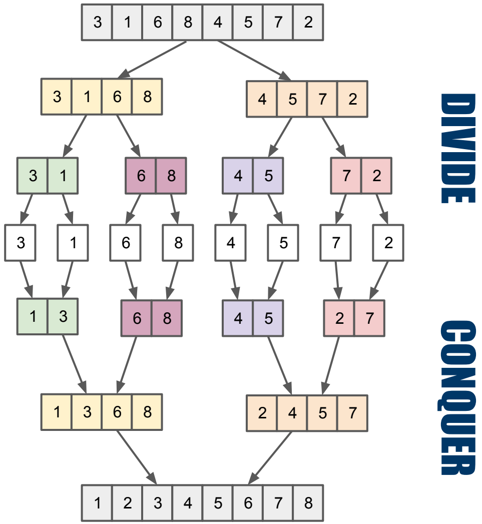
right

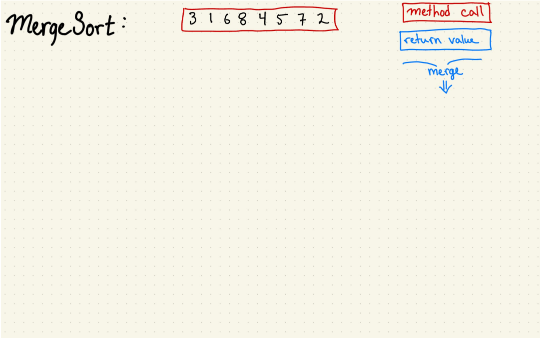
2	4	5	7

j = 0

result

[ ]





Example 3 - Mergesort

The Mergesort performs the sort:

- merging the sorted halves into a sorted array repeatedly, until the whole array is sorted;
    - this is the ***conquer*** part.

```python
def merge(left, right):
    result, i, j = [], 0 , 0
    while i < len(left) and j < len(right):
        if (left[i] <= right[j]):
            result.append(left[i])
            i += 1
        else:
            result.append(right[j])
            j += 1
    result += left[i:]
    result += right[j:]
    return result
```

left

1	3	6	8

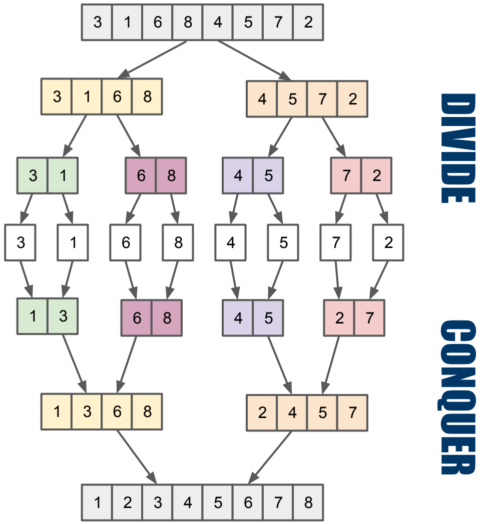
right

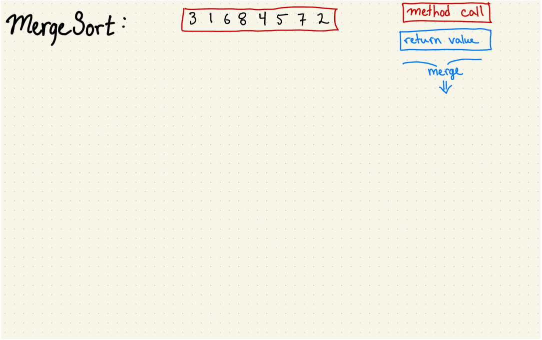
2	4	5	7

i = 1 ***j = 0***

result

1



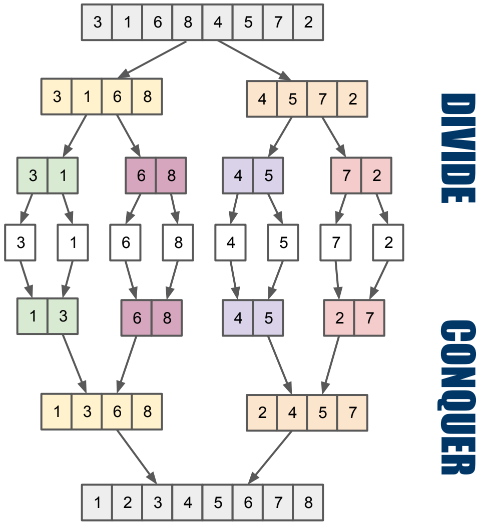


Step 2 of 9

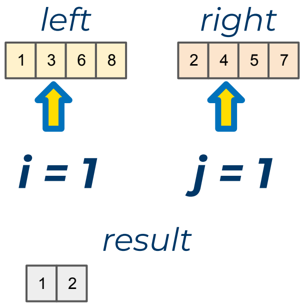
Example 3 - Mergesort

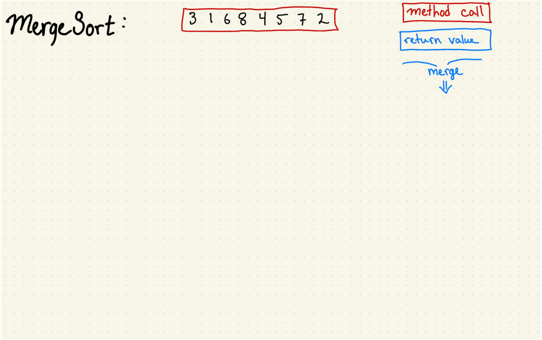
The Mergesort performs the sort:

- merging the sorted halves into a sorted array repeatedly, until the whole array is sorted;
  - this is the ***conquer*** part.




```python
def merge(left, right):
    result, i, j = [], 0 , 0
    while i < len(left) and j < len(right):
        if (left[i] <= right[j]):
            result.append(left[i])
            i += 1
        else:
            result.append(right[j])
            j += 1
    result += left[i:]
    result += right[j:]
    return result
```



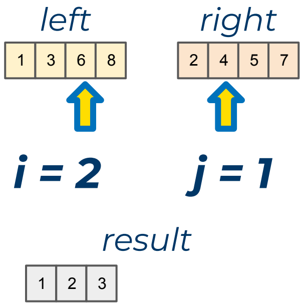


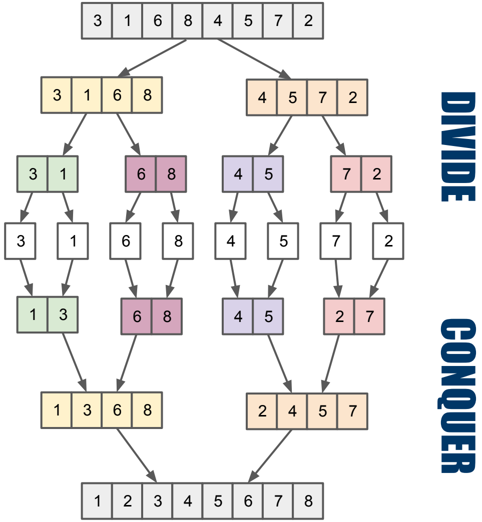
Example 3 - Mergesort

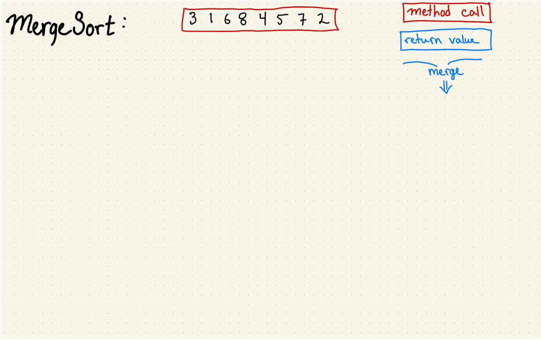
The Mergesort performs the sort:

- merging the sorted halves into a sorted array repeatedly, until the whole array is sorted;
    - this is the ***conquer*** part.

```python
def merge(left, right):
    result, i, j = [], 0 , 0
    while i < len(left) and j < len(right):
        if (left[i] <= right[j]):
            result.append(left[i])
            i += 1
        else:
            result.append(right[j])
            j += 1
    result += left[i:]
    result += right[j:]
    return result
```







Step 4 of 9

Example 3 - Mergesort

The Mergesort performs the sort:

- merging the sorted halves into a sorted array repeatedly, until the whole array is sorted;
  - this is the ***conquer*** part.

```python
def merge(left, right):
    result, i, j = [], 0 , 0
    while i < len(left) and j < len(right):
        if (left[i] <= right[j]):
            result.append(left[i])
            i += 1
        else:
            result.append(right[j])
            j += 1
    result += left[i:]
    result += right[j:]
    return result
```

left

1	3	6	8

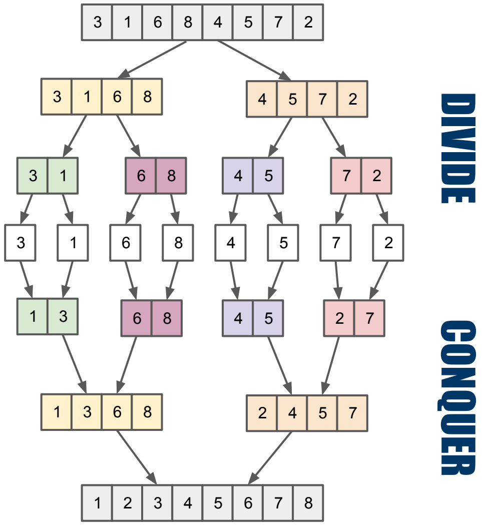
right

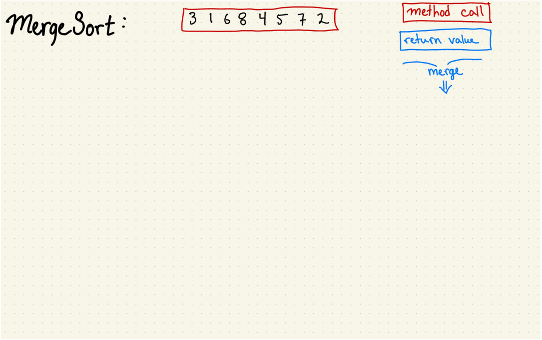
2	4	5	7

***i* = 2**   ***j* = 2**

result

1	2	3	4



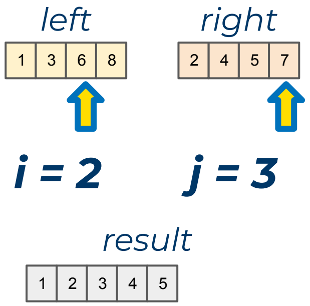


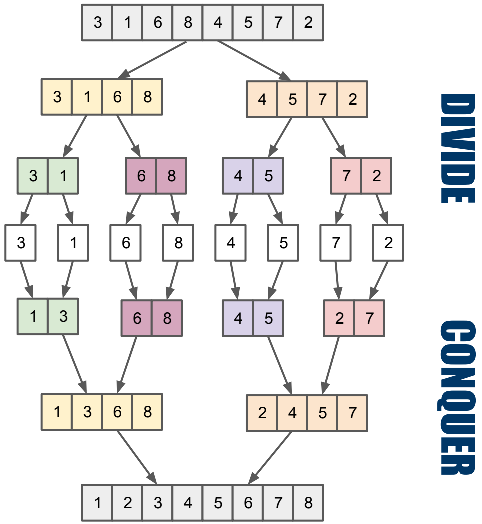
Example 3 - Mergesort

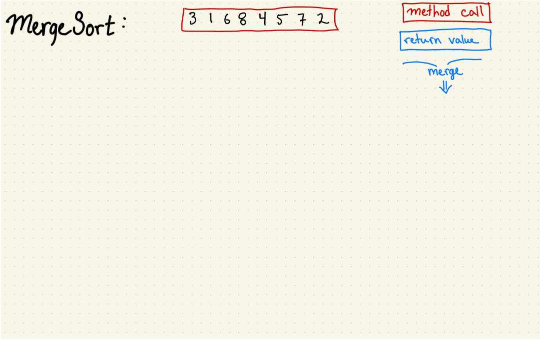
The Mergesort performs the sort:

- merging the sorted halves into a sorted array repeatedly, until the whole array is sorted;
    - this is the ***conquer*** part.

```python
def merge(left, right):
    result, i, j = [], 0 , 0
    while i < len(left) and j < len(right):
        if (left[i] <= right[j]):
            result.append(left[i])
            i += 1
        else:
            result.append(right[j])
            j += 1
    result += left[i:]
    result += right[j:]
    return result
```







MERRIMACK COLLEGE

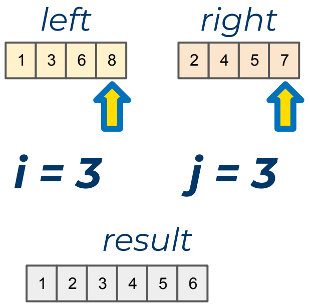
Step 6 of 9

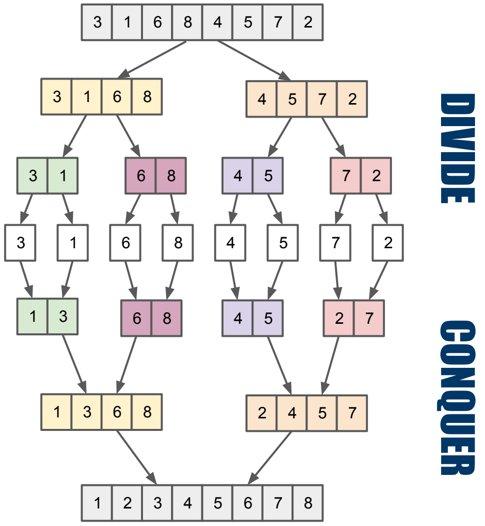
Example 3 - Mergesort

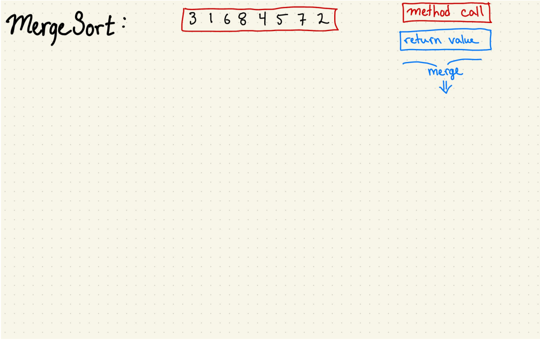
The Mergesort performs the sort:

- merging the sorted halves into a sorted array repeatedly, until the whole array is sorted;
    - this is the ***conquer*** part.

```python
def merge(left, right):
    result, i, j = [], 0 , 0
    while i < len(left) and j < len(right):
        if (left[i] <= right[j]):
            result.append(left[i])
            i += 1
        else:
            result.append(right[j])
            j += 1
    result += left[i:]
    result += right[j:]
    return result
```





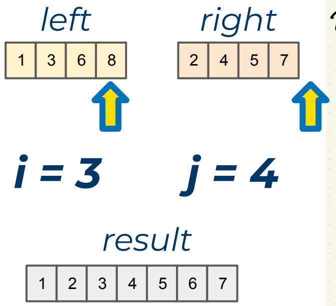


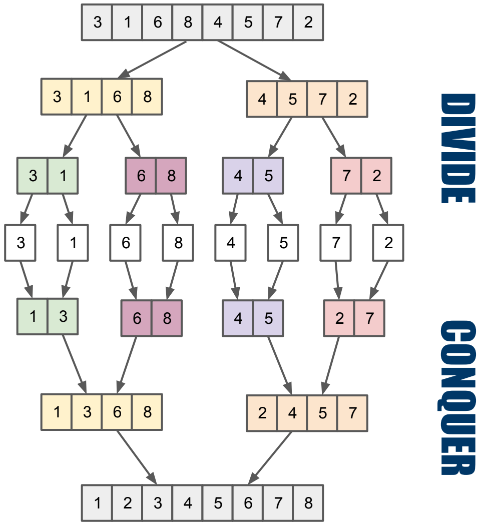
Example 3 - Mergesort

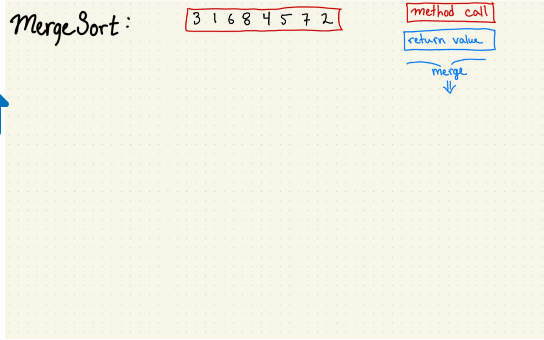
The Mergesort performs the sort:

- merging the sorted halves into a sorted array repeatedly, until the whole array is sorted;
    - this is the ***conquer*** part.

```python
def merge(left, right):
    result, i, j = [], 0 , 0
    while i < len(left) and j < len(right):
        if (left[i] <= right[j]):
            result.append(left[i])
            i += 1
        else:
            result.append(right[j])
            j += 1
    result += left[i:]
    result += right[j:]
    return result
```







Step 8 of 9

MERRIMACK COLLEGE

Example 3 - Mergesort

The Mergesort performs the sort:

- merging the sorted halves into a sorted array repeatedly, until the whole array is sorted;
    - this is the ***conquer*** part.

```python
def merge(left, right):
    result, i, j = [], 0 , 0
    while i < len(left) and j < len(right):
        if (left[i] <= right[j]):
            result.append(left[i])
            i += 1
        else:
            result.append(right[j])
            j += 1
    result += left[i:]
    result += right[j:]
    return result
```

left

1	3	6	8

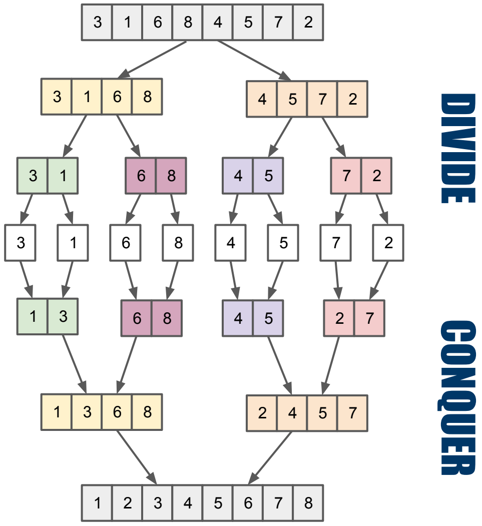
right

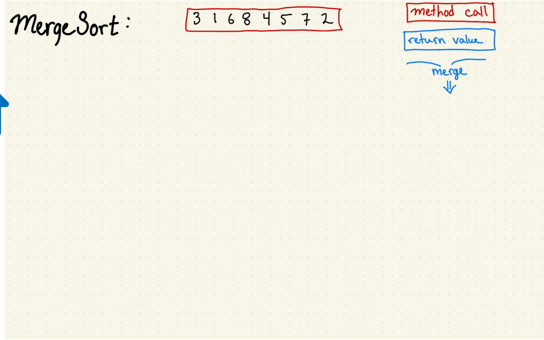
2	4	5	7

i = 3 ***j = 4***

result

1	2	3	4	5	6	7	8





Step 9 of 9

Example 3 - Mergesort

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

Master method:

- ***a=2, b=2, f(n)=n, log$_2$2=1***

MergeSort: 3 1 6 8 4 5 7 2

method call

return value

merge

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

The Mergesort algorithm requires some extra memory for the array copies created, thus, the memory complexity is also equal to ***O(n log n)***.

```python
def mergesort(A):
    if len(A) <= 1:
        return A
    else:
        mid = len(A) // 2
        left = mergesort(A[:mid])
        right = mergesort(A[mid:])
        return merge(left, right)

def merge(left, right):
    result, i, j = [], 0 , 0
    while i < len(left) and j < len(right):
        if (left[i] <= right[j]):
            result.append(left[i])
            i += 1
        else:
            result.append(right[j])
            j += 1
    result += left[i:]
    result += right[j:]
    return result

A = [3, 1, 6, 8, 4, 5, 7, 2]
A = mergesort(A)
print(A)
```

MERRIMACK COLLEGE

O(n log n) - Log Linear Algorithm (time complexity)

Divide-and-Conquer Algorithms Examples

The examples that will be seen are:

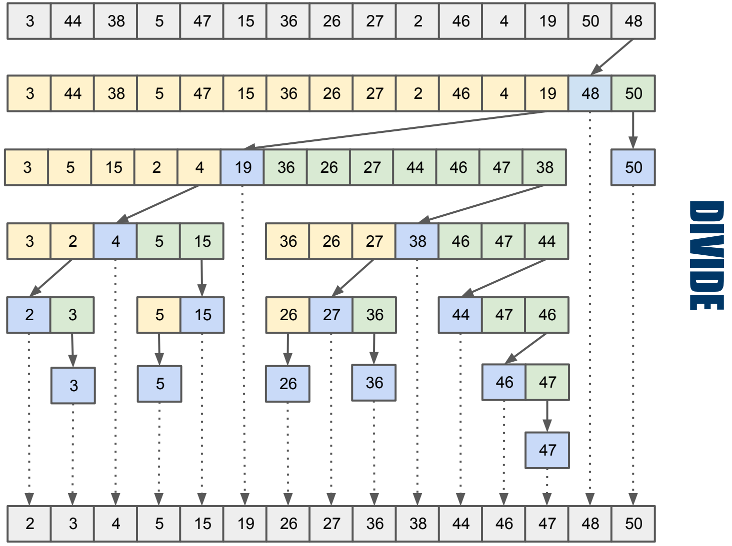
- Array sum
- Height of Binary Tree
- Mergesort
- **Quicksort**

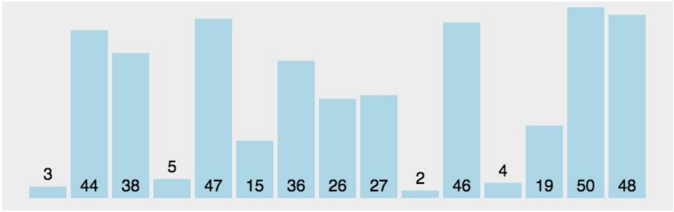
Example 4 - Quicksort

- Giving an array $\boldsymbol{A = [\, a_1, a_2, \dots a_n \,]}$
- Deliver a permutation of $\boldsymbol{A}$ where $\boldsymbol{a_i <= a_j}$ if $\boldsymbol{i < j}$

The Quicksort performs the sort:

- calling recursively a Lomuto partition (or other splitting an array in two two parts where one has smaller elements than the other using a pivot) until you have a single element part;
  - this is the ***divide*** part;
- Once this is done, the sort is done;
  - this is the (empty) ***conquer*** part.





Wikipedia: [Tony Hoare](#).

Example 4 - Quicksort

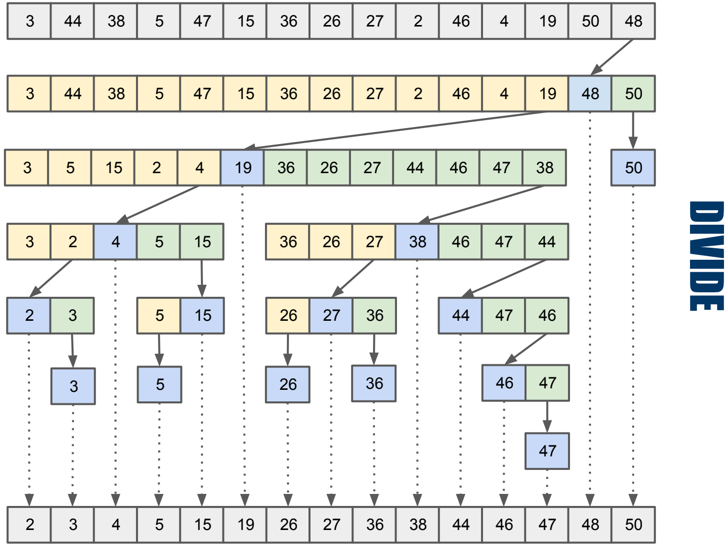
The Quicksort performs the sort:

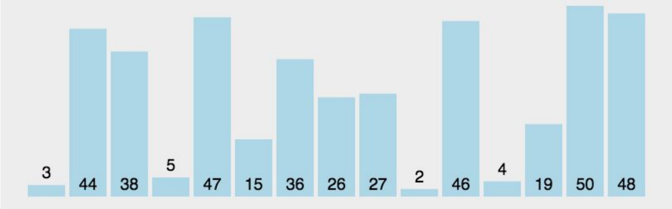
- calling recursively a Lomuto partition until you have a single element part;
    - this is the ***divide*** part.

```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
```

```python
A = [3, 44, 38, 5, 47, 15, 36, 26, 27, 2, 46, 4, 19, 50, 48]
quicksort(A, 0, len(A)-1)
print(A)
```

```python
def quicksort(A, left, right):
    if (left < right):
        mid = lomuto(A, left, right)
        quicksort(A, left, mid-1)
        quicksort(A, mid+1, right)
```





Video: Quicksort In Python Explained.

Example 4 - Quicksort

The Quicksort performs the sort:

- calling recursively a Lomuto partition until you have a single element part;
  - this is the ***divide*** part.

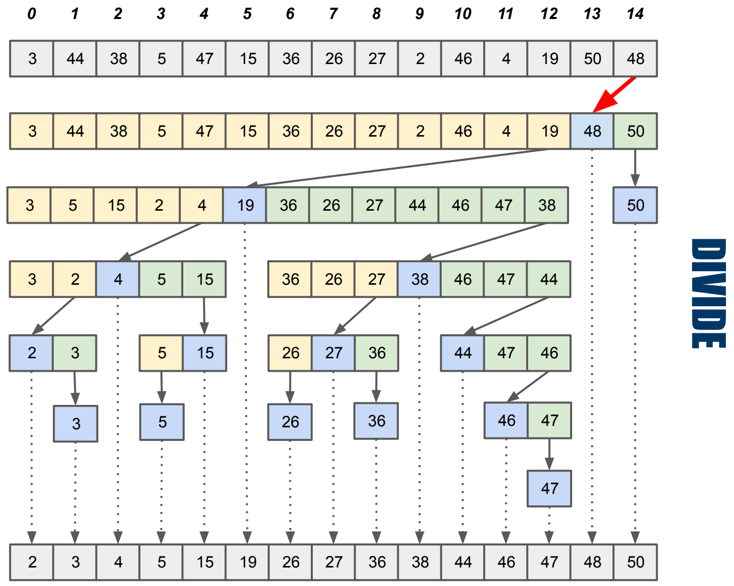
```python
def lomuto(A, left, right):
    p = A[right]
    i = left
    for j in range(left, right):
        if A[j] < p:
            A[i], A[j] = A[j], A[i]
            i += 1
    A[i], A[right] = A[right], A[i]
    return i
```

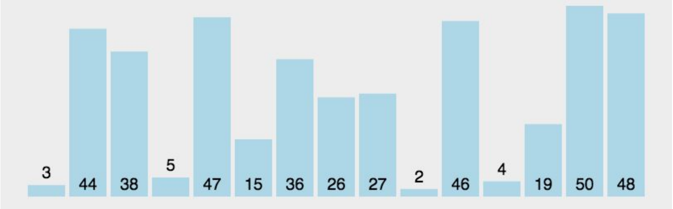
Call Lomuto for whole ***A*** (15 elements)

left = 0 - right = 14

```python
A = [3, 44, 38, 5, 47, 15, 36, 26, 27, 2, 46, 4, 19, 50, 48]
quicksort(A, 0, len(A)-1)
print(A)
```

```python
def quicksort(A, left, right):
    if (left < right):
        mid = lomuto(A, left, right)
        quicksort(A, left, mid-1)
        quicksort(A, mid+1, right)
```





Example 4 - Quicksort

The Quicksort performs the sort:

- calling recursively a Lomuto partition until you have a single element part;
  - this is the ***divide*** part.

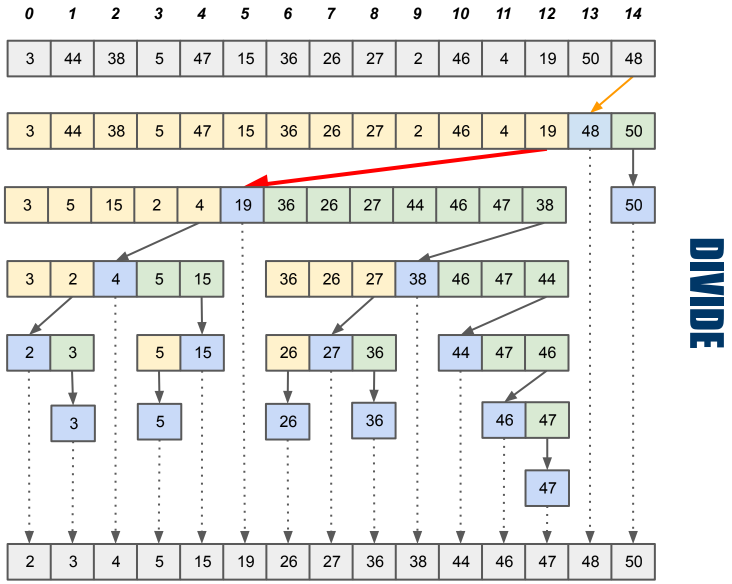
```python
def lomuto(A, left, right):
    p = A[right]
    i = left
    for j in range(left, right):
        if A[j] < p:
            A[i], A[j] = A[j], A[i]
            i += 1
    A[i], A[right] = A[right], A[i]
    return i
```

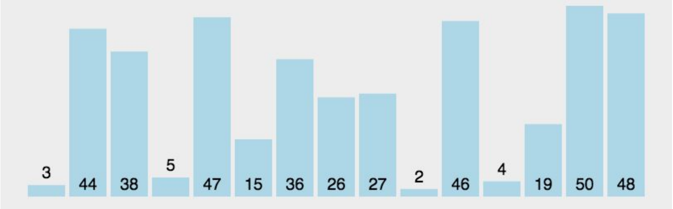
Call Lomuto for just 13 elements of ***A***

left = 0 - right = 12

```python
A = [3, 44, 38, 5, 47, 15, 36, 26, 27, 2, 46, 4, 19, 50, 48]
quicksort(A, 0, len(A)-1)
print(A)
```

```python
def quicksort(A, left, right):
    if (left < right):
        mid = lomuto(A, left, right)
        quicksort(A, left, mid-1)
        quicksort(A, mid+1, right)
```





MERRIMACK COLLEGE

Step 2 of 15

Example 4 - Quicksort

The Quicksort performs the sort:

- calling recursively a Lomuto partition until you have a single element part;
    - this is the ***divide*** part.

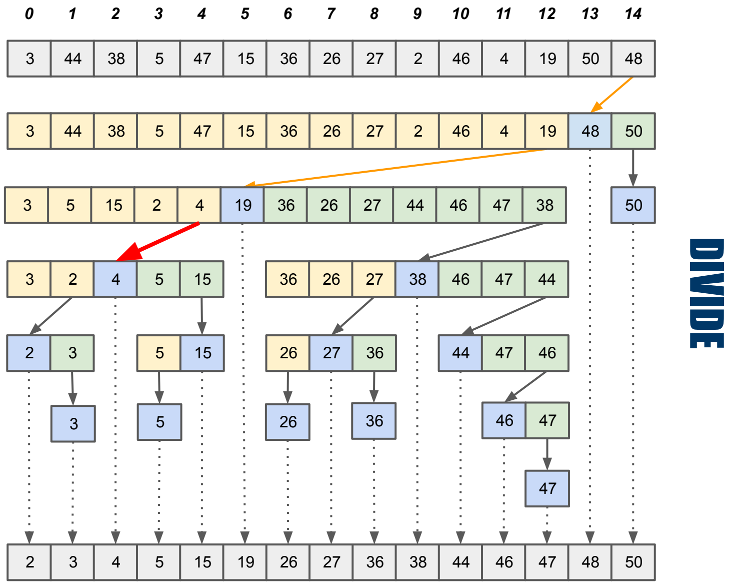
```python
def lomuto(A, left, right):
    p = A[right]
    i = left
    for j in range(left, right):
        if A[j] < p:
            A[i], A[j] = A[j], A[i]
            i += 1
    A[i], A[right] = A[right], A[i]
    return i
```

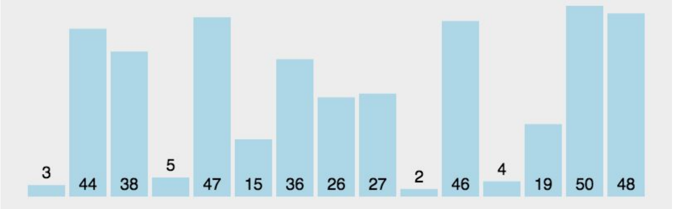
Call Lomuto for just 5 elements of ***A***

left = 0 - right = 4

```python
A = [3, 44, 38, 5, 47, 15, 36, 26, 27, 2, 46, 4, 19, 50, 48]
quicksort(A, 0, len(A)-1)
print(A)
```

```python
def quicksort(A, left, right):
    if (left < right):
        mid = lomuto(A, left, right)
        quicksort(A, left, mid-1)
        quicksort(A, mid+1, right)
```





Example 4 - Quicksort

The Quicksort performs the sort:

- calling recursively a Lomuto partition until you have a single element part;
    - this is the ***divide*** part.

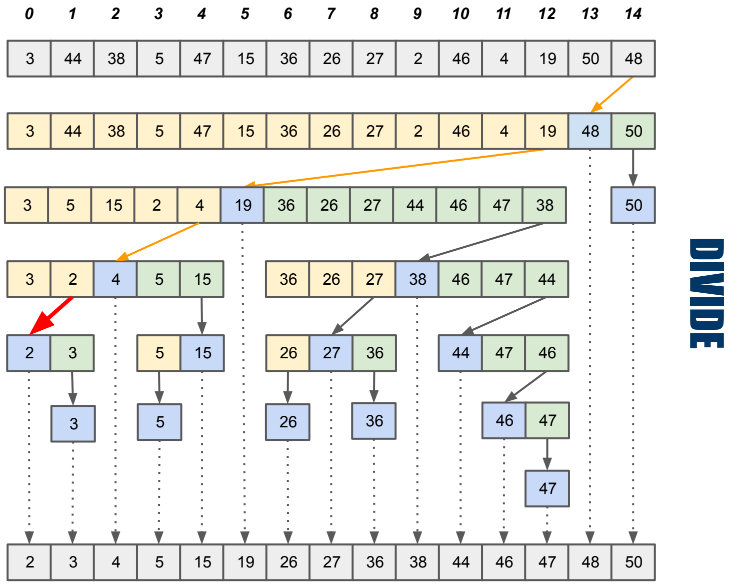
```python
def lomuto(A, left, right):
    p = A[right]
    i = left
    for j in range(left, right):
        if A[j] < p:
            A[i], A[j] = A[j], A[i]
            i += 1
    A[i], A[right] = A[right], A[i]
    return i
```

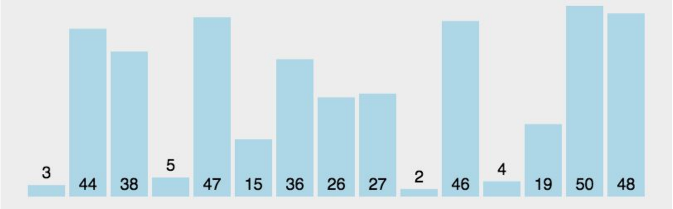
Call Lomuto for just 2 elements of ***A***

left = 0 - right = 2

```python
A = [3, 44, 38, 5, 47, 15, 36, 26, 27, 2, 46, 4, 19, 50, 48]
quicksort(A, 0, len(A)-1)
print(A)
```

```python
def quicksort(A, left, right):
    if (left < right):
        mid = lomuto(A, left, right)
        quicksort(A, left, mid-1)
        quicksort(A, mid+1, right)
```





Example 4 - Quicksort

The Quicksort performs the sort:

- calling recursively a Lomuto partition until you have a single element part;
  - this is the ***divide*** part.

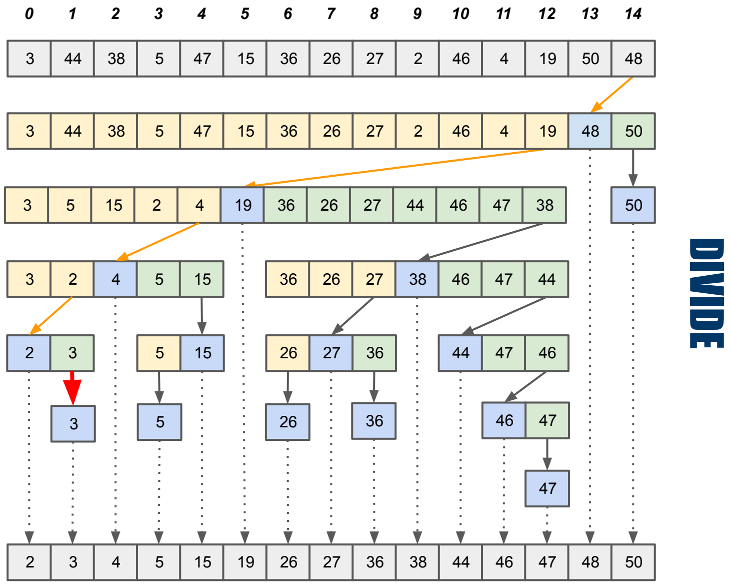
```python
def lomuto(A, left, right):
    p = A[right]
    i = left
    for j in range(left, right):
        if A[j] < p:
            A[i], A[j] = A[j], A[i]
            i += 1
    A[i], A[right] = A[right], A[i]
    return i
```

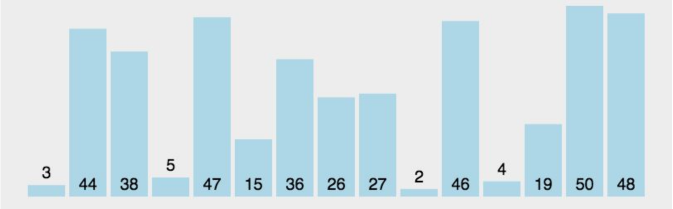
Call Lomuto for just 1 element of ***A***

left = 2 - right = 2

```python
A = [3, 44, 38, 5, 47, 15, 36, 26, 27, 2, 46, 4, 19, 50, 48]
quicksort(A, 0, len(A)-1)
print(A)
```

```python
def quicksort(A, left, right):
    if (left < right):
        mid = lomuto(A, left, right)
        quicksort(A, left, mid-1)
        quicksort(A, mid+1, right)
```





Example 4 - Quicksort

The Quicksort performs the sort:

- calling recursively a Lomuto partition until you have a single element part;
  - this is the ***divide*** part.

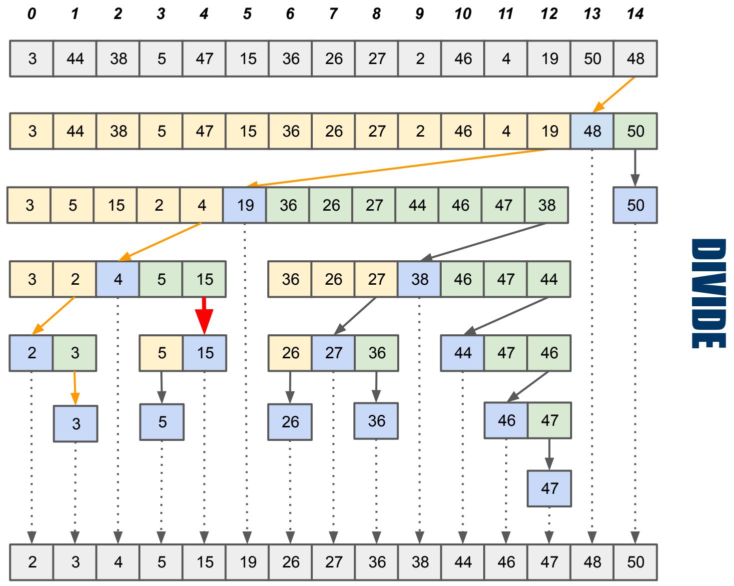
```python
def lomuto(A, left, right):
    p = A[right]
    i = left
    for j in range(left, right):
        if A[j] < p:
            A[i], A[j] = A[j], A[i]
            i += 1
    A[i], A[right] = A[right], A[i]
    return i
```

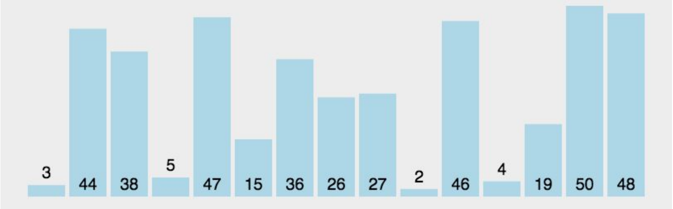
Call Lomuto for just 2 elements of ***A***

left = 3 - right = 4

```python
A = [3, 44, 38, 5, 47, 15, 36, 26, 27, 2, 46, 4, 19, 50, 48]
quicksort(A, 0, len(A)-1)
print(A)
```

```python
def quicksort(A, left, right):
    if (left < right):
        mid = lomuto(A, left, right)
        quicksort(A, left, mid-1)
        quicksort(A, mid+1, right)
```





Example 4 - Quicksort

The Quicksort performs the sort:

- calling recursively a Lomuto partition until you have a single element part;
  - this is the ***divide*** part.

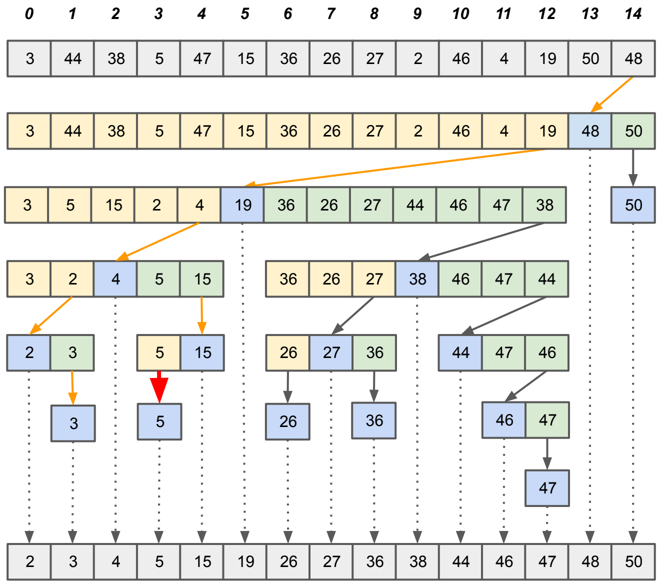
```python
def lomuto(A, left, right):
    p = A[right]
    i = left
    for j in range(left, right):
        if A[j] < p:
            A[i], A[j] = A[j], A[i]
            i += 1
    A[i], A[right] = A[right], A[i]
    return i
```

Call Lomuto for just 1 element of ***A***

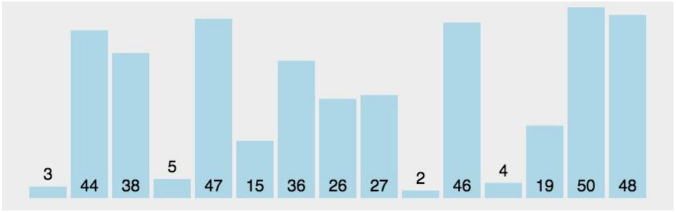
left = 4 - right = 4

```python
A = [3, 44, 38, 5, 47, 15, 36, 26, 27, 2, 46, 4, 19, 50, 48]
quicksort(A, 0, len(A)-1)
print(A)
```

```python
def quicksort(A, left, right):
    if (left < right):
        mid = lomuto(A, left, right)
        quicksort(A, left, mid-1)
        quicksort(A, mid+1, right)
```



DIVIDE



MERRIMACK COLLEGE

Step 7 of 15

Example 4 - Quicksort

The Quicksort performs the sort:

- calling recursively a Lomuto partition until you have a single element part;
    - this is the ***divide*** part.

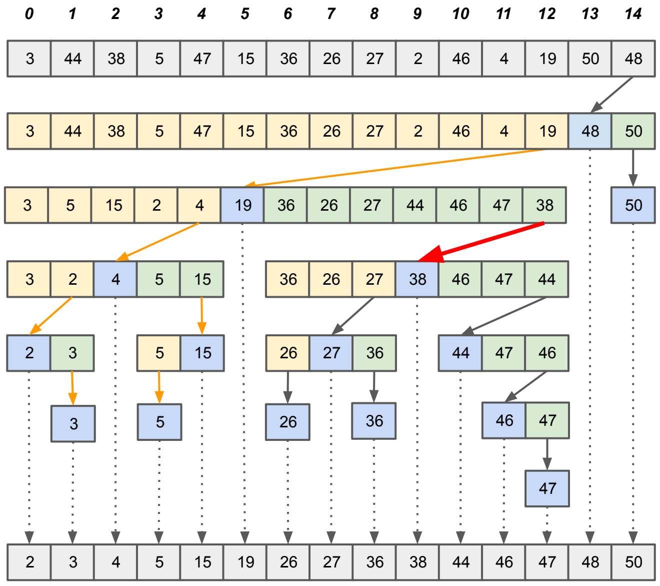
```python
def lomuto(A, left, right):
    p = A[right]
    i = left
    for j in range(left, right):
        if A[j] < p:
            A[i], A[j] = A[j], A[i]
            i += 1
    A[i], A[right] = A[right], A[i]
    return i
```

Call Lomuto for just 7 elements of ***A***

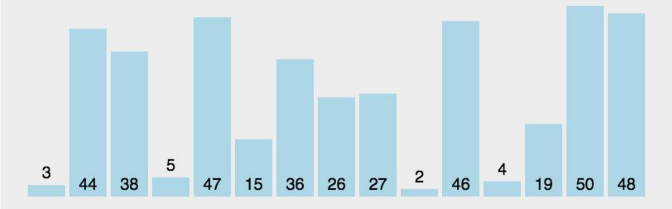
left = 6 - right = 12

```python
A = [3, 44, 38, 5, 47, 15, 36, 26, 27, 2, 46, 4, 19, 50, 48]
quicksort(A, 0, len(A)-1)
print(A)
```

```python
def quicksort(A, left, right):
    if (left < right):
        mid = lomuto(A, left, right)
        quicksort(A, left, mid-1)
        quicksort(A, mid+1, right)
```



DIVIDE



Example 4 - Quicksort

The Quicksort performs the sort:

- calling recursively a Lomuto partition until you have a single element part;
  - this is the ***divide*** part.

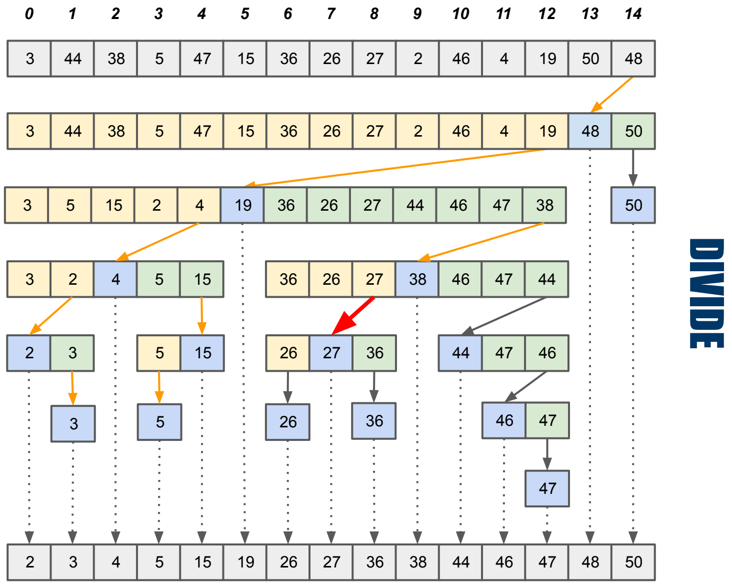
```python
def lomuto(A, left, right):
    p = A[right]
    i = left
    for j in range(left, right):
        if A[j] < p:
            A[i], A[j] = A[j], A[i]
            i += 1
    A[i], A[right] = A[right], A[i]
    return i
```

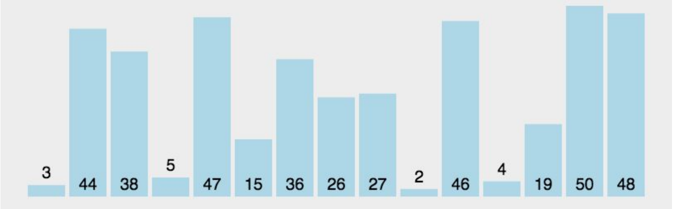
Call Lomuto for just 3 elements of ***A***

left = 6 - right = 8

```python
A = [3, 44, 38, 5, 47, 15, 36, 26, 27, 2, 46, 4, 19, 50, 48]
quicksort(A, 0, len(A)-1)
print(A)
```

```python
def quicksort(A, left, right):
    if (left < right):
        mid = lomuto(A, left, right)
        quicksort(A, left, mid-1)
        quicksort(A, mid+1, right)
```





MERRIMACK COLLEGE

Step 9 of 15

Example 4 - Quicksort

The Quicksort performs the sort:

- calling recursively a Lomuto partition until you have a single element part;
    - this is the ***divide*** part.

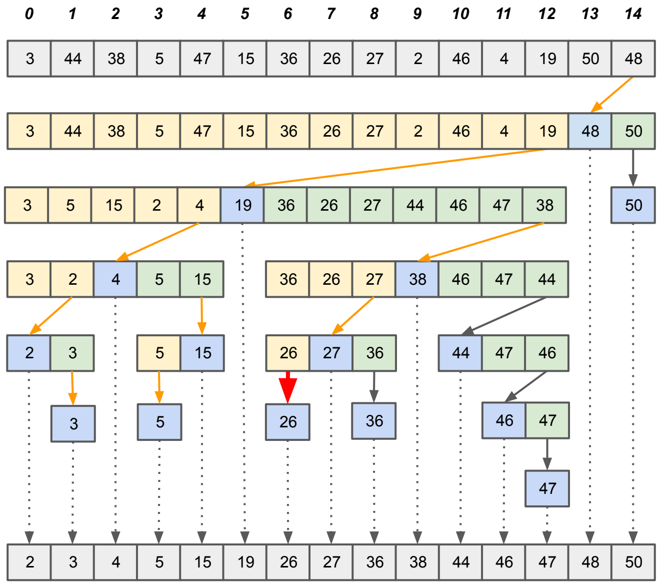
```python
def lomuto(A, left, right):
    p = A[right]
    i = left
    for j in range(left, right):
        if A[j] < p:
            A[i], A[j] = A[j], A[i]
            i += 1
    A[i], A[right] = A[right], A[i]
    return i
```

```python
A = [3, 44, 38, 5, 47, 15, 36, 26, 27, 2, 46, 4, 19, 50, 48]
quicksort(A, 0, len(A)-1)
print(A)
```

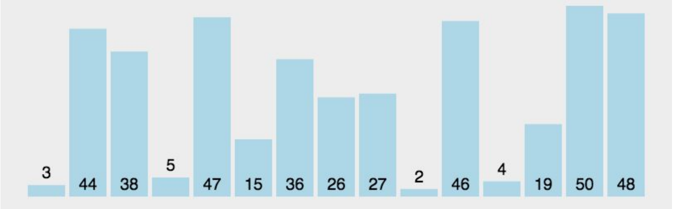
```python
def quicksort(A, left, right):
    if (left < right):
        mid = lomuto(A, left, right)
        quicksort(A, left, mid-1)
        quicksort(A, mid+1, right)
```

Call Lomuto for just 1 elements of ***A***

left = 6 - right = 6



DIVIDE



Step 10 of 15

Example 4 - Quicksort

The Quicksort performs the sort:

- calling recursively a Lomuto partition until you have a single element part;
  - this is the ***divide*** part.

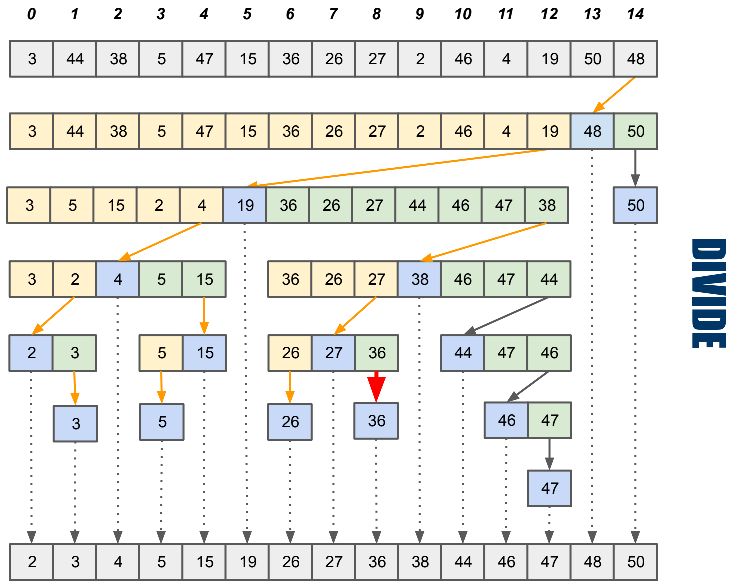
```python
def lomuto(A, left, right):
    p = A[right]
    i = left
    for j in range(left, right):
        if A[j] < p:
            A[i], A[j] = A[j], A[i]
            i += 1
    A[i], A[right] = A[right], A[i]
    return i
```

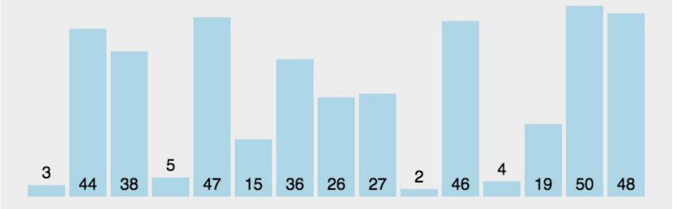
Call Lomuto for just 1 element of ***A***

left = 8   -   right = 8

```python
A = [3, 44, 38, 5, 47, 15, 36, 26, 27, 2, 46, 4, 19, 50, 48]
quicksort(A, 0, len(A)-1)
print(A)
```

```python
def quicksort(A, left, right):
    if (left < right):
        mid = lomuto(A, left, right)
        quicksort(A, left, mid-1)
        quicksort(A, mid+1, right)
```





Example 4 - Quicksort

The Quicksort performs the sort:

- calling recursively a Lomuto partition until you have a single element part;
    - this is the ***divide*** part.

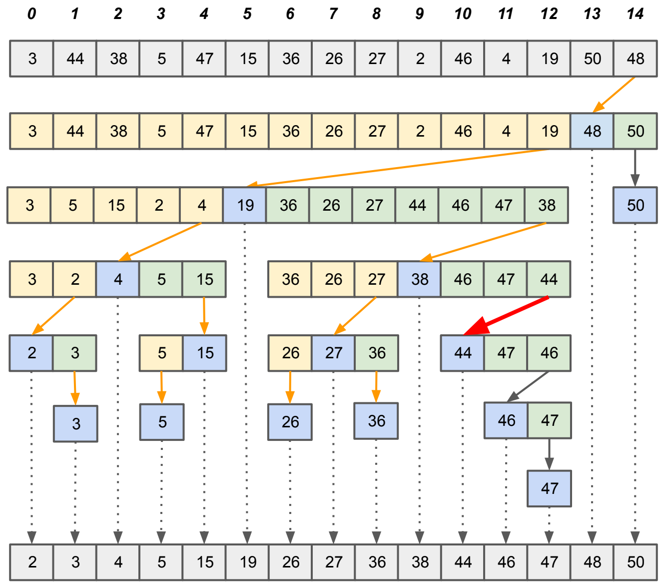
```python
def lomuto(A, left, right):
    p = A[right]
    i = left
    for j in range(left, right):
        if A[j] < p:
            A[i], A[j] = A[j], A[i]
            i += 1
    A[i], A[right] = A[right], A[i]
    return i
```

Call Lomuto for just 3 elements of ***A***

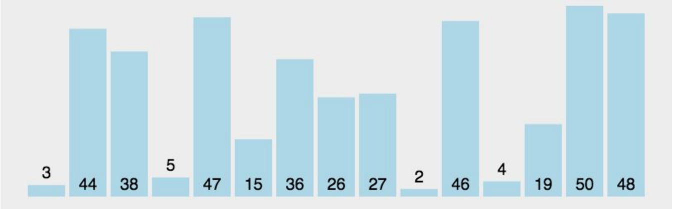
left = 10 - right = 12

```python
A = [3, 44, 38, 5, 47, 15, 36, 26, 27, 2, 46, 4, 19, 50, 48]
quicksort(A, 0, len(A)-1)
print(A)
```

```python
def quicksort(A, left, right):
    if (left < right):
        mid = lomuto(A, left, right)
        quicksort(A, left, mid-1)
        quicksort(A, mid+1, right)
```



DIVIDE



Example 4 - Quicksort

The Quicksort performs the sort:

- calling recursively a Lomuto partition until you have a single element part;
  - this is the ***divide*** part.

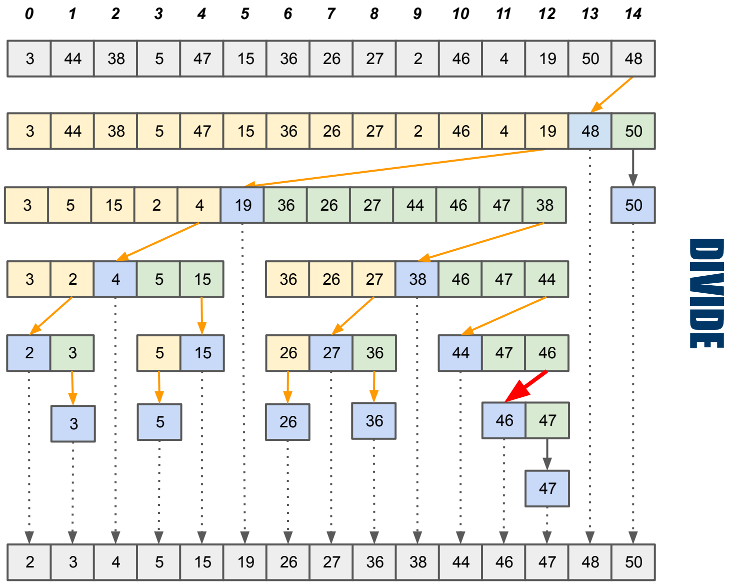
```python
def lomuto(A, left, right):
    p = A[right]
    i = left
    for j in range(left, right):
        if A[j] < p:
            A[i], A[j] = A[j], A[i]
            i += 1
    A[i], A[right] = A[right], A[i]
    return i
```

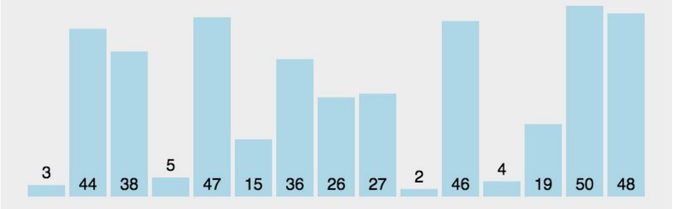
Call Lomuto for just 2 elements of ***A***

left = 11 - right = 12

```python
A = [3, 44, 38, 5, 47, 15, 36, 26, 27, 2, 46, 4, 19, 50, 48]
quicksort(A, 0, len(A)-1)
print(A)
```

```python
def quicksort(A, left, right):
    if (left < right):
        mid = lomuto(A, left, right)
        quicksort(A, left, mid-1)
        quicksort(A, mid+1, right)
```





Example 4 - Quicksort

The Quicksort performs the sort:

- calling recursively a Lomuto partition until you have a single element part;
    - this is the ***divide*** part.

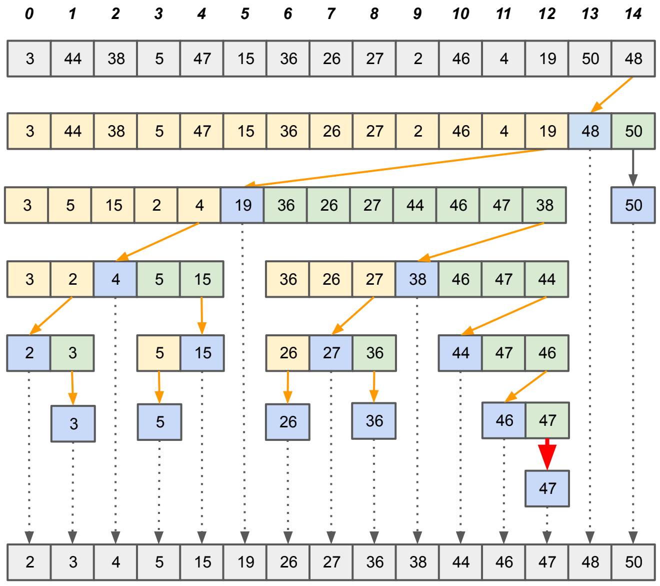
```python
def lomuto(A, left, right):
    p = A[right]
    i = left
    for j in range(left, right):
        if A[j] < p:
            A[i], A[j] = A[j], A[i]
            i += 1
    A[i], A[right] = A[right], A[i]
    return i
```

Call Lomuto for just 1 element of ***A***

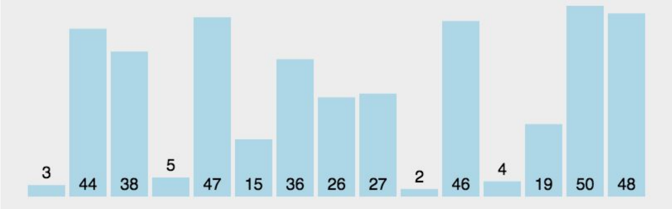
left = 12 - right = 12

```python
A = [3, 44, 38, 5, 47, 15, 36, 26, 27, 2, 46, 4, 19, 50, 48]
quicksort(A, 0, len(A)-1)
print(A)
```

```python
def quicksort(A, left, right):
    if (left < right):
        mid = lomuto(A, left, right)
        quicksort(A, left, mid-1)
        quicksort(A, mid+1, right)
```



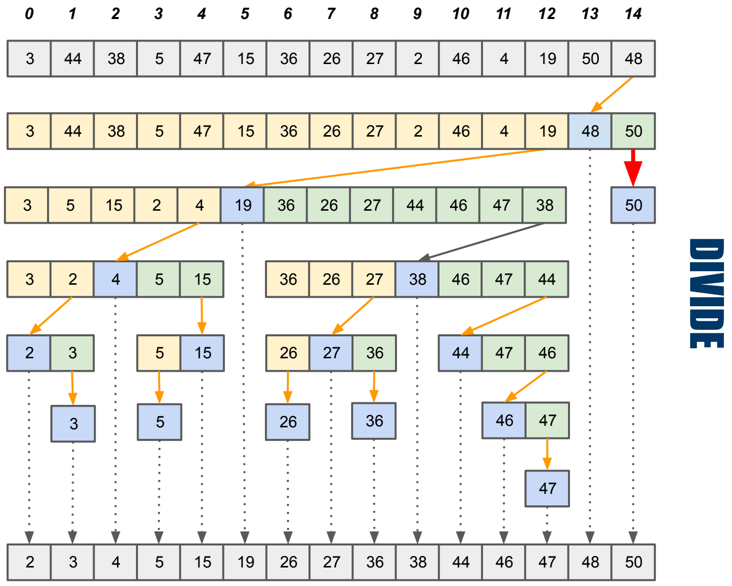
DIVIDE



Example 4 - Quicksort

The Quicksort performs the sort:

- calling recursively a Lomuto partition until you have a single element part;
  - this is the ***divide*** part.



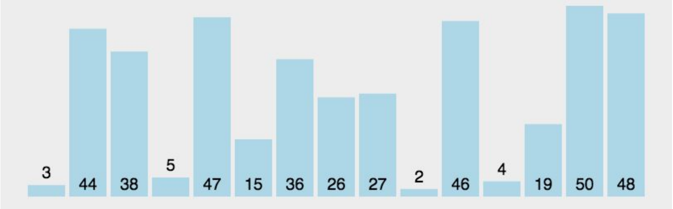

```
def lomuto(A, left, right):
    p = A[right]
    i = left
    for j in range(left, right):
        if A[j] < p:
            A[i], A[j] = A[j], A[i]
            i += 1
    A[i], A[right] = A[right], A[i]
    return i
```

Call Lomuto for just 1 element of ***A***

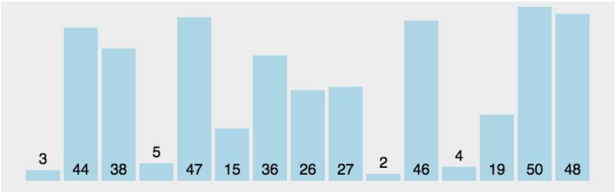
left = 12   -   right = 12

```
A = [3, 44, 38, 5, 47, 15, 36, 26, 27, 2, 46, 4, 19, 50, 48]
quicksort(A, 0, len(A)-1)
print(A)
```

```
def quicksort(A, left, right):
    if (left < right):
        mid = lomuto(A, left, right)
        quicksort(A, left, mid-1)
        quicksort(A, mid+1, right)
```



Example 4 - Quicksort



RARE

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

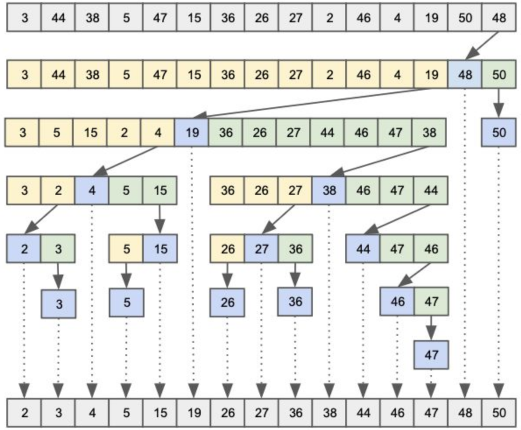
Worst case:

- ***T(n) = T(n-1) + n-1***
- Back-substitution:
  - ***O(n²)***

FREQUENT

Best case:

- ***T(n) = 2T(n/2) + n-1***
- Master method:
  - ***a=2, b=2, f(n)=n, log₂2=1***




```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

def quicksort(A, left, right):
    if (left < right):
        mid = lomuto(A, left, right)
        quicksort(A, left, mid-1)
        quicksort(A, mid+1, right)
```

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

It does not require extra memory, thus the memory complexity is constant: ***O(1)***.

MERRIMACK COLLEGE

O(n log n) - Log Linear Algorithm (time complexity)

This Week's tasks

- In-class Exercise E#7
- Coding Project P#7
- Quiz Q#7

Tasks

- Fill the worksheet as required.
- Develop 2 Python programs:
  - number of digits in a binary expansion;
  - sum of squares of positive Integers.
- Quiz #7 about this week topics.

In-class Exercise - E#7

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.

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

- (1) Trace the **recursive array sum** algorithm for the following arrays. Show to each recursive call the input array, the returned value, and the number of sums executed. At the end of the trace, present the total number of sums executed (the total of sums of all recursive calls.
  - A = [38, 21, 39, 60, -1, 10, 81, 23]
  - B = [2, 97, 5, 88, 9, 72, 12, 64, 17, 56, 21]
  - C = [100, 33, 22, 213, 65, 29, 153, 199, 47, 181, 85]
- (2) Trace the Mergesort algorithm for the following arrays. Show to each recursive call the two input and output arrays.
  - A = [38, 21, 39, 60, -1, 10, 81, 23]
  - B = [2, 97, 5, 88, 9, 72, 12, 64, 17, 56, 21]
  - C = [100, 33, 22, 213, 65, 29, 153, 199, 47, 181, 85]

CSC6013 - Worksheet for Week 7

Array Sum

1. Trace the recursive array sum algorithm for the following arrays. Show to each recursive call the input array, the returned value, and the number of sums executed. At the end of the trace, present the total number of sums executed (the total of sums of all recursive calls.
   a. A = [38, 21, 39, 60, -1, 10, 81, 23]
   b. B = [2, 97, 5, 88, 9, 72, 12, 64, 17, 56, 21]
   c. C = [100, 33, 22, 213, 65, 29, 153, 199, 47, 181, 85]

Mergesort

2. Trace the Mergesort algorithm for the following arrays. Show to each recursive call the two input and output arrays.
   a. A = [38, 21, 39, 60, -1, 10, 81, 23]
   b. B = [2, 97, 5, 88, 9, 72, 12, 64, 17, 56, 21]
   c. C = [100, 33, 22, 213, 65, 29, 153, 199, 47, 181, 85]

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

Seventh Coding Project - P#7

Develop one Python program to perform the Quicksort, but instead of sorting the elements in ascending order (from the smallest to the largest), the elements are sorted in the decrescent order (from the larger to the smallest). Your program also have to compute and print at the end the number of swaps performed and the number of recursive calls.

Test your program over the following arrays:

- A = [38, 21, 39, 60, -1, 10, 81, 23]
- B = [2, 97, 5, 88, 9, 72, 12, 64, 17, 56, 21]
- C = [100, 33, 22, 213, 65, 29, 153, 199, 47, 181, 85]

You have to submit the code (***.py*** file) of your algorithm, plus the ***.pdf*** of the trace for the three required tests.

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

Seventh Quiz - Q#7

- The seventh quiz in this course covers the topics of Week 7;
- The quiz will be available this Friday, and it is composed by 10 questions;
- The quiz should be taken on Canvas (Module 7), 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:

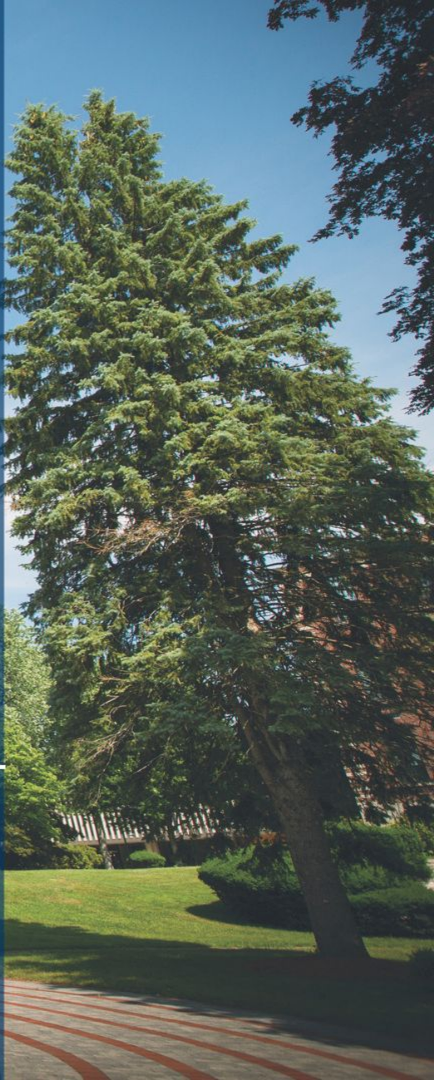
- Go to Canvas, answer the quiz and submit it within the deadline.

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

Welcome to CSC 6013

- **Do In-class Exercise E#7 until Friday;**
- **Do Quiz Q#7 (available Friday) until next Monday;**
- **Do Coding Project P#7 until next Monday.**

Next Week - Transform-and-conquer algorithms
... and the final exam!



MERRIMACK COLLEGE