Sorting Algorithms

- Selection Sort
- Insertion Sort
- Bubble Sort
- Merge Sort
- Quick Sort

Naive Sorting: Selection Sort

```
def selSort(nums):
   # sort nums into ascending order

   n = len(nums)

   # For each position in the list (except the very last)

   for bottom in range(n-1):
       # find the smallest item in nums[bottom]...nums[n-1]

       mp = bottom                       # bottom is smallest initially
       for i in range(bottom+1, n):      # look at each position
           if nums[i] < nums[mp]:        # this one is smaller
               mp = i                    # remember its index

       # swap smallest item to the bottom
       nums[bottom], nums[mp] = nums[mp], nums[bottom]
```

29, 64, 73, 34, **20**,
20, **64**, 73, 34, **29**,
20, 29, **73**, **34**, 64
20, 29, 34, **73**, **64**
20, 29, 34, 64, 73

가장 작은값을 찾아서 첫번째
자리에 있는 값과 교체

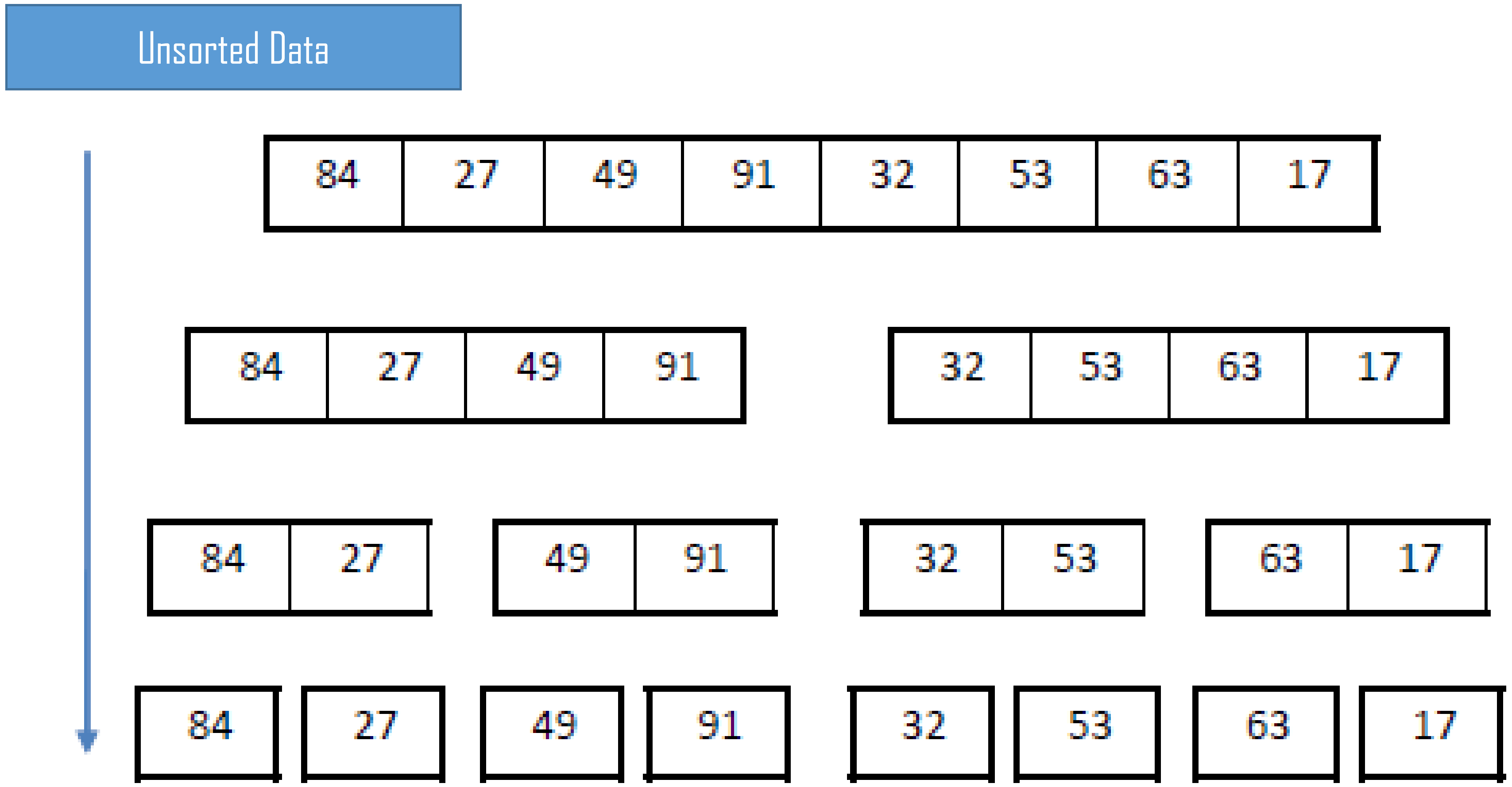
Python code for Merge-Sort [1/2]

```
def msort(list):
    if len(list) == 0 or len(list) == 1: # base case
        return list[:len(list)] # copy the input
    # recursive case
    halfway = len(list) // 2
    list1 = list[0:halfway]
    list2 = list[halfway:len(list)]
    newlist1 = msort(list1) # recursively sort left half
    newlist2 = msort(list2) # recursively sort right half
    newlist = merge(newlist1, newlist2)
    return newlist
```

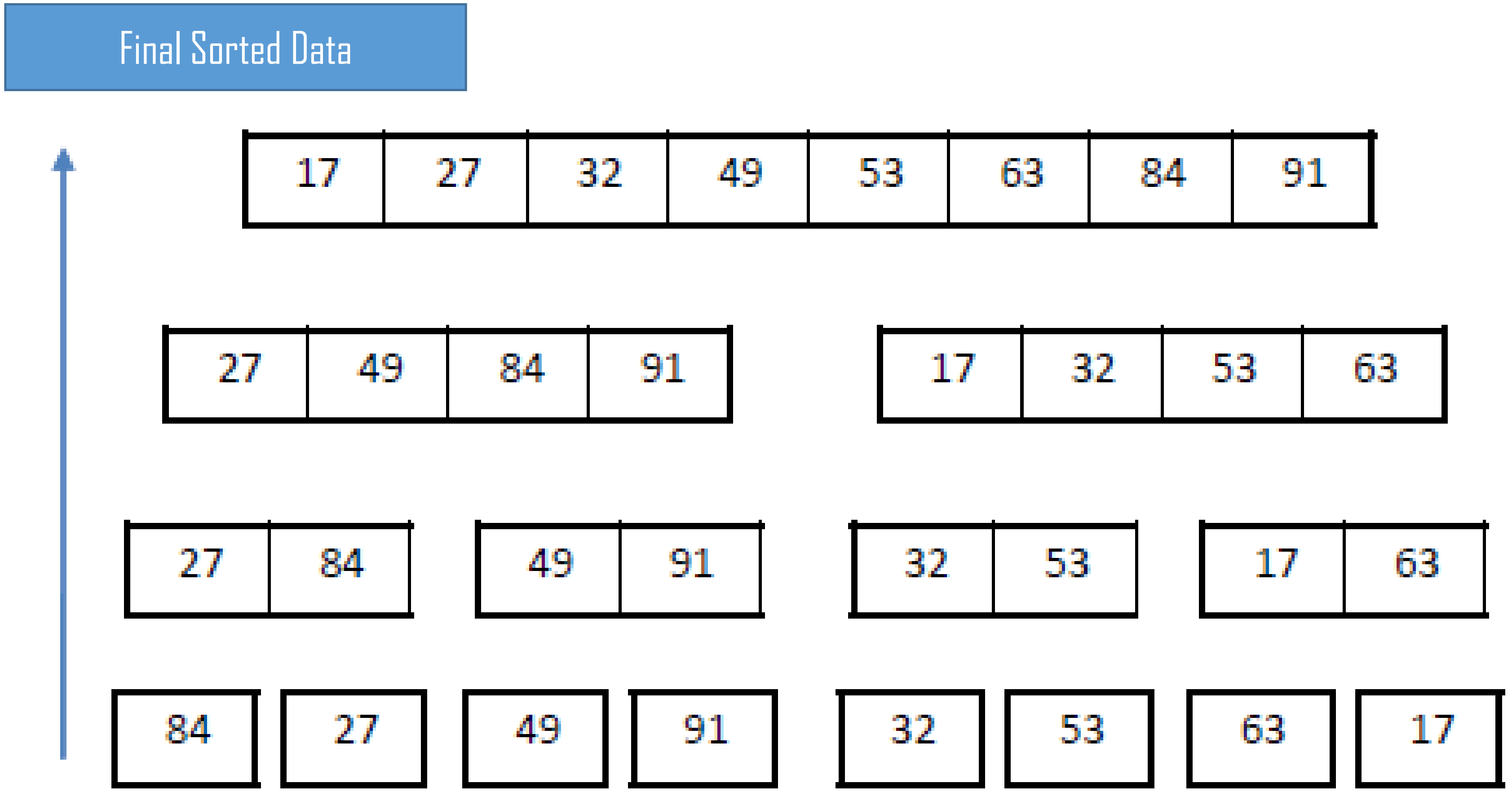
Python code for Merge-Sort [2/2]

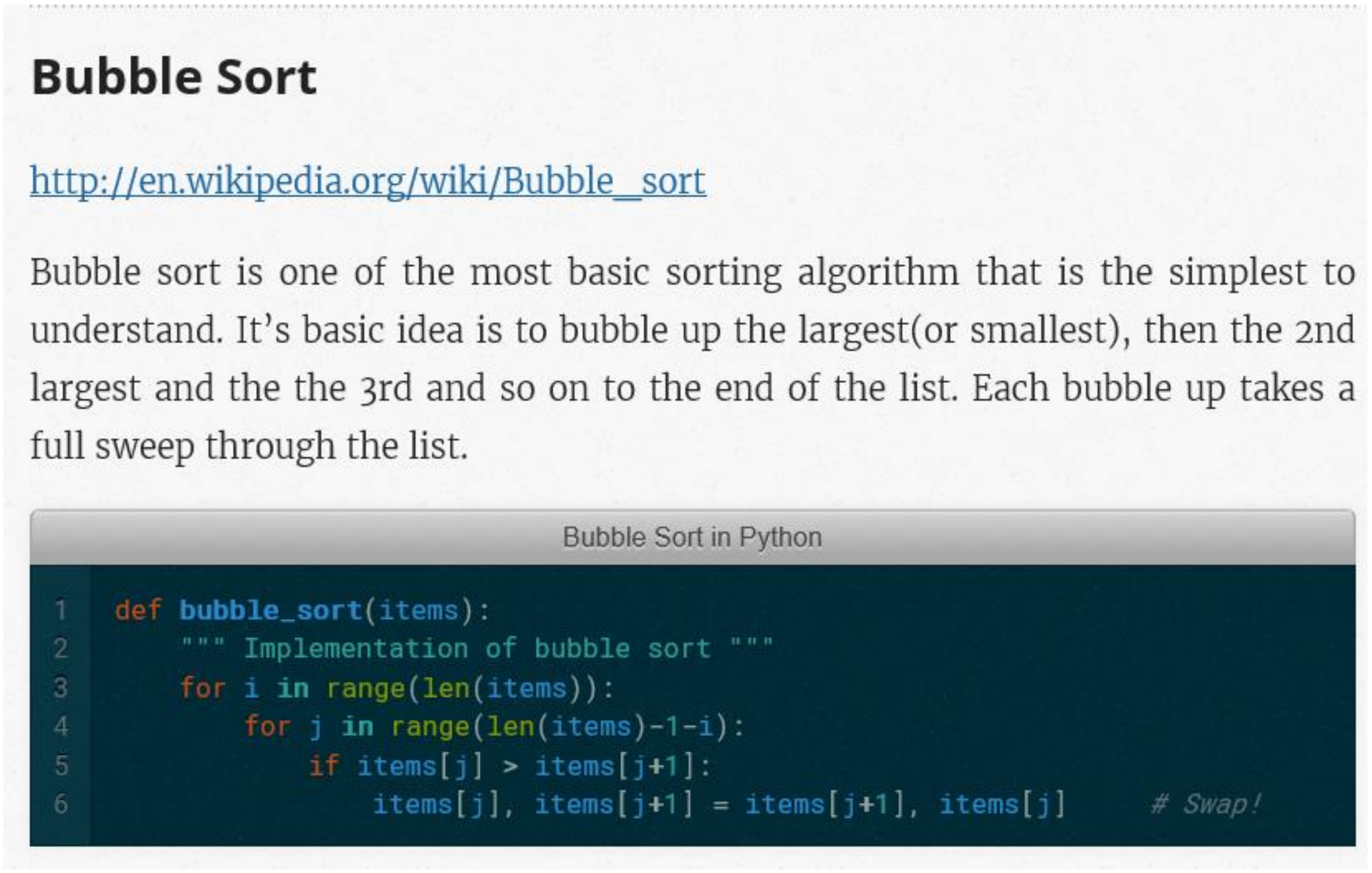
```python
def merge(a, b):
    index_a = 0 # the current index in list a
    index_b = 0 # the current index in list b
    c = []
    while index_a < len(a) and index_b < len(b):
        if a[index_a] <= b[index_b]:
            c.append(a[index_a])
            index_a = index_a + 1
        else:
            c.append(b[index_b])
            index_b = index_b + 1
    # when we exit the loop
    # we are at the end of at least one of the lists
    c.extend(a[index_a:])
    c.extend(b[index_b:])
    return c
```

Merge-Sort: Divide Step (Split)



Merge-Sort: Conquer Step (Merge)





Bubble Sort

http://en.wikipedia.org/wiki/Bubble_sort

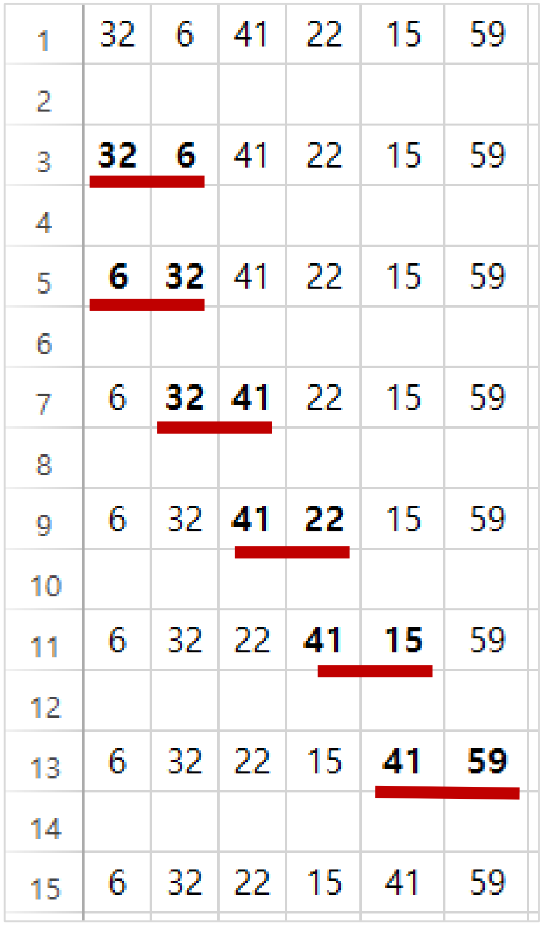
Bubble sort is one of the most basic sorting algorithm that is the simplest to understand. It's basic idea is to bubble up the largest(or smallest), then the 2nd largest and the the 3rd and so on to the end of the list. Each bubble up takes a full sweep through the list.

Bubble Sort in Python

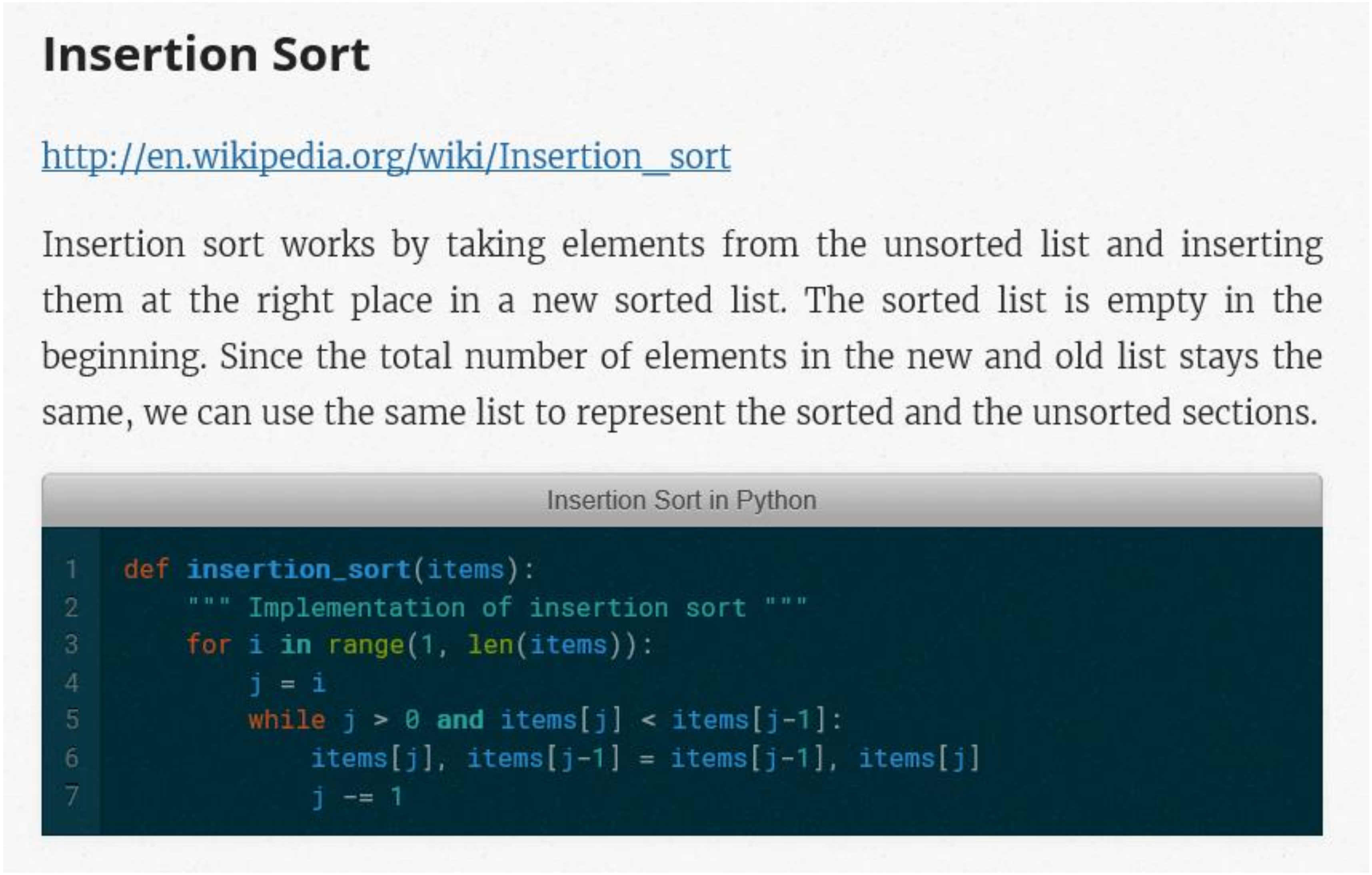
```python
def bubble_sort(items):
    """ Implementation of bubble sort """
    for i in range(len(items)):
        for j in range(len(items)-1-i):
            if items[j] > items[j+1]:
                items[j], items[j+1] = items[j+1], items[j]     # Swap!
```

1번째 2번째 비교 ➔ 필요한 swap 수행
2번째 3번째 비교 ➔ 필요한 swap 수행
…
(N-1)번째 N번째 비교 ➔ 필요한 swap 수행

Bubble Sort Shot



1	32	6	41	22	15	59
2						
3	**32**	**6**	41	22	15	59
4						
5	**6**	**32**	41	22	15	59
6						
7	6	**32**	**41**	22	15	59
8						
9	6	32	**41**	**22**	15	59
10						
11	6	32	22	**41**	**15**	59
12						
13	6	32	22	15	**41**	**59**
14						
15	6	32	22	15	41	59



Insertion Sort

http://en.wikipedia.org/wiki/Insertion_sort

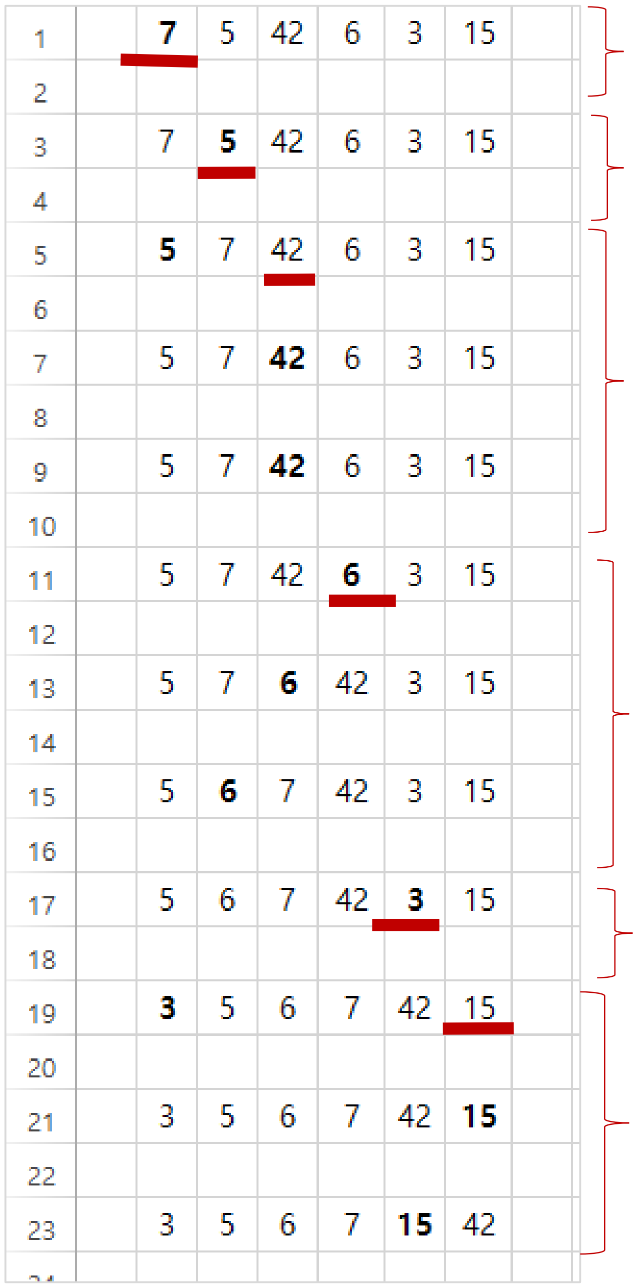
Insertion sort works by taking elements from the unsorted list and inserting them at the right place in a new sorted list. The sorted list is empty in the beginning. Since the total number of elements in the new and old list stays the same, we can use the same list to represent the sorted and the unsorted sections.

Insertion Sort in Python

```python
def insertion_sort(items):
    """ Implementation of insertion sort """
    for i in range(1, len(items)):
        j = i
        while j > 0 and items[j] < items[j-1]:
            items[j], items[j-1] = items[j-1], items[j]
            j -= 1
```

처음에는 empty인 sorted list를 두고, Unsorted list에서 1개씩 element를 sorted list에 이동하면서 sorted list를 유지하도록 insertion 수행

Insertion Sort Shot

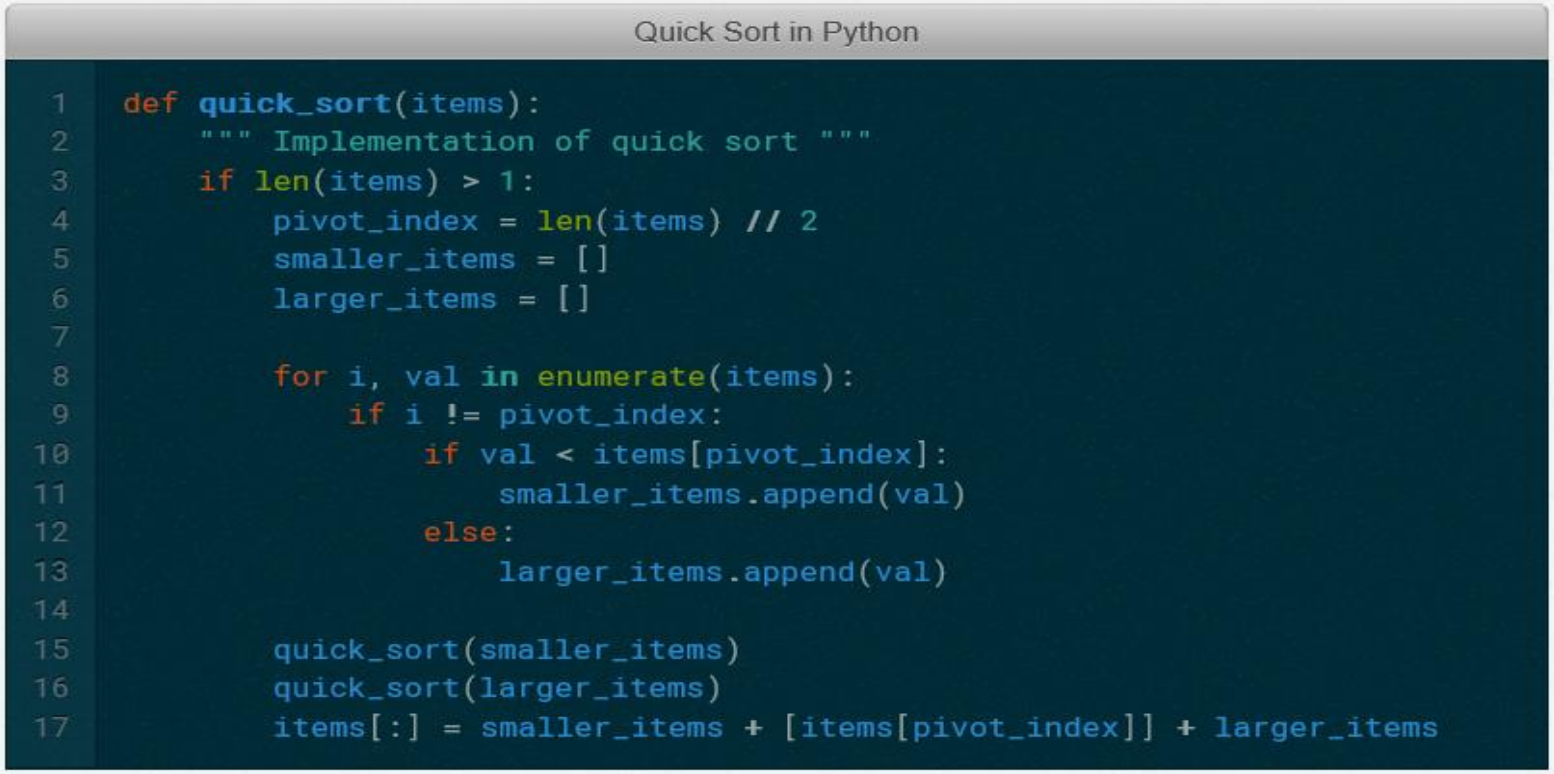


1	**7**	5	42	6	3	15
2						
3	7	**5**	42	6	3	15
4						
5	**5**	7	42	6	3	15
6						
7	5	7	**42**	6	3	15
8						
9	5	7	**42**	6	3	15
10						
11	5	7	42	**6**	3	15
12						
13	5	7	**6**	42	3	15
14						
15	5	**6**	7	42	3	15
16						
17	5	6	7	42	**3**	15
18						
19	**3**	5	6	7	42	15
20						
21	3	5	6	7	42	**15**
22						
23	3	5	6	7	**15**	42

Quick Sort

http://en.wikipedia.org/wiki/Quicksort

Quick sort works by first selecting a pivot element from the list. It then creates two lists, one containing elements less than the pivot and the other containing elements higher than the pivot. It then sorts the two lists and join them with the pivot in between. Just like the Merge sort, when the lists are subdivided to lists of size 1, they are considered as already sorted.



Quick Sort in Python

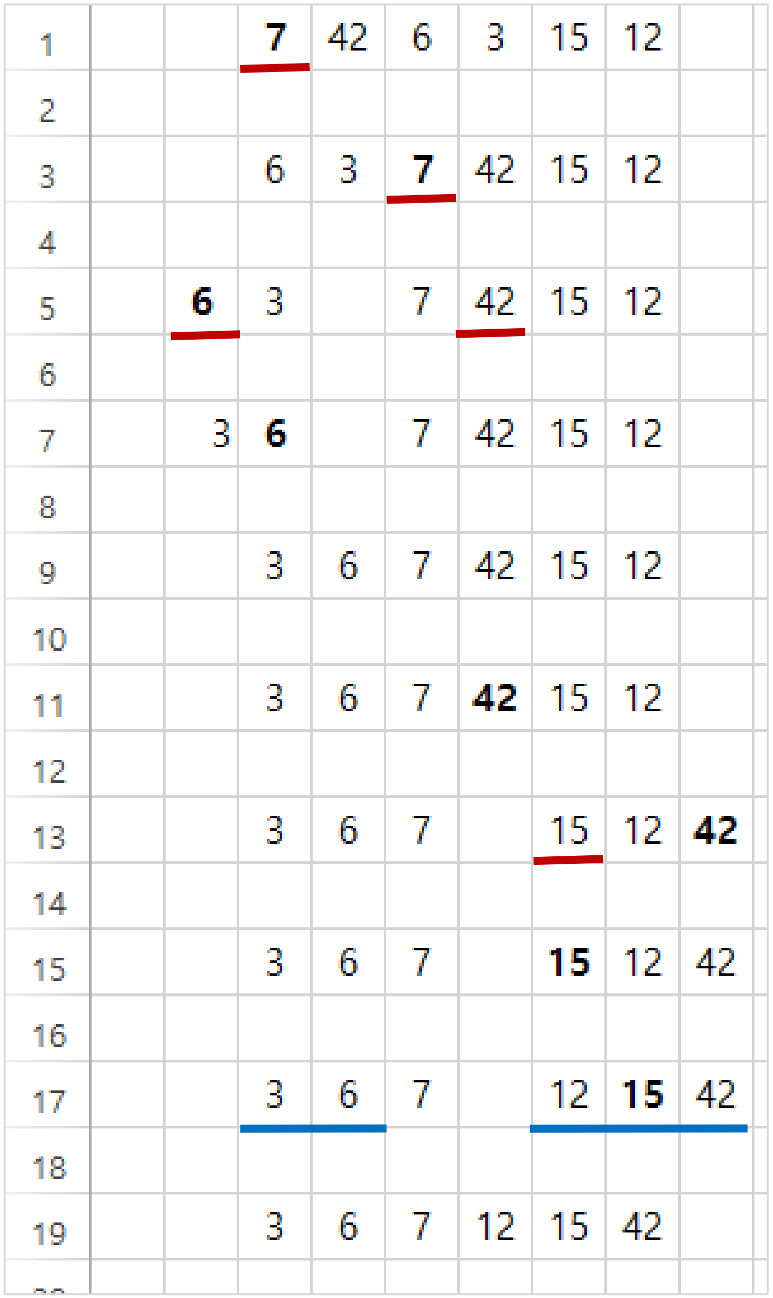
```python
def quick_sort(items):
    """ Implementation of quick sort """
    if len(items) > 1:
        pivot_index = len(items) // 2
        smaller_items = []
        larger_items = []

        for i, val in enumerate(items):
            if i != pivot_index:
                if val < items[pivot_index]:
                    smaller_items.append(val)
                else:
                    larger_items.append(val)

        quick_sort(smaller_items)
        quick_sort(larger_items)
        items[:] = smaller_items + [items[pivot_index]] + larger_items
```

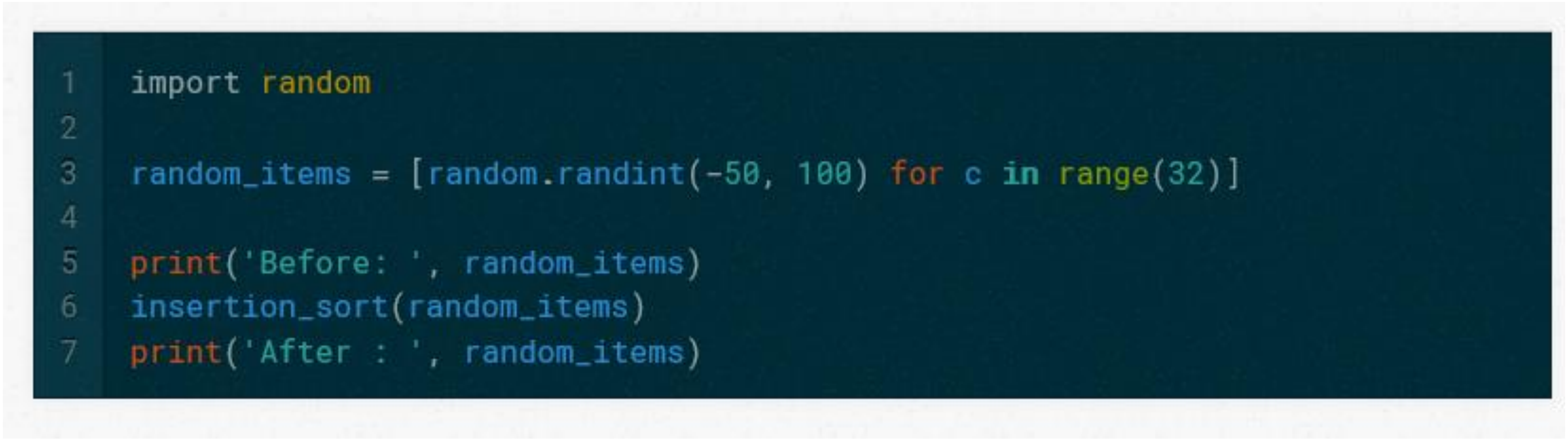
- Step1: Unsorted list의 첫번째 element를 중심으로 전체리스트를 left part와 right part로 재배열
- Step2: 나누어진 left part와 right part에서 step1을 수행

Quick Sort Shot



1			**7**	42	6	3	15	12	
2									
3			6	3	**7**	42	15	12	
4									
5		**6**	3		7	42	15	12	
6									
7		3	**6**		7	42	15	12	
8									
9			3	6	7	42	15	12	
10									
11			3	6	7	**42**	15	12	
12									
13			3	6	7		15	12	**42**
14									
15			3	6	7		**15**	12	42
16									
17			3	6	7		12	**15**	42
18									
19			3	6	7	12	15	42	

Test data and Time Measurement



```
import random

random_items = [random.randint(-50, 100) for c in range(32)]

print('Before: ', random_items)
insertion_sort(random_items)
print('After : ', random_items)
```

import time
startTime = time.clock()
sort_function_x(random_items)
endTime = time.clock()
elapsedTime = endTime – startTime
Print(“The elapsed time for sort_function_x is: “, elapsedTime)

Team Project 4 (Deadline: 5월 2일 낮12시) :

- (1번 100점) Lecture note (12)에 있는 Sorting Algorithm을 비교하는 Mission

- 5개 sorting algorithm들의 작동원리를 10개 data의 입력으로 그림으로 보인다
  - 한 개 algorithm에 ppt 4장 이하로 구성

- 성능평가는 아래 3경우로 한다

  (1) 100개의 Randomly Generated Data
  (2) 1000개의 Randomly Generated Data
  (3) 10000개의 Randomly Generated Data

- Python built-in function인 sorted()도 포함하여 총 6개 algorithm의 수행시간을 time library를 이용하여 측정비교 한다

- 5개 Algorithm에서 수행되는 comparison operation의 개수를 count해본다

- 팀별로 PPT로 만들어서 제출한다
  - PPT는 intuitive, informative, visual하게 만들어지는것이 중요함