Department of Computing

**Lab 12: Sorting & Searching Arrays**

CS-110: Fundamentals of Computer Programming

SE-7B  
Date: 4th January, 2017

Instructor: M. Muddassir Malik

## Introduction

## The purpose of this lab is to familiarize students with the implementation of Bubble sort.

## Tools/Software Requirement

## Python IDLE

## Description

The **bubble sort** makes multiple passes through a list. It compares adjacent items and exchanges those that are out of order. Each pass through the list places the next largest value in its proper place. In essence, each item “bubbles” up to the location where it belongs.

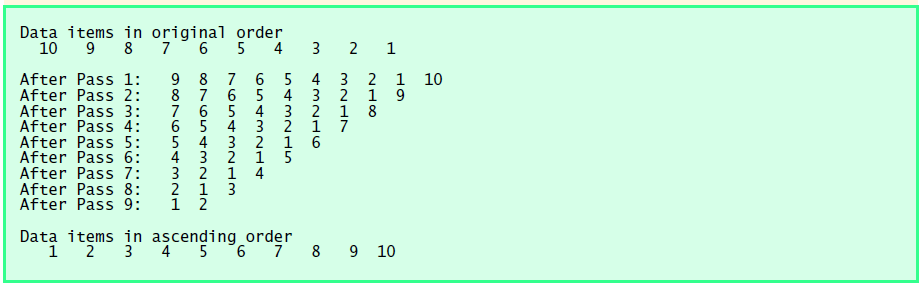
Below shows the first pass of a bubble sort. The shaded items are being compared to see if they are out of order. If there are *n* items in the list, then there are n−1n−1 pairs of items that need to be compared on the first pass. It is important to note that once the largest value in the list is part of a pair, it will continually be moved along until the pass is complete.



At the start of the second pass, the largest value is now in place. There are n−1 items left to sort, meaning that there will be n−2n−2 pairs. Since each pass places the next largest value in place, the total number of passes necessary will be n−1n−1. After completing the n−1n−1 passes, the smallest item must be in the correct position with no further processing required. [ActiveCode 1](https://interactivepython.org/runestone/static/pythonds/SortSearch/TheBubbleSort.html#lst-bubble) shows the complete bubbleSort function. It takes the list as a parameter, and modifies it by exchanging items as necessary.

# Lab Tasks

1. Read 10 random numbers from the user into an array. Sort the data in ascending order using the ***bubble sort algorithm***. Present the sorted data on the screen.
2. The bubble sort algorithm is inefficient for large arrays. Make the following simple modifications to improve the performance of the bubble sort.
3. After the first pass, the largest number is guaranteed to be in the highest-numbered element of the array; after the second pass, the two highest numbers are “in place,” and so on. Instead of making nine comparisons on every pass, modify the bubble sort to make eight comparisons on the second pass, seven on the third pass and so on.
4. Modify the sort to check at the end of each pass if any swaps have been made. If none has been made, then the data must already be in the proper order, so the program should terminate. If swaps have been made, then at least one more pass is needed.



Using **Recursion**, implement the following programs.

1. The factorial function is used frequently in probability problems. The factorial of a positive integer n (written n! and pronounced “n factorial”) is equal to the product of the positive integers from 1 to n.

Write a function **factorial** that accepts an integer as parameter and returns its factorial.

1. Write a function **sum** that accepts an integer and returns the sum from 1 upto that integer.   
     
   