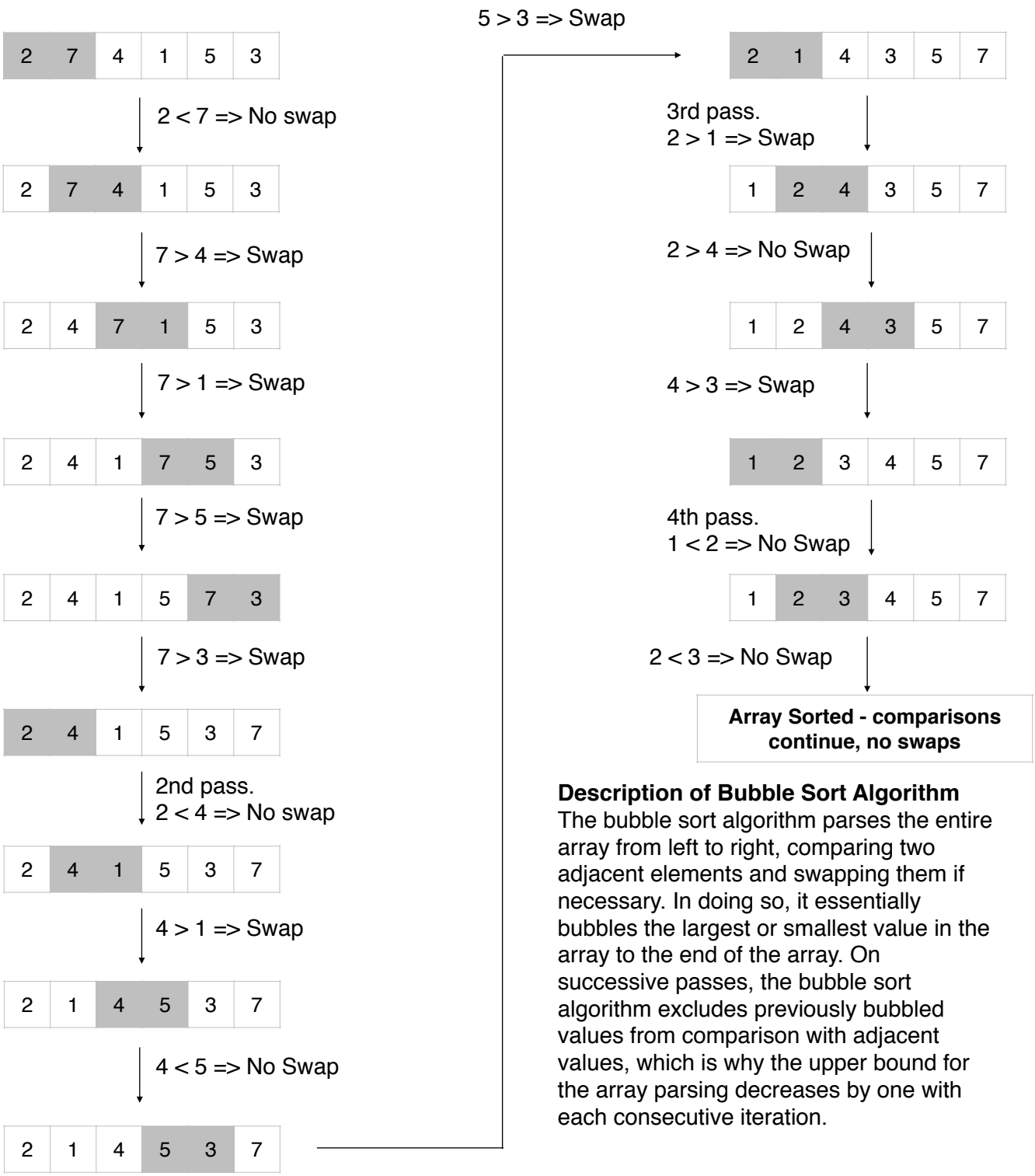
# 04 - Array Searching

## Pre-lab Exercises

## Exercise 1

**Describe how the Bubble Sort algorithm sorts an integer array of 6 elements. Give illustrations for each step of sorting.**




**Description of Bubble Sort Algorithm**

The bubble sort algorithm parses the entire array from left to right, comparing two adjacent elements and swapping them if necessary. In doing so, it essentially bubbles the largest or smallest value in the array to the end of the array. On successive passes, the bubble sort algorithm excludes previously bubbled values from comparison with adjacent values, which is why the upper bound for the array parsing decreases by one with each consecutive iteration.

# Exercise 2

**Take help from post-lab exercise of Lab 01 and write prototypes and definitions of a Bubble Sort function overloaded for int, double, and char types. You can reduce your work by utilising 'function template' method of defining functions, but this only optional.**

```
#include <iostream>
using namespace std;

template <typename T>
void printArray(T* myArray, int arraySize);

template <typename T>
void bubbleSort(T* myArray, int arraySize);

template <typename T>
void test(T* myArray, int arraySize);

//MAIN
int main()
{
    //CREATING THREE DIFFERENT TYPES OF ARRAYS
    int     intArray[] = {1, 10, 8, 7, 14, 16, 2, 5, 4, 9};
    char    charArray[] = {'a', 'l', 'g', 'o', 'r', 'i', 't', 'h', 'm',
's'};
    double  doubleArray[] = {99.8, 1.0, -2.39, 4.555, 1245.2, 88.9,
1.11, 2.8};

    test(intArray, sizeof(intArray)/sizeof(intArray[0]));
    test(doubleArray, sizeof(doubleArray)/sizeof(doubleArray[0]));
    test(charArray, sizeof(charArray)/sizeof(charArray[0]));

    cout << "END OF PROGRAMME" << endl;
    return 0;
}

//FUNCTION TEMPLATE DEFINITIONS
template <typename T>
void printArray(T* myArray, int arraySize)
{
    for (int i = 0; i < arraySize; i++)
        cout << myArray[i] << " ";
    cout << endl;
}//end printArray
```

```
template <typename T>
void bubbleSort(T* myArray, int arraySize)
{
    int i, j;
    for (i = 0; i < arraySize - 1; i++)
        for (j = 0; j < arraySize - i - 1; j++)
            if (myArray[j] > myArray[j + 1])
            {
                T temp = myArray[j];
                myArray[j] = myArray[j+1];
                myArray[j + 1] = temp;
            }//end swap
         //end inner for loop
    //end outer for loop
}//end bubbleSort


template <typename T>
void test(T* myArray, int arraySize)
{
    cout << "Before sorting, array is: " << endl;
    printArray(myArray, arraySize);
    cout << "After sorting, array is: " << endl;
    bubbleSort(myArray, arraySize);
    printArray(myArray, arraySize);
    cout << endl << endl;
}//end test
```

Output

```
Before sorting, array is:
1 10 8 7 14 16 2 5 4 9
After sorting, array is:
1 2 4 5 7 8 9 10 14 16


Before sorting, array is:
99.8 1 -2.39 4.555 1245.2 88.9 1.11 2.8
After sorting, array is:
-2.39 1 1.11 2.8 4.555 88.9 99.8 1245.2


Before sorting, array is:
a l g o r i t h m s
After sorting, array is:
a g h i l m o r s t


END OF PROGRAMME
Program ended with exit code: 0
```

# Exercise 3

See the C++ reference for using the the following <ctime> library functions: `time()`, `srand()`, and `rand()`. Show examples of using each of them in a single program. Also, analyse the given lines and describe their functionality.

```
#include<iostream>
#include<ctime>
#include<cstdlib>
using namespace std;

int main()
{
    int LIM=10;
    float num[LIM];
    srand(time(NULL));      //seed random number generator

    time_t start = time(0); //time at start of programme

    //EVEN NUMBERS
    cout << "Automatic initialization with even numbers" << endl;
    for(int i=0;i<LIM;i++)
        num[i]=(i+1)*2;
    cout << "Numbers generated. \nPress any key to continue";
    getchar();
    for(int i=0;i<LIM;i++)
        cout<<"\nElement no."<<i<<"="<<num[i];
    cout<<"\nPress any key to continue";
    getchar();

    cout<<endl<<endl;

    //RANDOM NUMBERS
    cout << "Automatic initialization with random numbers." << endl;
    for(int i=0;i<LIM;i++)
        num[i]=(rand()%100)+1;
    cout<<"Numbers generated.\nPress any key to continue";
    getchar();
    for(int i=0;i<LIM;i++)
        cout<<"\nElement no."<<i<<"="<<num[i];
    cout<<endl<<endl;

    time_t end = time(0);                   //time at end of programme
    cout << "Total time taken by programme:\t "
        << end - start << "s" << endl;    //total time taken by
programme

    cout << "END OF PROGRAMME" << endl;
    return 0;
}
```

## Program Output

```
Automatic initialization with even numbers
Numbers generated.
Press any key to continue
Element no.0=2
Element no.1=4
Element no.2=6
Element no.3=8
Element no.4=10
Element no.5=12
Element no.6=14
Element no.7=16
Element no.8=18
Element no.9=20
Press any key to continue

Automatic initialization with random numbers.
Numbers generated.
Press any key to continue
Element no.0=97
Element no.1=100
Element no.2=45
Element no.3=56
Element no.4=92
Element no.5=52
Element no.6=17
Element no.7=100
Element no.8=38
Element no.9=44

Total time taken by programme:    3s
END OF PROGRAMME
Program ended with exit code: 0
```

## Analysis

`time()`
Gets the current calendar time as a value of type time_t.

The function returns this value, and if the argument is not a null pointer, it also sets this value to the object point by timer.

The value returned generally represents the number of seconds since 00:00 hours, Jan 1, 197- UTC (i.e. the current unix timestamp). However, different libraries may use a different representation of time: portable programmes should not use the value returned by this function directly, but always rely on calls to other elements of the standard library to translate them to portable types such as `localtime`, `gmtime`, or `difftime`.

`rand()`
Returns a pseudo-random integral number in the range between 0 and RAND_MAX.
This number is generated by an algorithm that returns a sequence of apparently non-related numbers each time it is called. This algorithm uses a seed to generate the series, which should be initialized to some distinctive value using function srand.

RAND_MAX is a constant defined in <cstdlib>.

## srand()

The pseudo-random number generator is initialized using the argument passed as *seed*.

For every different *seed* value used in a call to srand, the pseudo-random number generator can be expected to generate a different succession of results in the subsequent calls to rand.

Two different initializations with the same *seed* will generate the same succession of results in subsequent calls to rand.

If *seed* is set to 1, the generator is reinitialized to its initial value and produces the same values as before any call to rand or srand.

In order to generate random-like numbers, srand is usually initialized to some distinctive runtime value, like the value returned by function time (declared in header <ctime>). This is distinctive enough for most trivial randomization needs.