COMP 2611 – Data Structures

Lab 3

Part 1: Recursion on Arrays

Using the code in `ArrayOperations.cpp`, write the following recursive functions:

```
void printArray (int a[], int i, int n);
        // displays the elements of the array on the monitor

bool containsArray (int a[], int i, int n, int key);
        // return true if key is present in the array and false, otherwise

int sumArray (int a[], int i, int n);
        // find the sum of the elements in the array (assume there is at least one)

int maxArray (int a[], int i, int n);
        // find the maximum element in the array (assume there is at least one)
```

In all the function prototypes above, *n* is the amount of elements in the array *a*.

Part 2: Merge Operation on an Array

A typical merge function accepts three arrays as parameters:

```
int mergeArray (int a[], int a_size, int b[], int b_size, int c[]);
```

The first two arrays, *a* and *b*, contain the values to be merged, in sorted order. The third array, *c*, stores the merged values. *a_size* is the amount of elements in *a* and *b_size* is the amount of elements in *b*.

(a) Examine the *mergeArray* function in `MergeArrays.cpp`. Run the program with different values in *a* and *b* (remember, the values must be sorted in both arrays).

(b) What is the purpose of inserting the value of *INT_MAX* in *a*[*a_size*] and *b*[*b_size*]?

```
a[a_size] = INT_MAX;
b[b_size] = INT_MAX;
```

(c) The following is another prototype for the *mergeArray* function:

```
int mergeArray2 (int a[], int p, int q, int r)
```

This version accepts only a single array, *a*, as a parameter. The values in *a* from location *p* to location *q* are in sorted order. The values from location *q+1* to location *r-1* are also in sorted order. However, the values from location *p* to location *r-1* are not sorted.

Write the *mergeArray2* function. Note that you will have to use temporary arrays to hold the values in *a* from locations *p* to *q*, and the values from locations *q*+1 to *r*-1. Use the *mergeArray* function from part(a) as a guide.

Part 3: Storing a Struct in a Linked List (Needed for Assignment #1)

Consider the following declarations for a node in a linked list:

```
struct Skill {
   string name;
   int years;
};

struct Node {
   Skill data;
   Node * next;
};
```

Each node in the linked list stores a struct which has two fields, *name*, and *years*. The *name* is the name of a particular skill and *years* is the amount of years a person has this skill. For example, a skill *name* could be "Project Management" and the *years* could be 5.

(a) Write code for the following functions:

```
Node * createNode (string skill_name, int skill_years);
Node * insertAtHead (Node * top, string skill_name, int skill_years);
bool contains (Node * top, string key);
int size (Node * top);
void printList (Node * top);
```

(b) Use the code in `Skills-Stub.cpp` to test the functions written in (a).