ECE 2574 ♦ Supplementary Notes for Homework 5

1. Merge Sort

Mergesort uses the idea that 2 sorted arrays can be merged into a single sorted array quickly. It divides a given array into 2 sub-arrays, and then recursively calls mergesort() on each sub-array. After this, a merge operation is carried out, where the sub-arrays are compared element by element, and assembled into a single array.

If the length of any of the sub-array is 1, mergesort() simply returns. You can imagine this recursion as a tree—each mergesort() call is represented by a node. Each mergesort() call makes 2 mergesort() sub-calls in turn, and these sub-calls are represented by the children of this node in the tree. Thus, if we consider the bottom-most tier in the recursion tree, we see that 2 sub-arrays of length 1 each are merged into a single sorted array of length 2.

Mergesort() is an example of a "divide-and-conquer" algorithm, where dividing an array into 2 sub-arrays is the "divide" step, and merging the 2 sub-arrays after they are sorted is the "conquer" step. Merge sort is an $O(n \log n)$ sorting algorithm. Merge sort was invented by John von Neumann in 1945. It is still studied as a typical example of a divide-and-conquer algorithm. Most implementations produce a "stable" sort, which means that the implementation preserves the input order of equal elements in the sorted output.

Below, an example header file, `mergesort.h`, for implementing the mergesort algorithm is given. Note that this is *only* an example; your implementation may be quite different from this code.

```
#include <iomanip>
#include <iostream>
#include <cmath>
#include <fstream>

using namespace std;

//struct used to store the input information. Each node contains a character and
// integer and pointer to the next node

struct node
{
	char letter;
	int frequency;
	node * next;
};

//class mergelist that is used to facilitate the mergesort
class mergelist
{

//overloaded << operator used to output the results
friend ofstream& operator<<(ofstream&, const mergelist&);

//private data members
private:
	node * head;		//head pointer, used to signify beginning of list
	node * cur;		//current pointer, used to store current position
	int size;		//size of linked list
	int sortsize;		//variable used to determine the number of times the array
```

```
                                //needs to be subdivided

public:

        //default constructor
        mergelist();

        //function used as an interface by the client code to initiate mergesort
        void sort();

        //recursive method which subdivides the large list into smaller lists, accepts
        //2 pointers as input and returns a pointer
        node * mergesort(node * subhead, node * sublast);

        //recursive method which puts the small lists back together into one list,
        //accepts 2 pointers as input and returns a pointer
        node * merge(node * lower, node * upper);

        //function used to add entries into the linked list
        void addnode(char let, int freq);

        //destructor
        ~mergelist();
};
```

Reference links:

1. http://en.wikipedia.org/wiki/Merge_sort