COSC 220 - Computer Science II
Lab 8

Dr. Joe Anderson

Due: 5 November 2019

1 Objectives

1. Practice using dynamically allocated arrays
2. Practice with random number generation
3. Practice with recursion and sorting algorithms

2 Tasks

1. You will implement the Quicksort algorithm, to sort an array of integers.
2. To implement Quicksort, you will implement three subroutines:
   (a) `void quicksort(int arr[], int start, int end)`
   (b) `int partition(int arr[], int start, int end)`
   (c) `void swap(int &val1, int &val2)`
3. First, the `swap` function has a single task: given two elements (by reference), swap their values in memory.
   (a) Try some test cases to be sure it works:
   ```
   int a = 10;
   int b = 5;
   std::cout << "a: " << a << " - " << "b: " << b << std::endl;
   // Prints "a: 10 - b: 5"
   swap(a,b);
   std::cout << "a: " << a << std::endl << "b: " << b << std::endl;
   // Prints "a: 5 - b: 10"
   ```
   (b) Test it on an array:
   ```
   int arr[] = {10, 5};
   swap(arr[0], arr[1]);
   ```
4. The `partition` function is what will take the most work. Its job is to choose the pivot, and, using only a single loop over the array, ensure that every element smaller than the pivot is on the left, and every element greater than the pivot is on the right. It should return the *index* of the pivot after the re-arranging.

   (a) A simple choice for this can be made by simply choosing the first element of the array as the pivot.
   (b) The function should then scan the array segment, placing everything less than the pivot on the left, and everything greater than the pivot on the right (see one possible algorithm from lecture).

5. Test your partition function on a few short arrays, printing the pivot, its location, and the arrays after calling partition once.
6. Finally, the function Quicksort will follow this general algorithm:
   (a) If `start` >= `end`, do nothing.
   (b) Otherwise, call `partition` to get the location of a pivot point
   (c) Call `quicksort(arr, start, pivot)`
   (d) Call `quicksort(arr, pivot+1, end)`
7. Test your full quicksort algorithm on a couple different hard-coded arrays. Try edge cases: arrays that are already sorted, arrays that are sorted backwards.
8. Try quicksorting larger arrays, with randomly generated values.
   (a) Write a function to dynamically allocate an array of a specified length that also assigns each element to a random integer
   (b) To generate a random integer, one way is to first `#include<stdio.h>` and also `#include<time.h>`
   (c) To seed the random number generator to something different every time, first add `srand(time(NULL))`. Otherwise, it may be the same every time you run your program.
   (d) To generate a random number between 1 and n, use `rand() % n + 1`
      i. `rand()` generates a random double between 0 and 1 in general

3 Submission

Upload your project files to the course canvas system in a single zipped folder: To zip on Linux:

1. To zip a single folder into a single archive called "myZip.zip":

```
zip -r myZip.zip folderName
```

2. To zip multiple files into a zip file called "myZip.zip":

```
zip myZip.zip file1.cpp file2.h file3 file4.cpp
```

Turn in (stapled) printouts of your source code, properly commented and formatted with your name, course number, and complete description of the code.

Also turn in printouts reflecting several different runs of your program (you can copy/past from the terminal output window). Be sure to test different situations, show how the program handles erroneous input and different edge cases.