CSCI 480 Algorithms Design and Analysis

Group Code Task 1
Description

Task: Implement Sorting Algorithms and Performance Analysis

Generate 1000 data points with a random number generator as input for the following sorting tasks:

1. **Implement Heapsort:** Implement the Heapsort algorithm in C++. Heapsort is an in-place sorting algorithm. Ensure that your implementation sorts the data input array (from above) in ascending order. After implementation, analyze and discuss the program's efficiency (computation and space) compared to the theoretical efficiency as discussed in the lecture.

2. **Implement Mergesort:** Implement the Mergesort algorithm in C++. Mergesort is known for its stability in sorting. Ensure that your implementation sorts the data input array (from above) in ascending order. Analyze and discuss your program's efficiency (computation and space) compared to the theoretical efficiency. Explore and discuss the stability of your Mergesort implementation, emphasizing its importance in sorting algorithms.

3. **Implement a Priority Queue using an Ordered Array:** Implement a priority queue using an ordered array in C++. The priority queue should have a fixed capacity and support the following operations:

   - `insert(int value)`: Adds an item to the end of the array.
   - `deleteMax()`: Searches through the entire array to find the maximum value, swaps it with the item at the end of the array before removing it, similar to swapping in sorting algorithms. After implementation, analyze and discuss your program's efficiency (computation) compared to the efficiency of your Heapsort from task 1.

4. **Implement Radix String Sort:** Implement Radix String Sort in C++. This sorting algorithm should work on an array of suffixes from an input string. The input string should be read from a text file provided as

a command-line argument. Suffixes are substrings that start at various positions within the original string. The first suffix is the entire string, and subsequent ones begin at positions one less than the previous suffix, gradually reducing in size. Ensure that your implementation sorts the suffixes lexicographically.

5. **Performance Analysis and Experimentation:** Create a singular driver program that conducts extensive experiments on the implemented sorting algorithms. Generate performance data by running the sorting algorithms on various input sizes and distributions. Gather performance metrics such as execution time, space usage, and stability. Write these performance data out in a coherent format to an output file. The input file name for the Radix String Sort should be provided as a command-line argument, as well as the output file name.

By completing these tasks and the performance analysis, you will not only demonstrate your proficiency in various sorting algorithms but also your ability to analyze their efficiency and gather experimental data for evaluation.

Submit 1 zipped archive containing all your files including report, please make sure you include a clang++ based makefile

No blacklisted libraries but I do require you implement the algorithms specified yourselves.

Due date: October 8th 23:99