Lab 8 - Sort

Due Date: 5:00 p.m., April 27, 2016
All public interfaces are minimum requirements. You may add additional public methods if needed as long as you follow the lab and style guidelines.

For Lab 8 we are going to implement several sorting algorithms on different kinds of data sets data sets.

Part A: Sorts

For Part A you are going to implement a Sort class that has 2 different sorts:

- class Sort
    - Public Methods
        - int* sortA(int * a, int size)
            - Should redirect to one of your chosen sorts
        - int* sortB(int * b, int size)
    - Private Methods (**choose two**)
        - *NOTE: because the below methods are private, you may alter their interface as needed. These are just suggestions.*
        - int* heapsort(int* a, int size)
            - Takes an array of ints and returns the sorted version of the array using the heapsort algorithm to sort
        - int* mergesort(int* a, int size)
            - Takes an array of ints and returns the sorted version of the array using the merge algorithm to sort
        - int* quicksort(int* array, int high, int low)
            - Takes an array of ints and returns the sorted version of the array using the quicksort algorithm to sort

Part B: Choose your Sort (Extra Credit)

Choose the 3rd sort (the one you didn't choose) and optimize the hell out of it.
Add the following to your Sort class:

- int * sortOptimized(int * a, int size)

Make it run as fast as you can by limiting memory usage, reducing operations, etc. I will run it against additional data sets (given in the driver code below) and take the average runtime.

You will get 5 points extra credit (half a lab) for implementing the third sort, regardless of optimization.

If you choose to compete for the fastest sort, you can use any optimization options you can find, including compile time optimizations. The top 3 in the class will get an additional 10 points (1 lab) extra credit.

Part C : Code Organization and Submission

- Required code organization:
  - lab8.cpp
  - Sort.h/.cpp
  - Makefile
    - **Your makefile must include c++11 extensions**
  - readme
- While inside your program2 folder, create a zip archive with the following command
  - zip -r lab8 *
    - This creates an archive of all file and folders in the current directory called lab8.zip
    - **Do not zip the folder itself, only the files required for the lab**
- Upload the archive to Mimir under 'lab8'

Tests

- Test Driver
  - lab8.cpp

Grading Guidelines

- **Part A**
  - Passes tests 1-9
    - 1 point for each test
- **Part B**
  - Passes Test 10 - 5 points extra credit for implementing third sort
  - 10 extra credit points for fastest run
- **Part C:**
  - Follows formatting guidelines, requested project structure and naming conventions, and submission does not include .o files or binary (1 point)

Formatting Guidelines

- Stores all values in a named variable.

    - No Magic Numbers.
- Uses indentation to identify code blocks.
    - Every Code block should be indented from it's parent block to identify scope.
- No single letter or non-descriptive variable names
    - The only exception to this rule is 'i' in a for loop
- Separates code blocks and logical sections with whitespace
    - Optimize your code for the reader, not the writer
- Output is formatted with an explanation of the output values
    - Format your output so that someone who does not know what the program is supposed to do would know what the output meant
- Each method is preceded by a comment explaining what the method does
- Each significant code block is preceded by a comment explaining what the code block does.
    - A significant code block is more than 3 lines performing a single logical operation
- CONSTANTS are in all caps
- Only data types start with a capital letter
    - Classes, Enums, Structs, etc.
- Do not use the 'using namespace' declaration in a header (.h) file
- In general we will follow the Google C++ style guidelines. If you want more info, you can view them here: https://google.github.io/styleguide/cppguide.html