Sorting

This assignment consists of:

1. Implementation of a framework for measuring sorting algorithms
2. Benchmarking the performance of sorting algorithms

(a) Implement classes `InsertionSort`, `QuickSort`, and `HeapSort`. Define an abstract base class, `Sort` that has the following members:

- `insertAllFromFile(char* fileName, int numItemstoLoad)`
  - Should be implemented
- `void print(ostream& out)`
  - Should be implemented
- `void sort()`
  - Should NOT be implemented-pure `virtual`
- constructor
  - Allocates an array data member
  - Constructor should take a parameter for maximum capacity of array
- destructor
  - Deallocates the array data member
- Derive three classes from class `Sort`
  - `InsertionSort`
  - `QuickSort` use median of three to select pivot and switch to selection sort when array slices are less than 16 items.
  - `HeapSort`
- Please indicate the big-oh values for the following functions in the comments
  - `Sort::insertAllFromFile`
  - `void InsertionSort::sort()`
  - `void QuickSort::sort()`
  - `void HeapSort::sort()`

(b) Benchmark the different sorting algorithms

- Similar to the previous two homeworks, test your sort implementations using the files `random.txt` and `words.txt`.
  - You may reuse your code from previous homeworks if helpful
  - To test, load the array using `insertAllFromFile`, which just puts each word at the end of the array list, then sort the words by calling the `sort` method. For these tests, make your array size 50,000.
  - You can do this by passing in a counter, `N`, to `insertAllFromFile` which indicates how many words to insert and then sort. That function will insert and then sort only the first `N` words from the specified file. Where you call `XAllWords` (where `XAllWords` you stands for `insertAllWords`, `findAllWords`, and `removeAllWords`) will place it in a loop with the loop control variable ranging from 1 to 10 and you will set this `N` parameter by multiplying the loop control variable times 1/10th of the large `N` which is 45,293. 1/10th of this big N is 4,529. Here is a sketch of how each will work:

Psuedocode:

- For each file random.txt and words.txt:
  - For each partition consisting of 1/10th, 2/10,...up to the full 10/10 file:
  - Measure and report the time for: `insertAllFromFile`

```
void Sort::insertAllFromFile(int partition, char *fileName){
  //open the file (previously, timer started here, - you can call it after insert too
  //insert the first partition*N/10 words into self
  //(optional: start timer here instead of before insert - will accept either)
  //call sort
  //stop the timer, close the file, report the time by printing to console/cout
}
void measureAll(char *fileName){
  for (int i=1; i<=10; ++i){
    InsertionSorter T1 = _______; // Start with an empty Sorter subclass here
    QuickSorter T2 = _______; // Start with an empty Sorter subclass here
    HeapSorter T3 = _______; // Start with an empty Sorter subclass here

    cout << "Time␣for␣InsertionSorter:␣";
    T1.insertAllFromFile(i, fileName);
    cout << "Time␣for␣QuickSorter:␣";
    T2.insertAllFromFile(i, fileName);
    cout << "Time␣for␣HeapSorter:␣";
    T3.insertAllFromFile(i, fileName);
  }
}
```

- For your report submit a script file with the valgrind output included as well as without valgrind.
- Include console output for each partition of the file and summarize it in your text file
  - random.txt partition 1/10. Sort: InsertionSort Time: 50s
  - random.txt partition 1/10. Sort: QuickSort Time: 60s
  - random.txt partition 1/10. Sort: HeapSort Time: 40s
  - random.txt partition 2/10. Sort: InsertionSort Time: 50s
  - ...

Make sure you submit the following files:

- `hw06_v.scr`
- `hw06.scr`
- `hw06.txt`
- `hw06.cpp`
- `hw06_f.cpp`
- `hw06.h`