Vincent Finn

Benchmarking

1. A short (one paragraph) explanation of why the benchmark harness is structured as it is. Why is there an initial run that isn't counted in the min/max/average? Why are we measuring multiple runs?

The benchmarks are structured this way with goal being to provide a decent estimate of how long the program takes to run. There is an Initial run called warmup, and this appears to always take longer than the max of the 10 that are run afterward. This probably means that there are some hardware aspects that take longer the very FIRST time the program run. Its like how a website takes a bit longer the first time, and subsequent visits have parts stored in the client's cache, so it is faster. It measures multiple runs in order to see the min, max, and average which gives you a better idea of the extremes the program might run at, as well as the normal pace.

2. Reasonable operational profiles for use of the RedBlackBST in the following use cases. Each of these profiles, for our purposes, is simply a percentage of insert/delete/lookup operations (3 numbers that sum to 100 in each case).
    1. A logging data structure, which mostly records new data, with occasional queries and edits (deletions)

       80% insert, 10% read, 10% delete.

    2. A read-heavy database, with occasional updates and deletions.

       80% read, 10% insert, 10% delete.

3. A description of what steps you took in preparation for your performance measurements. For example, what other software is or isn't running on your machine? (For this assignment, you may ignore our discussion of frequency scaling's effect on performance tests, since disabling it is quite inconvenient.)

I didn't do anything special before testing the performance, I had a few small programs open like word and chrome, but there wouldn't have been anything siphoning a lot from the cpu while it was running.

4. For each of your operational profiles, report the following, which should be produced by Benchmark.main() once you update the hard-coded operational profile.

1. Warmup time
    1. InsertHeavy: 156338740ns
    2. ReadHeavy: 101348356ns
2. Minimum iteration time
    1. InsertHeavy: 101132393ns
    2. ReadHeavy: 60340089ns
3. Maximum iteration time
    1. InsertHeavy: 113432973ns
    2. ReadHeavy: 69021012ns
4. Average iteration time
    1. InsertHeavy: 104574222ns
    2. ReadHeavy: 63171639ns

5. Which of your operational profiles has higher *throughput*? What about the red-black tree might explain this outcome?

Red black tree makes itself optimal for ready operations. Insert heavy programs have higher throughput because first has to perform a similar operation to read, and then may have to modify the tree to keep the correct structure of red black leaves.

6. Are your warmup times noticeably different from the "main" iteration times? Most likely yes, but either way, why might we *expect* a significant difference?

Warmup times are higher than the main iteration times. We might expect a difference due to the hardware needing to warmup before the main iteration starts to run.

7. When I run the test for either of my operational profiles with instrumentation-based profiling enabled, my measurements slow down by a factor of 10 or more. Why?

The try to time each time individually, those lines add up and make the program take significantly longer, especially if it is run hundreds of thousands of times.

8. Assume each run (each call to runOps) simulates the activity of one remote request. Based on the average execution time for each operational profile, what would the throughput be for each of your profiles? (i.e., requests/second)

Insert Heavy: 1 Requests / 104574222ns. About 9.56 Requests / second, or 1 Request / 0.105s

Read Heavy: 1 Request / 101348356ns. About 9.86 Requests / second, or 1 Request / 0.101s

9. Assuming each remote user makes 5 requests/minute, your program's resource usage scales linearly, and we are only interested in CPU execution time, how many concurrent remote users could you support on your machine (again, do this for each operational profile) without degrading performance or overloading the system?

I will round both to about 10 Request / second, or 600 Requests / minute. So that means we can handle about 120 concurrent users.

10. What aspects of load are we not testing, which could possibly reduce the capacity of your machine to service requests?

If remote access is still part of the project, then we aren't currently testing the time it takes to in between remote requests. Otherwise, the other thing that is not being tested is something like 100% insert to see what happens when the disk fills up maybe;