Ans1) The given benchmark harness is based on the instrumental-based profiling of the program where the harness is testing the Red-Black tree program based on the operational profile given. By default, the harness has read a heavy operational profile. The harness is structured like this because it does not miss short calls, in other words, it would count each and every branch statements and external calls. Ultimately, the benchmarks would include the OS functions call, object instantiation, branch statement, etc.

The initial run is the warm-up run done to pre-filling the disk, CPU, OS, and language-specific caches. This is also done to replicate a real-world running environment where users already have their caches warmed up. Thus, The benchmark produced after this warmup run would give us better performance measures. That's why this initial run is not counted in the subsequent max/min/average runs.

We are measuring over multiple runs to get some room for calculating the average for the operations. There is also a factor of randomly getting lucky or unlucky if testing over a single run, giving us un-realistic benchmarks. The purpose of the testing is to get repeatable, meaningful performance measurements, and these multiple runs help in achieving that purpose. Also, multiple measures replicate situations where caches are half-filled, dumped, and requested more space, etc. This overall helps in making our testing environment more similar to a real-world user environment.

Ans2) I have created three profiles matching our operational profiles in the benchmark code.

Profile 1: A logging data structure, which mostly records new data, with occasional queries and edits (deletions)

Reads: 10
Inserts: 80
Deletes: 10

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

Reads: 80
Inserts:10
Deletes: 10

Profile 3: A read-only database, with no modifications performed.

Reads: 100
Inserts: 0
Deletes: 0

Ans3) The following are the steps taken for the performance measurements:

1) Only the benchmark tests are running on the machine, no other software is running making the tests as much isolated as possible.
2) The number of operations performed is big enough to ensure each run is long enough to trigger possible issues.
3) Measurements are done following instrumental-based profiling so as to account for external call calculations giving more reliable results.

4) The machine is connected to the regular home Wi-fi network to replicate real-world operation.

Ans4)

```
Profile 1: A logging data structure, which mostly records new data, with occasional queries and edits (deletions)
Warmup round:      218071200ns
```

```
Average for 1000000 operations:     93697110ns
Minimum for 1000000 operations:     89833200ns
Maximum for 1000000 operations:     99198800ns

Profile 2: A read-heavy database, with occasional updates and deletions.
Warmup round:        61754100ns
```

```
Average for 1000000 operations:     56266780ns
Minimum for 1000000 operations:     54543900ns
Maximum for 1000000 operations:     58280000ns

Profile 3: A read-only database, with no modifications performed.
Warmup round:        10736900ns
```

```
Average for 1000000 operations:      9975800ns
Minimum for 1000000 operations:      9279300ns
Maximum for 1000000 operations:     12869300ns
```

Ans5) Profile 1 Throughput (Avg operations/sec) = 10,672,688 operations/sec
Profile 2 Throughput (Avg operations/sec) = 17,772,476 operations/sec
Profile 3 Throughput (Avg operations/sec) = 100,242,587 operations/sec

As we can see from the above screenshots the Profile 3(read-only) takes significantly lesser time (min, max, and avg) to do 1000000 operations than Profile 1(insert-heavy) and Profile 2(read-heavy). Thus, Profile 3 has higher throughput. Although, Profile 3 has technically a higher throughput, but Profile 3 does not represent replicate the real-world usage. Profile 1 and Profile 2 better represent actual real-world throughput so out of them, Profile 2 has higher throughput.

This tells us that the red-black tree program takes less time to do a read operation than an insert or delete operation. Generally, in a balanced red-black tree, read takes about O(log N) time, whereas delete and insert are more complicated as the

tree might need to be balanced or maintain red-black properties after any insert or delete operation. Our benchmark results also support this red-black functional property.

Ans6) Yes, the warmup times are significantly different from the benchmark runs. From the above screenshots, warmup times are generally bigger than that of benchmark runs. This is because during the warmup runs, the machine's CPU, OS, language-specific caches are cold so the program has to generate everything from scratch. With the caches warmed up, the program has less chance to re-generate objects, CPU, OS, and language-specific functions, unless the program becomes big enough to make the existing cache dump and generates a bigger one. Also for Java applications, warmup times are needed for various optimizations done by the Java compiler. These optimizations are later used in the subsequent runs to improve the performance of the runs. These results in noticeably bigger warmup iteration times than that of the subsequent runs.

Ans7) The operational profile (c read-only) tells us nothing about the performance of using the BST in a real application for a read-only workload. This is because, in the case of this operational profile, we are using 100 for reads, 0 for inserts and deletes. To read anything from BST, BST has to be pre-filled with some values to replicate the real-world application. In our case, the tree always remains empty for this operational profile resulting in repeatedly the same execution calls. Thus, get() always return null in this operational profile. In the real world, read operation represents users reading from a filled BST, but operational profile (c read-only) only reads from empty BST. That's why benchmarks produced from this operational profile aren't actually useful.

Ans8) Profile 1 Throughput (Avg requests/sec) = 1.06726878e+1 requests/sec
Profile 2 Throughput (Avg requests/sec) = 1.77724761e+1 requests/sec
Profile 3 Throughput (Avg requests/sec) = 1.00242587e+2 requests/sec

Ans9) The Formula for calculating # of Concurrent users = (Avg requests/min) / (users requests/min)

Profile 1 # of Concurrent users = 1.06726878e+1 * (60/5) ~= 128 users
Profile 2 # of Concurrent users = 1.77724761e+1 * (60/5) ~= 213 users
Profile 3 # of Concurrent users = 1.00242587e+2 * (60/5) ~= 1203 users

Ans10) The following are the aspects of load we are not testing that could possibly reduce the capacity of your machine to service requests:

1) We have not tested network bandwidth which would majorly impact the performance of send/receive users requests.
2) We have not tested the memory usage of different Red-black tree operations, memory usage has a huge impact on the program to service requests.
3) We have not tested the number of concurrent user requests supported without degrading performance or overloading the system.

4) We have not tested the average response latency times of different Red-black tree operations.
5) We have not tested for memory leaks which can give users unexpected results reducing the capacity of the machine to service requests.
6) We have not tested the performance of the operations when the machine is under-load from different processes.