Assignment 3: Searching a Set - Analysis Document

Muyuan Zhang

2022-11-16

1. If you had backed the sorted set with a Java List instead of a basic array, summarize the main points in which your implementation would have differed. Do you expect that using a Java List would have more or less efficient and why? (Consider efficiency both in running time and in program development time.)

- Running time: less efficient. Merge sort is used in the `Collections.sort()` method with O(n*log(n)) efficiency.
- Program development time: more efficient. For Java lists, we can utilize built-in methods to avoid overloading multiple methods.

2. What do you expect the Big-O behavior of BinarySearchSet's contains method to be and why?

- O(logN) as a binary search.
- Inside `contains()`, we have:

```
int low = 0;
int high = size_ - 1;

while (low <= high) {                              // O(logN)
    int mid = (high + low) / 2;                    // O(1)

    if (compare(element, set_[mid]) == 0) {        // O(1)
        // if the input element is found
        return true;
    }
    else if (compare(element, set_[mid]) > 0) {    // O(1)
        low = mid + 1;                             // O(1)
    }
    else {
        high = mid - 1;                            // O(1)
    }
}
```

3. Plot the running time of BinarySearchSet's contains method, using the timing techniques demonstrated in previous labs. Be sure to use a decent iteration count to get a reasonable average of running times. Include your plot in your analysis document. Does the growth rate of these running times match the Big-oh behavior you predicted in question 2?

- Run the program and record the average time for `contains()` on 16 set sizes: 2^{10}, 2^{11}, 2^{12}... 2^{24}, 2^{25}.

- Time `contains()` several thousand times for each set size, and take the average.
- Plot these times against the set size.

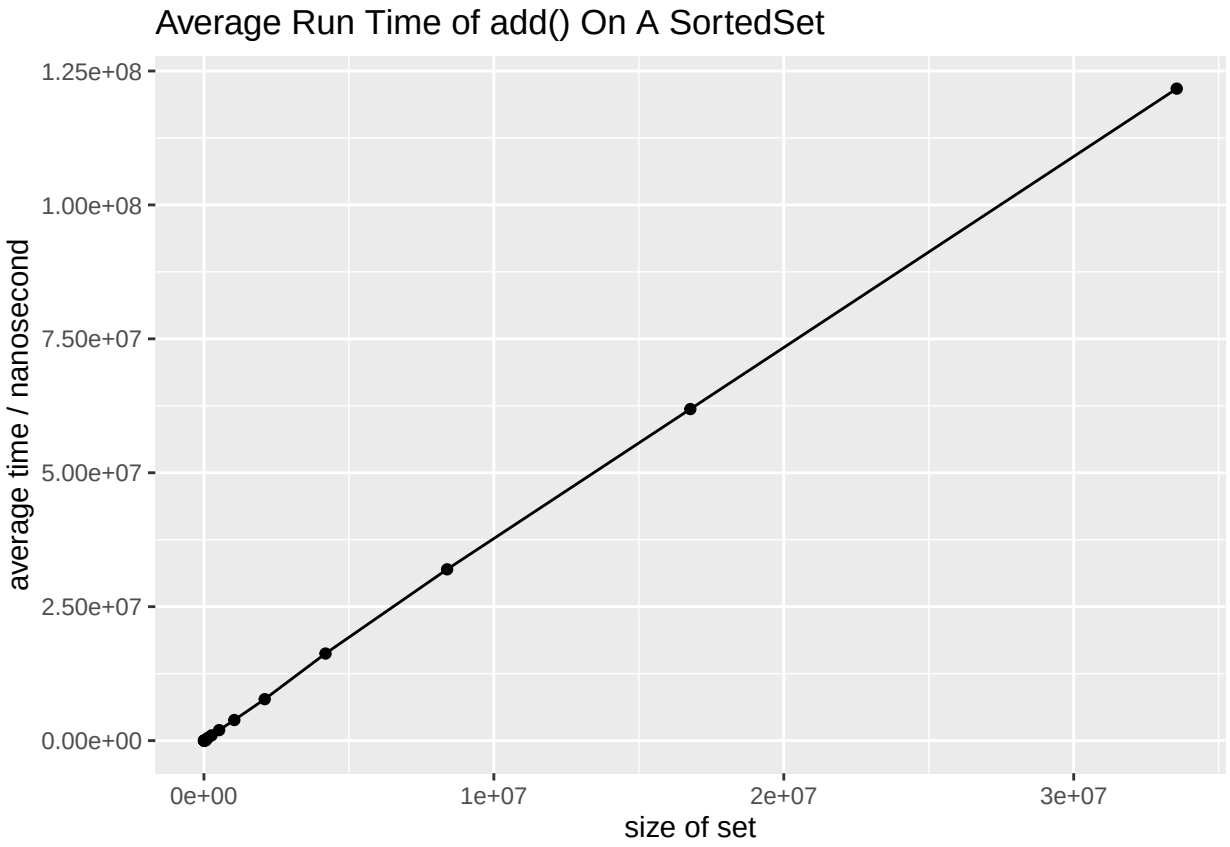
```
library(ggplot2)
size    <- c(1024, 2048, 4096, 8192, 16384, 32768, 65536, 131072, 262144,
             524288, 1048576, 2097152, 4194304, 8388608, 16777216, 33554432)
avgtime <- c(581.199, 168.935, 252.768, 197.701, 214.714, 242.99, 284.06,
             322.782, 200.279, 254.134, 449.349, 667.026, 772.696, 1032.701,
             1310.385, 1486.285)
timeingdata <- data.frame(size, avgtime)
ggplot(timeingdata, aes(size, avgtime)) + geom_line() + geom_point() + labs(
    x = "size of set",
    y = "average time / nanosecond",
    title = "Average Run Time of contains() On A SortedSet")
```

- Yes, the growth rate of running times matches O(logN).

4. Consider your add method. For an element not already contained in the set, how long does it take to locate the correct position at which to insert the element? Create a plot of running times. Pay close attention to the problem size for which you are collecting running times. Beware that if you simply add N items, the size of the sorted set is always changing. A good strategy is to fill a sorted set with N items and time how long it takes to add one additional item. To do this repeatedly (i.e., iteration count), remove the item and add it again, being careful not to include the time required to call remove() in your total. In the worst-case, how much time does it take to locate the position to add an element (give your answer using Big-oh)?

- For an element not contained in the set, it takes O(logN) to locate the insertion position.
- Run the program and record the average time for `add()` on 16 set sizes: 2^{10}, 2^{11}, 2^{12}... 2^{24}, 2^{25}.
- Time `add()` several thousand times for each set size, and take the average.
- Plot these times against the set size.

```
library(ggplot2)
size    <- c(1024, 2048, 4096, 8192, 16384, 32768, 65536, 131072, 262144,
             524288, 1048576, 2097152, 4194304, 8388608, 16777216, 33554432)
avgtime <- c(945.341, 1437.88, 2880.14, 6963.556, 11497.507, 22184.915,
             53365.695, 501644.539, 974700.702, 1961170.104, 3845698.366,
             7738428.494, 1.6254918414E7, 3.1975626294E7, 6.188944037E7,
             1.21704656138E8)
timeingdata <- data.frame(size, avgtime)
ggplot(timeingdata, aes(size, avgtime)) + geom_line() + geom_point() + labs(
    x = "size of set",
    y = "average time / nanosecond",
    title = "Average Run Time of add() On A SortedSet")
```



- In the worst-case, it takes O(logN) to locate the position and O(N) to add it.