EECS 560: Lab 7 – Comparing the performance of Min-Max-Heap and Min 5-Heap

Shane Chu

November 1, 2016

1 Overall organization of the experiment

1.1 Code arrangement

To run the experiment, we simply construct an integer array with the number of generations we specified, which is the set of integers $\{50000, 100000, 200000, 400000\}$, hard coded inside the array. Then we construct a nested for-loop to record the time for Min-Max-Heap and Min-5-Heap both on its build time and operation time.

Each element inserted into the structure is generated by a random integer between 1 and $4 \times n$, where n is the number of generations. To build the structure, Min-Max-Heap and Heap each inserts n number of elements. Note that Min-Max-Heap and Heap both use the *Heapify* method to build its structure.

1.2 Data

The time-measured data on both data structure is arranged to be output as a text file, which is taken care by using fstream library. That way, we could simplify the process of visualizing the data.

1.3 Run the program

Since a makefile is made, compile under the folder using:

```
make
```

Then run the executable lab7 to generate the data.

2 Data Generation

After running lab7, a text file result.txt is generated in the folder.

We process the output of the text file using programming language *python* and its library *matplotlib*. Further, we use *iPython notebook* (jupyter) so we could code and plot the result at the same time. The whole process is documented in parse-result.ipynb.

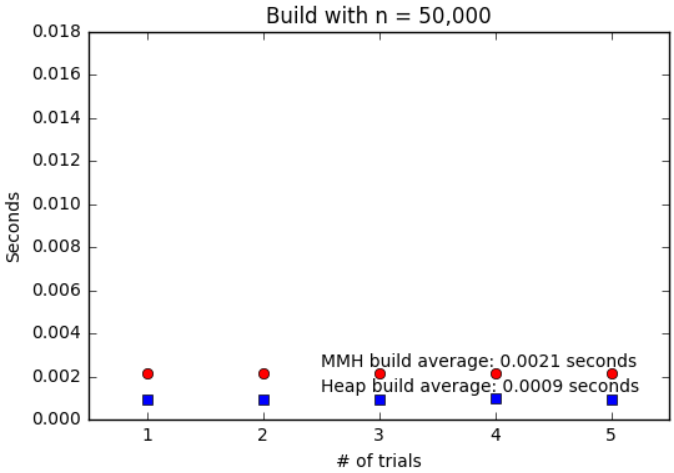
3 Results

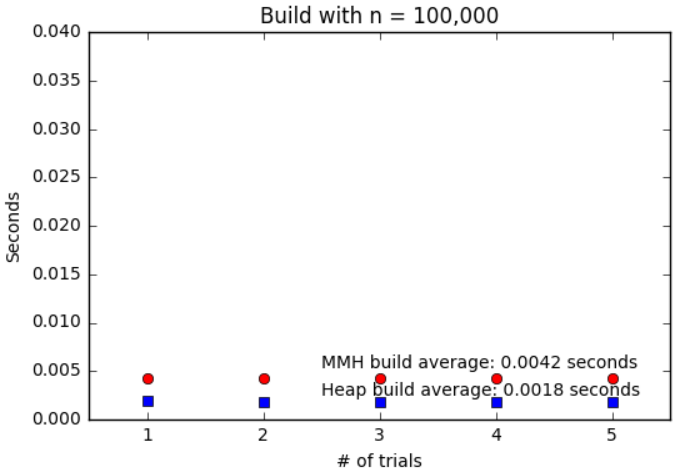
3.1 Data

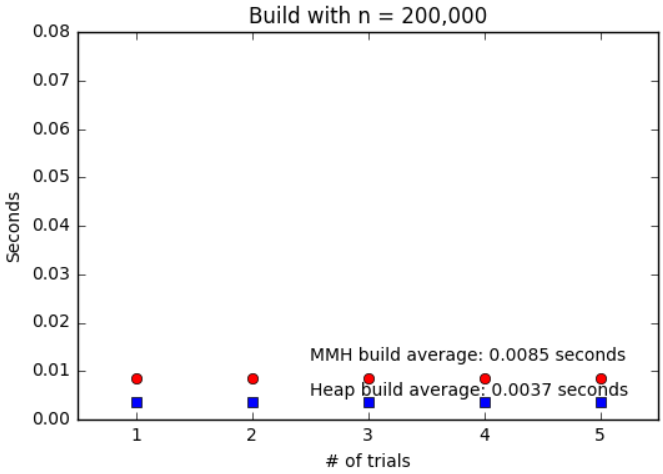
Please refer to the text file result.txt.

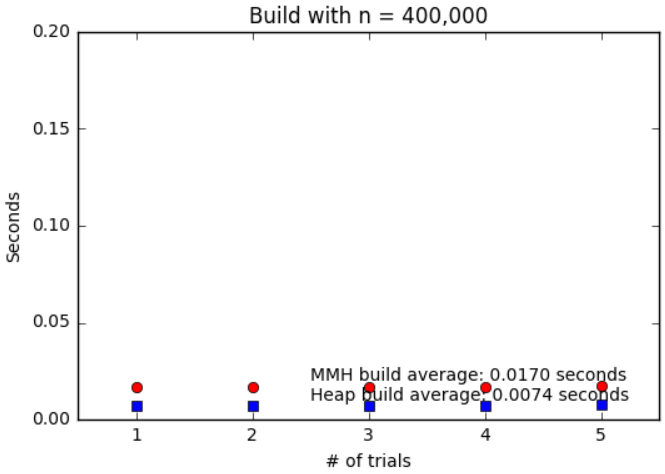
3.2 Build

Below are the graphs that show the results of building each data structures (Min-Max-Heap: MMH, Min-5-Heap: Heap) by inserting n elements (n is the number of random generations).



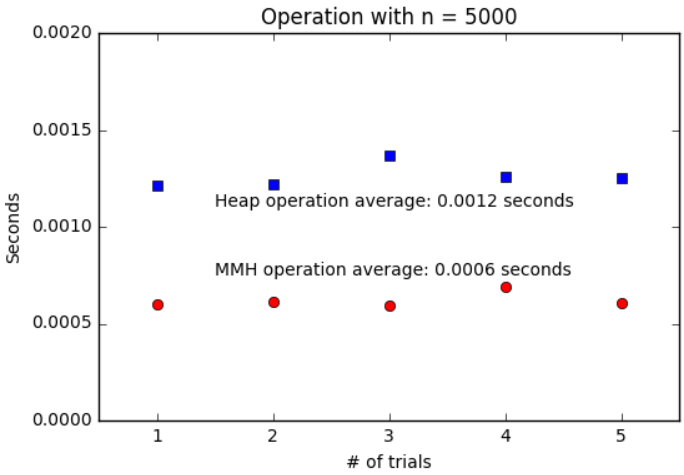


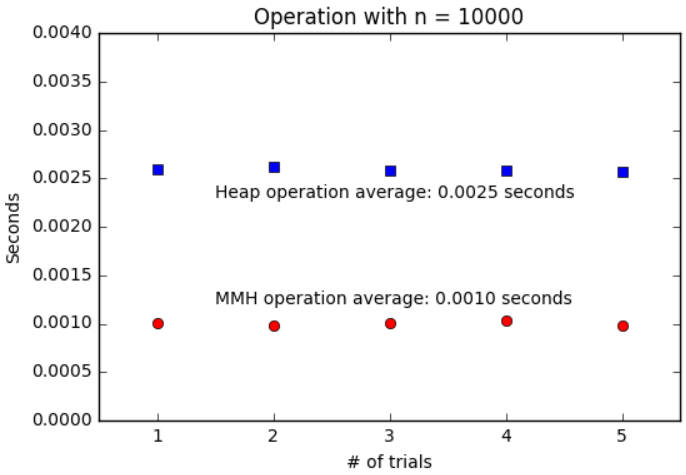


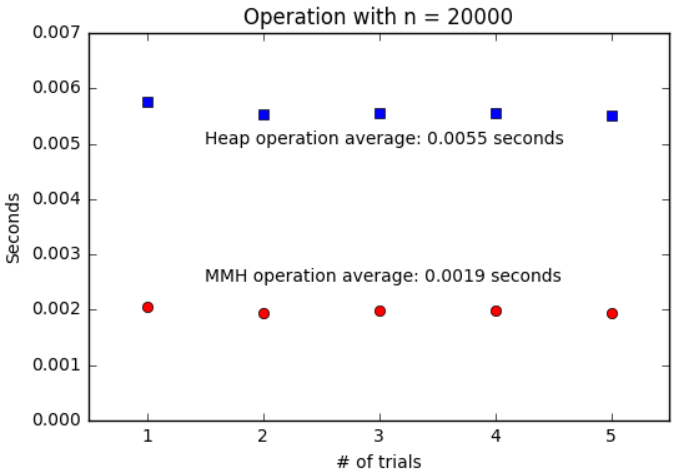


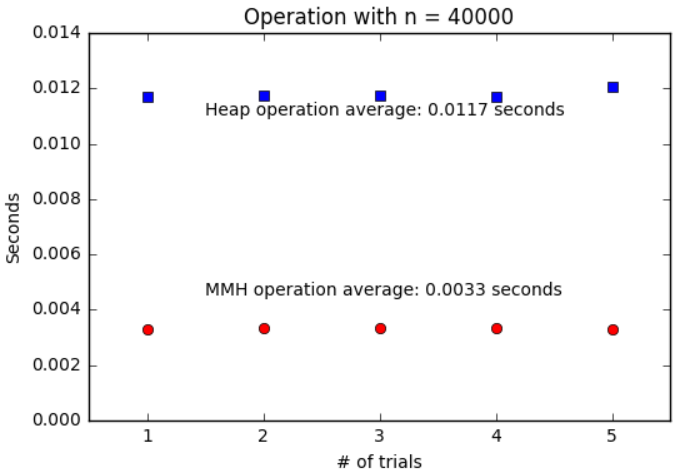
3.3 Operation

Below are the graphs that show the results of n operations on each data structures (n is the number of random generations divided by 10).

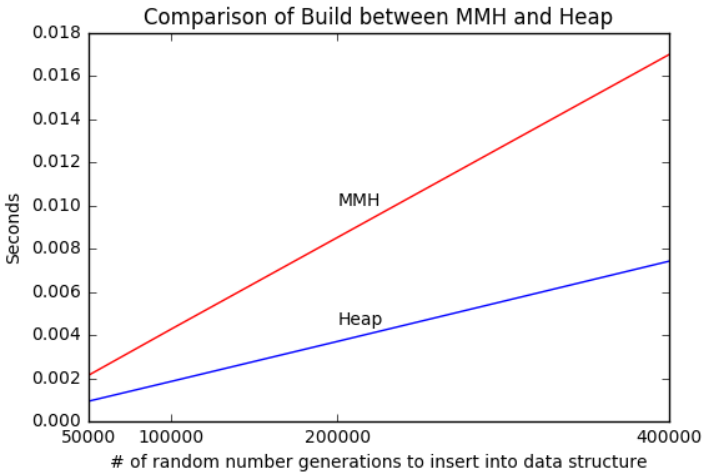


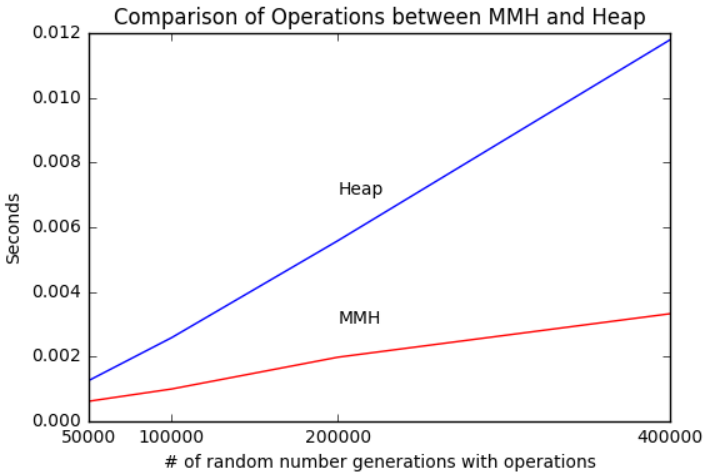






3.4 Performance Comparison





4 Conclusion

The *build* process takes longer time in Min-Max-Heap than in Min-5-Heap. Furthermore, as the input size increases, the time it takes to build a Min-Max-Heap becomes larger than it takes to build a Min-5-Heap. Theoretically, both takes $O(n)$ run time. A trivial fact can be shown in this graph – that the time difference between the two $O(n)$ algorithms with respect to the same operations need not to be constant.

The operations (*insert, deleteMin, deleteMax*) in Min-5-Heap takes longer time to complete than Min-Max-Heap. We can see that as the element of data structure increases, the difference between Min-5-Heap and Min-Max-Heap grows larger. Note that, we presume the result is due to deleteMax operation in Min-5-Heap, which takes $O(n)$ run time. All the other operations such as deleteMin and insert in Min-5-Heap, and deleteMin, deleteMax, and insert in Min-Max-Heap takes $O(logn)$ run time.