Offerdahl, Jay
EECS 560
Lab 9 Report

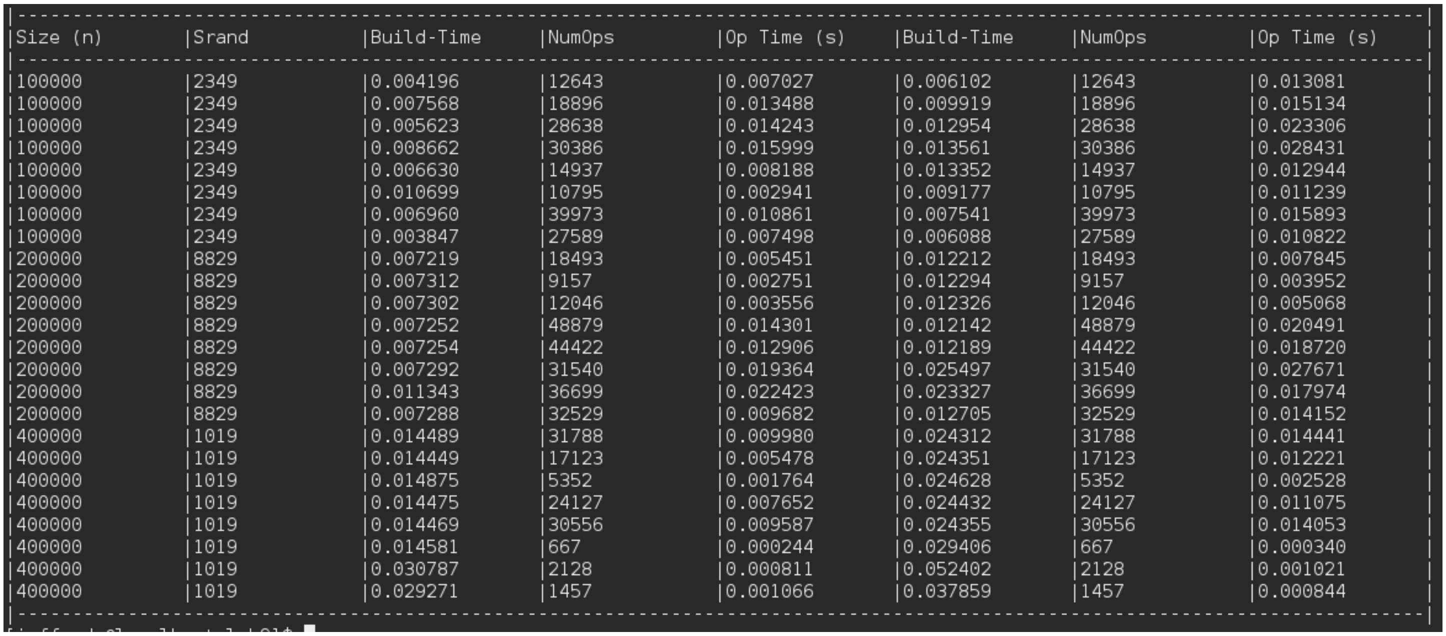
Lab 9 Report

To begin, I've included the code I used to structure the testing at the bottom of the document (because it's a bit long). Also, I've included a screenshot of my output. To generate this output yourself, simply run the program and the table will be printed to the console (which looks better if you make the console wide).

Looking at the results for these two data structures, it's clear that the min heap is quicker to build and, for the most part, quicker to operate on. There are very few examples where the min-max heap was quicker to perform the same operations on as the min heap. Part of this result stems from the fact that the min-max heap data structure is more bloated. In order to maintain the min-max property the min-max heap must execute more instructions every time an operation is carried out than the min heap. This is evident in the timing results. This is also the case for the build-time on both structures. Since the min-max heap must check if it needs to bubbleUpMin, or bubbleUpMax, or trickleDownMin, etc., there's more working being done relative to the min heap.

With that said, there's still significance to having a min-max heap. With a min-max heap, it's much easier to extract the largest value (deleteMax), which we did not test here. Arguably the time spread would be a bit more equal if we did test this function thought, because in order to find the largest value in a min heap, you'd have to search the entire array, which would take O(n) time, whereas finding the largest value in the min-max heap takes O(1) time. In the case of deleteMin, both structures are nearly identical except for the fact that the min-max heap must do a few extra checks to make sure it sends the last value in the heap up properly in bubbleUp.

All in all, this output was as expected with these data structures, and the lab itself was very informative. As always, any questions/concerns can be directed to me at jayofferdahl@ku.edu, and I've included a README.txt file with program specific instructions.



```
|-------------------------------------------------------------------------------------------------------------------------------|
|Size (n)       |Srand          |Build-Time     |NumOps         |Op Time (s)    |Build-Time     |NumOps         |Op Time (s)    |
|-------------------------------------------------------------------------------------------------------------------------------|
|100000         |2349           |0.004196       |12643          |0.007027       |0.006102       |12643          |0.013081       |
|100000         |2349           |0.007568       |18896          |0.013488       |0.009919       |18896          |0.015134       |
|100000         |2349           |0.005623       |28638          |0.014243       |0.012954       |28638          |0.023306       |
|100000         |2349           |0.008662       |30386          |0.015999       |0.013561       |30386          |0.028431       |
|100000         |2349           |0.006630       |14937          |0.008188       |0.013352       |14937          |0.012944       |
|100000         |2349           |0.010699       |10795          |0.002941       |0.009177       |10795          |0.011239       |
|100000         |2349           |0.006960       |39973          |0.010861       |0.007541       |39973          |0.015893       |
|100000         |2349           |0.003847       |27589          |0.007498       |0.006088       |27589          |0.010822       |
|200000         |8829           |0.007219       |18493          |0.005451       |0.012212       |18493          |0.007845       |
|200000         |8829           |0.007312       |9157           |0.002751       |0.012294       |9157           |0.003952       |
|200000         |8829           |0.007302       |12046          |0.003556       |0.012326       |12046          |0.005068       |
|200000         |8829           |0.007252       |48879          |0.014301       |0.012142       |48879          |0.020491       |
|200000         |8829           |0.007254       |44422          |0.012906       |0.012189       |44422          |0.018720       |
|200000         |8829           |0.007292       |31540          |0.019364       |0.025497       |31540          |0.027671       |
|200000         |8829           |0.011343       |36699          |0.022423       |0.023327       |36699          |0.017974       |
|200000         |8829           |0.007288       |32529          |0.009682       |0.012705       |32529          |0.014152       |
|400000         |1019           |0.014489       |31788          |0.009980       |0.024312       |31788          |0.014441       |
|400000         |1019           |0.014449       |17123          |0.005478       |0.024351       |17123          |0.012221       |
|400000         |1019           |0.014875       |5352           |0.001764       |0.024628       |5352           |0.002528       |
|400000         |1019           |0.014475       |24127          |0.007652       |0.024432       |24127          |0.011075       |
|400000         |1019           |0.014469       |30556          |0.009587       |0.024355       |30556          |0.014053       |
|400000         |1019           |0.014581       |667            |0.000244       |0.029406       |667            |0.000340       |
|400000         |1019           |0.030787       |2128           |0.000811       |0.052402       |2128           |0.001021       |
|400000         |1019           |0.029271       |1457           |0.001066       |0.037859       |1457           |0.000844       |
|-------------------------------------------------------------------------------------------------------------------------------|
```

Offerdahl, Jay
EECS 560
Lab 9 Report

How I structured the testing:

- 3 Rounds of 8 tests each (3 different data sizes, 8 tests for each one)
    - Set the seed, fill both arrays with the same data
    - Time building the min heap
    - Time building the min-max heap
    - Calculate the number of operations
    - Reset the seed, perform and time the operations on the min heap
    - Reset the seed, perform and time the operations on the min-max heap
    - Repeat until done
- Output the final table

```
for(int i = 0; i < 3; i++) {
        int currentSize = dataSizes[i];
        int seed = seeds[i], row, ops, val;
        srand(seeds[i]);

        // Do eight tests for this data size
        for(int j = 1; j <= 8; j++) {
            row = j + i * 8;
            arrSize1 = arrSize2 = 0;

            // Fill both arrays with the same data
            fillBoth(row, arr1, arr2, &arrSize1, &arrSize2, currentSize);
            output[row][0] = std::to_string(currentSize);
            output[row][1] = std::to_string(seed);

            // Build the min heap
            tim->start();
            min->buildHeap(arrSize1, arr1);
            duration = tim->stop();
            sprintf(buffer, "%5.6f", duration);
            output[row][2] = buffer;

            // Build the minmax heap
            tim->start();
            minmax->buildHeap(arrSize2, arr2);
            duration = tim->stop();
            sprintf(buffer, "%5.6f", duration);
            output[row][5] = buffer;

            // Calculate the number of operations to perform
            ops = rand() % 50000 + 1;
            output[row][3] = std::to_string(ops);
            output[row][6] = std::to_string(ops);

            // Reset the random number generator
            srand(seeds[i]);

            tim->start();
            for(int k = 0; k < ops; k++) {
                if((double) rand() / RAND_MAX < 0.5)
                    min->deleteMin(&arrSize1, arr1);
                else {
                    val = rand() % (currentSize * 2 + 1) - currentSize;
                    min->insert(val, &arrSize1, arr1);
                }
            }
            duration = tim->stop();

            sprintf(buffer, "%5.6f", duration);
            output[row][4] = buffer;

            // Reset the random number generator
            srand(seeds[i]);

            tim->start();
            for(int k = 0; k < ops; k++) {
                if((double) rand() / RAND_MAX < 0.5)
                    minmax->deleteMin(&arrSize2, arr2);
                else {
                    val = rand() % (currentSize * 2 + 1) - currentSize;
                    minmax->insert(val, &arrSize2, arr2);
                }
            }
            duration = tim->stop();

            sprintf(buffer, "%5.6f", duration);
            output[row][7] = buffer;
        }
    }
```