Daniel Murga

Organization

When creating the skew and leftist heap, a for loop is used based on n until 400,000, incrementing n by n = n * 2. Within the loop, a nested for loop from 1 to 5 sets the seed using srand(i) and gets random numbers and places them into an array. It then starts the timer, a for loop calls insert with each value on both heaps, populates them, and calculates the duration. To check deletion and insertion time, it creates an array containing operations to perform and random values if needed. The time to complete these operations is then output to the terminal and calculates the average with the stored durations for each heap.

Data Generation

To generate the data for the test, a for loop from 1 to 5 sets the seed in srand(i) based on the value of i. The next loop iterates from 0 to n and populates an array called random with values using rand(). To time the deletion and insertion operations, it creates an operations array to determine which operation to perform and an array for a random value if needed to insert or delete. A for loop to .10 * n checks each operation and executes deleteMin if $x < 0.5$ and inserts a random value otherwise. Each iteration doubles the size of n until 400,000.

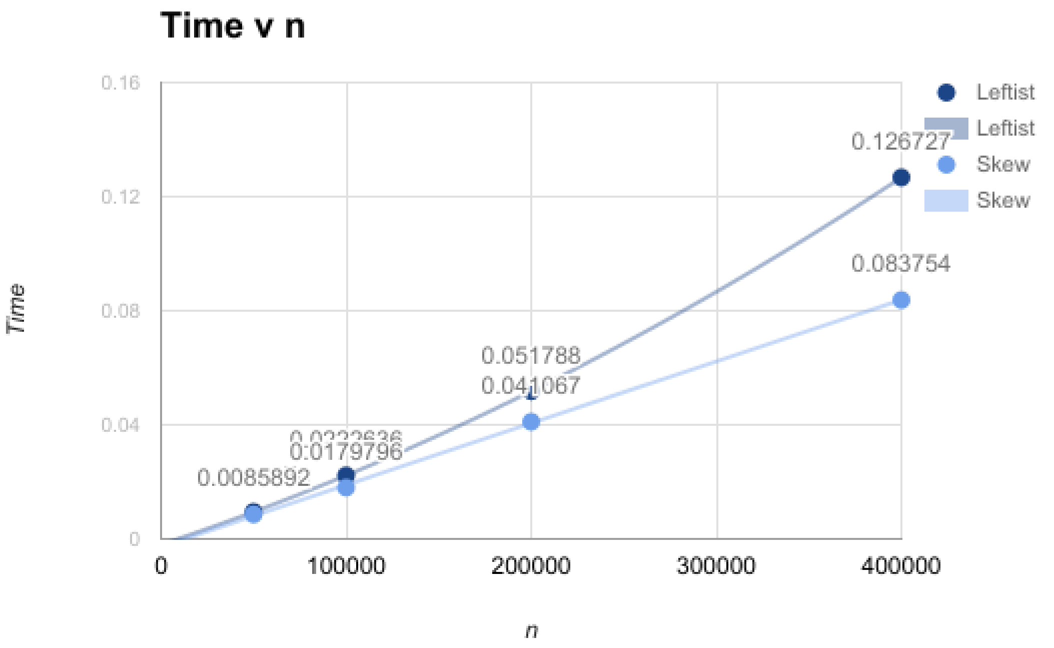
Summary of Results

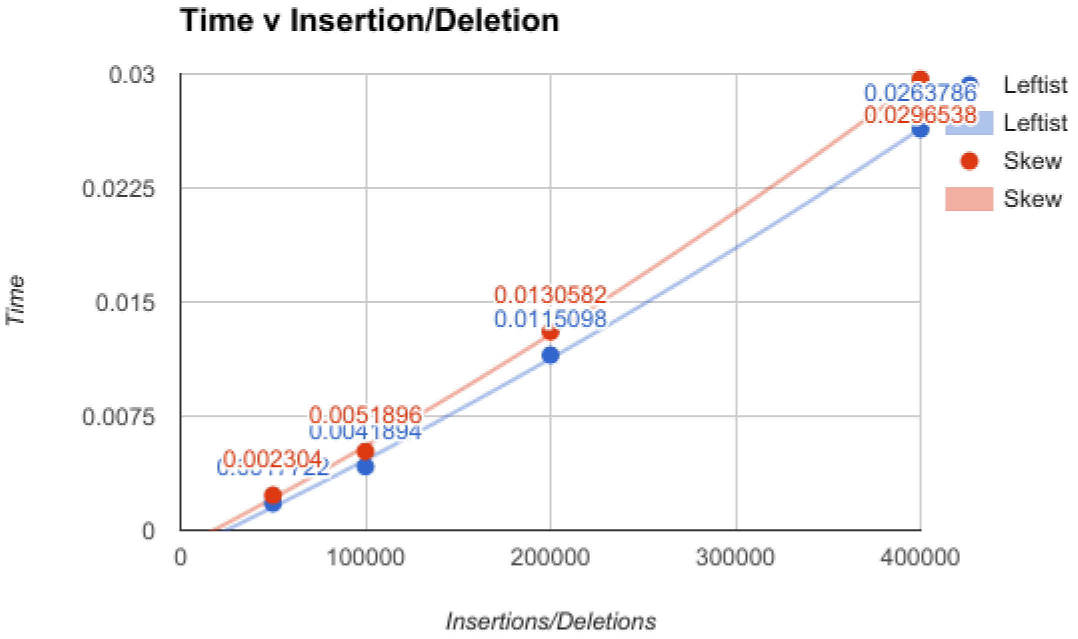
From the results, the average time to build the leftist heap was seemed linear as n doubles until n reached 200,000, starting at around 0.09, 0.022, 0.051 to 0.126 seconds at max n. When n was greater than 200,000, time grew exponentially. The average time to build a skew heap was significantly faster than the leftist at higher n values, going from 0.08, 0.017, 0.041, until 0.083 to build it at n = 400,000. When timing the duration of the same operations performed on both heaps, the skew heap was always slightly longer by about 5 percent. This results from the skew heap not being perfectly balanced, taking into account the randomness of the values inserted.

Observation and Conclusion

When building both heaps, the leftist heap time grew significantly faster than the skew, due to having to perform checks and merges, while the skew heap would always merge.. When performing operations on both heaps, the skew heap consistently lags behind the leftist heap, which is more efficient at performing operations like deleteMin since it has been created with the left root’s subtree always being longer than its right counterpart. The building and inserting into the leftist heap on average takes longer than the slowness of the deleteMin operations on the skew heap, and therefore a skew would be more efficient than the leftist heap in most scenarios.

Daniel Murga


Time v n


Time v Insertion/Deletion

Daniel Murga

AVG BUILD	Leftist	Skew
50000	0.0094678	0.0085892
100000	0.0222636	0.0179796
200000	0.051788	0.041067
400000	0.126727	0.083754

BUILD	Leftist	Skew
50000	0.009909	0.009288
	0.009297	0.008079
	0.009517	0.008556
	0.009153	0.008077
	0.009463	0.008946
100000	0.020893	0.018777
	0.023025	0.017768
	0.023224	0.017471
	0.021904	0.017659
	0.022272	0.018223
200000	0.047856	0.038873
	0.053198	0.040468
	0.050799	0.04067
	0.052151	0.042016
	0.054936	0.043308
400000	0.123304	0.082754
	0.137478	0.08734
	0.131931	0.080046
	0.123917	0.084919
	0.117003	0.083711

Daniel Murga

AVG OPS	Leftist	Skew
50000	0.0017722	0.002304
100000	0.0041894	0.0051896
200000	0.0115098	0.0130582
400000	0.0263786	0.0296538

OPERATIONS	Leftist	Skew
50000	0.00178	0.002236
	0.001767	0.002277
	0.00177	0.002324
	0.001757	0.00231
	0.001787	0.002373
100000	0.004218	0.004984
	0.004117	0.005129
	0.004157	0.005211
	0.004182	0.005274
	0.004273	0.00535
200000	0.011273	0.0121
	0.011307	0.012746
	0.011548	0.013288
	0.011666	0.013598
	0.011755	0.013559
400000	0.027317	0.029248
	0.027504	0.030714
	0.025548	0.028991
	0.025767	0.029432
	0.025757	0.029884