EECS 560: Lab 8 Report

Leftist Heap vs. Skew Heap

Haaris Chaudhry
KUID: 1312155

Date Submitted: 3/16/2017

Introduction and Background

In the first part of this lab, students were tasked with creating both a leftist heap and a skew heap and testing that these heaps worked with various values. This report considers the second part of the lab where students tested both heaps against each other to observe the difference in time between each heap during typical heap operations.

Implementation Process

Implementing the experiment consisted of first creating a timer and then the arrays that would hold the timing data. We know that we'll have to do 5 iterations for each of the 4 n values of {50000, 100000, 200000, 400000}, which means we'll have a total of 20 iterations. Therefore, each of the data arrays had 20 slots where the first 5 slots would represent the timings found while n was equal to 50000, the next 5 for n equal to 100000, and so on.

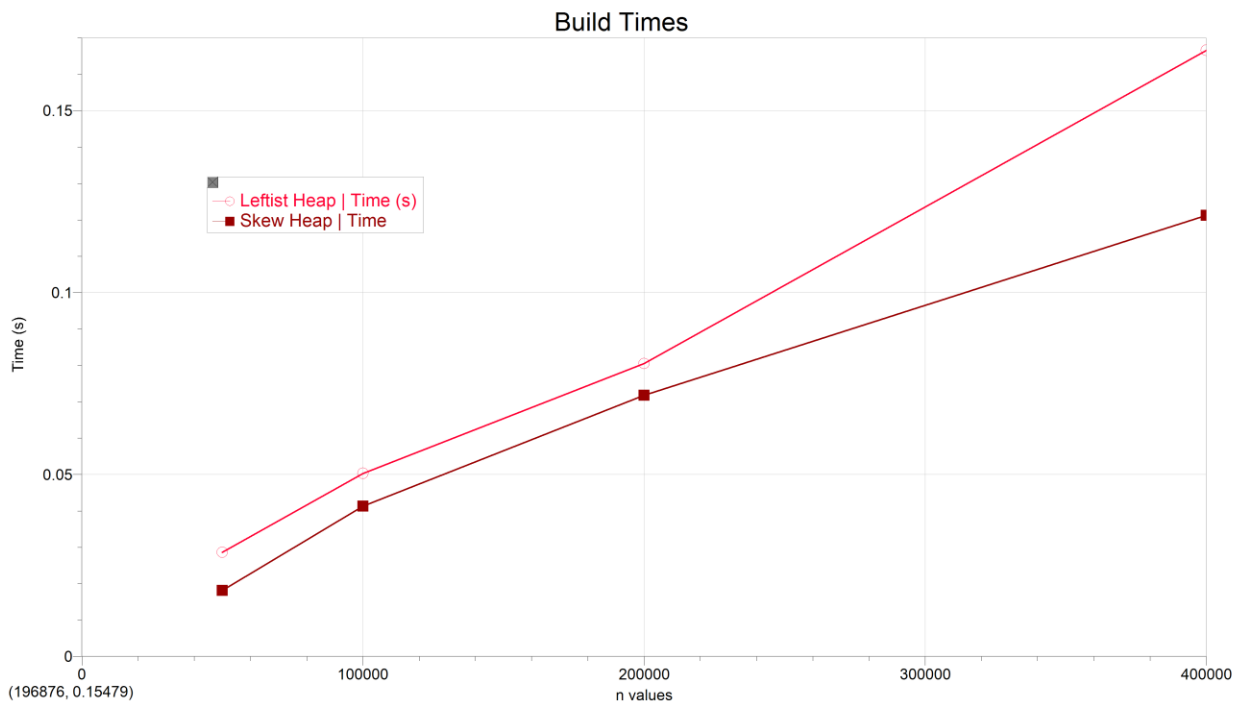
Clearly, we'll need two loops, one nested in the other. The outer loop (with the i counter) controls the n value while the inner loop (the loop with the j counter) controls the iterations for the n value. Each inner loop will control the seeding of the random number generator. Seed values used were in the integers in the range [0, 4].

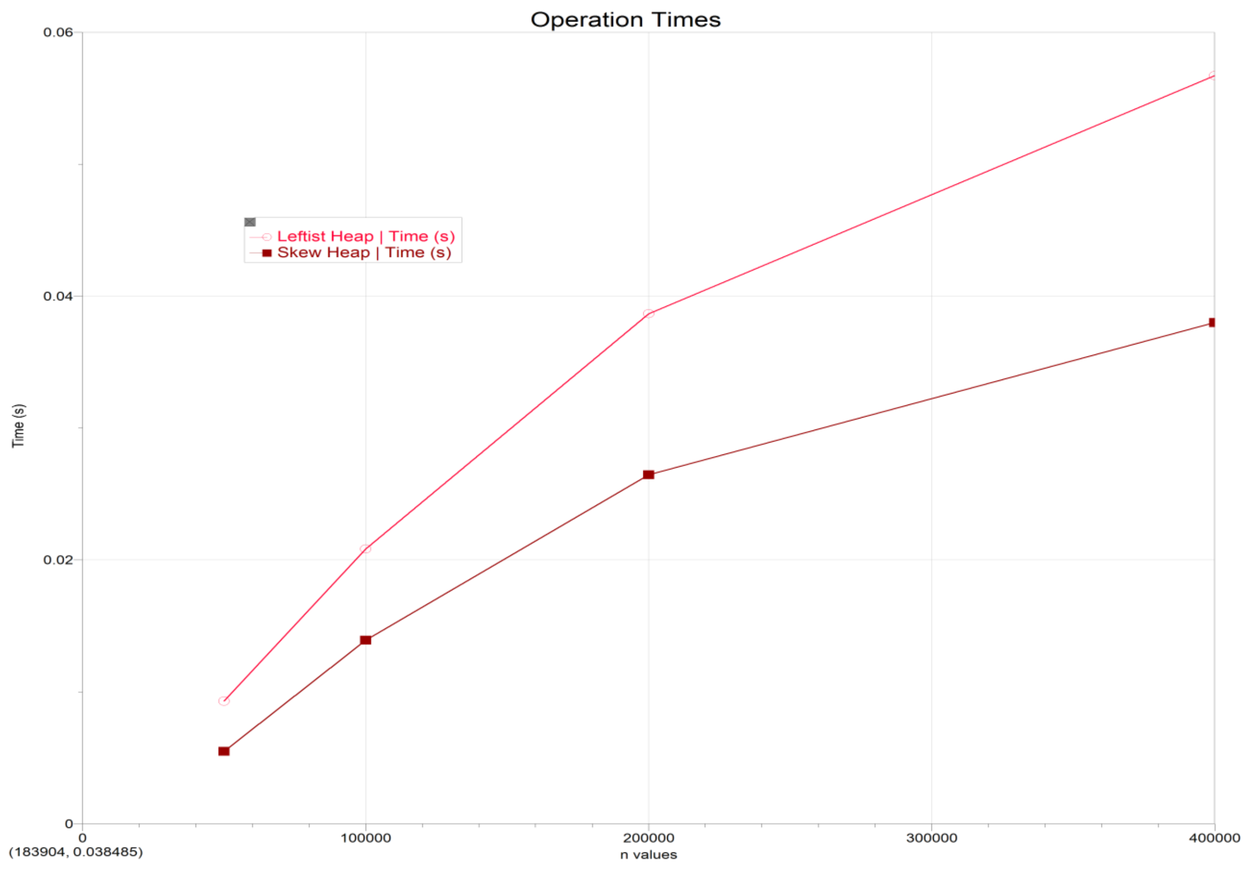
We then create an array dynamically with a size n and fill it with random values ranging from 1 to 4*n. Next, we build each heap, timing how long each heap takes to be filled with the random values. Lastly, we use another loop, the z-loop, that counts from 0 to .1*n and randomly assigns a process to each data structure during each iteration.

This process is then repeated per the limits of each loop. After gathering the data, it's displayed to the screen by summing up the array values 5 at a time and then averaging those values to display the average times for each n. We also show intermediate values for each seed value and each n value.

Results and Discussion

		Build Times (s)		**Operation Times (s)**	
n	**Seed**	*Leftist Heap*	*Skew Heap*	*Leftist Heap*	*Skew Heap*
50000	0	0.029849	0.019848	0.011196	0.006996
	1	0.0291	0.020388	0.008327	0.004867
	2	0.027615	0.019969	0.00846	0.004838
	3	0.02951	0.019332	0.011684	0.006708
	4	0.027169	0.011004	0.006862	0.00407
Average		0.0286486	0.0181082	0.0093058	0.0054958
% Difference		36.79202474		40.94220809	
100000	0	0.056699	0.044749	0.016044	0.010841
	1	0.049409	0.032938	0.017421	0.012274
	2	0.053068	0.03386	0.019848	0.013223
	3	0.046204	0.047967	0.026474	0.017239
	4	0.046247	0.04697	0.024376	0.016004
Average		0.0503254	0.0412968	0.0208326	0.0139162
% Difference		17.94044359		33.19988864	
200000	0	0.084126	0.09068	0.034347	0.021622
	1	0.072967	0.048708	0.035458	0.024545
	2	0.076754	0.058328	0.032387	0.022272
	3	0.086075	0.08502	0.046452	0.031276
	4	0.082658	0.076131	0.044664	0.032593
Average		0.080516	0.0717734	0.0386616	0.0264616
% Difference		10.85821452		31.55585904	
400000	0	0.177665	0.169666	0.063146	0.042806
	1	0.154661	0.115824	0.055305	0.03682
	2	0.162822	0.111434	0.054522	0.036618
	3	0.16843	0.10247	0.055528	0.037028
	4	0.169438	0.106618	0.05511	0.036701
Average		0.1666032	0.1212024	0.0567222	0.0379946
% Difference		27.25085713		33.01634986	


Build Times
Leftist Heap | Time (s)
Skew Heap | Time
Time (s)
0.15
0.1
0.05
0
0
100000
200000
300000
400000
n values
(196876, 0.15479)


Operation Times
Leftist Heap | Time (s)
Skew Heap | Time (s)
Time (s)
0.06
0.04
0.02
0
0
100000
200000
300000
400000
n values
(183904, 0.038485)

Perhaps not surprisingly, the skew heap performed better overall. Because the input data is randomized and because the skew heap does not have the extra overhead of having to keep track of and update ranks, the skew heap had a clear advantage. We can see that, percentage wise, the difference in times was significant.

Conclusion

It's obvious that the skew heap should be the preferred data structure to use if one had to choose between it and the leftist heap. A leftist heap could be a logical choice if the need for a data structure that does not get skewed (at least on its right subtree) is necessary.