

Yichen Dong

First of all, this project was harder than I expected. It was very hard to wrap my head around the recursive nature of the project at first. I finally sat down and played Towers of Hanoi for a bit, until it made sense to me. There was a pattern to the optimal movement of disks that could be translated into a recursive code. Eventually, I realized that the code was very simple. You started from the bottom and worked upward, doing the same things for the $n-1$ th disk.

The harder thing was trying to figure out how to do it iteratively. It wasn't possible to apply the recursive method to the iterative version, since there was no way to keep of which step it was on. For example, it was hard to know if the $n-1$ disk had completed all its movements so that the n th disk could be moved to its final destination. However, I realized that there was a trick to this as well. The cycles rotated between movements of 3 steps. For an odd number of disks, the first movement was always between the source and the destination, the second was always between the source and the auxiliary tower, and the last was between the destination and the auxiliary tower. The movement was different, since the smaller disk is always moved on top of the larger one, but it always involved the same towers. The even one started with the source and aux, with the 2nd step being the source and destination, and the last step remaining the same. It still followed the general pattern however.

I could not figure out how to perform the iterative solution without a stack. The recursive solution could be performed easily without a stack, since the recursion cascades down and each of the tower steps are performed in order. For the iterative version, however, since I did not know the exact movement of the disks at each step, I had to keep track of their relative sizes using stacks. This allowed me to separate the logic of the disk movement from the logic of which towers to look at. However, this probably increased the run time of the iterative solution by a large factor.

Which brings me to the runtimes, and the space and cost efficiencies. I tried to run it as high as possible, but due to space considerations, I could not go higher than 28. The space is a factor of $O(2^n)$, since for every increase in n the size doubled. This can be seen in the screenshot of the file sizes in the appendix. The runtime cost is not as exact as the space considerations, but this could be due to the natural fluctuations of the processor. My computer was not doing just the calculations, so random confounding variables could have affected the results. That being said, at the higher levels (24-28), the runtime roughly doubled for each successive increase. I would estimate the cost of runtime for both the iterative version and the recursive version at (2^n) . This is because the number of steps does not change for either version. The number of steps needed to be printed is still 2^{n-1} .

The iterative solution appears to be doing worse runtime wise compared to the recursive solution. This is true even at higher numbers of n . I am not particularly surprised, since the Towers of Hanoi problem lends itself so well to recursion. In addition, the stack calls probably increase the runtime of the iterative version compared to the recursive version, which only needs to write to the file. It is possible that the iterative version can work with large numbers of n than the recursive solution due to memory concerns, but the file size limits and the limits of my poor hard drive did not allow me to test further.

To improve this program, I would try to find a way to test the performance of each without having to write to a file. I tried many different methods, such as removing the outputting entirely. However, this seriously skewed the runtime to the recursive method, since it was basically doing nothing, while the iterative method still had to go through its stack pops and pushes. I also tried to find ways to suppress printing to a console while still having it go through the motions, but no such luck. I even tried deleting

files as they were made, but due to the fact that the file size doubled with each iteration, I found that it was still unwieldy, since the size of n is greater than the sum of $n-1$ files.

I would also try to find a way to do the iterative technique without using stacks. Perhaps there is a method that I have not thought of, that would be able to keep track of the disk movements without having to compare sizes. I think this could help the iterative function keep on par with the recursive function.

Before I tried the `write.append()` function, I had used a `StringBuilder` like with the last assignment. This worked for values up to $n=23$, after which I found a cryptic error. It turned out I had run out of memory and had an array overflow for the `StringBuilder`. This was not something that I would have known about had it not been for this project. I also realized that printing to the console takes A LOT longer than writing to a file. The same thing that takes seconds to write to a file could take a minute to output to the console.

I added a try-parse to the beginning of the program to check to see if the user entered a valid integer or not. The program would return an error and quit if the user entered a string instead. I also created an array for the runtimes so that I could output it as a nice csv file and open it in Excel.
























Another option I would like to do next time is to give the option of running just the n number of disks, or everything under it. To do this, I would probably use a Scanner function, which I did not have access to for this lab.

Appendix

Size	Recursive	Iterative
1	0.007665	0.035258
2	0.006387	0.015585
3	0.013796	0.016862
4	0.030658	0.03117
5	0.048543	0.076136
6	0.108839	0.173988
7	0.161213	0.466267
8	0.345165	0.893444
9	0.963448	1.169883
10	1.619032	2.304508
11	2.239358	2.033435
12	2.921003	3.170615
13	4.956993	7.414027
14	9.32866	9.429067
15	14.43716	11.8069
16	23.73797	21.74848
17	35.34278	40.1733
18	67.75536	96.66953
19	121.5074	98.48171
20	175.9875	187.9603
21	408.1724	375.9471
22	704.9243	722.9442
23	1539.288	1547.222
24	3271.464	3218.481
25	5873.401	6386.057
26	10926.05	11800.71
27	23677.31	23834.95
28	44354.36	48246.27

Time is in number of milliseconds

Space

 Output of size 6.txt	4/3/2018 9:42 PM	Text Document	5 KB
 Output of size 7.txt	4/3/2018 9:42 PM	Text Document	10 KB
 Output of size 8.txt	4/3/2018 9:42 PM	Text Document	19 KB
 Output of size 9.txt	4/3/2018 9:42 PM	Text Document	38 KB
 Output of size 10.txt	4/3/2018 9:42 PM	Text Document	75 KB
 Output of size 11.txt	4/3/2018 9:42 PM	Text Document	149 KB
 Output of size 12.txt	4/3/2018 9:42 PM	Text Document	297 KB
 Output of size 13.txt	4/3/2018 9:42 PM	Text Document	593 KB
 Output of size 14.txt	4/3/2018 9:42 PM	Text Document	1,185 KB
 Output of size 15.txt	4/3/2018 9:42 PM	Text Document	2,369 KB
 Output of size 16.txt	4/3/2018 9:42 PM	Text Document	4,737 KB
 Output of size 17.txt	4/3/2018 9:42 PM	Text Document	9,473 KB
 Output of size 18.txt	4/3/2018 9:42 PM	Text Document	18,946 KB
 Output of size 19.txt	4/3/2018 9:42 PM	Text Document	37,891 KB
 Output of size 20.txt	4/3/2018 9:42 PM	Text Document	75,781 KB
 Output of size 21.txt	4/3/2018 9:42 PM	Text Document	151,561 KB
 Output of size 22.txt	4/3/2018 9:42 PM	Text Document	303,121 KB
 Output of size 23.txt	4/3/2018 9:42 PM	Text Document	606,241 KB
 Output of size 24.txt	4/3/2018 9:42 PM	Text Document	1,212,481 KB
 Output of size 25.txt	4/3/2018 9:42 PM	Text Document	2,424,961 KB
 Output of size 26.txt	4/3/2018 9:43 PM	Text Document	4,849,921 KB
 Output of size 27.txt	4/3/2018 9:43 PM	Text Document	9,699,841 KB
 Output of size 28.txt	4/3/2018 9:45 PM	Text Document	19,399,681 KB