Lab report / Write up
Assignment 3:
Despina Patronas

Design Complexity:

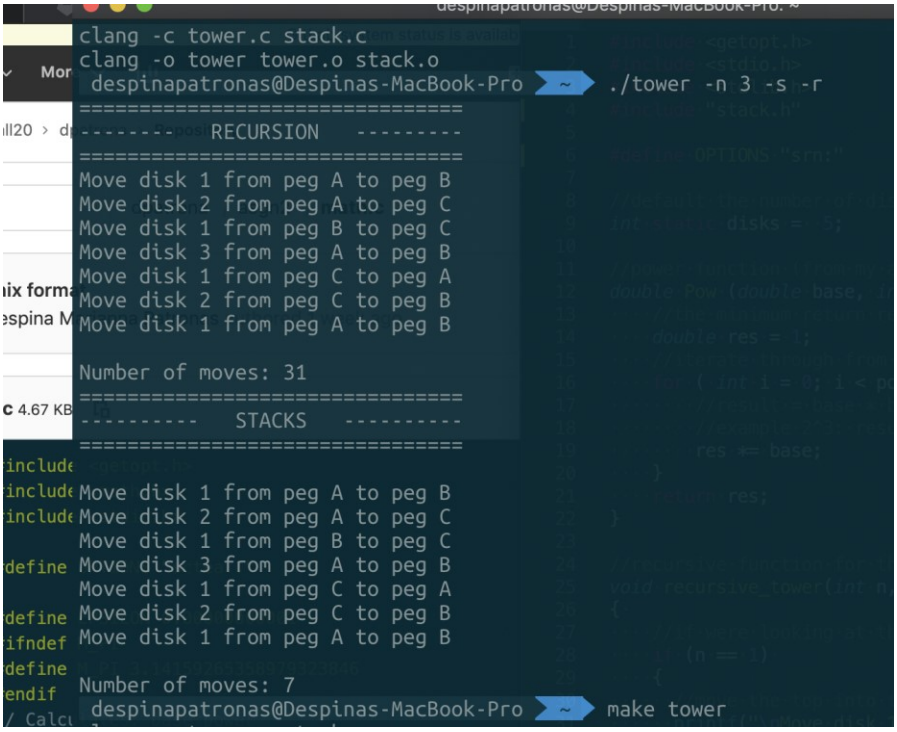
The tower.c Stack solution is rather intuitive and logically sequential methodology that lays out the tower of Hanoi algorithm step by step. Unfortunately, this approach relies heavily on branching statement and conditions which could be simplified to reduce the complexity. As a result of using pointers, passing by reference, and many branching conditional statements, valgrind has issued errors

```
Number of moves: 31
==31173==
==31173== HEAP SUMMARY:
==31173==     in use at exit: 0 bytes in 0 blocks
==31173==   total heap usage: 13 allocs, 13 frees, 252 bytes allocated
==31173==
==31173== All heap blocks were freed -- no leaks are possible
==31173==
==31173== For counts of detected and suppressed errors, rerun with: -v
==31173== Use --track-origins=yes to see where uninitialised values come from
==31173== ERROR SUMMARY: 262 errors from 44 contexts (suppressed: 0 from 0)
-bash-4.2$
```

Errors:

```
Move disk 1 from peg B to peg C
==22838== Conditional jump or move depends on uninitialised value(s)
==22838==    at 0x401217: s_pop (in /afs/cats.ucsc.edu/users/n/dpatrona/dpatrona/asgn3/tower)
==22838==    by 0x400A6E: stack_tower (in /afs/cats.ucsc.edu/users/n/dpatrona/dpatrona/asgn3/tower)
==22838==    by 0x401077: main (in /afs/cats.ucsc.edu/users/n/dpatrona/dpatrona/asgn3/tower)
==22838==
==22838== Use of uninitialised value of size 8
==22838==    at 0x40123A: s_pop (in /afs/cats.ucsc.edu/users/n/dpatrona/dpatrona/asgn3/tower)
==22838==    by 0x400A6E: stack_tower (in /afs/cats.ucsc.edu/users/n/dpatrona/dpatrona/asgn3/tower)
==22838==    by 0x401077: main (in /afs/cats.ucsc.edu/users/n/dpatrona/dpatrona/asgn3/tower)
==22838==
==22838== Conditional jump or move depends on uninitialised value(s)
==22838==    at 0x401171: s_push (in /afs/cats.ucsc.edu/users/n/dpatrona/dpatrona/asgn3/tower)
==22838==    by 0x400A79: stack_tower (in /afs/cats.ucsc.edu/users/n/dpatrona/dpatrona/asgn3/tower)
==22838==    by 0x401077: main (in /afs/cats.ucsc.edu/users/n/dpatrona/dpatrona/asgn3/tower)
==22838==
==22838== Use of uninitialised value of size 8
==22838==    at 0x4011CA: s_push (in /afs/cats.ucsc.edu/users/n/dpatrona/dpatrona/asgn3/tower)
==22838==    by 0x400A79: stack_tower (in /afs/cats.ucsc.edu/users/n/dpatrona/dpatrona/asgn3/tower)
==22838==    by 0x401077: main (in /afs/cats.ucsc.edu/users/n/dpatrona/dpatrona/asgn3/tower)
==22838==
```

However, since these supposed errors do not show affect the flow of the program and functionality (as seen by the output below), I have not changed the program itself but tried to track down what may be the cause of this error.



I believe these valgrind errors (uninitialized value and conditional jumps depending on such errors) are occurring within this loop.

I chose declare values set to zero initially which are passed by reference into a function which determine how to interpret the items[element] size.

```
else {
    for ( int i = 0; i < num_moves; i++ ) {
        int move = i % 3;

        switch (move) {
            //first even move (s<->e)
            case 0:
                //find the appropriate legal move
                compare(extra, source, &ce, &cs);

                if (cs < ce ) {
                    //(source->extra)
                    display(source, extra);

                    //push the valur that is popped from source i
                    s_push(extra, s_pop(source));
                }
                //otherwise (extra->source)
                else {
                    display(extra, source);
                    //(goal->source)push the value that is poppec
                    s_push(source, s_pop(extra));
                }
                break;
```

```
void compare(Stack *s1, Stack *s2, int *c1, int *c2) {
  //if the first peg has no disks it has value empty =
  //disk n < disk n-1 (empty) always causes a move to c
  int empty = disks+1;  //empty = disks+1

  if ( s_empty(s1)) {
      //derefence c1 pointer to hold value of empty
      *c1 = empty;
  }
  else if (s_empty(s2)) {
      *c2 = empty;
  }
  else {
      *c1 = s_peek(s1);
      *c2 = s_peek(s2);
  }
}
```

This program, despite these issues, works well on both my local machine and the unix server Without any memory leaks.

As far as time complexity is concerned the tower of Hanoi time is based on the formula to calculate the moves:

Formula = (2^n)-1

From which we can drop the constants and just say

T(n) = (2^n)

So: Big O = O(2^n)