11. Recursion

Thursday, May 26, 2022 4:24 AM

- Good for tree & graph manipulation, sort, search, D.P.
  - makes it a lot simpler

* iterative > recursive, space-wise because we allocate memory for recursive stack
- stack overflow

Simply:
- defines in term of itself
  - function refers to itself inside the function

- Some examples:
  - DOM Traversal, file tree manipulation

* needs base case (exit condition) or call stack overflow will happen

Anatomy
1) Identify base case
2) Identify recursive case
3) 2 return statements, 1 for base & 1 recursive case
   - return recursive case, so result will go up the stack
   recurs ie return should call itself while getting closer to base case

Practice factorial
- factorial (5) < stack
  factorial(4) < factor ... STOP STOP 5!

- factorial ( ) factorial(4) - factorial(3) - factorial(2) - factorial(1)

stop stop

5 * 4 * 3 * 2 * 1

fac(5);

```
public static int fac(int d){
    if (d == 1){
      return 1;
    }
    return d * fac(d-1);
  }
```

5 * 24 = 120

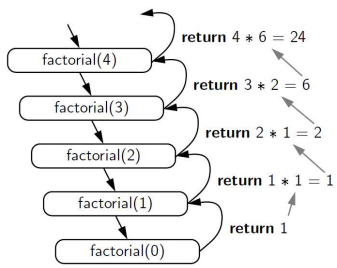
5 . fac(4) return 4 * 6 = 24

↳ 4 . fac(3) return 3 * 2 = 6

↳ 3 . fac(2) return 2 * 1

↳ 2 . fac(1) return 1

↳ 1




```
(factorial 6)
(* 6 (factorial 5))
(* 6 (* 5 (factorial 4)))
(* 6 (* 5 (* 4 (factorial 3))))
(* 6 (* 5 (* 4 (* 3 (factorial 2)))))
(* 6 (* 5 (* 4 (* 3 (* 2 (factorial 1))))))
(* 6 (* 5 (* 4 (* 3 (* 2 1)))))
(* 6 (* 5 (* 4 (* 3 2))))
(* 6 (* 5 (* 4 6)))
(* 6 (* 5 24))
(* 6 120)
720
```

down then up

Practice Fibonacci

input: int n, where n = index value of fib sequence

output: f(n)

where f(n) = f(n-1) + f(n-2), for n > 1

sum of previous 2 numbers

e.g. 0, 1, 1, 2, 3, 5, 8, 13, 21, 34

n = 2
output = 1, f(2) = f(1) + f(0) = 1 + 0 = 1

n = 3
output = 2, f(3) = f2 + f(1) = 1 + 1 = 2

n = 4 f(4) = f(3) + f(2) = 2 + 1 = 3

fib(4)
↳ fib(3) + fib(2)
↳ fib(2) + fib(1) + fib(1) + fib(0)

$n = 4$

output $= F(4) = F(3) + F(2) = 2 + 1 = \ldots$

class Solution {

Space: $O(n)$,

```
  public int fib(int n) {
  if (n == 0 ){
     return 0;
  }
  if (n < 3){
     return 1;
  }

   return fib(n-1) + fib(n-2);
  }
}
class Solution {
  public int fib(int n) {
  if(n <= 1)
        return n;

   return fib(n-1) + fib(n-2);
  }
}
```

or $n < 2$

or

Time: $O(2^n)$, exponential time bc tree expands exponentially,

Space: $O(n)$, for recur stack

iterative is $O(n)$ time,

Divide & conquer w/ Recursion

1) Divide into a number of subproblems that are smaller instances of the same problems
2) Each instance of the subproblem is identical in nature
3) The solution of each subproblem can be combined to solve the problem at hand

- Merge Sort, Quick Sort, Tree Traversal, Graph Traversal

iterative (in main) → iterative (in function) → identify minimal problem (base case) ⇒ convert remaining logic to recursion

reverse string

recursive

```
void reversed(char[] s, int first, int last){
   if (first > last) {
      return;
   }
   char temp = s[first];
   s[first] = s[last];
   s[last] = temp;
   reversed(s, ++first, --last);
}
```