Problem Set 6: Computational Complexity

Matthew Hrones

STAT 535: Statistical Computing

4/12/20

In [1]:
```
import math
import random
```

Part (a): When you are able to divide a problem into subproblems that themselves look just like the original with different inputs, what kind of algorithm is used?

Dividing a problem into subproblems is called using a recursive algorithm

Part (b): What is/are the base case(s)?

The recursive power algorithm has two base cases.

- Base Case 1: y == 0, where 1 is returned
- Base Case 2: y == 1, where x is returned

Part (c): Write a simple recursive formulation of the power function

In [2]:
```
def recursive_power(x,y):

    if y == 0:     # First Base Case
        return 1
    if y == 1:     # Second Base Case
        return x

    if y % 2 == 0: # Determining if y is even or odd
                   # If even, we create subproblems [x^(y/2)] and return[x^(y/2)]^2
        return recursive_power(x, y//2) * recursive_power(x, y//2)
    else:
                   # If odd, we multiply by an extra x term in the return statement
        return recursive_power(x, y//2) * recursive_power(x, y//2) * x
```

Part (d): Refactor your simple recursive formulation to a memoized dynamic programming formulation

In [3]:
```
memo_dict = {}

def dynamic_power(x,y):

    xy_pair = (x, y)             #generate key entry for memo_dict

    if xy_pair in memo_dict:     # check to see if the key exists in memo_dict
        return memo_dict[xy_pair]

    if y == 0:                   # Base Case 1
        return 1
    if y == 1:                   # Base Cases 2
        return x

    if y % 2 == 0:               # Determining if y is even or odd
                                 # if even, create an entry for the result [x^y]^2 and return it
        memo_dict[xy_pair] = dynamic_power(x, y//2) * dynamic_power(x, y//2)
        return memo_dict[xy_pair]
    else:                        # if odd, create an entry for the result ([x^y]^2)*(x) and return it
        memo_dict[xy_pair] = dynamic_power(x, y//2) * dynamic_power(x, y//2) * x
        return memo_dict[xy_pair]
```

Part (e): Refactor your simple recursive formulation to a bottom-up (iterative) programming formulation

In [4]:
```
iter_dict = {}

def iterative_power(x, y):
    xy_pair = (x, y)             #generate key entry for iter_dict

    iter_dict[(x, 0)] = 1        # loading initial values into iter_dict
    iter_dict[(x, 1)] = x

    for y_val in range(2, y+1):  # looping through the y_values, from 2 to y

        if y_val % 2 == 0:       # if even, create an entry for the result [x^y]^2
            iter_dict[(x, y_val)] = iter_dict[(x, y_val//2)] * iter_dict[(x, y_val//2)]

        else:                    # if odd, create an entry for the result ([x^y]^2)*(x) and return it
            iter_dict[(x, y_val)] = iter_dict[(x, y_val//2)] * iter_dict[(x, y_val//2)] * x


    return iter_dict[(x, y)]     # return the value for (x, y) once x^y has been computed for all y in range (0, y)
```

TESTING

In [5]:
```
def power_test(func):

    print("------------------------------")

    print("FUNCTION TEST: ", func)

    # Calculating x^y, for all x, y in range (0, 100)
    for x in range(0,100):
        for y in range(0, 100):

            display = random.randint(0,30) # generating random number that will decide if the result is printed

            power = x**y                   # finding result using python's built in exponent operator
            result = func(x, y)            # finding result using the selected method

            if power != result:            # checking to see if the results match
                print("FAIL")              # If they do not, break
                return

            elif display == 10 and y < 10 and x < 15:            # Code that will print occasionally print test results
                print("Displaying Randomly Selected Test")       # Will not print insanely large exponents
                print("x = "+ str(x)+ ", y = " + str(y) + ", Result: " + str(result))
                print("SUCCESS\n")


    print("COMPLETE SUCCESS!")
    print("------------------------------")
```

In [6]:
```
power_test(recursive_power)
```

```
------------------------------
FUNCTION TEST:  <function recursive_power at 0x7f872426ac10>
Displaying Randomly Selected Test
x = 0, y = 9, Result: 0
SUCCESS

Displaying Randomly Selected Test
x = 1, y = 2, Result: 1
SUCCESS

Displaying Randomly Selected Test
x = 10, y = 7, Result: 10000000
SUCCESS

Displaying Randomly Selected Test
x = 11, y = 8, Result: 214358881
SUCCESS

Displaying Randomly Selected Test
x = 14, y = 2, Result: 196
SUCCESS

COMPLETE SUCCESS!
------------------------------
```

In [7]:
```
power_test(dynamic_power)
```

```
------------------------------
FUNCTION TEST:  <function dynamic_power at 0x7f872426a940>
Displaying Randomly Selected Test
x = 1, y = 4, Result: 1
SUCCESS

Displaying Randomly Selected Test
x = 4, y = 6, Result: 4096
SUCCESS

Displaying Randomly Selected Test
x = 7, y = 4, Result: 2401
SUCCESS

Displaying Randomly Selected Test
x = 9, y = 7, Result: 4782969
SUCCESS

Displaying Randomly Selected Test
x = 10, y = 4, Result: 10000
SUCCESS

Displaying Randomly Selected Test
x = 11, y = 1, Result: 11
SUCCESS

Displaying Randomly Selected Test
x = 12, y = 1, Result: 12
SUCCESS

Displaying Randomly Selected Test
x = 13, y = 5, Result: 371293
SUCCESS

COMPLETE SUCCESS!
------------------------------
```

In [8]:
```
power_test(iterative_power)
```

```
------------------------------
FUNCTION TEST:  <function iterative_power at 0x7f872422c5e0>
Displaying Randomly Selected Test
x = 1, y = 6, Result: 1
SUCCESS

Displaying Randomly Selected Test
x = 2, y = 7, Result: 128
SUCCESS

Displaying Randomly Selected Test
x = 5, y = 4, Result: 625
SUCCESS

Displaying Randomly Selected Test
x = 6, y = 5, Result: 7776
SUCCESS

Displaying Randomly Selected Test
x = 8, y = 8, Result: 16777216
SUCCESS

Displaying Randomly Selected Test
x = 9, y = 4, Result: 6561
SUCCESS

Displaying Randomly Selected Test
x = 11, y = 0, Result: 1
SUCCESS

Displaying Randomly Selected Test
x = 11, y = 4, Result: 14641
SUCCESS

COMPLETE SUCCESS!
------------------------------
```