Practice for Graded Exercise 2

CSC140 Foundations of Computer Science

26 February 2020

1. Which of the following statements describes how software engineers test software?

   (a) They demonstrate that the program returns correct outputs with all possible inputs.

   (b) They prove the complete absence of defects.

   (c) They verify that the program computes all numerical results with perfect precision.

   (d) They continue repeating a test after removing a defect that caused the test to fail.

- They continue repeating a test after removing a defect that caused the test to fail.

2. If you are given a choice between two algorithms that solve the same problem and differ only by the fact that the time complexity of one is $O(N^2)$ and the time complexity of the other is $O(2^N)$, which should you choose?

   It might help to calculate N^2 and 2^N for several values of N.

   These reminders might help with the calculations...

   - we add exponents when we multiply so that $10^x \times 10^y = 10^{x+y}$ and $2^x \times 2^y = 2^{x+y}$
   - 2^x is very roughly equal to $10^{x/3}$ so that $2^{10} \approx 10^3$ and $2^{20} \approx 10^6$

We will prefer the algorithm whose time complexity is $O(N^2)$ because we expect that algorithm to be much faster for all but the smallest values of N.

N	N^2	2^N
0	0	1
1	1	2
2	4	4
8	64	256
10	100	1024
20	400	1048576
32	1024	4294967296
64	4096	$\approx 1.8 \times 10^{19}$

To say that the time complexity of an algorithm is $O(N^2)$ means that the computer will, when N is larger than some threshold, execute a number of instructions that is no greater than some constant times N^2. 2^N grows much faster than N^2, so no matter what constant of proportionality is chosen, for sufficiently large N, N^2 will be less than 2^N.

3. What will the computer print when it executes this code?

```
max_power = 10
powers = [ 2 ** n for n in range( max_power + 1 ) ]
print( powers )
```

```
[ 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024 ]
```

4. The logarithm base 2 of 1024 is 10: $\log_2 1024 = 10$.

   Logarithms appear in the characterization of the complexity of many divide-and-conquer algorithms. These algorithms achieve efficiency by repeatedly dividing a big problem into smaller problems until they produce a trivial problem.

   How many times must one divide 1024 by 2 before getting down to 1?

It takes 10 divisions to reduce 1024 to 1.

Division #	Quantity
	1024
1	512
2	256
3	128
4	64
5	32
6	16
7	8
8	4
9	2
10	1

5. What does the computer print when it executes this code?

```
a = [ 0 ] * 10
b = [ 0 for i in range(10) ]

print( a )
print( b )
```

The expressions on the right hand sides of both assignment statements create lists that contain ten zeroes.

```
[ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ]

[ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ]
```

6. Write the comment that the "TO-DO" requests.

```
import random

# TRIALS is the size of the experiment---it
# is the count of the random numbers that will
# be generated and measured
TRIALS = 100
BINS = 10
```

```
freq = [ 0 ] * BINS

for i in range(TRIALS):
    # produce a random number in the interval [0.0, 1.0),
    # with equal probability of any number in the range
    random_number = random.random()
    # extract the first digit of the random number
    index = int(random_number * BINS)
    # TO-DO Add a comment that explains what this
    # next statement accomplishes
    freq[ index ] += 1

print( freq )
```

The code produces a histogram—it counts how many of the random numbers begin with a 0, how many begin with a 1, how many begin with a 2, and so on.

(All of the random numbers are fractions that are greater than or equal to zero and less than one. "Begin with" then means the first digit after the decimal point.)

```
    # Increase the count of the number of random
    # numbers that begin with a particular digit
    # (the value of index).
```

7. Define a function that finds the value of the largest integer in a list of integers.

```
def largest( values ):
    best_guess_so_far = values[0]

    for i in range(1, len(values) ):
        if values[i] > best_guess_so_far:
            best_guess_so_far = values[i]

    return best_guess_so_far
```

8. Modify this code so that it prints pairs of primes whose difference is 2.

   The code shown here prints:

   2 3 5 7 11 13 17 19 23 29 31 37 41 43 47

   The modified code should print:

   (3 5) (5 7) (11 13) (17 19) (29 31) (41 43)

```
UPPER_LIMIT = 50

check_list = [ False for i in range(UPPER_LIMIT) ]

check_list[0] = True
check_list[1] = True

for i in range( 2, UPPER_LIMIT ):
    if check_list[i] == False:
        for j in range( i + i, UPPER_LIMIT, i ):
            check_list[j] = True

result = ""
for i in range( UPPER_LIMIT ):
    if check_list[i] == False:
        result += str(i) + " "

print( result )
```

```
UPPER_LIMIT = 50

check_list = [ False for i in range(UPPER_LIMIT) ]


check_list[0] = True
check_list[1] = True

for i in range( 2, UPPER_LIMIT ):
    if check_list[i] == False:
        for j in range( i + i, UPPER_LIMIT, i ):
            check_list[j] = True
```

```
result = ""
#for i in range( UPPER_LIMIT ):
#     if check_list[i] == False:
#          result += str(i) + " "

for i in range( UPPER_LIMIT - 2 ):
     if check_list[i] == False and check_list[i + 2] == False:
          result += "(" + str(i) + " " + str(i + 2) + ") "

print( result )
```

9. There are searches within the selection and insertion sort algorithms.

   (a) In which of the two sorting algorithms is the search through the sorted part of the list?

   (b) In which of the two sorting algorithms is the search through the unsorted part of the list?

   (c) In which of the two sorting algorithms is the search from left to right (toward the last element of the list)?

   (d) In which of the two sorting algorithms is the search from right to left (toward the first element of the list)?

   (e) In which of the two sorting algorithms does the search always reach one end of the list (the first or last element)?

   (f) In which of the two sorting algorithms does the search sometimes stop before reaching one end of the list?

   (a) The insertion sort searches through the sorted part of the list.

   (b) The selection sort searches through the unsorted part of the list.

   (c) The selection sort searches from left to right.

   (d) The insertion sort searches from right to left.

   (e) The search in the selection sort always goes all of the way to the last element in the list.

   (f) The search in the insertion sort does not always go all of the way to the first element in the list.

10. Define a function that will shuffle a list of integers.

- begin with a list of integers
- shuffle the list in n steps, where n is the length of the list
- at the i^{th} step
  - draw at random a number j from the set $\{i, i+1, i+2, \ldots n-1\}$, with an equal probability of drawing any number in the set
  - exchange the values at positions i and j in the list

```
def swap( some_list , i , j ):
    temp = some_list[i]
    some_list[i] = some_list[j]
    some_list[j] = temp

def shuffle( some_list ):
    for i in range( len(some_list) ):
        j = random.randint( i , len(some_list) - 1 )
        swap( i , j )
```