CS566 Mid-Term Assignment Jixing (Jacey) Man

Question 1.

1. $g_1(n) = lg(lg(n))$
2. $g_2(n) = 2^{lg(n)}$
3. $g_3(n) = 2^{\sqrt{2 \times lg(n)}}$
4. $g_4(n) = n^{5.2} + lg(n)$
5. $g_5(n) = n^{6.2} + n!$
6. $g_6(n) = n^{1024} + n^{512} + n \times lg(n)$
7. $g_7(n) = e^n + e^{ln(n)}$
8. $g_8(n) = \sqrt{lg(n)}$

Ranking from No. 1 to No.8, by order of growth. So usually those with exponents are higher growth functions, those with "n" as exponents should come next, then those with log exponents are not as high growth, square root should be lower growth compare to exponents. Order from highest growth to lowest see below:

1. No.6 g6(n) = n^1024 +n^512 +n×lg(n)
2. No.5 g5(n) = n^6.2+n!
3. No.4 g4(n) = n^5.2+lg(n)
4. No.7.g7(n) = e^n + e^ln(n)
5. No.2.g2(n) = 2^lg(n)
6. No.3.g3(n) = 2 ^Sqrt(2×lg(n))
7. No.1 g1(n) = lg(lg(n))
8. No.8 g8(n) = sqrt(lg(n))

Question 2.

Rules and reason to prove pair functions:
Big O(g(n)) = {f (n) there exists constants c, n0 > 0 such that
0≤f(n)≤c×g(n) for all n≥n0} f grows no faster than g (Big O), so limit f(n)/g(n) = 0

Big theta Θ(g(n)) = { f(n): there exist positive constants c1, c2 and n0 suchthat0≤c1g(n)≤f(n)≤c2g(n)for alln≥n0 } (big Theta) F grow the same as G

Big Omega(g(n)) = f a running time is Ω(f(n)), then for **large** enough n, the running time is at least k▪f(n) for some constant k. (Big Omega), so limit f(n)/g(n) = (no limit)

Little o = Upper bound that can not be tight, loose upper bound. Let f(n) and f(n) be functions that map positive integers to positive real numbers. We say that f(n) is (g(n)) (or f(n)Єo(g(n))) if for any real constant >0, there exists an integer constant$n_0 \geq 1$ such that f(n)< c∗g(n) for every integer$n \geq n_0$

Little omega = lower bound that is not tight . Let f(n) and g(n) be functions that map positive integers to positive real numbers. We say that f(n) is ω(g(n)) (or f(n)Єω(g(n))) if for any real constant c >0, there exists an integer constant$n_0 \geq 1$ such that f(n)> c·g(n) for every integer$n \geq n_0$

Check if each pair fit the rules:

f(n) = 128^(n/4) vs g(n) = (512)^(n/8) limit f/g is unlimited

1. $f(n)=\Theta(g(n)$ **Big Theta**: No,

2 $f(n)=O(g(n))$ **Big O**: No,

3. $f(n)=\Omega(g(n))$ **Big Omega**: Yes,

4. $f(n)=o(g(n))$ **Little o**: no,

5. $f(n)=\omega(g(n)$ **Little w**: Yes

f(n) = (n^64) * (2^n) vs g(n) = lg(n) * (2^n) + (4^n) + (n^32) limit f/g is unlimited

1. **1.** $f(n)=\Theta(g(n)$ **Big Theta**: No,
2. **2** $f(n)=O(g(n))$ **Big O**: No,
3. **3.** $f(n)=\Omega(g(n))$ **Big Omega**: Yes,
4. **4.** $f(n)=o(g(n))$ **Little o**: no,
5. **5.** $f(n)=\omega(g(n)$ **Little w**: Yes

f(n) = n^1024 vs g(n) = 2^(lg(n)*lg(n)) limit f/g is 0

6. **1.** $f(n)=\Theta(g(n)$ **Big Theta**: No,
7. **2** $f(n)=O(g(n))$ **Big O**: Yes,
8. **3.** $f(n)=\Omega(g(n))$ **Big Omega**: No,
9. **4.** $f(n)=o(g(n))$ **Little o**: Yes,
10. **5.** $f(n)=\omega(g(n)$ **Little w**: No

Question 3:
The question is asking for “n” big o notation.
Answer: O(log(n)), because there is only 1 while loop that runs “n” times

```
y = 0   this is constant
j = 1    this is constant
while (j*j =< n) this is while loop that runs “n” time as long as j*j > n
          y = y +1 this is constant
          j = j +1    this is constant
end while
```

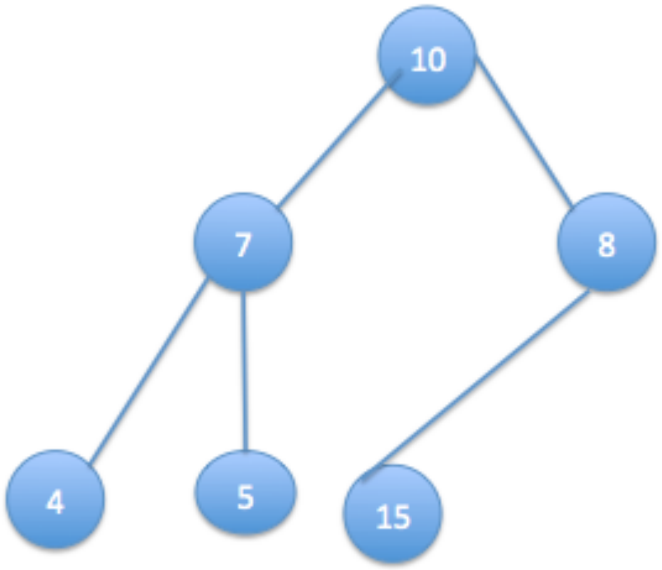
Question 4:
Answer: ((log(n)*n^2) + n^2) so the Big O of n is O(log(n)*n^2)

```
a = 0     this is constant
k = n^2     this is  constant with n^2
while (k >1) do      this is outer while loop that runs the inner for loop, log(n)
          for j – 0 to n^2 do     this is the first inner loop with n^2
                    a = a + I      this is constant
          end for         end inner for loop
          k = k/2    this is constant
end while
```

Question 5:
Answer: n*n*(n/3) = n^2*(n/3) = (n^3)/3 so the Big O of n for the python code is O(n^3)

```
Def isThis(n):
          Print(n)
          If (n == 1):       this is linear so just “n”
                    Return True
          If ( n == 0):    this is linear so “n”
                    Return False
          If (( n%3) == 0):
                    Return (isThis(n/3))   here is a (n/3) but also linear
          Else:
                    Return False
```

Question 6:
For an arbitrary input array, A that in visualized Heap form does not satisfy the max-heap property. Describe when the worst case of HEAP-EXTRACT-MAX happens. Provide an example.
Answer: The property of max-heap is that the root node has the maximum value, the value of each node is =< to the value of the parent node, and it should be a complete binary tree. The worst-case for Heap-Extract-Max is O(log n).
For example: if we have a max heap, see below, and we try to extract this heap to make it valid. Right now it has a root node of 15 which exceeds the parent node 10, so it violated the rule that parent node must be largest, so we need to keep swap and delete nodes this until the 15 reach the parent node



Question 7: Name two properties that a good hash function should have? Describe your answer (If possible with an example).

1. The function should uniformly distribute the keys, it should avoid collisions
2. The hash function has low computation cost.

We can use heuristic techniques to check if the hash functions can perform well. There are two heuristic methods, one is hashing by division and the other is hashing by multiplication.
For example, if we use hashing by division to check the function, a good function should be able to get the same mod reminder. So if the table size is 17, and R = 256, so for both key = 37596, 37596 % 17 = 12, and if we test for key = 573, so 573 % 12 = 12. The reminders are the same.
As for low computation cost, than it should meet the properties of the min/max heap, the root node should be smallest/largest depends on max or min, complete binary tree.

Question 8: When you use double hashing in a hash table, is the number of potential collisions depending on the sequence of key insertions into the table? In other words, if you change the sequence of insertions would the number of collisions change?
Answer: No, if I change the sequences of the insertions, the number of collisions will not change.
Support your answer using an example:
When I am insert keys into hash table, the total number will not change, because if we have a collision, a new location need to be there for the key that caused it, every slot in the hash table can only store a single element, so if the hash table's slot number does not change, the number of potential collisions will not change.