CS566 Fall22

Midterm Assignment

Prof. Ming Zhang

Yiduo Feng

10/24/2022

Tasks

1. Ordering by asymptotic growth rates (4 points): Rank the following functions by order of growth. This means finding an arrangement $g1$, $g2$, … , $g9$ of the functions that satisfy $g1 = \Omega(g2)$, $g2 = \Omega(g3)$, . . . $g7 = \Omega(g8)$

1. $g_1 = \log(\log(n))$
2. $g_2 = 2^{\log(n)}$
3. $g_3 = 2^{\sqrt{2\times\log(n)}}$
4. $g_4 = n^{8.2} + \log(n)$
5. $g_5 = n^{8.2} + \mathrm{n!}$
6. $g_6 = n^{2024} + n^{512} + \mathrm{n} \times \log(n)$
7. $g_7 = e^{\mathrm{n}} + e^{\ln(n)}$
8. $g_8 = \sqrt{\log(n)}$

Tip: Work in stepwise format. Find the fastest growing function first, and then the next fast one
Solution: g5, g7, g2, g3, g6, g4, g8, g1
G2 = 2^log(n) = n
G7 = e^n + n
We know 1 < log(n) < sqrt(n) <n< n*log(n) < n^2 < 2^n < n!
So the fastest growing one is g5 since g5 = O(n!)
Since g7 = O(e^n) so the second one is g7, and flowing with g2 and g3.
Since n^2024 is faster n^8.2 , so g6 is faster than g4.
Since log(n) < sqrt(n), g8 is faster than g1.

2. Asymptotic Notations (6 points) For each pair of the following functions $f(n)$ and $g(n)$, check if
1. $f(n) = \theta(g(n))$?
2. $f(n) = O(g(n))$?
3. $f(n) = \Omega(g(n))$?

4. $f(n)$ = $o(g(n))$? Little o?
5. $f(n)$ = $\omega(g(n))$? Little omega?
Functions $f(n)$ and $g(n)$ are:

1. $f(n) = (128)^{(\frac{n}{4})}$ $vs.$ $g(n) = (512)^{(\frac{n}{8})}$
2. $f(n) = n^{64} \times 2^n$ $vs.$ $g(n) = \log(n) \times 2^n + 4^n + n^{32}$
3. $f(n) = n^{1024}$ $vs.$ $g(n) = 2^{\log(n) \times \log(n)}$

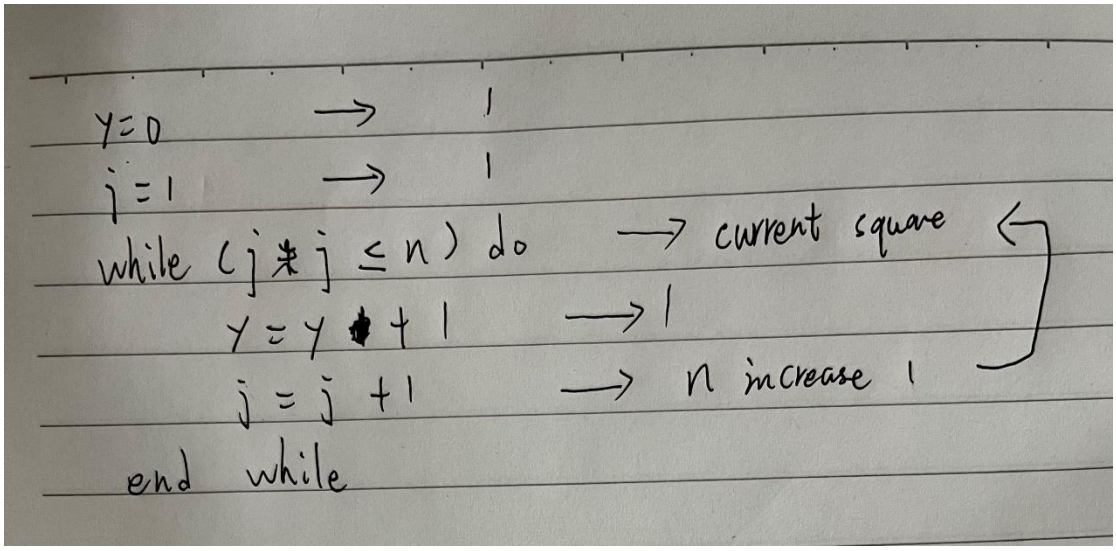
You do not need to provide a formal proof. Describe your justifications in text form.

1. Solution: f(n)= $\Omega(g(n))$, f(n) = $\omega(g(n))$(little omega);
   F(n) = (2*2^(3/4))^n; g(n) = (2*2^(1/8))^n;
   Since 2*2^(3/4) > 2*2^(1/8), so there must be some c*g(n) which is smaller than f(n) (for example c = 1).
   F(n) = 2^(7n/4); g(n) = 2*(9n/8);
   Since lim f(n)/g(n) = lim 2^(7n/4) / 2*(9n/8) = (7n/4)/(9n/8) = infinity, f(n) = $\omega(g(n))$.
2. Solution: $f(n) = O(g(n))$, $f(n) = o(g(n))$.
   We know that the order of growth rate is 1 < log(n) < n ….< n^2 < 2^n < 3^n.
   F(n) = (n^64)*(2^n), we can find that f(n) = O((n^64)*(2^n));
   G(n) = log(n)*(2^n)+4^n+n^32, we can find that g(n) =O(4^n);
   So there must be some c*g(n) which is larger than f(n).
   Since lim f(n)/g(n) = lim ((n^64)*(2^n))/ (4^n) = 0, $f(n) = o(g(n))$.
3. Solution: f(n)= $\Omega(g(n))$, f(n)= $O(g(n))$, $f(n) = \theta(g(n))$,f(n) = o($g(n)$)
   F(n) = n^1024; g(n) = 2^(log(n))^2 = n^(log(n))
   So f(n) = O(n^1024), and g(n) = O(n^(log(n)))
   so there must be some c*g(n) which is larger than f(n) when range of n > 2^1024.
   Since lim f(n)/g(n) = lim (n^1024)/ (n^(log(n))) = n^(1024-log2(n)) = 0, $f(n) = o(g(n))$.

3. Big O notation (2 points): What is the running time of this code in Big O notation regarding n?

Algorithm 1 What is the running time of this code in Big O notation regarding n?

```
1: y = 0
2: j = 1
3: while (j * j ≤ n) do
4:     y = y + 1          ▷ Note: line 4 is some constrant cost
5:     j = j + 1
6: end while
```

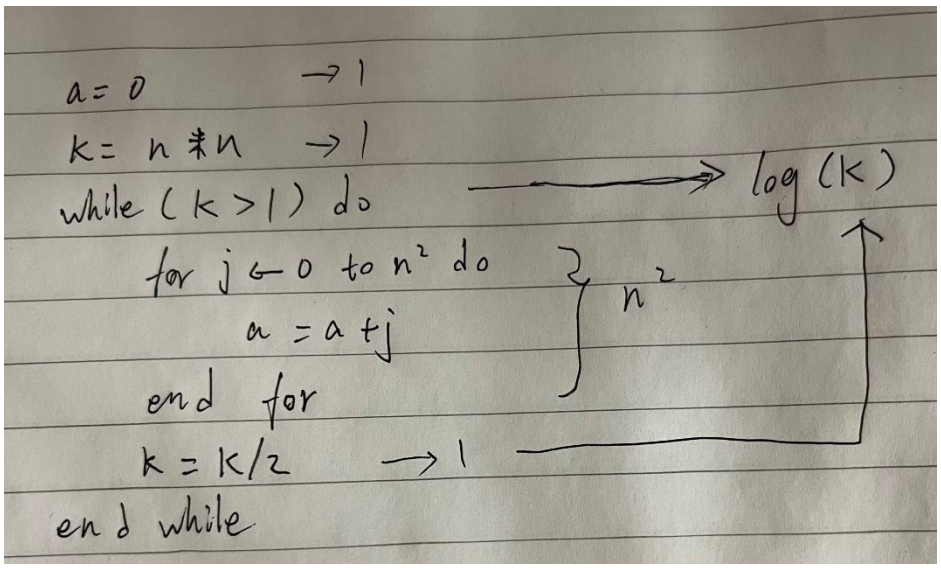


Thus, it is O(sqrt(n)).

4. Big O notation (2 points): What is the running time of this code in Big O notation regarding n?

Algorithm 2 What is the running time of this code in Big O notation regarding n?

```
1: a = 0
2: k = n × n
3: while (k > 1) do
4:     for j ← 0 to n^2 do
5:         a = a + j
6:     end for
7:     k = k/2
8: end while
```



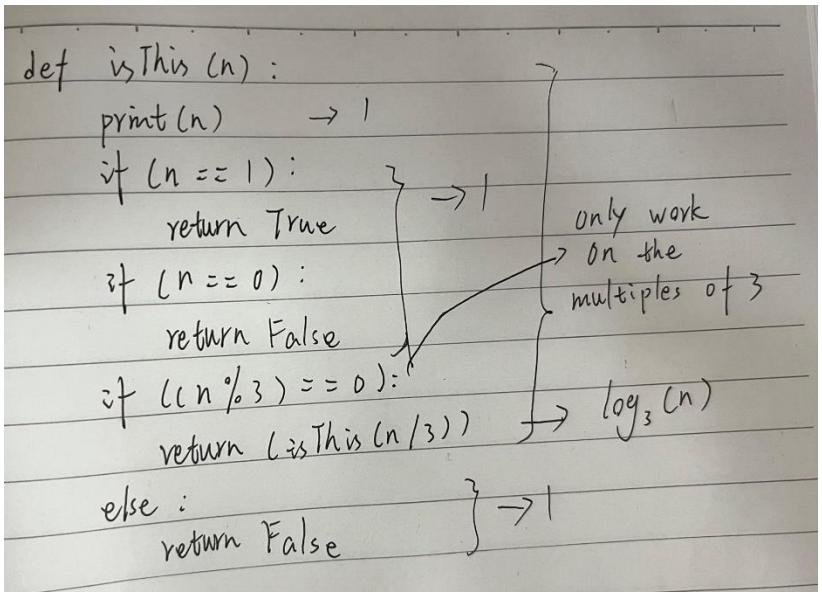
O(k*(log(k)).
So according to the analysis above, the runtime is O((n^2)*log(k)).
Since k = n*n here, so we can also say that the runtime is O(2(n^2)*(log(n)) =

O(k*(log(k))

5. Recursive Function (2 points): What is the running time of the following python code in Big O notation regarding n?

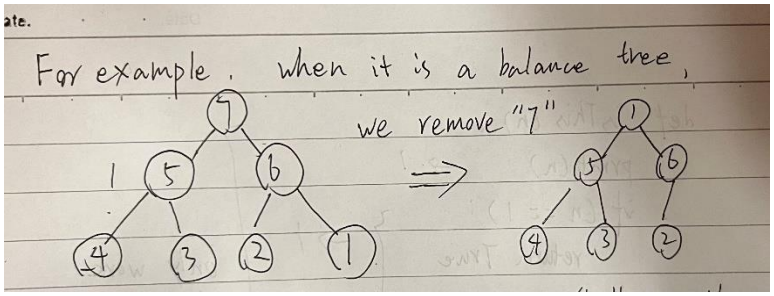
```
def isThis(n):
  print(n)
  if (n == 1):
    return True
  if ( n == 0) :
    return False
  if ((n % 3) == 0):
    return (isThis(n / 3))
  else:
    return False
```

Listing 1: What is the running time of this code in Big O notation regarding n?



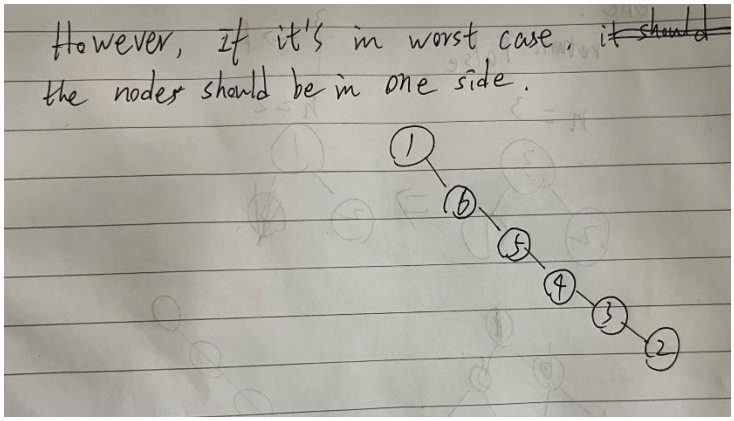
When the input is 0 or other numbers which are not the multiples of 3, the runtime is O(1).
When the input is the multiples of 3, the runtime is O(log3(n))

6. HEAP-EXTRACT-MAX worst case (1 point):
Describe when the worst case of HEAP-EXTRACT-MAX happens after you removed a max value from a heap. Provide an example.
The worst case is O(n), and all of the nodes in one side of the heap.



When we use heapify recursion to move “1” to the bottom of the heap, we only need

to move twice.



However, if it is in worst case, all of nodes are in one side of heap, so the recursion need to visit all of the nodes and the runtime will be O(n).

7. Hash Function (1 point): Name two properties a good hash function should have. Describe your answer (If possible, with an example)

- It is efficient to search element and compute.
- It can deal with collisions.
  When the position is duplicate, we need the method like double hashing, linear probing to deal with collisions.
  For example: when h(k) = h%10.
  1 and 11 will be in the same position ,we can use linked list by linear probing to solve this problem.

8. Double Hashing (2 points): When you use double hashing in a hash table, does the number of potential collisions depend 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? Support your answer using an example.
Yes, if changing the sequence of insertions, the number of collisions will change.
For example, we use double hashing formular below:

$$h(k,i) = (h_1(k) + i \times h_2(k)) \bmod m)$$
$$h_1(k) = k \bmod 7$$
$$h_2(k) = 1 + (k \bmod 3)$$

And the inserting keys are {3, 5, 31}

3: 3%7 = 3

0	1	2	3	4	5	6
			3			

5:5%7 = 5

0	1	2	3	4	5	6
			3		5	

31: 31%7 = 3(collision)

H2 = 1 + 31%3 = 2

(3+2)%7 = 5(collision)

(3+4)%7 = 0

0	1	2	3	4	5	6
31			3		5	

The number of collision is 2.

Then we change the order as {31, 5, 3}

31: 31%7 = 3

0	1	2	3	4	5	6
			31			

5:5%7 = 5

0	1	2	3	4	5	6
			31		5	

3: 3%7 = 3(collision)

H2 = 1 + 3%3 = 1

(3+1)%7 = 4

0	1	2	3	4	5	6
			31	3	5	

The number of collision is 1.

Thus, if changing the sequence of insertions, the number of collisions will change.