CSE 327 Introduction to Algorithm Design Homework #7

Berkan Aslan
27704073

Question #7

$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \begin{cases} c, \text{where } c>0; & f(n) \in \Theta(g(n)) \\ \infty; & f(n) \in \omega(g(n)) \\ 0; & f(n) \in O(g(n)) \end{cases}$

a) $f(n) = 2^n \quad g(n) = 2^{2n}$

Limit Analysis

$$\frac{f(n)}{g(n)} = \frac{2^n}{2^{2n}} = \frac{2}{2^{2n}} = 0$$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0 \Rightarrow f(n) \in O(g(n))$$

b) $f(n) = n^2 \quad g(n) = n^3$,

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \frac{2n^2}{n^3} = 0$$

$$\lim \frac{f(n)}{g(n)} = 0 \Rightarrow f(n) \in O(g(n))$$

c) $f(n) = 3n+7 \quad g(n) = 2n-5$

$$\lim_{n\to\infty} = \frac{3n+7}{2n-5} = \frac{3}{2} \Rightarrow f(n) \in \Theta(g(n))$$

d) $f(n) = 4n^2 \quad g(n) = n^2$

$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \frac{4n^2}{n^2} = 4 \Rightarrow f(n) \in \Theta(g(n))$

L'Hospital

e) $f(n) = \log_2(n) \quad g(n) = \log_{10}(n)$

Logarithm Rule

$\log_b a = \frac{\log_c a}{\log_c b} \Rightarrow f(n) = \frac{\ln(n)}{\ln(2)} \quad g(n) = \frac{\ln(n)}{\ln(10)}$

$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \frac{\frac{\ln(n)}{\ln(2)}}{\frac{\ln(n)}{\ln(10)}} = \frac{\ln 10}{\ln 2} \Rightarrow f(n) \in \Theta(g(n))$ → Real number

f) $f(n) = 2^n \quad g(n) = 3^n$

$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \frac{2^n}{3^n} = 0 \Rightarrow f(n) \in O(g(n))$

g) $f(n) = n^3 \quad g(n) = 7000n^2$

$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \frac{n^3}{7000n^2}$

L-Hospital rule

$\frac{f'(n)}{g'(n)} = \frac{3n^2}{2000n}$

$\frac{f''(n)}{g''(n)} = \frac{6n}{2000} = \infty \Rightarrow \infty ; f(n) \in \omega(g(n))$

h) $f(n) = 5n+4$ $g(n) = 2n+2$

$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \frac{5n+4}{2n+2} = \frac{5}{2} \Rightarrow f(n) \in \Theta(g(n))$

i) $f(n) = \sqrt{n}$ $g(n) = \log_2(n)$

$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \frac{\sqrt{n}}{\log_2(n)} = \frac{\frac{1}{2\sqrt{n}}}{\frac{1}{n\ln(2)}} = \frac{n\ln(2)}{2\sqrt{n}} = \frac{\ln(2)}{2} \lim_{n\to\infty} \sqrt{n} = \infty$

$\Rightarrow f(n) \in \Omega(g(n))$

j) $f(n) = 2^n$ $g(n) = 2^{n+1}$

$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \frac{2^n}{2^{n+1}} = \frac{2^n}{2.2^n} = \frac{1}{2} \Rightarrow f(n) \in \Theta(g(n))$

Question #2

* $\lim_{n\to\infty} \frac{\frac{1}{2n}}{\log n} = 0 \Rightarrow \frac{1}{2n} < \log n$
* $\lim_{n\to\infty} \frac{\log n}{\sqrt{n+5}} = 0 \Rightarrow \log n < \sqrt{n+5}$
* $\lim_{n\to\infty} \frac{\sqrt{n+5}}{n+1} = 0 \Rightarrow \sqrt{n+5} < n+1$
* $\lim_{n\to\infty} \frac{n+1}{n^2 \log(n)} = 0$ $n+1 < n^2\log(n)$
* $\lim_{n\to\infty} \frac{n^2\log(n)}{2^n} = 0$ $n^2\log(n) < 2^n$
* $\lim_{n\to\infty} \frac{2^n}{10^n} = 0 \Rightarrow 2^n < 10^n$
* $\lim_{n\to\infty} \frac{10^n}{n!} = 0 \Rightarrow 10^n < n!$

* $\lim_{n \to \infty} \frac{n!}{2^{2^n}} = 0 \Rightarrow n! < 2^{2^n}$ (1)

ALL Growth Rate

$\frac{1}{2^n} < \log(n) < \sqrt{n+5} < n+7 < n^2 \log(n) < 2^n < 70^n < n! < 2^{2^n}$

=

Question #3

a) pseudo code

```
merge(BST1, BST2)
    For each element E in BST1:
        Insert(E, BST2)

Insert(Element, BST)
IF BST is empty:
  create a new node with Element and set it as the root
  of BST
otherwise:
  if Element is less then the root's value:
      Insert(Element, left subtree of BST)

Else:
  Insert(Element, right subtree of BST)
```

The insertion operation BST has an average-case time complexity $O(\log n)$. worst case $O(n)$

we are inserting each element from BST1 into BST2 so for n elements in BST1 the overall time complexity for merging would be $O(n \log n)$ on average

Worst case is $O(n^2)$

b) Pseudo code

```
kthSmallest(BST, k)
  create a variable count and initialize it to 0
  create a variable result

  InorderTraversal(BST, k, count, result)
  return result


InorderTraversal(node, &count, result)
  if node is null or count >= k:
    return

  InorderTraversal(node.left, k, count, result)

  count = count + 1;

  if count == k
    result = node.value
    return

  InOrderTraversal(node.right, k, count, result).
```

This code performs on in-order traversal of the BST which visit nodes is ascending order. when the count reaches k, it sets the result to the kth smallest element The in-order traversal takes O(n) time in the worst case where n is the number of nodes in the BST

Time complexity of finding the kth smallest element in a BST is O(n) in worst case and it's generally an efficient method for this

c) Balanced BST (BST)

```
nodes = InorderTraversal(BST)
SortedArray = Sort(nodes)
return BuildBalancedBST(SortedArray)

BuildBalancedBST(SortedArray):
  if SortedArray is empty:
    return null

  middle = length(SortedArray) / 2
  root = new Node(SortedArray[middle])
  root.left = BuildBalancedBST(SortedArray[0:middle])
  root.right = BuildBalancedBST(SortedArray[middle+1:])
  return root
```

Time Complexity

- InorderTraversalBST = O(n)
- SortedNodes = O(nlogn) (merge sort)
- BuildBalancedBST(SortedArray)

O(n) (InorderTraversal) + O(nlogn) (Sort) + O(n) (BuildBalancedBST)

= O(nlogn)

d) Pseudo Code

```
RangeSearch(BST, min, max, Result)
  if BST is empty:
    return

  if BST.value > min:
    RangeSearch(BST.left, min, max, result)

  if BST.value >= min and BST.value <= max:

  if BST.value <= max
    RangeSearch(BST.right, min, max, result)
```

Time complexity O(kth) k is number of element within min, max range and h is height of tree
worst case comes is all element O(nth)

Question 4

```
i = 2
while i <= n
  if i % 2 == 0:
    i = i - 1
  else
    i = i * i
    i = i + 1
print(i)
```

Time complexity is $\log_2(n)$ because i^2 decrease.

$O(n)$ → $O(\log_2 n)$

n^2 decrease = $\log_2(n)$

For:

First step $i = 2^* 2 = 4$

Second step $i = 4^* 4 = 16$

Third step $i = 16^* 16 = 256$

continue

$2^x = n$

$x = \log_2(n)$

Question 5

1) Initialize index variable

2) Iterate through the array elements on by one, starting from index 0

3) For each element:
   a. check if it's even (i.e element % 2 == 0)
   b. if it's even, return the element, terminate the loop
   c. if it's not even, increment the index by 1 to move to the next element

4) If you reach the end of the array without finding an even element return a message

worst case = $O(n)$

Best case = $\Omega(1)$

Average case = $\Theta(n)$

First index = 1 % 2 = 0

Second element = 1 * 0 : 1 % 2 = 0 = 72%

Third element = 1 * 0 : 1 % 2 = 0, 1 % 2 = 0 = 72.8%