CSE 321 Homework I

① In each of the following situations, determine whether $f \in O(g)$, $f \in \Omega(g)$ or both (in which case $f \in \Theta(g)$). Provide explicit explanations for your answers. For at least half of the examples, perform a limit analysis.

a) $f(n) = 2^n$, $g(n) = 2^{2n}$ Analyze the limit of: $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{2^n}{2^{2n}}$ simplify

$\lim_{n\to\infty} \frac{1}{2^n} = 0$ since limit is equal to 0, $\boxed{f(n) \in O(g(n))}$

b) $f(n) = n^2$, $g(n) = n^3$ Analyze the limit of: $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{n^2}{n^3}$ simplify

$\lim_{n\to\infty} \frac{1}{n} = 0$ since limit is equal to 0, $\boxed{f(n) \in O(g(n))}$

c) $f(n) = 3n+1$, $g(n) = 2n-5$ Analyze the limit of: $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{3n+1}{2n-5}$ simplify

$\lim_{n\to\infty} \frac{3+\frac{1}{n}}{2-\frac{5}{n}}$ both of them approach 0. therefore $\lim_{n\to\infty} \frac{3+0}{2+0} = \frac{3}{2}$

since the limit is a finite positive constant, $\boxed{f(n) \in \Theta(g(n))}$

d) $f(n) = 4n^2$, $g(n) = n^2$ Analyze the limit of: $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{4n^2}{n^2}$ simplify

$\lim_{n\to\infty} \frac{4\cancel{n^2}}{\cancel{n^2}} = 4$ since the limit is a finite positive constant, $\boxed{f(n) \in \Theta(g(n))}$

e) $f(n) = \log_2(n)$, $g(n) = \log_{10}(n)$ Analyze the limit of: $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{\log_2(n)}{\log_{10}(n)}$ simplify, we going to change bases

$\left.\begin{array}{l} \log_2(n) = \frac{\ln(n)}{\ln(2)} \\ \log_{10}(n) = \frac{\ln(n)}{\ln(10)} \end{array}\right\} \lim_{n\to\infty} \frac{\frac{\ln(n)}{\ln(2)}}{\frac{\ln(n)}{\ln(10)}} = \lim_{n\to\infty} = \frac{\ln(10)}{\ln(2)}$

therefore, $f(n)$ and $g(n)$ same asymptotic growth rate $\boxed{f(n) \in \Theta(g(n))}$

f) $f(n) = 2^n$ $g(n) = 3^n$ Analyze the limit of: $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{2^n}{3^n}$ simplify,

$\lim_{n\to\infty} \left(\frac{2}{3}\right)^n$

it goes to 0 because $\frac{2}{3}$ less than 1, so this limit goes to 0

$\boxed{f(n) \in O(g(n))}$

g) $f(n) = n^3$, $g(n) = 1000n^2$ Analyze the limit of: $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{n^3}{1000n^2}$ simplify

$\lim_{n\to\infty} \frac{1}{1000} \cdot \frac{n^3}{n^2} = \frac{1}{1000} \cdot \lim_{n\to\infty} \frac{n^2}{n^2} n = \frac{1}{1000} \cdot \infty = \infty$

since the limit is infinity this means $f(n)$ grows faster than $g(n)$ $\boxed{f(n) \in \Omega(g(n))}$

h) $f(n) = 5n+4$ $g(n) = 2n+2$ Analyze the limit of: $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{5n+4}{2n+2}$ simplify,

$\lim_{n\to\infty} \frac{5+\frac{4}{n}}{2+\frac{2}{n}}$ both of them approach to 0. $= \lim_{n\to\infty} \frac{5+0}{2+0} = \frac{5}{2}$

since the limit is finite positive constant, $\boxed{f(n) \in \Theta(g(n))}$

i) $f(n) = \sqrt{n}$ $g(n) = \log_2(n)$ Analyze the limit of: $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{\sqrt{n}}{\log_2 n}$ simplify

First apply L'Hopital Rule: $\lim_{n\to\infty} \frac{\frac{1}{2\sqrt{n}}}{\frac{1}{n\ln(2)}} = \frac{1}{2}\ln(2)\sqrt{n} = \lim_{n\to\infty}\left(\frac{1}{2}\ln(2)\sqrt{n}\right)$

$\frac{1}{2}\ln(2) \cdot \lim_{n\to\infty}\sqrt{n}$

$\frac{1}{2}\ln(2) \cdot \sqrt{\lim_{n\to\infty} n}$ ∞

it approaches to ∞

so, $\boxed{f(n) \in \Omega(g(n))}$

j) $f(n) = 2^n$ $g(n) = 2^{n+1}$ Analyze the limit of: $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{2^n}{2^{n+1}}$ simplify

$\lim_{n\to\infty} \frac{\cancel{2^n}}{\cancel{2^n}\cdot 2} = \frac{1}{2}$ → since the limit is finite positive constant $\boxed{f(n) \in \Theta(g(n))}$

---

③ List the following functions in order of their growth and provide proof for your claims.

$\frac{1}{2n}$, $\log(n)$, $\sqrt{n+5}$, $n+1$, 10^n, $n^2\log(n)$, 2^n, $n!$, n^{2^n}

linear ⇐ | ⇒ polynomial   polynomial ⇐ | ⇒ exponential

$\underbrace{\frac{1}{2n}}_{O(n)} < \underbrace{\log(n)}_{O(n)} < \underbrace{\sqrt{n+5}}_{O(n)} < \underbrace{n+1}_{O(n)} < \underbrace{n^2\log(n)}_{O(n^2)} < \underbrace{2^n}_{O(2^n)} < \underbrace{10^n}_{O(10^n)} < \underbrace{n!}_{O(n!)} < \underbrace{n^{2^n}}_{O(n^{2^n})}$ (from slowest to fastest)

the slowest. it grows linearly

it grows slower than any polynomial function

polynomial function and grows faster than linear function

exponential functions grows faster than polynomial functions

grows even faster than exponential functions

function grows fastest among the ones listed

---

④ Calculate the time complexity (in terms of big-Oh notation) of the following program.

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

Analyze the growth of i: $2 \to 4 \to 16 \to 256$ → its has logarithmic growth for n

so simplify: $\left(\left(2^2\right)^2\right)^{2\cdots}$ and goes on, therefore; $(2^2)^x$ → step number, $\log_2(\log_2 n) = x$

hence it's big-Oh notation is $\boxed{O(\log(\log(n)))}$

(5) Suppose you have an array of n elements, where each element can be either even or odd with a probability distribution of %20 even and %80 odd. Propose an algorithm that identifies the first even element in the array. Describe the algorithm and analyze its average-case time complexity.

Algorithm:
1) Start at first element of the array.
2) Iterate through the array from the begginy.
3) For each element, check if it is even.
4) If program finds an even element, returns its index.
5) If program reaches the end without finding an even element, it returns signal or message.

```
for i in range(len(arr)):
    if arr[i] % 2 == 0
        return i
return -1
```

If we consider the odd possibility;
first step (odd chance) = %80
second step (" ") = %80 . %80
third step (" ") = $\%80^3$ ... and goes on.
at certain point odd possibility nearly 0. so in constant time we reach the even number.

Therefore, average case time complexity is O(1)

(3) Provide pseudo code for the following operations on a given binary search tree (BST) with a height of n. Analyze the time complexity (in terms of Big-Oh notation) of your code for each following
a) Merging with another BST of height n.
b) Finding the kth smallest element in the BST.
c) Balancing the BST.
d) Finding elements within a specified value range.

a)
```
def Merge(tree1, tree2)
    for each_node in tree2:
        tree1.insert(node)
    return tree1
```

b)
```
def KthSmallestElement(root, k)
    treeStack = []
    count = 0
    while root is not null or stack is not empty:
        while root is not null:
            treeStack.push(root)
            root = root.left.
        root = treeStack.pop()
        count += 1
        if count == k:
            return root.value
        root = root.right
    return None.
```

c)
```
def balance(root):
    nodes = inOrderTraversal(root)
    return sortedArray(nodes)

def inOrderTraversal(node):
    if node is null:
        return []
    return inOrderTraversal(node.left) + [node] + inOrderTraversal(node.right)
```

```
def sortedArray(arr)
    if arr is empty:
        return null
    mid = len(arr) // 2
    root = TreeNode(arr[mid])
    root.left = sortedArray(arr[:mid])
    root.right = sortedArray(arr[mid+1:])
    return root
```

d)
```
def findElements(root, low, high)
    elements = []
    findInRange(root, low, high, elements)
    return result

def findInRange(node, low, high, elements)
    if node is null:
        return
    if node.value >= low:
        findInRange(node.left, low, high, elements)
    if low <= node.value <= high
        elements.append(node.value)
    if node.value <= high:
        findInRange(node.right, low, high, elements)
```

Time complexity analysis and executable code in pseudo_python.py.