CSE 321 Homework 1

Answers

1-) Answer in detail the questions that are shown below by using asymptotic notations, yes / no answers and plagiarisation from the web will not be accepted.

a) Is it meaningless to say:“The running time of algorithm A is at least $O(n^2)$”?

The running time of algorithm A is at least $O(n^2)$ is meaningless. Formally, $O(n^2)$ states at most, covering upper bound. If an algorithm’s running time is $O(n^2)$, it means the algorithm is at most quadratic. So stating an upper bound as a lower bound by “at least” is meaningless.

b) Are the following true?

i) $2^{n+1} = O(2^n)$?

$2^{n+1} = 2^n . 2 \Rightarrow 2.2^n \leq c. 2^n$ where c=2, for all $n \geq 0$. Therefore, this statement is true.

ii) $2^{2n} = O(2^n)$?

$2^{2n} = 2^n . 2^n \Rightarrow 2^n . 2^n \leq c. 2^n$, there is no such a constant for the c that provides the equation, so this statement is not true.

c) Let f (n) and g(n) be asymptotically nonnegative functions. Is that equation: $\max(f(n), g(n)) = \Theta(f(n) + g(n))$ true?

By the basic definition of Θ() notation:
If $\max(f(n), g(n))$ is $\Theta(f(n) + g(n))$ then $\max(f(n), g(n))$ always should be between $C_1 . (f(n) + g(n))$ and $C_2 . (f(n) + g(n))$. Then;

Let say $f(n) > g(n)$; then $f(n) + g(n) < 2 . f(n) \Rightarrow (f(n) + g(n)) / 2 < f(n)$ and f(n) was our max (f(n), g(n)).
So $\max(f(n), g(n)) > C_1 . (f(n) + g(n))$ is always true when $C_1 \leq 1/2$ (1)

Let say $f(n) > g(n)$; then $f(n) < f(n) + g(n)$, since g(n) is non negative function $\Rightarrow f(n) < f(n) + g(n)$ and f(n) was our max (f(n), g(n)).
So $\max(f(n), g(n)) < C_2 . (f(n) + g(n))$ is always true when $C_2 \geq 1$ (2)

The statement is true since condition (1) and (2) have been provided.

2-) In each of the following situations, indicate whether f ϵ O(g), or f ϵ Ω(g), or both (in which case f ϵ Θ(g)).

	f(n)	g(n)
a)	$n^{1.01}$	$n\log^2 n$
b)	n!	2^n
c)	$\sqrt{n}$	$(\log n)^3$
d)	$n2^n$	3^n
e)	$\sum^{n}_{i=1} i^k$	n^{k+1}
f)	2^n	2^{n+1}
g)	$n^{1/2}$	$5^{\log_2 n}$
h)	log2n	log3n

Answers at the end of the paper.

3-) List the following functions according to their order of growth and prove your assertions.

Logn, $\sqrt{n}$, n + 10, 10^n, 100^n, n^2logn, $32^{\log n}$, n^6

Explanations attached to the end of the file.

Logn < $\sqrt{n}$ < n + 10 < n^2logn < $32^{\log n}$ < n^6 < 10^n < 100^n

4-) Analyze the complexity in time (big -Oh notation) of the following operations at a given binary search tree (BST) that has height n:

a) FindMin.
b) Searching a node.
c) Delete a leaf node.
d) Merging with another BST that has height n.

a)

```
find_min(node):

    current = node
    while(current.left is not None):
        current = current.left
    return current.data
```

If the height is n, so there is 2^n -1 nodes. On the worst case the tree hasn't any right branch so the worst case of the algorithm is $O(2^n)$. If the tree is balanced then there could be n left nodes. So that the worst case become O(n) on balanced trees.

b)

```
search(root,key):

    if root is None or root.val == key:
        return root
    if root.val < key:
        return search(root.right,key)
    return search(root.left,key)
```

If the height is n, so there is 2^n -1 nodes. On the worst case the item could be the last item so the worst case of the algorithm is O(2^n). If the tree is balanced then the algorithm finds the node at most n step since n is the height. So that the worst case become O(n) on balanced trees.

c)

```
deleteNode(root, key):
    if root is None:
        return root
    if key < root.key:
        root.left = deleteNode(root.left, key)
    elif(key > root.key):
        root.right = deleteNode(root.right, key)
    else:
        if root.left is None :
            temp = root.right
            root = None
            return temp
        elif root.right is None :
            temp = root.left
            root = None
            return temp
        temp = minValueNode(root.right)
        root.key = temp.key
        root.right = deleteNode(root.right , temp.key)
    return root
```

If the height is n, so there is 2^n -1 nodes. On the worst case the node could be the last node and the tree must reorganize all nodes again so the worst case of the algorithm is O(2^n). If the tree is balanced then the algorithm deletes the node at most n step since n is the height. So that the worst case of deletion become O(n) on balanced BST.

d)

merge_trees(tree1, tree2):

convert tree1 and tree2 to ordered lists

```
tree3 = new tree()
    i = 0;
    j = 0;
    while (i < m && j < n)
        if (tree1[i] < tree2[j]):
            tree3.Add(tree1[i])
            i++
        else:
            tree3.Add(list2[j])
            j++
    while (i < m)
        tree3.Add(tree1[i])
        i++
    while (j < n):
        tree3.Add(tree2[j])
        j++
    return list3
```

If the height of tree1 is n, so there is 2^n -1 nodes and the number of nodes in tree2 is 2^m -1 since the height of the tree2 is m. Converting the trees to ordered lists takes O(n) and O(m) times by using inorder traversal. Lastly merging ordered lists takes O(m + n) times by using two cursor that represent current inserted items from the lists. So, the worst case of the algorithm becomes O(n) + O(m) + O(m + n) = O(m+n).

5-) Find the complexity in time (big -Oh notation) of the following program.

```
void function(int n)
{
    int count = 0;
    for (int i = 2; i <= n; i++)
        if  (i % 2 == 0)
        {
                count++;
        }
        else
        {
                i = (i – 1) * i;
        }
}
```

The for loop can be modified as the following and the both loop is exactly the same with together:

for (int i = 2; i <= n; i^2+i)

count++;

Time Complexity of a loop is considered as O(LogLogn) if the loop variables is reduced / increased exponentially by a constant amount. In this statement; the cursor (i) increases exponential as it increases i^2 + i on each step. So time complexity of the code is O(LogLogn). Examine the following explanation if you want to see how it becomes true. Remember that; we can ignore +i part because i^2 dominates the + i part.

In the loop, i takes values 2, 2^k, $(2^k)^k$ = ${}_2k^2$, ..., ${}_2k^{logk(log(n))}$. The last term must be less than or equal to n, and we have ${}_2k^{logk(log(n))}$= $2^{log(n)}$= n, which completely agrees with the value of our last term. So there are in total $log_k(log(n))$ many iterations, and each iteration takes a constant amount of time to run, therefore the total time complexity is O(log(log(n))).

Q-2) ①

a) $n^{1.01}$ $\quad n\log^2 n \Rightarrow \lim_{n\to\infty} \frac{n^{1.01}}{n\log^2 n} = \lim_{n\to\infty} \frac{n^{0.01}}{\log^2 n} \Rightarrow \lim_{n\to\infty} \frac{0.01 \cdot n^{-0.99}}{\frac{2}{n}\log n}$

$$\Rightarrow \lim_{n\to\infty} \frac{0.01\, n^{0.01}}{2\log n} = \lim_{n\to\infty} \frac{10^{-4}\, n^{-0.99}}{\frac{2}{n}} = \lim_{n\to\infty} \frac{10^{-4}\, n^{0.01}}{2}$$

this limit is equal to $+\infty$

$\boxed{\text{So } n^{1.01} = \Omega(n\log^2 n)}$

b) $n!$ $\quad 2^n$

let say S(1)
n-4 term

$$n! = \overbrace{n\cdot(n-1)\cdots}^{\text{let say } S(1),\ n-4 \text{ term}} \underbrace{4\cdot3\cdot2\cdot1}_{24}$$

$$2^n = \underbrace{2\cdot2\cdot2\cdots\cdots}_{(n-4)\text{ times, let say } S(2)} \underbrace{2\cdot2\cdot2\cdot2}_{16}$$

$\Rightarrow$ Each term in S(1) is bigger then terms of S(2) So multiplication of terms in S(1) bigger then multip. of terms S(2). (1)

$\Rightarrow$ 24 is bigger then 16 (2)

$\Rightarrow$ Statement 1 x 2 [redacted] is equal to results of both n! and 2^n and n! dominates 2^n in both statement. So

$\boxed{n! = \Omega(2^n) \text{ where } n \geq 4}$

c) $\sqrt{n}$ $\quad (\log n)^3$

$\Rightarrow$ let say $n = 2^k$ where k is even;

$= \sqrt{2^k}$ vs $(\log_2 2^k)^3$

$= 2^{k/2}$ vs k^3 and $\lim_{k\to\infty} \frac{2^{k/2}}{k^3} = \infty$

$\boxed{\text{So, } \sqrt{n} = \Omega((\log n)^3)}$

d-) $n \cdot 2^n$ $\quad 3^n$ $\quad \lim_{n\to\infty} \frac{n2^n}{3^n} = \lim_{n\to\infty} n \cdot \left(\frac{2}{3}\right)^n$ => the question can be solved by using L' Hopital rule. On the other hand we know that Exponential functions grows faster than polynomial functions ②

So that $\left(\frac{2}{3}\right)^n$ dominates n

and that means the limit goes 0 faster than ∞. Therefore the answer is $\boxed{n2^n = O(3^n)}$

e-) $\sum_{i=1}^{n} i^k$ $\quad n^{k+1}$

$$\Rightarrow \sum_{i=1}^{n} i^k = \underbrace{\overbrace{1^k + 2^k + 3^k + \cdots + n^k}^{S(1)}}_{n \text{ term}}$$

$$n^{k+1} = n \cdot n^k = \underbrace{\overbrace{n^k + n^k + n^k + \cdots + n^k}^{S(2)}}_{n \text{ term}}$$

Since $n>0$; each term in $S(2)$ is bigger than all the terms of $S(1)$ and number of the sets are equal. Therefore, Sum (S1) must be less than sum (S2). So the answer is $\boxed{\sum_{i=1}^{n} i^k = O(n^{k+1})}$

f-) 2^n $\quad 2^{n+1}$ $\quad$ => Bonus :) $\quad c_1 \cdot 2^{n+1} \leq 2^n \leq c_2 \cdot 2^{n+1}$

$\boxed{\text{So } 2^n = \Theta(2^{n+1})}$

g-) $n^{1/2}$ $\quad 5^{\log_2 n}$ => $5^{\log_2 n} = n^{\log_2 5} \cong n^{2.32}$

$\Rightarrow \lim_{n\to\infty} \frac{n^{\frac{1}{2}}}{n^{2.32}} = 0$ $\quad \boxed{\text{So } n^{1/2} = O(5^{\log_2 n})}$

h-) $\log 2n$ $\quad \log 3n$ $\quad$ => Bonus 2 :) $\quad c_1 \cdot \log 2n \leq \log 3n \leq c_2 \cdot \log 2n$

$\boxed{\text{So } \log 2n = \Theta(\log 3n)}$

$\Rightarrow$ let say $n = 2^{2k}$ $\Rightarrow$ (1) $\{\log n$ vs $\sqrt{n}\}$

$\log 2^{2k}$ $\quad$ $\sqrt{2^{2k}}$

$= 2k$ $\quad$ $2^k \Rightarrow \lim_{k\to\infty} \frac{2k}{2^k} = 0$ so that;

$\boxed{\sqrt{n} > \log n}$

$\Rightarrow$ (2) $\sqrt{n}$ $\quad$ $n+10$

$= 2^k$ $\quad$ $2^{2k}+10 \Rightarrow$ We already compare them on q1-part b, therefore;

$\boxed{n+10 > \sqrt{n}}$

$\Rightarrow 32^{\log n}$ $\quad$ n^6 $\Rightarrow 32^{\log n} = n^{\log 32} = n^5$

$\lim_{n\to\infty} \frac{n^5}{n^6} = 0$ so that $\boxed{n^6 > 32^{\log n}}$

$\Rightarrow 10^n$ $\quad$ $100^n \Rightarrow$ Same with q1-part b $= 10^n$ vs 10^{2n}. so that

$\boxed{100^n > 10^n}$

$\Rightarrow n+10$ $\quad$ $10^n \Rightarrow \lim_{n\to\infty} \frac{n+10}{10^n} = \lim_{n\to\infty} \frac{1}{10 \cdot n^3} = 0$ so; $\boxed{10^n > n+10}$

$\Rightarrow$ let say $n = 2^k$ $\Rightarrow$ (1) $n^2 \log n$ $\quad$ $n+10$

$2^{2k} \cdot \log 2^k$ $\quad$ $2^k + 10 = k \cdot 2^{2k}$ vs $2^k + 10$; we did the proof several times so;

$k \cdot 2^{2k} > 2^k + 10 \Rightarrow \boxed{n^2 \log n > n+10}$

(2) $n^2 \log n$ $\quad$ 10^n

$(2^k)^2 \cdot k$ $\quad$ 10^{2^k} $\Rightarrow 2^{2k} \cdot k$ vs 10^{2^k}, both base and power of 10^{2^k} term are bigger than 2^{2k} so; $\boxed{10^n > n^2 \log n}$

(3) $n^2 \log n$ $\quad$ $32^{\log n}$ $\Rightarrow$ $2^{2k} \cdot k$ vs $2^{2k} \cdot 2^{3k}$

$2^{2k} \cdot k$ $\quad$ $(2^k)^5$

$\lim_{k\to\infty} \frac{2^{2k} \cdot k}{2^{2k} \cdot 2^{3k}} = \lim_{k\to\infty} \frac{k}{2^{3k}} = 0$, so; $\boxed{32^{\log n} > n^2 \log n}$

$\Rightarrow n^6$ $\quad$ $10^n \Rightarrow n^6$ $\quad$ 10^n

$\lim_{n\to\infty} \frac{n^6}{10^n} = \frac{6n^5}{10^n \cdot \ln 10} = \; = 0$ so $\Rightarrow \boxed{10^n > n^6}$