CSE373 Assignment 1

Aditya Balwani
SBUID: 109353920

February 25, 2016

1 Question 1

1.1 Part 1

We need to prove that

$$\frac{3n^3 + 9n^2 + n - 1}{n^3} \leq c \text{ for all } n \geq 1$$

We know that

$$3 + 9 + 1 - 1 = \frac{3n^3 + 9n^3 + n^3 - 1n^3}{n^3} > \frac{3n^3 + 9n^2 + n - 1}{n^3} \text{ for all } n \geq 1$$

$$12 > \frac{3n^3 + 9n^2 + n - 1}{n^3} \text{ for all } n \geq 1$$

So let constant be 12

$$\frac{3n^3 + 9n^2 + n - 1}{n^3} \leq 12 \text{ for all } n \geq 1$$

Thus $f(n) = O(g(n))$

1.2 Part 2

We need to prove that

$$\frac{5n\log_2 n + 8n - 200}{n\log_2 n} \leq c \text{ for all } n \geq 1$$

We know that

$$5 + 8 - 200 = \frac{5n\log_2 n + 8n\log_2 n - 200n\log_2 n}{n\log_2 n} > \frac{5n\log_2 n + 8n - 200}{n\log_2 n} \text{ for all } n \geq 1$$

$$187 > \frac{5n\log_2 n + 8n - 200}{n\log_2 n} \text{ for all } n \geq 1$$

So let constant be -187

$$\frac{5n\log_2 n + 8n - 200}{n\log_2 n} \leq -187 \text{ for all } n \geq 1$$

Thus $f(n) = O(g(n))$

2 Question 2

The correct order of growth rate is :

- $O(logn)$
- $O(\sqrt{n})$
- $O(n)$
- $O(nlogn)$
- $O(n^1.9)$
- $O(n^2)$
- $O(n^3)$
- $O(2^n)$
- $O(n!)$

3 Question 3

- $O(n^1.1)$
- $O(n^3)$
- $O(n^3)$
- $O(2^n)$
- $O(n^2)$

4 Question 4

$$T(1) = 1$$

$$T(n) = 1 + n^2 + 2T(\frac{n}{2})$$

$$= 1 + n^2 + 2((\frac{n}{2})^2 + 1 + 2T(\frac{n}{4}))$$

$$= 1 + n^2 + 2(\frac{n^2}{2^2})^2) + 2 + 4T(1 + \frac{n}{4}^2 + 2T(\frac{n}{4}))$$

Using this pattern we see that :

$$= (1 + 2 + 4...) + (n^2 + 2\frac{n^2}{2^2} + 4\frac{n^2}{4^2} + 2^iT\frac{n}{2^i}$$

The series $1 + \frac{1}{2} + \frac{1}{4}$ .. will gives us 2 $= (2^i - 1) + 2n^2 + 2^iT\frac{n}{2^i}$

The base case is T(1) = 1

Using this we find i

$\frac{n}{2^i} = 1 => log_2 n = i$

$$T(n) = 2^{log_2 n} - 1 + 2n^2 + 2^iT(1)$$

$$= 2^{log_2 n} - 1 + 2n^2 + 2^{log_2 n}c$$

$$= n - 1 + 2n^2 + n$$

This means that this merge sort is $O(n^2)$

5 Question 5

5.1 Part 1

5.2 Part 2

$$
\begin{aligned}
T(n) &= 4T(\frac{n}{2}) + n^2\sqrt{n} \\
&= 4(T(\frac{n}{4}) + (\frac{n}{2})^2\sqrt{\frac{n}{2}}) + n^2\sqrt{n} \\
&= 4T(\frac{n}{4}) + 4(\frac{n}{2})^2\sqrt{\frac{n}{2}} + n^2\sqrt{n} \\
&= 4(T(\frac{n}{8}) + (\frac{n}{4})^2\sqrt{\frac{n}{4}}) + 4(\frac{n}{2})^2\sqrt{\frac{n}{2}} + n^2\sqrt{n} \\
&= 4T(\frac{n}{8}) + 4(\frac{n}{4})^2\sqrt{\frac{n}{4}} + 4(\frac{n}{2})^2\sqrt{\frac{n}{2}} + n^2\sqrt{n} \\
&= 4^iT(\frac{n}{2^i}) + in^2.5 \qquad \text{Using this we find i} \\
&\frac{n}{2^i} = 1 => log_2 n = i \\
&\text{The base case is T(1) <c} \\
&\text{Plugging in the value of i} \\
T(n) &= 4^{log_2 n}T(1) + log_2 n n^{2.5} \\
&= 4^{log_2 n}c + n^{2.5}log_2 n
\end{aligned}
$$

This means that this is $O(n^{2.5}log_2 n)$

6 Question 6

```
def compareLists (list1, list2, n):
        list1 = mergeSort(list1)
        list2 = mergeSort(list2)

        for(i=0;i<n;i++):
                if(list2[i] != list2[i])
                        return True;
```

The running time of the merge sort is $O(nlogn)$ and the comparison is $O(n)$ So the running time of the program is $O(nlogn)$

7 Question 7

In a pivoted array, all elements to the left of the pivot are less than the pivot and all to the right are greater. In this array 42, 55 and 73 could have been the pivots

8 Question 8

findIndexOfLargestElement was defined in class

```
def flip(array, n):
        start = 0
        while start < n:
                temp = array[start]
                array[start] = array[n]
                array[n] = temp
                start+=1
                n-=1
```

```
def pancakeSort(array, n):
        for size in reversed(range(n), 1):
                maxIndex = findIndexOfLargestElement(array)
                if(maxIndex != size-1):
                        flip(array, maxIndex)
                        flip(arr, size-1)
```

The total time complexity for this is $O(n^2)$

9 Question 9

One possible way to do this is to do a binary search on the 100 floors to see where the jar breaks. The max number of drops for that is 7 we'll end up breaking more than 2 jars. Another start at some value i, and then double it if it doesn't break and if it does break then we know it lies between the original x and the incremented value so we start at x and keep incrementing by 1 until it breaks. For a 100, thats not the optimal solution because the worse case is 19 drops. However we noticed that if the egg breaks in an early drop then we have more drops to work with to not hit the worse case. So by using algebra we get : $x + (x-1) + (x-2) .. >= 100$ $x(x+2) >= 100$ Solving for x gives us 14. Which means we start droping at the 14th floor and then if it breaks we start 1 and go up till 14 till we see the breaking point and if it doesn't break then we move up another 13 floors and repeat.

Algorithm for N floors :

```
def glassJar(x):
        increment = x * (x + 1) / 2
        floor = x
        while floor < x:
                if jar breaks on drop[floor]:
                        floor = floor + 1
                        while(floor < floor + increment - 1):
                                if jar breaks on drop[floor]:
                                        return floor
                                floor = floor + 1
                floor = floor + increment - 1
                increment = increment - 1
        return x
```