Exercise 1 True/False Questions (2+2+2+2+2+2=12 Marks)

Decide whether each of the following statements is True or False. Justify your answer.

a) If $lim_{n\to\infty}(\frac{f(n)}{g(n)}) = \boxed{c}$ where c>0, then $f(n) = O(g(n))$ and $g(n) = O(f(n))$.

$f(n) = \Theta(g(n)) \rightarrow O(g(n))$

$g(n) = \Theta(f(n)) \rightarrow O(f(n))$ True

b) $(n+1)! = \Theta(n!)$

$C_2 n! \leq (n+1)! \leq C_1 n!$ $\frac{C_2 n!}{n!} \leq \frac{n!(n+1)}{n!} \leq \frac{C_1 n!}{n!}$

$C_2 \leq n+1 \leq C_1$ False, n is not bounded

c) If $f(n) = O(g(n))$ and $g(n) = O(f(n))$ then $f(n) = g(n)$.

$f(n) \leq g(n)$ and $g(n) \leq f(n)$

So, yes $f(n) = g(n)$ True

d) The best case running time of Merge Sort is different from its worst case running time.

False

e) A sorted array is the best case input for the Quick Sort algorithm.

False

f) The worst case running time of Quick Sort is asymptotically the same as its average case running time.

False, average case is $n\log(n)$ worst case is n^2

Second loop:

loop invariant: At the start of each iteration i, we Know that buckets[0--i-1] is sorted.

Initialization: At each iteration i we sort bucket[i], therefore loop invariant is maintained for the next iteration

Termination: After loop terminates, i=K so buckets[0--K-1] are sorted.

we proved that each bucket is sorted with respect to the others and also each bucket itself is sorted so returning the concatenation of the buckets results in a sorted array.

Exercise 2 Bucket Sort (3+5+4+4=16 Marks)

Consider the following sorting algorithm referred to as *Bucket Sort*. In this exercise, assume that all arrays are 0-indexed and that the input array A is composed of only positive numbers.

```
BucketSort(A, k)
buckets ← new array of k empty arrays   [ [ ] , [ ], - - ]
max ← maximum value in array A
for i = 0 to length(A) − 1 do
    Insert A[i] in buckets[$\lfloor \frac{A[i]}{max} * (k-1) \rfloor$]
end
for i = 0 to k − 1 do
    Sort buckets[i]
end
return the concatenation of buckets[1], . . . , buckets[k]
```

a) Trace the operation of bucket sort given $A = [7, 2, 6, 1, 4, 3]$ and $k = 5$.

intial

max = 7

buckets = [[ ],[ ],[ ],[ ],[ ]]

i=0, insert 7 in bucket 4

buckets = [[ ],[ ],[ ],[ ],[7]]

i=1, insert 2 in bucket 1

buckets = [[ ],[2],[ ],[ ],[7]]

i=2, insert 6 in bucket 3

buckets = [[ ],[2],[ ],[6],[7]]

i=3, insert 1 in bucket 0

buckets = [[1],[2],[ ],[6],[7]]

i=4, insert 4 in bucket 2

buckets = [[1],[2],[4],[6],[7]]

i=5, insert 3 in bucket 1

buckets = [[1],[2,3],[4],[6],[7]]

buckets = [[1], [2,3] ,[4] ,[6] ,[7]]

first loop

b) Prove that **BucketSort** is correct.

loop invariant: At the start of each iteration we Know that for any j where 0<j<K (K is number of buckets) the minimum element in bucket j is strictly greater than maximum element of bucket j-1.

Initialization: Before the first loop all the buckets are empty, so loop invariant is trivially satisfied.

Maintenance: The loop invariant is maintained at each iterative step, because we insert A[i] in the right bucket by Knowing the ratio of A[i] with respect to max(A) and then we insert by multiplying by the max index.
If there was some A[i] and A[j] where A[j] > A[i] and A[j] is in a bucket < the bucket of A[i], this is impossible because the ratio of $\frac{A[j]}{max} > \frac{A[i]}{max}$ contradiction.

Termination: upon termination we will have all elements of Array A inserted in the correct buckets maintaing loop invariant.

c) What is the best case scenario of **BucketSort**? What is the best case time complexity? Show all of your workout.

Best cas depends on the sorting function Sort that sort each bucket. If it is merge sort then no difference.

d) What is the worst case scenario of **BucketSort**? What is the worst case time complexity? Show all of your workout.

assuming sort is bubble sort

line	cost	times
2	C_1	1
3	C_2	1
4	C_3	A+1
5	C_4	A
7	C_5	K+1
8	C_6	KX
10	C_7	1

sorting of bucket → 8

← X is runing time of sorting bucket

$$C_1(1)+C_2(1)+C_3(A+1)+C_4(A)+C_5(K+1)+C_6(Kx)+C_7$$

$$= C_1+C_2+AC_3+C_3+AC_4+KC_5+C_5+KxC_6+C_7$$

same for best case? →

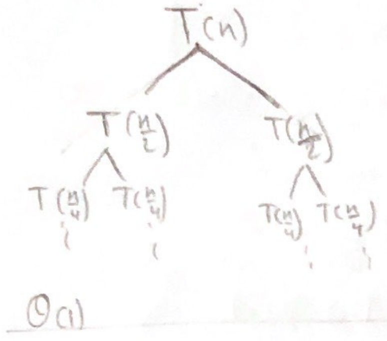
$$= (C_1+C_2+C_3+C_5+C_7)+C_4A+C_5K+C_6(KX)$$

Exercise 3 Recurrence Trees (8+4=12 Marks)

a) Obtain a **tight bound** Θ on the running time of the following recurrence by using the recursion tree method. Draw the recursion tree and show all of your workout.

$$T(n) = 2T(\frac{n}{2}) + n\ log_2(n)$$

$$T(n) = \begin{cases} 2T(\frac{n}{2}) + n\log_2(n), & \text{otherwise} \\ \Theta(1) & , n=1 \end{cases}$$



height	size	Cost
$i=0$	n	$(2^0)n\log_2(n)$
$i=1$	$\frac{n}{2}$	$(2)\frac{n}{2}\log_2(\frac{n}{2})$
$i=2$	$\frac{n}{4}$	$(4)\frac{n}{4}\log_2(\frac{n}{4})$
	$\vdots$	$\vdots$
	$\frac{n}{2^i}$	$(2^i)\frac{n}{2^i}\log_2(\frac{n}{2^i})$
$i=h$	$\frac{n}{2^h}$	$(2^h)\frac{n}{2^h}\log_2(\frac{n}{2^h})$

$$C.L = n\log_2\left(\frac{n}{2^h}\right)$$
$$= n\left[\log_2(n) - \log_2(2^h)\right]$$
$$= n\left[\log_2(n) - h\right]$$
$$= n\left[\log_2(n) - \log_2(n)\right]$$
$$C.L = 0$$

$$\frac{n}{2^h} = 1$$
$$n = 2^h$$
$$\boxed{h = \log_2(n)}$$

$$Cost = \sum_{i=0}^{i=h-1} n\log_2\left(\frac{n}{2^i}\right) + \cancel{C.L}_{=0}$$
$$= \sum_{i=0}^{i=\log_2(n)-1} n\left[\log_2(n) - i\right]$$
$$= \sum_{i=0}^{i=\log_2(n)-1} n\log_2(n) - \sum_{i=0}^{i=\log_2(n)-1} i$$
$$= n\log_2(n)\left[\log_2(n)\right] - \frac{(\log_2(n)-1)(\log_2(n))}{2}$$
$$Cost = n\log_2^2(n) - \frac{\log_2^2(n)}{2} + \frac{\log_2(n)}{2}$$

b) Can the above recurrence be solved using the master theorem? Justify your answer.

$a=2,\ b=2,\ f(n) = n\log_2(n)$

$C.R = f(n) = n\log_2(n)$

$C.L = n^{\log_b a} = n^{\log_2(2)} = n$

$C.R > C.L$ looks like Case 3

$f(n) = \Omega(n^{1+\varepsilon})$

$\cancel{n}\log_2(n) \geq \cancel{n}n^e$

$\log_2(n) \geq n^e$

Falls in a gap between Case 3 and Case 2

Exercise 4 Master Theorem (3+3+3+3=12 Marks)

Can the following recurrences be solved using the master theorem? If yes, solve them. If not, explain why. Show all of your workout.

a) $T(n) = 16T(\frac{n}{4}) + n^3$

$a = 16, b = 4, f(n) = n^3$

$C.R = f(n) = n^3$

$C.L = n^{\log_b a} = n^{\log_4 16} = n^2$

$C.R > C.L$ looks like Case 3

$f(n) = \Omega(n^{2+\varepsilon})$

$n^3 \leqslant n^2 n^{\varepsilon}$, $\varepsilon > 0$

$\varepsilon = 1$

$n^3 \leqslant n^3$

$af(\frac{n}{b}) \leqslant cf(n)$, $0 < c < 1$

$16 (\frac{n}{4})^3 \leqslant cn^3$

$\cancel{16} \frac{\cancel{n^3}}{\cancel{16}(4)} \leqslant c\cancel{n^3}$

$c \geq \frac{1}{4} \rightarrow c = \frac{1}{4}$

$T(n) = \Theta(n^3)$

b) $T(n) = 5T(\frac{n}{2}) + n^2$

$a = 5, b = 2, f(n) = n^2$

$C.R = n^2$

$C.L = n^{\log_b a} = n^{\log_2(5)} = n^{2.3}$

$C.L > C.R$ looks like Case 1

$f(n) = O(n^{2.3-\varepsilon})$

$n^2 \leqslant \frac{n^{2.3}}{n^{\varepsilon}}$

$\varepsilon = 0.1$

$n^2 \leqslant n^{2.2}$

$T(n) = \Theta(n^{\log_2(5)})$

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

$a = 1, b = 4, f(n) = C$

$C.R = C$

$C.L = n^{\log_b a} = n^{\log_4 1} = n^0 = 1 = C$

$C.R = C.L$

$T(n) = \Theta(C\log(n)) = \Theta(\log(n))$

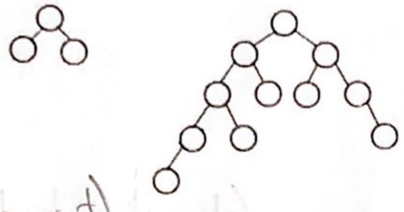
d) $T(n) = 2T(\frac{n}{2}) + T(\frac{n}{4}) + n$

Con not be solved using master theorem due to eneven splitting

Exercise 5 Divide and Conquer Algorithm Design (8+10=18 Marks)

a) Design a divide and conquer algorithm for computing the maximum number of levels in a binary tree.

For example: for the left tree below, the maximum number of levels is 2 while for the right tree, the maximum number of levels is 5. The algorithm must return 0 for empty trees and 1 for a single-node trees, respectively. You can use express your algorithm in English or using Pseudo Code. **Write the recurrence expressing the running time of your proposed algorithm.**



```
Compute_height (Node root)
{
    if (node not null)
    {
        left = compute_height(root.left)
        right = compute_height(root.right)
        return max(left, right) +1
    }
    return 0
}
```

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

b) Given a 2D map with countries represented as points by their x and y coordinates, describe a divide and conquer algorithm to find the closest two countries. Your algorithm must be in $O(n\ log^2(n))$. You can express your algorithm in English or using Pseudo Code. **Write the recurrence expressing the running time of your proposed algorithm.**