Homework Assignment 1

CS 430 Introduction to Algorithms

Name : Karthik Mahesh

CWID : A20383027

1.Solution :

We have 2 eggs and n floors.

Let's assume that, in the best strategy, the worst case is 'i'.

So we start at the i^{th} floor. Now if egg breaks, we have to check all the floors from 1 to i. If the egg does not break, we have to check $(i + (i-1))^{th}$ floor. We are using i+ i-1 instead of i+i because we have already checked 1 floor and we do not want to exceed 'i' attempts.

Again if it does not break, we have to check $(i+(i-1)+(i-2))^{th}$ floor .

Similarly we go on till the nth floor.

As we have n floors, sum of this series should not be less than n.

So we get

i + (i-1) + (i-2) + …. + 2 + 1 >= n

now we need to find the minimum value of i such that

i(i+1) / 2 >= n

i.e, $i^2 + i - 2n = 0$

Substituting the value for n, we can solve for i.

When we start from i^{th} floor, we can determine the highest floor of a building of N floors from which the egg will survive the fall with the least number of tests.

2.Solution :

For n = 1 to 4, algorithm with running time $200n^3$ runs faster than an algorithm whose running time is 1.5^n^2.

For n > 4, algorithm with running time $200n^3$ runs slower than an algorithm whose running time is 1.5^n^2.

So '**<u>5</u>**' is the smallest value of n such that an algorithm, whose running time is 200 n^3, runs faster than an algorithm whose running time is 1.5^n^2 on the same machine.

When n= 5, algorithm with running time 200n^3 will be 25000 and 1.5^n^2 will be 25251.

3.Solution :

(a) Not always True
$f(n) = O(g(n))$ 'does not' imply $g(n) = O(f(n))$.

Proof : proof by contradiction

Suppose $f(n) = n$ and $g(n) = 2n$

Here $n <= 2n$ ($f(n) <= g(n)$) is true

but $2n <= n$ ($g(n) <= f(n)$) is false.

Hence $f(n) = O(g(n))$ 'does not' imply $g(n) = O(f(n))$.

(b) True
$f(n) = O(g(n))$ implies $\log f(n) = O(\log g(n))$.

$f(n) = O(g(n))$ implies $g(n)$ is always greater than or equal to $f(n)$.

Assume
$f(n) = 1$
$g(n) = 2$

$1 <= 2$
Also $\log 1 <= \log 2$
Which means $\log f(n) = O(\log g(n))$.

For any value of $f(n)$ and $g(n)$, where $f(n)<=g(n)$
$\log(f(n)) <= \log(g(n))$

So, $f(n) = O(g(n))$ implies $\log f(n) = O(\log g(n))$.

(C) Not always true
$f(n) != O((f(n))^2)$ when $f(n) = 1/n$

Proof : Proof by contradiction

When $f(n) = 1/n$
$(f(n))^2 = (1/n)^2 = 1/n^2$

Assume $n = 2$
$f(n) =1/n = ½$,
$(f(n))^2 = 1/n^2 = ¼$
$½ > ¼$
$f(n) > (f(n))^2$
i.e, $f(n) != O((f(n))^2)$.

(d) True

$f(n) + g(n) = \Theta(\max(f(n), g(n)))$.

$f(n) <= \max(f(n), g(n))$ …. 1
$g(n) <= \max(f(n), g(n))$ …. 2
Adding 1 and 2 we get
$f(n) + g(n) <= 2 \max(f(n), g(n))$
Therefore,
$f(n) + g(n) = \mathbf{O}(\max(f(n), g(n)))$ with $c=2$ ….. 3

$f(n) + g(n) >= f(n)$ … 4
$f(n) + g(n) >= g(n)$ … 5
From 4 and 5, we can conclude that
$f(n) + g(n) >= \max(f(n), g(n))$ since f() and g() are non-negative increasing functions.
Therefore,
$f(n) + g(n) = \mathbf{\Omega}(\max(f(n), g(n)))$ ….. 6

From 3 and 6, we can conclude that
$f(n) + g(n) = \Theta(\max(f(n), g(n)))$

References used :

1. CLRS3 text book
2. Design and Analysis of Algorithms text book