(1) Prove the following statement using method of induction.
For $\forall n \geq 1$, & $n, k \in R^+$, $\lceil n/k \rceil = \left\lfloor \frac{n+k-1}{k} \right\rfloor$

ans:- Given:- For $\forall n \geq 1$ & $n, k \in R^+$, $\lceil n/k \rceil = \left\lfloor \frac{n+k-1}{k} \right\rfloor$

Base step:- let $n = 1$, then

$$\lceil 1/k \rceil = \left\lfloor \frac{1+k-1}{k} \right\rfloor \Rightarrow \lceil 1/k \rceil = \lfloor k/k \rfloor$$

$$\Rightarrow \lceil 0.... \rceil = \lfloor 1 \rfloor$$

$$\Rightarrow 1 = 1 \text{ (proved)}.$$

Induction hypothesis :- for $n = k$, $\lceil n/k \rceil = \lfloor n+k-1/k \rfloor$ where $n, k \in R^+$.

Induction step :- let $n = k+1$, then

$$\text{LHS} :- \lceil n/k \rceil = \lceil k+1/k \rceil = \lceil 1 + 1/k \rceil = 2.$$

$$\text{RHS} :- \left\lfloor \frac{n+k-1}{k} \right\rfloor = \left\lfloor \frac{k+1+k-1}{k} \right\rfloor = \left\lfloor \frac{2k}{k} \right\rfloor = 2.$$

So, LHS = RHS.

Hence, as induction step is true, we can say

$$\forall n \geq 1 \text{ \& } n, k \in R^+, \lceil n/k \rceil = \left\lfloor \frac{n+k-1}{k} \right\rfloor.$$

(2) Prove the following statement using method of induction.
For every non-negative integer, n, $\sum_{i=0}^{n} 3^i = \frac{3^{n+1}-1}{2}$.

ans:- Given:- non-negative integer n, $\sum_{i=0}^{n} 3^i = \frac{3^{n+1}-1}{2}$.

Base step:- let $n = 0$, then

$$\text{LHS} :- \sum_{i=0}^{0} 3^0 = 3^0 = (1)$$

$$\text{RHS} :- \frac{3^{0+1}-1}{2} = 2/2 = (1) \quad \text{So, LHS = RHS (proved)}$$

Induction hypothesis :- let $n=k$,

So, for $n=k$ $\sum_{i=0}^{k} 3^i = \frac{3^{k+1}-1}{2}$.

Induction step :- let $n=k+1$,

LHS : $\sum_{i=0}^{k+1} 3^i = \sum_{i=0}^{k} 3^i + 3^{k+1}$

$= \frac{3^{k+1}-1}{2} + 3^{k+1}$ ⟶ (From IH)

$= \frac{(3)^{k+1} - 1 + 2(3^{k+1})}{2}$

$= \frac{3^{k+1}[1+2] - 1}{2}$

$= \frac{3^{(k+1)+1} - 1}{2}$

= RHS ⟶ Hence proved.

So, for every non negative integer n $\sum_{i=0}^{n} 3^i = \frac{3^{n+1}-1}{2}$.

(3) Prove the following statement using method of Induction.

Every non-negative integer can be written as the sum of distinct powers of 2.

For ex :- $23 = 2^4 + 2^2 + 2^1 + 2^0$.

ons :- Base step :- $n=1$, which can be written as $2^0 = 1$.

So, $1 = 2^0$, $2 = 2^1$ & $3 = 2^0 + 2^1$

So, Induction hypothesis :- $n=k$, $\forall\ k \geq 1$, k can be written as :- $2^{x_0} + 2^{x_1} + 2^{x_2} \cdots 2^{x_n}$,

Induction step :- let $n=k+1$

We need to show that, k+1 can also be written as the sum of distinct powers of 2.

Case I:- If (k+1) is odd.

$k+1 = 2^0 + k$; since k is already a sum of distinct powers of 2 (From IH),

So, for any odd number (k+1), the claim is true.

Case 2:- If (k+1) is even.

Since k+1 is even,

$\Rightarrow \frac{k+1}{2} \in Z$, strictly less than k.

So, by applying IH, $\frac{k+1}{2} \leq 2^{x_0} + 2^{x_1} + \cdots 2^{x_n}$

$$\Rightarrow 2 \times \frac{k+1}{2} \leq 2^{x_0+1} + 2^{x_1+1} + \cdots 2^{x_n+1}$$

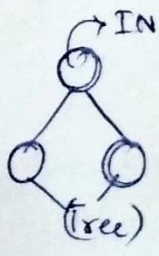
$$\Rightarrow k+1 \approx 2^{x_0+1} + \cdots 2^{x_n+1}$$

Since for both the cases, the claim is held true, it is true for any non-negative integer n, it can be written as a sum of distinct powers of 2.

(4) A Binary tree is full if every node has either two children (internal node) or no children (leaf node). Prove that in full binary tree, the number of leaves is exactly one more than the number of internal nodes.

ans:- Base step:- Let n=1,

When n=1, no. of internal nodes, then there are 1+1 = 2 leaves



Induction hypothesis :- let $n=k$,

So, $n=k$ no. of internal nodes, there are $(k+1)$ no. of leaves.

Induction step :- let $n=k+1$, no. of internal nodes

From, IH, upto 'k' no. of internal nodes we have $(k+1) \rightarrow$ no. of leaves.

So, for $(k+1)^{th}$ internal node, we have 2 leaves (As it is a full binary tree)

So, total no. of leaves :- $(k+1)+2-1 = \underline{k+2}$

(We are substracting 1, as we have considered $(k+1)^{th}$ as leaf node earlier, so, substracting it from total).

Hence proved

So, we can say in any full binary tree, the no. of leaves is exactly one more than the no. of internal nodes.

(5) Prove by mathematical induction that a decimal no. is divisible by 3 if & only if the sum of its digits is divisible by 3.

ans:- Base step :- let n be any decimal no. say, $n=12$

So, $12 \equiv 0 \pmod 3$ → as the remainder is 0,

So, 12 is divisible by 3

Also, $1+2 = 3 \equiv 0 \pmod 3$ → divisible by 3

Induction hypothesis :- PTO.

Let $n = a_0 a_1 \ldots a_k$ be the decimal integer n, such that,

$n \equiv 0 \pmod 3 \rightarrow$ divisible by 3, where $0 \leq a_i < 10$.

Also, $a_0 + a_1 + \ldots a_k \equiv 0 \pmod 3 \rightarrow$ divisible by 3

Induction step: Let $n = a_0 a_1 \ldots a_k a_{k+1}$ be the decimal integer n, ~~such that,~~

~~A~~ Expanding n, we get,

$n = a_{k+1} + a_{k-0}(10)^1 + a_{k-1}(10)^2 + \ldots + a_0(10)^{k+1}$, where $[0 \leq a_i < 10]$

$= a_{k+1} + a_{k-0}(9+1) + a_{k-1}(99+1) + \ldots a_0(9\ldots(k+1 \text{ times}) + 1)$

$= 9a_{k-0} + 99a_{k-1} \ldots + (9\text{-}k+1 \text{ times})a_0 + (a_{k+1} + a_{k-0} + a_{k-1} + \ldots a_0)$

So, $[9a_k + 99a_{k-1} + \ldots (9\ldots(k+1) \text{ times})a_0 \equiv 0 \pmod 3$ ↳ divisible by 3$]$

Also, From, IH: $a_k + a_{k-1} + \ldots a_0 \equiv 0 \pmod 3$ ↳ divisible by 3

So, $a_{k+1} + a_k + \ldots + a_0$ will only be divisible by 3, if (a_{k+1} is a multiple/divisible by 3)

Hence, for $n = a_0 a_1 \ldots a_{k+1}$ to be divisible by 3, the sum of digits needs to be divisible by 3 i.e.

$a_{k+1} + a_k + \ldots + a_0 \equiv 0 \pmod 3$, then only that no. will be divisible by 3

Hence proved.

So, a decimal no. is divisible by 3 if & only if the sum of its digits is divisible by 3.

(6) For two non-negative integers a & b, their GCD is the largest integer c, s.t. c divides both a, b (both a/c, b/c are integers). Euclid's algorithm described below can compute the GCD efficiently.

```
GCD (a,b)
If b == 0, then return a
Else
   return GCD (b, a%b)
```

Prove that Euclid's algorithm computes GCD (a,b) in time O (log (a+b)) assuming computing a%b takes 1 unit of time.

ans:- **Base Case** :- Take any value of a & b

say, a = 12, b = 4.

then 12 mod 4 = 0 → working of algorithm

4 mod 0 = 0.

4 = GCD

To proof :- Time → O (log (a+b)).

Proof :- One trick for analyzing the time complexity of Euclid's algorithm is to follow what happens over two iterations:-

(a, b)

↓

(b, a%b)

(a%b, (b%(a%b)) → 2nd iteration

Now a & b will both decrease, instead of only one, which makes the analysis easier. We can divide it into cases:-

(1) $2a <= b$

(2) $2b <= a$

(3) $2a > b$ but $a < b$

(4) $2b > a$ but, $b < a$

(5) $a == b$

Now, Case I :- $b \% (a \% b) < a$ & $2a <= b$, so b is decreased by atleast half, so (a+b) decreased by atleast 25%.

Case II :- $a \% b < b$ & $2b <= a$, so a is decreased by atleast half, so (a+b) decreased by atleast 25%.

Case III :- b will become (b-a), which is less than b/2, decreasing a+b by atleast 25%

Case IV :- a will become (a-b), which is less than a/2, decreasing a+b, by atleast 25%.

Case V :- a+b drops to 0, which is obviously decreasing (a+b) by atleast 25%.

So, every single case decreases the total (a+b) by atleast a quarter.

There's a maximum no. of times this can happen before a+b is forced to drop below 1. The total no. of steps (S) until we hit 0 must satisfy $(4/3)^S <= A+B$.

$\Rightarrow (4/3)^S = A+B$

$\rightarrow S <= \log_{[4/3]}(A+B)$ $\Rightarrow S = O(\log(A+B))$ (Hence proved)

(7) The Fibonacci numbers 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89... are recursively defined as follows:

$$F_n = \begin{cases} 0 & \text{if } n=0 \\ 1 & \text{if } n=1 \\ F_{n-1} + F_{n-2} & \text{if } n \geq 2 \end{cases}$$

Prove the following statements using mathematical induction.

(i) For all non-negative integers n, F_n is even if & only if n is divisible by 3.

(ii) $\sum_{i=0}^{n} F_i = F_{n+2} - 1$.

(iii) $F_n^2 - F_{n+1} F_{n-1} = (-1)^{n+1}$

(iv) If n is an integer multiple of m, then F_n is an integer multiple of F_m.

ans: (i) Base step: let $n = 3$,

So, $F_3 = 2$, & $2 \equiv 0 \pmod 2$ → divisible by 2 hence even

Also $n = 3 \equiv 0 \pmod 3$ → divisible by 3

Induction hypothesis: let for $n = k$, for $k \geq 3$ s.t. F_k is even & k is divisible by 3.

Induction step: let us consider for (k+1), (k+2) & (k+3) cases s.t. $k \geq 3$.

when $n = (k+1)$

then, $F_{k+1} = F_k + F_{k-1}$
$= \text{Even} + \text{Odd}$
$= \text{Odd}$

As (k-1) is not divisible by 3 So, F_{k-1} is odd.

So, for, $n = k+2$,

$F_{k+2} = F_{k+1} + F_k$

$= \text{Odd} + \text{Even}$

$= \text{Odd}$.

Now, $n = k+3$,

$F_{k+3} = F_{k+2} + F_{k+1}$

$= \text{Odd} + \text{Odd} = \text{Even}$

Also $k+3 \equiv 0 \pmod 3 \rightarrow$ divisible by 3

Hence proved.

$F(n)$ is even if & only if n is divisible by 3.

ii) Base step : $n = 0$

LHS : $\sum_{i=0}^{0} F_i = F_0 = 0$.

RHS : $F_{n+2} - 1 = F_2 - 1 = 1 - 1 = 0$

So, LHS = RHS

Induction hypothesis : for $n = k$, $\sum_{i=0}^{k} F_i = F_{k+2} - 1$.

Induction step : let $n = k+1$,

LHS : $\sum_{i=0}^{k+1} F_i = \sum_{i=0}^{k} F_i + F_{k+1}$

$= F_{k+2} - 1 + F_{k+1}$ (From IH)

$= (F_{k+2} + F_{k+1}) - 1 = \left[F_{(k+3)-1} + F_{(k+3)-2}\right] - 1$

$= (F_{k+3}) - 1$ → (From question)

$= F_{(k+1)+2} - 1 = \text{RHS}$ → Hence proved

iii) $F_n^2 - F_{n+1} F_{n-1} = (-1)^{n+1}$

Base step : $n = 0+1 = 1$ ($n \neq 0$ as F_{-1} not possible)

LHS : $F_1^2 - F_2 \cdot F_0 = 1 - 1(0) = 1$.

RHS : $(-1)^{1+1} = (-1)^2 = 1$

So, LHS = RHS → (proved)

Induction hypothesis : For $n = k$, we have,

$$F_k^2 - F_{k+1} F_{k-1} = (-1)^{k+1}$$

$$\Rightarrow F_k^2 = (-1)^{k+1} + F_{k+1} F_{k-1}$$

Induction step : Let $n = k+1$

So, ~~LHS~~ : → ~~$F_{k+1}^2 - F_{k+2} \cdot F_k$~~ we have to get : $F_{k+1}^2 = (-1)^{k+2} + F_{k+2} \cdot F_k$

We have : let : $F_{k+2} \cdot F_k + (-1)^{k+2}$

$$= F_k \cdot (F_{k+1} + F_k) + (-1)^{k+2}$$

$$= F_k \cdot F_{k+1} + F_k^2 + (-1)^{k+2}$$

$$= F_k (F_k + F_{k-1}) + F_k^2 + (-1)^{k+2}$$

$$= F_k^2 + F_k F_{k-1} + F_k^2 + (-1)^{k+2}$$

$$= F_k^2 + F_k F_{k-1} + (F_k^2 - (-1)^{k+1})$$

$$= F_k^2 + F_k F_{k-1} + F_{k+1} F_{k-1} \quad \text{(From IH)}$$

$$= F_k^2 + F_k \cdot F_{k-1} + F_{k-1} (F_k + F_{k-1})$$

$$= F_k^2 + F_k \cdot F_{k-1} + F_{k-1} F_k + F_{k-1}^2$$

$$= F_k^2 + 2 F_k F_{k-1} + F_{k-1}^2$$

$$= (F_k + F_{k-1})^2 = F_{k+1}^2 \rightarrow \text{Proved}$$

(iv) Base step :- Let $n = 6$, $m = 3$

So, $F_n = F_6 = 8$, $F_m = F_3 = 2$

So, F_n is an integer multiple of F_m.

Inductive hypothesis :- For $m = k$ & $n = kp$, where $p \in z^+ \& k \geq 0$

F_{kp} is an integer multiple of F_k.

Induction step :- Let $m = k+1$ & $n = (k+1)p = kp + p$

where $p \in z^+$

From IH, F_k is ~~divided~~ divisor of F_{kp}.

So F_{k+1} is divisor of $F_{(k+1)p} = F_{kp+p}$

So, F_{kp+p} is an integer multiple of F_k.

Hence proved.

If n is an integer multiple of m, then F_n is an integer multiple of F_m.

(8) Prove by mathematical induction that the sum of the cubes of three successive natural numbers is divisible by 9.

ans:- Base step :- For $1^3 + 2^3 + 3^3 = 1 + 8 + 27 = 36 \equiv 0 \pmod 9$

↳ divisible 9

Inductive hypothesis :- For $n = (k), (k+1), \& (k+2)$ 3 successive natural numbers

So, $(k)^3 + (k+1)^3 + (k+2)^3 = 9P$, $P \in N$

Inductive step :- For $n = (k+1), (k+2), \& (k+3)$ 3 successive

natural nos.

We need to show that, sum of their cubes is divisible by 9

$= (k+1)^3 + (k+2)^3 + (k+3)^3$

$= (k+1)^3 + (k+2)^3 + (k^3 + 27k + 9k^2 + 27)$

$= k^3 + (k+1)^3 + (k+2)^3 + 9(k^2 + 3k + 3)$

$= 9P + 9(k^2 + 3k + 3)$

$= 9(P + k^2 + 3k + 3) \longrightarrow$ which is divisible by 9

Hence, true.

Hence, by the principle of induction sum of the cubes of three consecutive natural numbers from n is divisible by 9.

(9) For every non-negative integer n, $\sum_{i=0}^{n} (2i+1)^2 = \frac{(n+1)(2n+1)(2n+3)}{3}$

ans: Base step : For $n=0$,

LHS : $\sum_{i=0}^{0} (2i+1)^2 = (2(0)+1)^2 = (1)$

RHS : $\frac{(0+1)(0+1)(0+3)}{3} = 3/3 = (1)$.

So, LHS = RHS (proved)

Inductive step : For $n=k$, (hypothesis)

$\sum_{i=0}^{k} (2i+1)^2 = \frac{(k+1)(2k+1)(2k+3)}{3}$

Inductive hypo step : For $n=k+1$

LHS : $\sum_{i=0}^{k+1} (2i+1)^2 = \sum_{i=0}^{k} (2i+1)^2 + [2(k+1)+1]^2$

$$= \frac{(k+1)(2k+1)(2k+3)}{3} + (2k+3)^2 \rightarrow \text{(From IH)}$$

$$= \frac{(k+1)(2k+1)(2k+3) + 3(2k+3)^2}{3}$$

$$= \frac{(2k+3)(2k^2+9k+10)}{3}$$

$$= \frac{(2k+3)(2k+5)(k+2)}{3}$$

$$= \frac{[(k+1)+1][2(k+1)+1][2(k+1)+3]}{3}$$

$= \underline{\text{RHS}} \rightarrow$ Hence proved.

Hence, for every non-negative integer n, $\sum_{i=0}^{n}(2i+1)^2 = \frac{(n+1)(2n+1)(2n+3)}{3}$

(10) Prove that $|A \times B| = |A| * |B|$ for all finite sets A & B

Ans:- To prove :- $|A \times B| = |A| \times |B|$

Given :- A & B are finite sets.

Proof :- Let us consider 3 cases :-

Case I :- Let $A = \{x\}$ → element x in A where $x > 0$

$B = \{y\}$ → element y in B where $y > 0$

LHS :- $|A \times B| = |x \times y| = x \times y$

RHS :- $|A| \times |B| = |x| \times |y| = x \times y$.

So, LHS = RHS (proved).

Case II: Let $A = \{x\}$, where $x<0$

$B = \{y\}$, where $y<0$

LHS: $|A \times B| = |-x \times -y|$

$= |-(x \times y)|$

$= x \times y$

RHS: $|A| \times |B| = |-x| \times |-y|$

$= x \times y = x \times y$

So, LHS = RHS → proved.

Case III: Let $A = \{x\}$, where $x<0$

$B = \{y\}$ where $y>0$

LHS: $|A \times B| = |-x \times y|$

$= x \times y$.

RHS: $|A| \times |B| = |-x| \times |y| = x \times y$. → LHS = RHS proved

So, all the 3 cases has been proved

hence, $|A \times B| = |A| \times |B|$ for all finite sets A & B.

(11) Prove or disprove: For every $n \in N$, $n^2 - n + 11$ is prime

ans: Let the value of $n = 11$, s.t. $n \in N$

then $n^2 - n + 11 = 11^2 - 11 + 11 = 121$

$\neq$ prime number

Hence, disproved.

(12) For every $x, y \in N$, $2^{2x} + 3^{2y+1}$ is prime. Prove or Disprove.

ans:- Let the value of $x=3$ & $y=4$ $\forall\ x, y \in N$.

then $2^{2(3)} + 3^{2(4)+1}$

$= 2^6 + 3^9$

$= 64 + 19681 = 19747 \rightarrow$ is not a prime number, as $19747 \equiv 0 \pmod 7$.

↳ divisible by 7

So, hence, disproved.

(13) For every $m \in I$, there exists an $n \in N$, s.t. $\left|\frac{1}{m} - \frac{1}{n}\right| > \frac{1}{2}$

ans:- Let the value of $m = 2$ & $n = 3$ s.t. $m \in I$ & $n \in N$.

So, LHS :- $\left|\frac{1}{m} - \frac{1}{n}\right|$

$= \left|\frac{1}{2} - \frac{1}{3}\right| = \left|\frac{1}{6}\right| = \frac{1}{6}$

RHS :- $\frac{1}{2}$

So, $\boxed{\frac{1}{6} \not> \frac{1}{2}}$, So, disproved.

(14) If n is an integer & n^2 is divisible by 4, then n is divisible by 4.

ans:- Let the value of $n = 6$

So, $n^2 = 36$ is divisible by 4.

but, $n = 6$ is not divisible by 4

i.e. $6 \equiv 2 \pmod 4 \rightarrow$ not divisible

Hence, disproved.

Prove or Disprove:

(15) $(a+b)^2 = (a^2+b^2)$ is not an algebraic identity $\forall a, b \in R$.

ans:- The algebraic eq^{n}s which are valid for all values of variables in them are called algebraic identities.

So, let us take $a=1$, $b=2$

So LHS :- $(1+2)^2 = 9$

RHS :- $(1)^2 + (2)^2 = 5$ So as LHS $\neq$ RHS

So, it is not an algebraic identity.

Hence proved.

(16) For $\forall\ a \in N$, $\frac{1}{x+2} = \frac{1}{x} + \frac{1}{2}$ is not an identity

Prove or Disprove

ans:- The algebraic eq^{n}s which are valid for all values of variables in them are called algebraic identities.

So, let us take, $x=1$,

So LHS :- $\frac{1}{1+2} = \frac{1}{3}$

RHS :- $\frac{1}{1} + \frac{1}{2} = \frac{3}{2}$, So as LHS $\neq$ RHS

So, it is not an algebraic identity

Hence, proved.

(17) Prove or Disprove :- If $pq = x$, then $p = \frac{x}{q}$ $\forall\ p, q, x \in R$

ans:- let us take, values $p=10$, $q=0$ & $x=0$, s.t. $p, q, x \in$

but, $p = \frac{x}{q}$

$\Rightarrow 10 = \frac{x}{0} \rightarrow$ (not defined)

So, disproved, if $p, q, x \in R$.

(18) Show using counter example that "a+b can be less than (min) (a, b) $\forall a, b, \in R$".

ans:- To show :- $a+b < \min(a,b)$.

Let us take the values $a=1$, $b=3$

So, $a+b = 1+3 = 4$

but $\min(1,3) = 1$

So, $4 \not< 1$, → if $a, b \geq 0$, then counter example not possible)

But, if $a, b < 0$

So, $a = -1$ & $b = -2$

So, $a+b = -1-2 = -3$ → (i)

So, $\min(-1,-2) = -2$ → (ii)

Now, from (i) & (ii), $-3 < -2$ → (proved, only for negative nos.)

But in the question, asked for $\forall a, b \in R$, hence $\forall, a, b$ is not possible

Hence, disproved

(19) Prove or disprove :- For all real no. x, if x^2 is rational then x is rational.

ans:- If the value of $x^2 = 2$, → which is rational

But ⇒ $x = \sqrt{2}$ → irrational no. (not rational)

Hence, due to the counterexample, → disproved.

(20) For all real numbers, x, y, $\lceil x+y \rceil = \lceil x \rceil + \lceil y \rceil$

Prove or disprove

ans:- Let the value of $x = 2.3$ & $y = 4.2$

LHS:- $\lceil x+y \rceil = \lceil 2.3+4.2 \rceil = \lceil 6.5 \rceil = 7$

RHS:- $\lceil x \rceil + \lceil y \rceil = \lceil 2.3 \rceil + \lceil 4.2 \rceil = 8$

So, LHS $\neq$ RHS, $\forall\ x, y \in R$

Hence disproved.

(21) Check for the correctness of the following algorithm that performs conservation from ~~binary~~ decimal to binary.

Algorithm Decimal to Binary

Input: n, a positive integer.

Output: b, an array of bits, the bin ~~rapes~~ repr. of n, stasring with the least significant bits.

```
t:=n;
k:=0;
while (t>0) do
    b[k]:=t mod 2;
    t:t div 2;
    k:=k+1
end.
```

ans:- * Invariant:- Before starting of the loop, we have,

n (positive integer) i.e. $n \geq 0 \Rightarrow t = n \geq 0$

& A[k], where $k \geq 0$, A[k] = 0/1.

Let us proof it, by taking an example, say n = 10.

* Initialization:- here, invariant, $n = 10 > 0 \Rightarrow t = 10 \geq 0$

& A[k], $(k \geq 0)$ = empty

or we can say 0.

Hence, before the starting of the loop, invariant is true.

* Maintainance:- We have n = 10

So, t = n = 10, k = 0.

First iteration:- b[0] = 0

t = 10/2 = 5.

k = 0+1 = (1).

Here $t = 5 \geq 0$ & b[0] / A[0] = 0 (So, invariant is true).

Hence, invariant remains true before the next iteration of the loop

* Termination:-

Last iteration:- b[3] = 1 mod 2 = (1)

t = 1/2 = 0

k = 3+1 = 4.

So, $t = 0 \geq 0$ & b[3] / A[3] = 0 (invariant is true)

Hence, invariant remains true when the loop terminates at $t > 0$, So, the above algorithm is corrected.

(22) Check the correctness of the following algorithm :-

```
Max
percolate (A,n) /* A is an array of n integers */
i=0
while i < n-1 do
    if A[i] > A[i+1]
     swap ( A[i], A[i+1] )
i = i+1
return A[n-1]
```

ans:- Let us consider an array A = | 2 | 1 | 3 | 4 | (0 1 2 3), i=0, n=4.

Working the algorithm :- i = 0,

First iteration ←
- i < 3
- A[0] > A[1] (true)
- So, A[1] = 2 & A[0] = 1.
- i = 0+1 = 1.

Array after 1st iteration :- | 1 | 2 | 3 | 4 | (0 1 2 3), i=1, n=4.

2nd iteration →
- i < 3 (True)
- A[1] > A[2] → false
- So, array will remain same.
- i = 1+1 = 2.

Array after 2nd iteration :- | 1 | 2 | 3 | 4 | (0 1 2 3), i=2, n=4

Last iteration / 3rd iteration →
- i < 3 → (True)
- A[2] > A[3] → false
- So, no swap :..
- i = 2+1 = 3.

Array after last iteration :

0	1	2	3
1	2	3	4

i=3 $\quad n=4$

* Invariant : From A[0 : i] , $\Rightarrow$ A[i] is the largest / max.

* Initialization : From the example done above, we have, seen before the starting of the loop, invariant remains true.

i.e. A[0 : 0] $\Rightarrow$ A[0] = 2. (single element)
So, it is largest.

* Maintanence : Invariant remains true before the next iteration

A[0 : i] $\Rightarrow$ A[0 : 1] (in first iteration)

$\Rightarrow$ A[1] = 2 (largest as compared to A[0]).

* Termination : Invariant ~~must be~~ remains true when the loop terminates

loop will terminate, when $i \geq n-1$,

Here A[0 : i] $\Rightarrow$ A[0 : 3] (in last iteration)

$\Rightarrow$ A[3] = 4 (largest as compared to other elements).

Hence, correctness proved.

(23) Check the correctness of the following algorithm (code) that performs in-place reversal of an array of size n.

```
//inputs: array A of size n
void reverse array (int *A, int n);
  int i = (n-1)/2
  int j = n/2;
  int tmp;
```

```
while (i>=0 && j<=(n-1))
    tmp = A[i];
    A[i] = A[j];
    A[j] = tmp;
    i--;
    j++;
```

ans:- let us consider an array $A = \boxed{1|2|3|4}$ $n = 4$.

Working on algorithm :-

$i = (4-1)/2 = 1$

$j = 4/2 = 2$

first iteration → $(i>=0, j<=3) \rightarrow$ true

$\boxed{1|2|3|4}$ (i, j)

tmp = A[1] = 2

A[1] = A[2] = 3 ⇒ A[1] = 3

A[2] = tmp = 2 ⇒ A[2] = 2

i = 1-1 = 0, j = 2+1 = 3.

Array after 1st iteration :- $\boxed{1|3|2|4}$ → n = 4

(i, j)

last iteration = $(i>=0, j<=3) \rightarrow$ true $[i=0, j=3]$

So, A[0] = 4, A[3] = 1 → swapping

i = -1, j = 4 → terminate.

Array after last iteration :- $\boxed{4|3|2|1}$, n = 4.

* Invariant :- $\boxed{i+j = n-1}$, where n = size of array A.

$i = n-1/2$ & $j = n/2$.

* Initialization :- Invariant is true before starting of the loop.

Before starting of the loop, $n=4$, $i=1$, $j=2$

So, $i+j = 2+1 = 3 = n-1$

↳ So, invariant is true.

* Maintanence :- Invariant remains true before the next iteration

i.e. After 1st iteration, $i=0$, $j=3$, $n=4$

So, $i+j = 3 = n-1$

↳ So, invariant is true.

* Termination :- Invariant ~~must be~~ remains true when the loop terminates

loop will terminate, when $i<0 \;\&\&\; j>n-1$

i.e. After last iteration, $i=-1$, $j=4$, $n=4$

So $i+j = 3 = n-1$

↳ So, invariant is true

& also, we got the array reversed, so the algorithm is corrected.

(24) Check the correctness of the following algorithm (code) that performs iterative binary search on an array of size n to find the element target.

```
def binary_search (A, target)
    lo = 0
    hi = len(A) - 1
    while (lo <= hi)
        mid = (lo + hi) / 2
        if A[mid] == target;
```

return mid

else if A[mid] < target;

lo = mid + 1

else

hi = mid - 1

ans:- Let us consider an Array A =

0	1	2	3	4
1	2	3	4	5

n = 5

target = 4.

Working on algorithm :-

lo = 0

hi = 5-1 = 4

first iteration = lo = 0, hi = 4

- lo <= hi (true)
- mid = 0+4/2 = (2).
- if (A[2] == 4) → (false)
- else if (A[2] < 4) → true
  - lo = 2+1 = 3.

last / 2nd iteration = lo = 3, hi = 4

- lo <= hi (true)
- mid = 3+4/2 = 3.
- if (A[mid] == 4) (true).
  - return (3).

* Invariant :- → Array A is sorted in ascending order.

→ $0 \le lo \le hi \le A.length - 1$

→ If target is there in Array, then, { target ∈ A[lo ... hi] }.

* Initialization :- Invariant is true before starting of the loop.

→ Array 'A' is sorted, (A.length = 5)

Also, lo = 0, hi = 5-1 = 4.

So, $0 \le 0 \le 4 \le 4$ → true.

So, invariant is true.

* Maintanence :- Invariant remains true before the next iteration of the loop.

After First iteration → Array 'A' is sorted → Also A.length = 5

Also, lo = 0, hi = 4

So, $0 \le 3 \le 4 \le 4$ → true

↳ So, invariant is true.

* Termination :- Invariant remains true ~~before~~ when the loop terminates, the loop will terminate when lo > hi. Or, when it find the search.

At last iteration → Array A is sorted → Also A.length = 5.

Also, lo = 3, hi = 4

So, $0 \le 3 \le 4 \le 4$ → true

↳ So invariant is true.

Hence, after termination, we find the targeted element

(25)

```
float power (float x, int a)
// input condition : a >= 0
{
if (a == 0)
    return 1.0;
```

```
else if (a%2)  // a is odd.
    return x*power(x, a-1);
else
    return power(x*x, a/2);
```

(i) Prove the correctness of the above function to compute x^a using mathematical induction.

(ii) Write the iterative version of the program & prove the correctness using loop invariant.

Ans:- (i) Base step :- For $a=0$,

then, $x^a = x^0 = 1$.

From the program :- 1.0 → (proved)

Induction hypothesis :- For a = even

$= 2k$ (say)

then we get from the program :- x^{2k}.

For a = odd

$= 2k+1$

then, we get from the program :- x^{2k+1}

Induction step :- let for a = even

$= 2(k+1)$ → (say)

then, from, the program we get :- ~~a%2~~ $a/2 = (k+1)$

return :- $(x*x)^{a/2} = (x^2)^{k+1}$

$= x^{2(k+1)}$

let a = odd

$= 2(k+1)+1 = \underline{2k+3}$

then, from the program, we get :- a%2 = (1).

So, return :- $x * x^{2(k+1)+1-1}$

$= x^{2(k+1)+1}$

$= x^{2k+3}$

Hence, proved.

(11) Iterative version of the program :-

```
float power (float x, int a) {
    int pow = 1;
    for (int i = 1; i <= a ; i++)
        pow *= x
    return pow;
}.
```

* Invariant :- Result (R) holds the multiplication of (x) by itself (a-1) times.

* Initialization :- when (a = 1) no multiplication is needed & the result evaluates to (x), so let us initialize the result as (R = x). This requires us to initialize (i = 1). With (R = x, & i = 1), the Invariant holds true before entering the loop because we need i - 1 = 1 - 1 = 0 multiplications.

Maintanence :- Raising (x) to the power of (a) is calculated by multiplying (x) by itself a times. In order to do that we need a statement like ($R = R^* x$). We can put this statement before or after we increment the loop index (i) because the loop invariant is going to hold true in either case. The reason why the invariant holds true in either case is because the statement ($R = R^* x$) doesn't depend on (i)

Termination :- the loop terminates when $k = n$, so, we have, $i - 1 = a - 1$ multiplications by the time we finish the loop. This is exactly the goal of the algorithm (post condition) explained, & the loop invariant holds true.

(26) For each of the following pairs of functions, determines whether $f(n) = O(g(n))$ or $g(n) = O(f(n))$.

a) $f(n) = n(n-1)/2$ & $g(n) = 6n$

b) $f(n) = n + 2\sqrt{n}$ & $g(n) = n^2$

c) $f(n) = n + \log n$ & $g(n) = n\sqrt{n}$

d) $f(n) = n \log n$ & $g(n) = n\sqrt{n}/2$

e) $f(n) = 2(\log n)^2$ & $g(n) = \log n + 1$

ans:- a) $f(n) = n(n-1)/2$ & $g(n) = 6n$

Let us assume:- $f(n) = O(g(n))$

$\Rightarrow f(n) \leq c \cdot g(n)$

$\Rightarrow \frac{n(n-1)}{2} \leq c\,(6)(n)$

$\Rightarrow \frac{n-1}{12} \leq c$

$\Rightarrow n/12 - 1/2 \leq c \Rightarrow n/12 \leq c \Rightarrow \boxed{c \geq n/12}$

when $n \to \infty$, then there doesn't exists any no. c which is greater than ∞.

So, $\Rightarrow \frac{n(n-1)}{2} \not\leq (c)(6)(n)$.

$\Rightarrow f(n) \not\leq c \cdot g(n)$. (N.P)

Let us assume, $g(n) = O(f(n))$.

$\Rightarrow g(n) \leq c \cdot f(n)$

$\Rightarrow [6(n)] \leq c \cdot \frac{n(n-1)}{2}$

$\Rightarrow \left(\frac{12}{n-1}\right) \leq c \Rightarrow \boxed{c \geq \frac{12}{n-1}}$

when $n \to \infty$, then $\boxed{c \geq 0}$, which is possible

Hence, $c \cdot \frac{(n)(n-1)}{2} \geq 6n \Rightarrow 6n \leq \frac{c \cdot (n)(n-1)}{2}$

$\Rightarrow g(n) = O(f(n)) \rightarrow$ (proved)

(b) $f(n) = n + 2\sqrt{n}$ & $g(n) = n^2$.

ans; let us assume : $f(n) = O(g(n))$.

$\Rightarrow f(n) \leq c \cdot g(n)$

$\Rightarrow n + 2\sqrt{n} \leq c \cdot n^2$

$\Rightarrow \boxed{1/n + 2/n^{3/2} \leq c}$ If $n \to \infty$, then $c \geq 0$ which is possible.

Hence $f(n) = O(g(n))$ → proved.

Now, let us assume, $g(n) = O(f(n))$.

$\Rightarrow g(n) \leq c \cdot f(n)$

$\Rightarrow n^2 \leq c \cdot (n + 2\sqrt{n})$

$\Rightarrow 1 \leq c \cdot \left(\frac{1}{n} + \frac{2}{n^{3/2}}\right)$

$\Rightarrow c \geq \frac{1}{\left(\frac{1}{n} + \frac{2}{n^{3/2}}\right)}$ when $n \to \infty$ then $c \geq \infty$ (which is not possible)

Hence, $g(n) \neq O(f(n))$.

c) $f(n) = n + \log n$ & $g(n) = n\sqrt{n}$.

Let us assume, $f(n) = O(g(n))$

$\Rightarrow n + \log n \leq c \cdot n\sqrt{n}$

$\Rightarrow \frac{n}{n} + \frac{\log n}{n} \leq \sqrt{n} \cdot c$

$\Rightarrow 1 + \frac{\log n}{n} \leq \sqrt{n} \cdot c$

As $\log n / n \to 0$, when $n \to \infty$

So, $\boxed{1 \leq c\sqrt{n}}$ → which is possible

Hence: $f(n) = O(g(n))$.

Let us assume: $g(n) = O(f(n))$

$\Rightarrow n\sqrt{n} \leq c \cdot (n + \log n)$

$\Rightarrow \sqrt{n}/c \leq 1 + \log n / n$. So, when $n \to \infty$, then $\log n / n \to 0$

$\Rightarrow \boxed{\sqrt{n}/c \leq 1}$ → (which is not possible)

So, $g(n) \neq O(f(n))$

(d) $f(n) = n\log n$ & $g(n) = n\sqrt{n}/2$.

For $f(n) = O(g(n))$ (let us assume).

$\Rightarrow f(n) \leq c \cdot g(n)$

$\Rightarrow n\log n \leq c \cdot n\sqrt{n}/2$

$\Rightarrow \log n \leq c\sqrt{n}$

$\Rightarrow c \geq \frac{\log n}{\sqrt{n}}$ when $n \to \infty$, then $\frac{\log n}{\sqrt{n}} \to 0$

$\Rightarrow \boxed{c \geq 0}$ → which is possible

Hence, $f(n) \leq c \cdot g(n) \Rightarrow \boxed{f(n) = O(g(n))}$

Let us assume:- $g(n) = O(f(n))$

$\Rightarrow g(n) \leq c \cdot f(n)$

$\Rightarrow \frac{n\sqrt{n}}{2} \leq c \cdot n\log n$

$\Rightarrow c \geq \frac{\sqrt{n}}{2\log n}$, so, when $n \to \infty$, then $\frac{\sqrt{n}}{2\log n} \to \infty$

$\Rightarrow \boxed{c \geq \infty}$

(which is impossible)

Hence, $g(n) \neq O(f(n))$

(e) $f(n) = 2(\log n)^2$ & $g(n) = \log n + 1$

Let us assume :- $f(n) = O(g(n))$

$\Rightarrow 2(\log n)^2 = O(\log n + 1)$

$\Rightarrow 2(\log n)^2 \leq c \cdot \log n + 1$

$\Rightarrow c \geq 2(\log n)^2 / \log n + 1$ $\Rightarrow$ when $n \to \infty$ then $c \geq \infty$ (N.P)

So, $f(n) \not\leq c\,g(n) \Rightarrow f(n) \neq O(g(n))$.

Let us assume $\div g(n) \leq c f(n)$.

$$\Rightarrow \log n + 1 \not\leq c \cdot 2(\log n)^2$$

$$\Rightarrow \frac{\log n + 1}{2(\log n)^2} \leq c$$

$$\Rightarrow \frac{1}{2\log n} + \frac{1}{2(\log n)^2} \leq c$$

$\Rightarrow n \to \infty$, then $\log_2 (\log n)^2 \to \infty$

$\Rightarrow$ $0 + 0 \leq c$ → which is possible

So, $g(n) = O(f(n))$ → (proved).

(27) State TRUE or FALSE justifying your answer with proper reason.

a) $2n^2 + 1 = O(n^2)$

ans: We can write $\div\ 2n^2 + 1 \leq 2n^2 + n^2$

$$\Rightarrow 2n^2 + 1 \leq 3n^2$$

where $c = 3$ & $f(n) = n^2$.

So $\forall\ n \geq 1$, we have $T(n) = O(n^2)$

So answer $\div$ TRUE.

b) $n^2(1 + \sqrt{n}) = O(n^2)$

ans: We can write it as $\div\ n^2(1+\sqrt{n}) = n^2 + n^2\sqrt{n}$

→ We have :- $n^2 + n^2\sqrt{n} \leq n^2\sqrt{n} + n^2\sqrt{n}$

$\Rightarrow n^2 + n^2\sqrt{n} \leq 2n^2\sqrt{n}$

$c = 2, \; f(n) = n^2\sqrt{n}$.

Hence for $\forall n \geq 1$, we have :- $T(n) \leq 2n^2\sqrt{n}$

$\Rightarrow T(n) = O(n^2\sqrt{n})$

$\neq O(n^2)$ (As :- $n^2\sqrt{n} \geq n^2(1)$ (Asymptotically))

So answer is False.

(c) $n^2(1+\sqrt{n}) = O(n^2 \log n)$.

ans:- We can write it as :- $n^2(1+\sqrt{n}) \leq n^2(\sqrt{n} + \sqrt{n})$

$\Rightarrow n^2 + n^2\sqrt{n} \leq 2n^2\sqrt{n}$

$c = 2$ & $f(n) = n^2\sqrt{n}$.

Hence for $\forall n \geq 1$, we have, $T(n) \leq 2n^2\sqrt{n}$

$\Rightarrow T(n) = O(n^2\sqrt{n})$

$\neq O(n^2 \log n)$

(As $n^2\sqrt{n} \geq n^2(\log n)$ (Asymptotically))

So answer is FALSE.

(d) $3n^2 + \sqrt{n} = O(n + n\sqrt{n} + \sqrt{n})$

ans:- We can write it as :- $3n^2 + \sqrt{n} \leq 3n^2 + n^2$

$\Rightarrow 3n^2 + \sqrt{n} \leq 4n^2$

$c = 2$ & $f(n) = n^2$.

Hence for $\forall n \geq 1$, we have, $T(n) \leq 4n^2$

$\Rightarrow T(n) = O(n^2)$.

We have been given $O(n + n\sqrt{n} + \sqrt{n})$

$= O(n\sqrt{n})$ (Ignoring the lower order constants)

Now, $T(n) = O(n.n)$

$\neq O(n.\sqrt{n})$

As $(n^2 (n.n) \geq n.\sqrt{n}$ (Asymptotically))

Hence, answer is FALSE.

(e) $\sqrt{n} \log n = O(n)$

ans: We know by, Growth rate of function:-

$\log n \leq \sqrt{n}$ (Asymptotically)

$\Rightarrow \sqrt{n} \log n \leq \sqrt{n}.\sqrt{n}$

$\Rightarrow \sqrt{n} \log n \leq (1)\, n$

So, $c = 1$, & $f(n) = n$.

Hence $\forall\, n \geq 1$, we have $T(n) \leq n$

$\Rightarrow T(n) = O(n)$

Hence answer is TRUE.

(f) $\lg n \in O(n)$.

ans: From the Growth rate of fun^c :-

$$1 < \lg n < \sqrt{n} < n < n \log n < n^2 < n^3 \text{ ----}$$

So, $\lg n = O(\lg n\, n)$ → Hence, answer is TRUE

$= O(n)$ (As $\lg n \leq n$ (Asymptotically))

(g) $n \in O(n \lg n)$

ans:- From the series of Growth rate of fun^c, we have:-

$$1 \leq \log n$$

$$\Rightarrow (1)(n) \leq (n)\log n.$$

So, $c = 1$, & $f(n) = n \log n$.

Hence $\forall\ n \geq 1$, we have $T(n) \leq n \log n$.

$$\Rightarrow T(n) = O(n \lg n).$$

Hence answer is TRUE.

(h) $n \lg n \in O(n^2)$

ans:- From bit (f), we have:-

$$\lg n \leq n$$

$$\Rightarrow (n) \lg n \leq (n)(n)$$

$$\Rightarrow n \lg n \leq n^2$$

So, $c = 1$, & $f(n) = n^2$

Hence, $\forall\ n \geq 1$, We have, $T(n) \leq n^2$

$$\Rightarrow T(n) = O(n^2)$$

Hence answer is TRUE.

(i) $2^n \in \Omega(6^{\ln n})$

ans:- We have:- $6^{\ln n} = n^{\ln 6} = n^{1.79}$.

So, from the series of Growth rate of fun^c we have:-

$$n^{1.79} \leq 2^n.$$

$$\Rightarrow 2^n \geq n^{1.79} \quad \text{So, } c = 1,\ f(n) = 2^n.$$

So, $\forall n \geq 1$, We have: $T(n) \geq n^{1.99}$

$\Rightarrow T(n) = \Omega(n^{1.99})$

$= \Omega(6^{\ln n})$

Hence, answer is TRUE.

(i) $\lg^3 n \in o(n^{1/2})$.

ans: From the series of Growth rate of fun^c, we have:

$\lg n \leq \sqrt{n}$

$\Rightarrow (\lg n)(\lg n)(\lg n) \leq (\sqrt{n})(\sqrt{n})(\sqrt{n})$

$\Rightarrow \lg^3 n \leq n^{3/2}$ → $c=1$ & $f(n) = n^{1/2}$.

$\Rightarrow \boxed{\lg^3 n < n^{1/2}}$. (Strictly greater)

Hence, $\forall n \geq 1$ we have, $T(n) < n^{1/2}$

$\Rightarrow T(n) = O(n^{3/2})$

$= o(n^{1/2})$ (Strictly greater)

Hence, answer is TRUE.

(28) For each of the following pair of functions $f(n)$ & $g(n)$ determine whether $f(n) = O(g(n))$ or $f(n) = \Theta(g(n))$ or $f(n) = \Omega(g(n))$.

a) $f(n) = \sqrt{n}$, $g(n) = \log(n+3)$.

ans: We know, $\sqrt{n} \geq \log n$, $\forall n \geq 1$.

$\Rightarrow \sqrt{n} \geq (1) \log(n+3)$ → $c=1$ & $f(n) = \log(n+3)$

So, $\forall n \geq 1$, we have, $\boxed{f(n) = \Omega g(n)}$.

(b) $f(n) = n\sqrt{n}$, $g(n) = n^2 - n$

ans:- Time complexity for $f(n) = \Theta(n\sqrt{n}) = O(n\sqrt{n})$

$T(n)$ for $g(n) = O(n^2)$

Now, $\boxed{\sqrt{n} \geq 1}$ → from the series of Growth of func.

$\Rightarrow \sqrt{n} \leq n$

$\Rightarrow \boxed{n\sqrt{n} \leq n^2}$

$\Rightarrow n\sqrt{n} \leq c \cdot (n^2 - n)$

$\Rightarrow f(n) \leq c \cdot (g(n))$

$\Rightarrow \underline{f(n) = O(g(n))}$ → (answer)

(c) $f(n) = 2^n - n^2$, $g(n) = n^4 + n^2$.

ans:- $T(n)$ for $f(n) = O(2^n)$

$T(n)$ for $g(n) = O(n^4)$

So, from the growth of rate of function :-

$2^n \geq c \cdot n^4$

$\Rightarrow f(n) \geq c \cdot g(n)$

$\Rightarrow f(n) = \Omega\, g(n)$

(d) $f(n) = n^2 + 3n + 4$, $g(n) = 6n^2$

ans:- $T(n)$ for $f(n) = O(n^2)$

$T(n)$ for $g(n) = O(n^2)$

From the series of growth rate of function :-

$\Rightarrow n^2 \leq c \cdot n^2$

$\Rightarrow n^2 = n^2$

OR $f(n) = \Theta\, g(n)$ → (answer).

(e) $f(n) = n + n\sqrt{n}$, $g(n) = 4n \log(n^2+1)$.

ans:- $f(n) = n + n\sqrt{n}$

$T(n)$ for $f(n) = O(n\sqrt{n})$

$g(n) = 4n \log(n^2+1)$.

$T(n)$ for $g(n) = O(n \log n)$.

From the series of growth of rate of functions :- we have:-

$$\log n \leq \sqrt{n}$$

$$\Rightarrow \sqrt{n} \geq \log n$$

$$\Rightarrow n\sqrt{n} \geq n \log n$$

$$\Rightarrow f(n) \geq c \cdot g(n)$$

So, $f(n) = \Omega\, g(n)$ → (answer).

(29) Prove the following statements:-

(a) If $f_1(n) = \Omega(g_1(n))$ & $f_2(n) = \Omega(g_2(n))$ then $f_1(n) + f_2(n) = \Omega(g_1(n) + g_2(n))$.

ans:- PTO.

ans:- We have:- $f_1(n) = \Omega(g_1(n))$

$f_2(n) = \Omega(g_2(n))$.

We can write it as:-

$0 \leq c_1 g_1(n) \leq f_1(n) \rightarrow (I)$

& $0 \leq c_2 g_2(n) \leq f_2(n) \rightarrow (II)$.

Adding (I) & (II), we get:-

$0 \leq c_1 g_1(n) + c_2 g_2(n) \leq f_1(n) + f_2(n)$

$\Rightarrow 0 \leq (g_1(n) + g_2(n)) c_3 \leq f_1(n) + f_2(n)$

$\Rightarrow f_1(n) + f_2(n) = \Omega(g_1(n) + g_2(n))$.

↳ Hence proved.

(b) If $f_1(n) = \Omega(g_1(n))$ & $f_2(n) = \Omega(g_2(n))$

then $f_1(n) + f_2(n) = \Omega(\min(g_1(n) + g_2(n)))$

ans:- Given:- $f_1(n) = \Omega(g_1(n))$.

$\Rightarrow 0 \leq c_1 g_1(n) \leq f_1(n) \rightarrow (I)$.

$f_2(n) = \Omega(g_2(n))$

$\Rightarrow 0 \leq c_2 g_2(n) \leq f_2(n) \rightarrow (II)$.

Adding (I) & (II) we get:-

$0 \leq c_1 g_1(n) + c_2 g_2(n) \leq f_1(n) + f_2(n)$

$\Rightarrow 0 \leq c_3 (g_1(n) + g_2(n)) \leq f_1(n) + f_2(n)$

$\Rightarrow 0 \leq c(g_1(n) + g_2(n)) \leq f_1(n) + f_2(n)$.

Here c is the minimum value s.t. if is less than $f_1(n) + f_2(n)$.

→ It is happening as $c_3 > c$, but still less than $f_1(n) + f_2(n)$

So, $c(g_1(n) + g_2(n)) \leq f_1(n) + f_2(n)$.

Hence, $f_1(n) + f_2(n) = \Omega(\min(g_1(n) + g_2(n)))$.

↳ (proved)

c) If $f_1(n) = O(g_1(n))$ & $f_2(n) = O(g_2(n))$ & then $f_1(n) \cdot f_2(n) = O(g_1(n) \cdot g_2(n))$.

ans: Given: $f_1(n) = O(g_1(n))$

$\Rightarrow 0 \leq f_1(n) \leq c_1(g_1(n)) \rightarrow (1)$

Also, $f_2(n) = O(g_2(n))$

$\Rightarrow 0 \leq f_2(n) \leq c_2(g_2(n)) \rightarrow (II)$.

Multiplying (1) & (2), we get:

$0 \leq f_1(n) \cdot f_2(n) \leq \boxed{c_1 c_2}(g_1(n)) \cdot (g_2(n))$.

$\Rightarrow 0 \leq f_1(n) \cdot f_2(n) \leq c_3(g_1(n)) \cdot (g_2(n))$.

Hence: $f_1(n) \cdot f_2(n) = O(g_1(n) \cdot g_2(n))$

↳ from the def^n of Big 'O'

(d) If $f(n) = O(g(n))$, then $f(n)^k = O(g(n)^k)$.

ans: Given: $f(n) = O(g(n))$.

$\Rightarrow 0 \leq f(n) \leq c_1(g(n))$.

Multiplying the power k to inequalities, we get

$$0^k \leq f(n)^k \leq c_1^k g(n)^k$$

$$\Rightarrow \quad 0 \leq f(n)^k \leq c_2 g(n)^k$$

So, $f(n)^k = O(g(n)^k)$ → (proved)

e) If $f_1(n) = \theta(g_1(n))$, $f_2(n) = \theta(g_2(n))$, ..., $f_k(n) = \theta(g_k(n))$ where $k \in I^+$ then prove that

$$\sum_{i=1}^{k} f_i(n) = \theta\left(\max_{1 \leq j \leq k} (g_j(n))\right)$$

ans: We have: $f_1(n) = \theta(g_1(n))$

$f_2(n) = \theta(g_2(n))$

$\vdots$

$f_k(n) = \theta(g_k(n))$.

So, we can write it as:

$$0 \leq c_1 [g_1(n)] \leq f_1(n) \leq c_1' [g_1(n)]$$

$\vdots$

$$0 \leq c_k [g_k(n)] \leq f_k(n) \leq c_k' (g_k(n))$$

Note: Here $c_1', c_2' \ldots c_k'$ can be max, as, if

$f_k(n), f_1(n) < c_k', c_1' < c'_{k\max} g_k[n]$, $c'_{1\max} g_1[n]$.

$g_1(n)$ $g_k(n)$

So, adding all the above eq's:

$$0 \leq c_{\min} [g_1(n) + g_2(n) + \cdots g_k(n)] \leq f_1(n) + \cdots f_k(n)$$

$$\leq c'_{\max} [g_1(n) + g_2(n) + \cdots g_k(n)]$$

$\Rightarrow \sum_{i=1}^{k} f_i(n) = \Theta\left(\max_{i \le j \le k} (g_j(n))\right)$.

↳ proved.

(30) Given $f(n) = (n+a)^b$ for any real constants a & b, where $b > 0$. Prove that $f(n) \in \Theta(n^b)$

ans: $(n+a)^b = n^b + c_1 n^{b-1} a + \dots = \Theta(n^b)$ → we need to prove this.
↳ (1)

From eq^n (1), we can write, $0 \le c_1 n^b \le (n+a)^b \le c_2 n^b$.

$\forall\ n \ge n_0$

$n+a \ge n - |a|$

$n+a \ge n/2$, $|n| \le n/2$.

Let us consider: $f(n) = (n+a)^b$ & $g(n) = n^b$.

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{(n+a)^b}{n^b} = \lim_{n\to\infty} \frac{n^b (1 + a/n)^b}{n^b}$$

$$= \lim_{n\to\infty} \left(1 + \frac{a}{n}\right)^b$$

$$= 1.$$

$\Rightarrow f(n) = O(g(n))$ → (ii)

Now, $\lim_{n\to\infty} \frac{g(n)}{f(n)} = \lim_{n\to\infty} \frac{n^b}{(n+a)^b} = \lim_{n\to\infty} \frac{n^b}{n^b (1+a/n)^b} = \frac{1}{1^b} = 1$

$\Rightarrow f(n) = \Omega(g(n))$ → (iii)

Both (ii) & (iii) gives that →

$f(n) = \Theta(g(n)) \Rightarrow (n+a)^b \in \Theta(n^b)$.

(31) Prove that $a^n \in o(b^n)$ for $b > a > 0$.

ans:- Given:- $b > a > 0$.

$\therefore b^n > a^n$

$\Rightarrow a^n < (1) b^n$ (Strictly less)

$c = 1$ & $f(n) = b^n$

So, acc. to little 'o' defination:- $c > 0$ & $n \geq 1$, $n_0 = 1$.

So, $[a^n) \in o(b^n)]$ → proved.

(32) Prove that $\lg n = O(\sqrt{n})$, however $\sqrt{n} \neq O(\lg n)$.

ans:- We know:- $\lg n \leq n$ (from mathematics)

$\Rightarrow \lg \sqrt{n} \leq \sqrt{n}$

$\Rightarrow 2 \lg \sqrt{n} \leq 2\sqrt{n}$

$\Rightarrow \lg (\sqrt{n})^2 \leq 2\sqrt{n}$

$\Rightarrow \lg n \leq 2\sqrt{n}$

$c = 2$, $f(n) = 2\sqrt{n}$

$\forall n \geq 1$, we have $T(n) \leq 2\sqrt{n}$

$\Rightarrow T(n) = O(\sqrt{n})$.

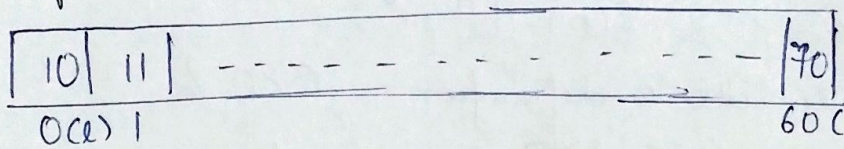
Also, $\lim_{n \to \infty} \frac{f(n)}{g(n)} = \lim_{n \to \infty} \frac{\log n}{\sqrt{n}} = 0$.

$\Rightarrow \lg n = O(\sqrt{n})$.

Hence proved.

(33) Let A[1 ... 60] = 10, 11, ..., 70. How many comparisons are performed by algorithm Binary-Search when searching for the elements present in the first, middle & last position of the array? Formulate & explain by taking a suitable example.

ans:-



(1) if the element is present in the middle, say this above array, then no comparisons are needed & element would be searched at once.

For ex:- Here $l=0, h=60$

$$mid = \left\lceil \frac{l+h}{2} \right\rceil = 30. \left(A[mid] == \text{targeted element}\right)$$

(2) Suppose the element is present at the beginning or at the end of the sorted array, we first compare the search element with $A[n/2]$ & recurse either on the first half $A[0, \ldots n/2-1]$ or 2nd half $A[n/2, \ldots n-1]$.

Thus, we can find the reccurence to be $T(n) = T(\lceil n/2 \rceil) + O(1)$. The first part of the recurrence should be obvious. The $O(1)$ part comes because we have to determine which half of the array to recurse on. This comparison can be done in constant time.

Using master's theorem, to solve this problem,

$$a=1, b=2, c=0.$$

Note:- $T(1)=1$, let $n=2^k \Rightarrow k = \lceil \log_2 n \rceil$

$$T(2^k) = T(2^k/2) + 1$$

$$= T(2^{k-1}) + 1$$

$$= [T(2^{k-2}) + 1] + 1$$

$$= T(2^{k-2}) + 2$$

$$= [T(2^{k-3}) + 1] + 2$$

$$\vdots$$

$$= T(2^{k-k}) + k$$

$$= T(2^0) + k$$

$$= T(1) + k$$

$$= (1+k).$$

So, $T(2^k) = 1+k \Rightarrow T(n) = \lceil \log_2 n \rceil + 1$.

Therefore, performing binary search, you should expect to have $\lceil \log_2 n \rceil + 1$ comparisons where n is the number of elements in your search space.

In the previous array, there were total 60 elements,

So, total no. of comparisons = $\lceil \log_2 60 \rceil + 1$

$= 6 + 1 = (7)$.

Hence, there will be 7 comparisons in the array, if it is located at beginning or end & not in the mid.

(34) Suppose you are choosing b/w the following 3 algorithms.

- Algorithm A solves problem by dividing them into 5 sub-problems of half the size, recursively solving each sub-problem & then combining the solution in linear time.
- Algorithm B solves problem size n by recursively solving two sub problems of size n by ~~recursively~~ solving two-sub problems of size n-1 & then combining the solutions in constant time.
- Algorithm C solves problems of size n by dividing them into sub problems of size n/3, recursively solving each sub problem & then combining the solutions in $O(n^2)$ times.

What are the running times of each of these algorithms (in big-O notation), & which would you choose?

ans:- * Recurrence relation for algorithm A is represented by using Master's theorem:-

$T(n) = 5T(n/2) + n, \quad T(1) = 1.$

In the above rel^n $(n/2)$ is taken because each time in sub problem input becomes half.

Then as per the Master theorem, $a=5$, $b=2$, & $f(n)=n$

So, $T(n) \rightarrow$ running time for Algorithm A $= O(n^{\log_b a})$

$= O(n^{\log_2 5})$

* Recurrence rel^n for algorithm B is:-

$T(n) = 2T(n-1) + c$

$T(1) = 1$

Again the next recurring terms $T(3)$ & $T(4)$ for algorithm B will be:-

$T(3) = 2T(2) + (2)(2)T(1) + c$

$T(4) = 2T(3) + (2)(2)(2)T(2) + (2)(2)T(1) + c$

Generalizing, the above recurring relation, it should be noted that the relation is subdivided & deepens in the multiple of 2^n.

Therefore, the running time of algorithm B is $O(2^n)$.

* The recurrence relation for algorithm C as per the Master theorem will be →

$T(n) = 9T(n/3) + (n^2)$, $T(1) = 1$.

Acc. to Master theorem, $a=9$, $b=3$, $k = \log_3^9 = 2$

$f(n) = n^2$.

So, $T(n) = O(n^2 \log n)$.

Since algorithm C linearly divides the problem into 9 sub-problems of size $(n/3)$ & takes running time in terms of $(\log_3 n)$ which consumes less memory space in comparison to the recursive approach.

So, C will be an appropriate algorithm for implementation.