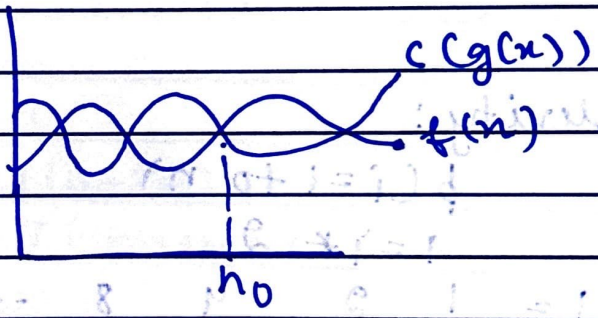
1. · Asymptotic notations are used to represent the complexities of algorithms for asymptotic analysis.
· These notations are mathematical tools to represent complexities.

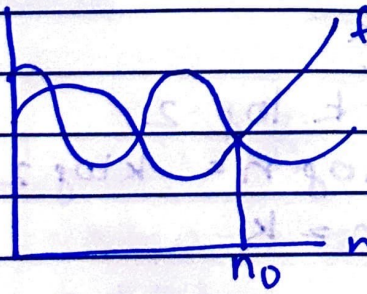
- Big Oh notation:
· Gives an upper bound for a fn f(n) within a constant factor.

$f(n) = O(g(n))$
if $f(n) \leq c\,g(n)$
for $c > 0$ & $n \geq n_0$.



- Big Omega notation:
· Gives lower bound for a fn f(n) within a constant factor.

$f(n) = \Omega(g(n))$
if $f(n) \geq g(n)$
f(n) for $c > 0$ & $n \geq n_0$



- Big Theta notation:
  - Gives bound for a f^n $f(n)$ within a constant factor.

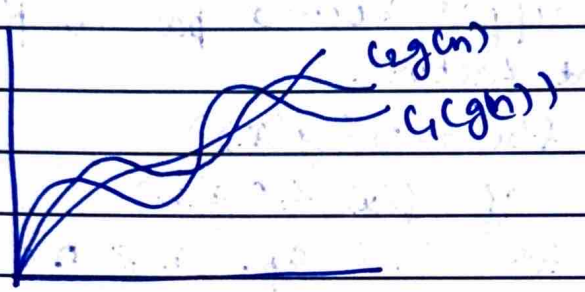
$f(n) = O(g(n))$

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

If $c_1 g(n) = f(n) \leq c_2 g(n)$

$c_1 > c_2 > 0$

$\forall n \geq n_0$



2.

time complexity:

```
for(i=1 to n)
    i=i*2
```

i = 1 2 4 8 ---- n

2^0 2^1 2^2 2^3 ---- 2^k

$GP = a r^{k-1}$

$n = 1 \cdot 2^{k-1}$

$n = \frac{2^k}{2}$

$2n = 2^k$

$\log 2n = k \log 2$

$\log 2 + \log n = k \log 2$

$\log n = k$

$\therefore T(n) = O(\log(n))$

3. $T(n) = 3T(n-1)$, $n > 0$, otherwise

$T(0) = 1$

$T(1) = 3T(0)$

$= 3$

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

$= 9 = 3^2$

$T(3) = 3T(2) = 27 = 3^3$

$T(n) = 3^n$

$= O(3^n)$

4. $T(n) = 2T(n-1) - 1$ — (1), $n > 0$, otherwise

Let $n = n-1$

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

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

Put $T(n-1)$ in (1)

$T(n) = T(n-2) - 3$ — (2)

Put $n = n-2$

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

$= 2T(n-3) - 1$

Put in (2) —

$T(n) = 4(2T(n-3) - 1) - 1$

$= 8T(n-3) - 4 - 1$

$= 8T(n-3) - 5 = 2^k T(n-k) - 1$

$n - k = 1 \quad k = (n-1)$

$T(n) = 2^{n-1} T(n-n+1) - 5$

$= 2^{n-1} T(1) - 5$

$= \frac{2^n}{2} = 2^n = O(2^n)$

5.

```
while (s <= n)
{
  i ++;
  s = s+i;
  printf (#n);
}
```

i=1 = i++ , i= 2

s = 3

i = 3

s = 6

i = 4

s = 10

i = 5

s = 15

i = 2 3 4 5 ⟶

s+1+2 s+1+2+3 s+1+2+3+4 ---

$s = s+1+2+3+4 \cdots k$

$s(k) = k(k+1)/2 \leq n$

$k^2 + k/2 \leq n$

$k^2 \leq n$

$k \leq \sqrt{n}$

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

6. $i = 1 \quad 2 \quad 3 \quad 4 \quad \cdots$

$i^2 = 2 \quad 4 \quad 9 \quad 16 \quad \cdots \quad k^2$

$k^2 \leq n$

$k \leq \sqrt{n}$

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

7. $T(n) = T(n/2) * \log(n) * \log(n)$

$= n/2 * \log n^2$

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

8. $T(n) = n * n * [T(n-3)]$

$T(n) = n^2 + T(n-3)$

$= O(n^2)$

9. $i=1$, $j = 1, 2, 3, 4, \cdots n \rightarrow n/1$

$i=2$, $j = 1, 3, 5, 7, \cdots n \rightarrow n/2$

$i=3$, $j = 1, 4, 7, 11, \cdots n \rightarrow n/3$

$i = n-1$, $j = 1, n \cdots n/n = 1$

$n + \frac{n}{2} + \frac{n}{3} \cdots \frac{n}{2} = \log(n)$

(maximum series)

$T(n) = n * \log n = O(n \log n)$

10. $n = 1$

$n^k = 1^k \cdot c^n = c$

$n = 2$

$nk = 2^k$, $c^n \in c^2$

$n = k$, $n^k - k^k$, $c^n = c^k$

we can say that

for any value of $n > 0$

$n^k \geq c^n$

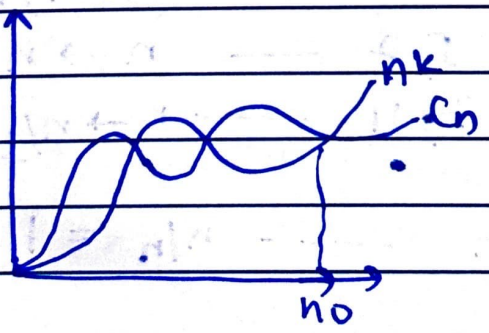
Let $n^k = f(x) \cdot c^n = \log(cn)$

$\therefore f(x) \geq \log(cn)$

$c_0 > 0$, $n_0 \geq n_0$

$\therefore f(n) = O(\log(n))$

$\therefore n^k = O(c^n)$



11. extract min →

```
int extractmin(vector<int>& heap)
{
  if(heap.empty())
  {
    return -1;          → O(1)
  }
  swap(heap[0], heap.back()); → O(1)

  int minElement = heap.back();
  heap.popback(); → O(1)
  heapify(hap, 0); → O(log n)
  return min element;
}
```

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

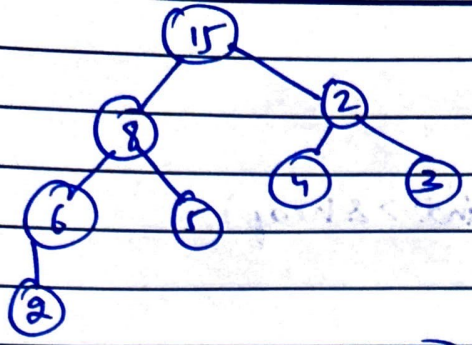
no of cary →

$(1+\frac{n}{4}) + \frac{2+n}{1} + 1 \times \frac{n}{16}$

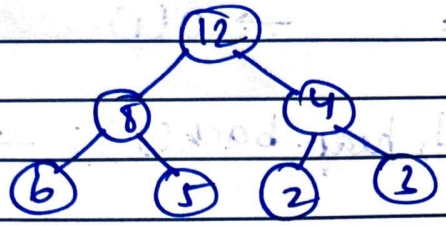
$+ (n-1) \forall 1)$

$= \log(n)$ harmonic mean

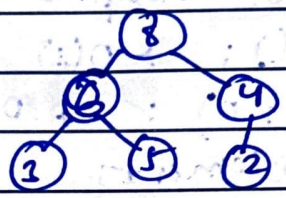
12.



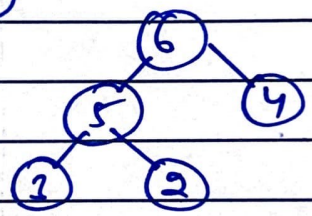
Delete Root (15)



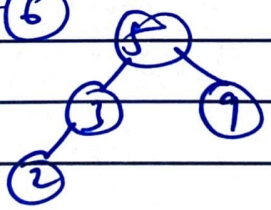
Delete root (12)



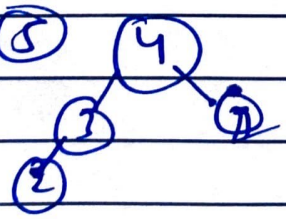
Delete Root (8)



Delete Root (6)



Delete Root (5)



Delete Root (4)

Delete (3) (2)

Delete (2)

heap is complete