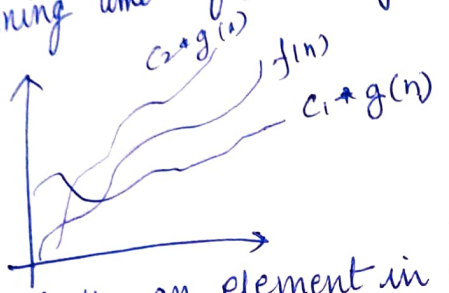
Design and Analysis of Algorithm

Divakar Pandey
DS I
44

Assignment – 01

Ans:- Asymptotic Notation are mathematical tools used to describe the efficiency of algorithm, their running time, as the input size tends toward infinity.

1) Theta Notation (θ) :- It represent the upper and lower bound of running time of an algorithm.



$c_1 g(n) * f(n) \leq c_2 * g(n)$ for all $n \geq 0$

Ex:- Finding an element in an unsorted array using linear search has a Time complexity of $O(n)$

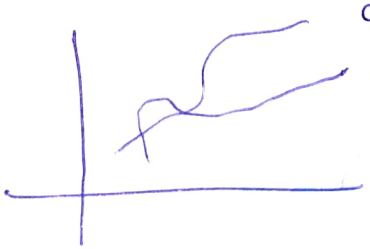
2) Omega Notation:- This represent the lower bound of an algorithm's running time.

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

$0 \leq cg(n) \leq f(n) \quad n \geq 0$

Example:- Sorting an array using selection sort has time complexity of $\Omega(n^2)$

3) Big O Notation:- This represent the upper bound of an algorithm running time.



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

$0 \leq f(n) \leq c \cdot g(n)$

The Time complexity of a linear search is $O(n)$.

Ans2 The iteration in the power of 2.

1 ... 2 ... 4 ... 8 ... 16 ..... n

for $n = 2^{k-1}$ $\quad 2n = 2^k$

$\log n = k \log_2 2$

$\log n = k$

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

Ans3 $T(n) = 3(T(n-1))$ if $n>0$ otherwise 1

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

$T(n) = 3[3T(n-2)]$

$T(n) = 3 \cdot 3 \cdot 3 \cdot [3T(n-3)]$

$T(n) = 3^k T(n)$

$T(n) = 3^k O(3^k)$

Ans4) $T(n) = 2T(n-1) - 1$ $\quad n>0$ & $n=1$

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

$T(n) = 2 \cdot 2 \{ 2(T(n-3) - 1) - 1$

$T(n) = 2^k [2(T-k) - 1) - k$

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

Ans5:-

```
int i=1, s=1;
while (s<=n){
    i++;        → n times
    s=s+i;      → n times
    printf("#"); → n times.
}
```

T.C. ~~O(n)~~

Time complexity is $O(n)$

Ans6:-

```
void function (int n)
{
    int i, count = 0;
    for (int i=0 ; i*i < n ; i++)
    {
        count ++;
    }
}
```

The loop iterate up to square root of n. Then the time complexity is $O(\sqrt{n})$.

Ans7:-

```
void function (int n)
{
    int i, j, k, count = 0;
    for (i = n/2 ; i <= n ; i++)
    {
        for (j = 1 ; j <= n ; j = j * 2)
        {
            for (k = 1 ; k <= n ; k = k * 2)
            {
                count ++
            }
        }
    }
}
```

The outer loop iterate n/2 times then Time complexity is $O(\frac{n}{2})$.

The middle loop iterate in power of 2 so it form a G.P. then Time complexity is $O(\log_2 n)$.

The inner loop ~~ite~~ also has Time complexity of $O(\log_2 n)$.

Time complexity = $\frac{n}{2} * \log_2 n \times \log_2 n$

$= \frac{n}{2} \log_2^2 n$

$O(n \log_2^2 n)$

Ans 9:

```
void function (int n)
{
    for (i = 1; i<=n; i++)
    {
        for (j = 1; j<=n; j = j+i)
        {
            printf ("*");
        }
    }
}
```

The outer loop runs n times.

The inner loop runs j = 1 to n by increasing j+i value of i.

for i=1 the inner loops runs n times

i=2 the inner loop runs n/2 time.

i=3 " " " " n/3 times.

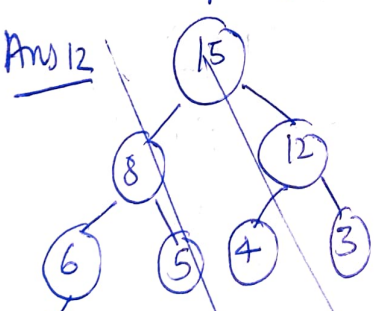
for i=n the loop runs once.

$$n + \frac{n}{2} + \frac{n}{3} + \frac{n}{4} + \cdots 1$$

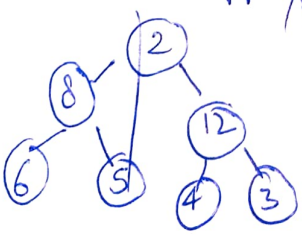
$$k = \log n$$

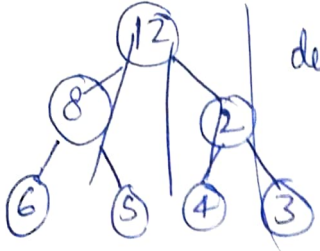
Time complexity of program is $O(n \log n)$

Ans 12

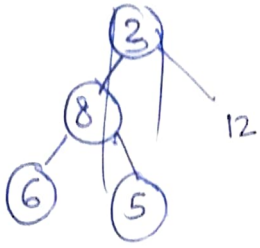


swapping 15 with 2 and deleting 15





deleting 15

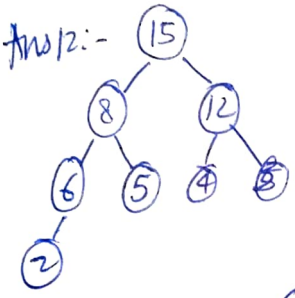


Ans 10 · In this case c^n grow exponentially and n^k grows polynomially. The asymptotic relation is that c^n is asymptotically larger than n^k

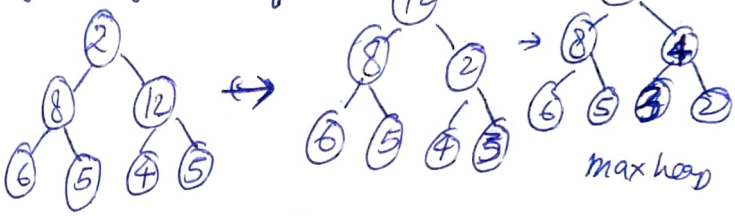
$$c^n = \Omega(n^k)$$

Exponential growth will eventually dominate polynomial growth for sign sufficiently large value of n.

Ans 12:-

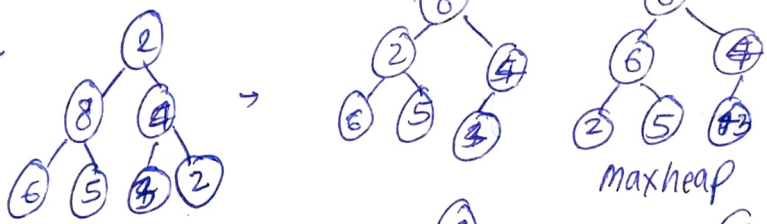


deleting 15



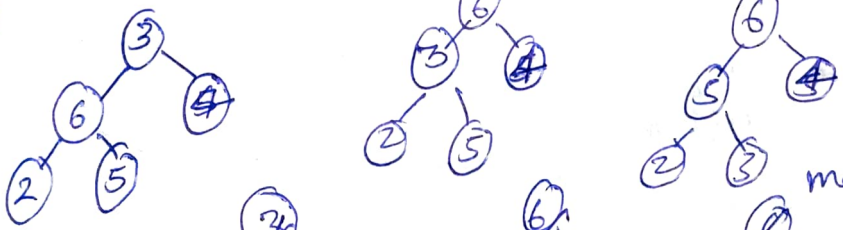
max heap

deleting 12



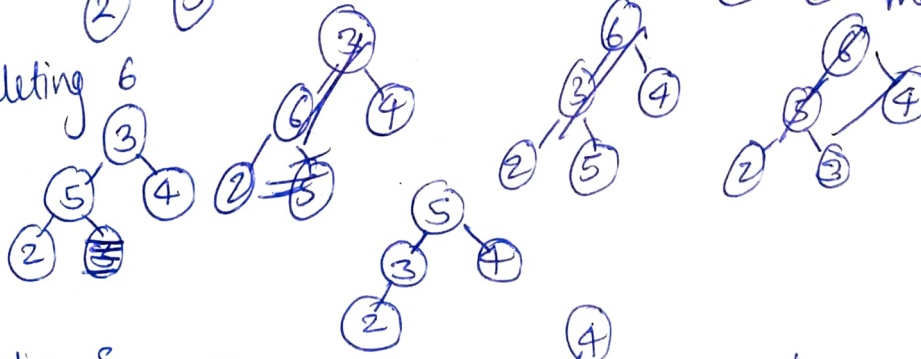
maxheap

deleting 8



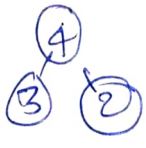
max heap

deleting 6

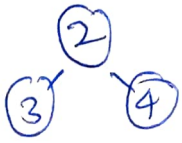


max heap

deleting 5



max heap



deleting 4

deleting 3

Ans 8:-

```
void function (int n)
{
    if (n==1)
      return;
    for (i=1; i<=n; i++) {
      for (j=1; j<=n; j++)
      {
          printf ("*");
      }
    }
    function (n-3);
}
```

The nested loop result has time complexity of $O(n^2)$

The function call itself with $n-3$

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

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

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

The dominant term is $O(n^3)$

Time complexity is $O(n^3)$.

Ans 11: The time complexity of the extractMin() operation in min heap is $O(\log n)$.

extractMin() is used to extract the minimum element from min heap which is usually root of heap.

After extracting the minimum element, the last element of the heap is moved to root and a heapify operation is performed to move this element down the heap to its correct position

The extraction of minimum element and heapify operation has time complexity proportional to height of heap i.e $O(\log n)$.