Chapter 3

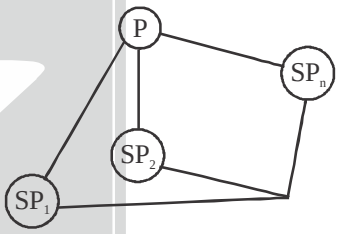
Recurrence relation & Recursion

Recurrence relation :
A recurrence is an equation or inequality that describes a function in term of itself with its smaller inputs.

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

The problem of size n is divided into '2' subproblems each of size n/2 and the cost of dividing + combining the solution is 'n'/

$T(n) = 2T\left(\frac{n}{2}\right) + n$ the cost of dividing + combining.



There are **THREE** phases :
(1) Divide $\rightarrow$ break complex problem
(2) Conquer $\rightarrow$ solve recursively.
(3) Combine $\rightarrow$ Combined solutions of all sub problems.
Examples:

$$T(n) = \begin{cases} \theta(1) & \text{If n} = 1 \\ 4\text{T}\left(\frac{n}{2}\right) + \theta(n) & \text{If } n > 1 \end{cases}$$

Or, in general, $T(n) = aT(n/b) + f(n)$ where $a \geq 1$ and $b > 1$

In the above recurrence the whole problem of input size n is divided into a subproblems of $\frac{n}{b}$ size and the cost of dividings and combining the solution is f(n).

Methods to solve recurrence relation :
(1) Backward substitution
(2) Recurrence tree method
(3) Master method
(4) Akra Bazzi method.

Backward substitution methods:
In this method we reduce value of n in backward way untill the size of problem becomes '1'.
Let us consider the following recurrence.

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

We can further expand it as follows:

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

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

:

:

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

$$= 1 + 2 + 3 + 4 + ... + n \left[\text{if } T(1) = 1 \right]$$

$$= \frac{n(n+1)}{2} = \theta(n^2)$$

Recurrence Tree Method :

Three step process :

(1) Construct the recurrence tree

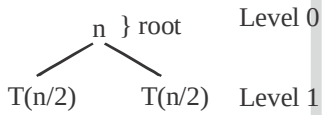
(2) Add the cost at each levels of tree

(3) Add the cost of all levels to get overall cost.

e.g. RR given : $T(n) = 2T\left(\frac{n}{2}\right) + n \qquad T(1) = 1$

In above rr. $T(n) = 2T\left(\frac{n}{2}\right) + (n)$

This 'n' is free from I, so we'll treat this n as root of tree.



Children according to the number of sub-problems here it is $\left(\frac{n}{2}\right) \rightarrow 2$ sub problem.

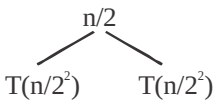
If we add both children and root, we will get the original R.R.

Now we have to grow this tree. So, we will calculate value of child leaf and grow it downward.

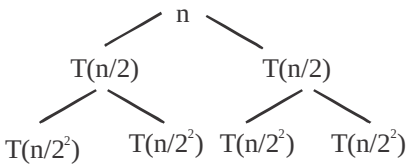
$$T\left(\frac{n}{2}\right) = 2T\left(\frac{n}{u}\right)$$

Or, $$T\left(\frac{n}{2}\right) = 2T\left(\frac{n}{2^2}\right) + \left(\frac{n}{2}\right) \rightarrow \text{cost of dividing 4}$$

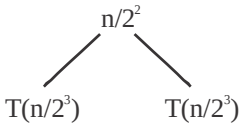
Now, this is a root c_3 it is free from T.



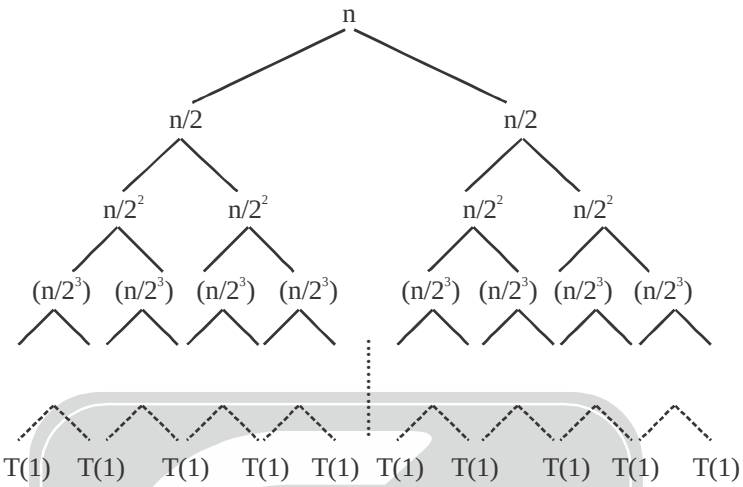
Now attach this partial tree to above parts tree.



Similarly : $$T\left(\frac{n}{2^2}\right) = 2T\left(\frac{n}{2^3}\right) + \frac{n}{2^2}$$

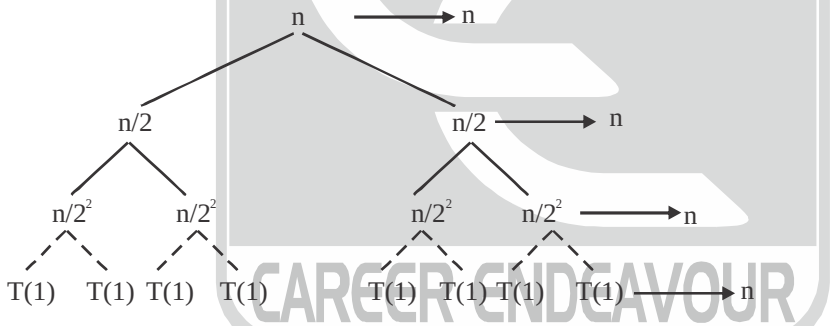


Now combine this partial tree into previous.



Now we create recurrence tree we will proceed to next step :

Step - 2 : Add cost of each level.



Step - 3 : Add all cost

$T(n) = \cos t = n + n + n.....n$ times

Now, how many times we will add n depend on HEIGHT OF TREE

At each level n is divided into 2 parts i.e.

$$\frac{n}{1} \to \frac{n}{2} \to \frac{n}{2^2} \to \frac{n}{2^3} \to \frac{n}{2^4} \ldots\ldots\ldots \frac{n}{2^i}$$

$$\text{level} \to 0 \quad 1 \quad 2 \quad 3 \quad 4 \quad\quad i$$

At last level that is $\frac{n}{2^i}$ value $\frac{n}{2^i} = 1$

$$n = 2^i$$

Put log on both sides we get

$$\log_2 n = \log_2 2^i \Rightarrow \log_2 n = i \log_2 2 \quad \because \log_2 2 = 1$$

$$\log_2 n = i$$

i = height of the tree = $\log_2 n$.
so, total cost will be

$$T(n) = n + n + n + n \ldots\ldots + \log_2 n \text{ times}$$
$$= O(n \log n)$$

Master Method :
Master method is used for the following recurrences of the form.

$T(n) = aT\left(\frac{n}{b}\right) + f(n)$ where $a \geq 1$ and $b > 1$ are constants and f(n) is an asymptotically positive function then.

Case 1: If $f(n) = O\left(n^{\log_b^a - \varepsilon}\right)$ for some constant $\varepsilon > 0$ then $T(n) = \theta\left(n^{\log_b^a}\right)$

Case 2: If $f(n) = \Omega\left(n^{\log_b^a}\right)$ then $T(n) = \theta\left(n^{\log_b^a} \log n\right)$

Case 3: If $f(n) = \theta\left(n^{\log_b^a + \varepsilon}\right)$ for some constant $\varepsilon > 0$, then $T(n) = \theta\left(f(n)\right)$

Examples:

(a) $T(n) = 9T(n/3) + n$

here a = 9, b = 3 and f(n) = n.

$n^{\log_b^a} = n^{\log_3^9} = n^2$ since $f(n) = O\left(n^{\log_3^9 - \varepsilon}\right)$

Where $\varepsilon = 1$ we can apply case 1 of master method and calculate that $T(n) = \theta\left(n^2\right)$.

(b) $T(n) = T\left(\frac{2n}{3}\right) + 1$

here a = 1 b = $\frac{3}{2}$ $f(n) = 1$; $n^{\log_b^a} = n^{\log_{3/2}^1} = n^0 = 1$

Case 2 of master method applies and thus the solution to the recurrence relation is $T(n) = \theta(\log n)$

(c) $T(n) = 3T(n/4) + n \log n$

We have $a = 3, b = 4$; $f(n) = n \log n$

$n^{\log_b^a} = n^{\log_4^3} = O\left(n^{0.793}\right)$ Since $f(n) = \Omega\left(n^{\log_4^3 + \epsilon}\right)$

Where $\varepsilon \approx 0.2$ case 3 applies as

$af(n/b) = 3\left(\frac{n}{4}\right)\log\left(\frac{n}{4}\right) \leq n \log n = c\ f(n)$ for $c = \frac{3}{4}$

There for by case of master theorem:

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

(d) $T(n) = 2T(n/2) + n\log n$

Here a = 2, b = 2 and $f(n) = n\log n$

$n^{\log_b^a} = n^{\log_2^2} = n$ as $f(n)$ is not polynomially larger than $n^{\log_2^2}$. Therefore it does not follow any case of master theorem.

(e) $T(n) = 2T\left(\lfloor \sqrt{n} \rfloor\right) + \log n$

Soln. Let $n = 2^m$ and ignoring the $\lfloor\ \rfloor$ sign.

$$T\left(2^m\right) = 2T\left(2^{m/2}\right) + m$$

Now let $T\left(2^m\right) = S(n)$ then

$$S(m) = 2S(m/2) + m$$

here $a = 2$ $b = 2$ and $f(m) = m$

$m^{\log_b^a} = m^{\log_2^2} = m = f(m)$ so by case II of master method.

$S(m) = \theta(m\log m)$ Now changing back.

$$T(n) = S\left(2^m\right) = \theta(m\log m)$$
$$= \theta(\log n \log\log n)$$

Akra bazzi Method :

If the recurrence has form

$$T(n) = \sum_{i=1}^{n} a_i\left(T\left(\frac{n}{b^i}\right)\right) + f(n) \qquad a_i > 0,\ b_i > 1$$

With $\sum_{i=1}^{k} \frac{a_i}{b_i^p} = 1 \Rightarrow \frac{a_1}{b_1^p} + \frac{a_2}{b_2^p} + \frac{a_3}{b_3^p} + \ldots\ldots\ldots + \frac{a_k}{b_k^p} = 1$

where, p = row.

$$T(n) = \theta\left(n^p\left(1 + \int_1^n \frac{f(u)}{u^{p+1}}du\right)\right) \qquad f(n) = f(u)$$

We will consider the equation and find value of P put it in formula, perform integration and then get the value.

e.g. $T(n) = T\left(\frac{3n}{4}\right) + T\left(\frac{n}{4}\right) + n$

$$T(n) = a_1 T\left(\frac{n}{b_1}\right) + a_2 T\left(\frac{n}{b_2}\right) + f(n)$$

$a_1 = 1$ $a_2 = 1$

$b_1 = \frac{4}{3}$ $b_2 = 4$

$$\frac{a_1}{b_1^p}+\frac{a_2}{b_2^p}=1$$

$$\frac{1}{\left(\frac{4}{3}\right)^p}+\frac{1}{4p}=1 \Rightarrow \left(\frac{3}{4}\right)^p+\left(\frac{1}{4}\right)^p=1 \quad \Rightarrow \quad p=1 \quad \frac{3}{4}+\frac{1}{4}=1$$

$$T(n)=\theta\left(n^1\left(1+\int_1^n \frac{u}{u^{1+}}du=\theta\left(n+n\int_1^n \frac{u}{u^2}du=\theta\left(n+n\left[\log u\right]_1^n\right.\right.\right.$$

$$=\theta\left(n+n\left(\log n-\log 1\right)=\theta\left(n+n\log n\right)\right.$$

$$n\log n > n$$

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

e.g. $T(n)=T(n-1)+T(n-2)$

$T(1)=1$ let $T(n)=x^n$

$T(2)=1$

$f_n = f_{n-1} + f_{n-2}$

1, 1, 2, 3, 5, 8, ...

$$x^n = x^{n-1} + x^{n-2} \quad \Rightarrow \quad x^n = \frac{x^n}{x}+\frac{x^n}{x^2}$$

$$1=\frac{1}{x}+\frac{1}{x^2} \quad \Rightarrow \quad 1=\frac{x+1}{x^2} \qquad x^2-x-1=0$$

$$x=\frac{-b\pm\sqrt{b^2-4ac}}{2a} \quad \Rightarrow \quad x=\frac{1\pm\sqrt{5}}{2} \quad \Rightarrow \alpha=\frac{1+\sqrt{5}}{2} \quad \beta=\frac{1-\sqrt{5}}{2}$$

Solution is $T(n)=c_1\alpha^n+c_2\beta^n=c_1\left(\frac{1+\sqrt{5}}{2}\right)^n+c_2\left(\frac{1-\sqrt{5}}{2}\right)^n$

$$T(n)=\theta\left(1+\frac{\sqrt{5}}{2}\right)^n=\theta\left(\frac{1+2.14}{2}\right)^n=\theta\left(\frac{3}{2}\right)^n$$

Concept of Recurrsion:

Recursion means self-reference. A recursive function is a function which calls itself during its execution. It must have following properties.

(a) There must be certain criteria, called base criteria for which function does not call itself.

(b) Each time the function call itself (directly or indirectly), it must be closer to the base criteria i.e. it uses the arguments smaller than the one it was given.

A recursive function with these properties is said to be well defined.

Examples:

(a) Factorial function:

$n!=1.2.3.4.....n$ and 0!=1

$$n!=\begin{cases}1 & \text{if } n=0\\ 1.2.3 \quad .....n & \text{if } n\geq 1\end{cases}$$

Computing factorial n by function (Recursive)

```
int factorial (int n)
{
If (n = = 0)}                    //Base criteria.
return 1;
else
return n*factorial (n–1);  Inductive step
}
```

How it executes if n = 4

1. 4! = 4 .3!
2. 3! = 3.2!
3. 2! = 2.1!
4. 1! = 1.0!
5. 0!=1 (Termination)
6. 1!=1.1 =1
7. 2! = 2×1 = 2
8. 3! = 3×2 = 6
9. 4! = 4×6 = 24

(b) Fibonacci Numbers:

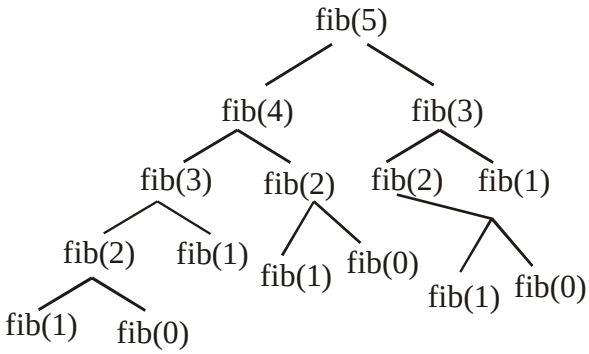
A fibonacci sequence f_0, f_1, f_2, f_3 ..... fn is defined as 0, 1, 1, 2, 3, 5, 8, 3, ...
That is $f_0 = 0$, $f_1 = 1$, $f_2 = 1$ and the subsequent terms are sum of 2 preceding terms.

$$f_n = \begin{cases} n & \text{if } n \le 1 \\ f_{n-1} + f_{n-2} & \text{if } n > 1 \end{cases}$$

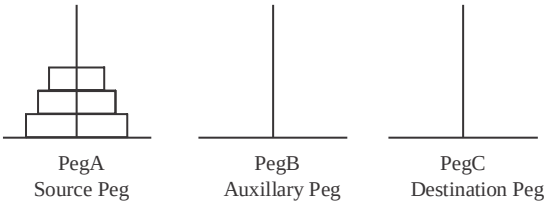
Computing n^{th} fibonacci number by recurrence.

```
Int fib (int n)
{
        If (n ≤ 1) } // Base criteria.
        return n;
else
        return fib (n–1) + fib (n–2);     (Inductive step)
}
```

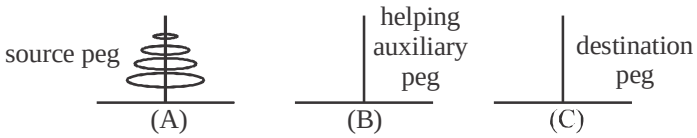
fib (5) will be computed as follows:



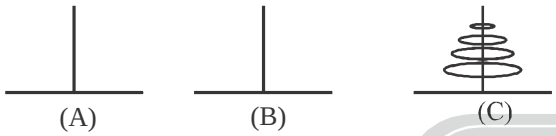
TOWER OF HANOI PROBLEMS



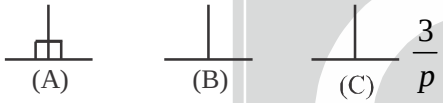
Initial configuration of TOH problem is :

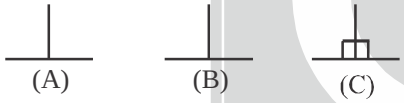


Final configuration of TOH problem :



e.g. **For One disk :**

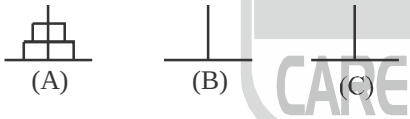
Initial Stage :  $\frac{3}{p}$ $\boxed{move : A \to C}$

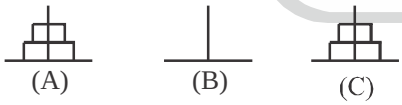
Final stage :  $\frac{0}{p}$

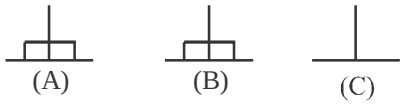
When n = 1

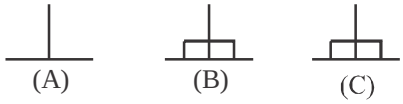
number of movement = $1\left(2^n - 1\right)$

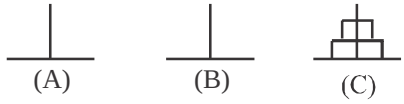
For two disk :

Initial Stage : 

Final state : 

Moves : **(i)**  $A \to B$

(ii)  $A \to C$

(iii)  $B \to C$

So, when n = 2

Number of movement (3) $\left(2^n - 1\right)$

Goal: All discs are sent to Peg C from PegA by movements of disc in other pegs.

(i) Only one disc movement is allowed at a time.

(ii) A larger disc cannot be placed over a smaller disc.

Number of movement of n disc in tower of Hanoi problem is $2^n - 1$.

$T(n)$ = total numebr of disc movement

Number of disc = n in THP

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

Recursion or Iteration :

A recursive function works through the process of calling itself until a condition is met. Iteration uses a looping control structure (while, do while, for) to repeat a section of code until a condition is met. Recursion is best when you have a lot of variables to keep track of with each loop, and it's using a "divide and conquer" type of algorithm. Recursion helps to trim down the frame of reference (the portion of the problem you're now interested in), very easily. When we have few variables to keep track of, or the algorithm isn't some type of "divide and conquer", then iterative looping is better. Iteration is slightly faster, but that should not be the criteria used to choose it. The difference is very small, and the clarity and shortness of the recursive code, can be quite helpful for debugging. With iteration it is the same variable whose value is constantly changed, and you can access its value at each stage. With recursion, a different local variable (by the same name) is created with each call.

SOLVED PROBLEMS

1. Let T(n) be a function defined by recurrence **[GATE - 2005]**

$$T(n) = 2T\left(\frac{n}{2}\right) + \sqrt{n} \;\forall\; n \geq 2$$

(a) $T(n) = \theta\left(\sqrt{n}\right)$ (b) $T(n) = \theta(n)$ (c) $T(n) = \theta(\log n)$ (d) None of these

Soln. $T(n) = 2T\left(\frac{n}{2}\right) + \sqrt{n} \;\forall\; n \geq 2$

$f(n) = \sqrt{n} \Rightarrow n\log_b^a = n\log_2^2 = n \Rightarrow f(n) < n\log_b^a$

So, $T(n) = \theta\left(\sqrt{n}\right)$

Correct option is (a)

2. The recurrence relation **[GATE - 1996]**

$$T(1) = 2 \Rightarrow T(n) = 3T\left(\frac{n}{4}\right) + n$$

(a) $\theta(n)$ (b) $\theta(\log n)$ (c) $\theta\left(n^{3/4}\right)$ (d) None of these

Soln. (a)

3. For $n = 2^k$ for some $k \geq 0$ the recurrence relation

$$T(n) = \sqrt{2}\, T\left(\frac{n}{2}\right) + \sqrt{n}$$

$$T(1) = 1$$

(a) $\sqrt{n}(\log n + 1)$ (b) $\sqrt{n}(\log n)$ (c) $\sqrt{n}\log\sqrt{n}$ (d) $n\log\sqrt{n}$

Ans. (a)

4. The running time of an algorithm is represented by the following recurrence relation **[GATE 2009]**

$$T(n)=\begin{cases} n \le 3 \\ T\left(\frac{n}{3}\right)+c_n, & \text{otherwise} \end{cases}$$

which of the following represents the complexity of the above recurrence relation.

(a) $\theta(n)$ (b) $\theta(n\log n)$ (c) $\theta(n^2)$ (d) $\theta(n^2\log n)$

Soln. $$T(n)=\begin{cases} n \le 3 \\ T\left(\frac{n}{3}\right)+c_n, & \text{otherwise} \end{cases}$$

Solve using master theorem.

$$T(n)=T\left(\frac{n}{3}\right)+\infty \qquad \Rightarrow T(n)=aT\left(\frac{n}{b}\right)+f(n)$$

$$n\log_b^a \Rightarrow n\log_{31} \qquad \Rightarrow n$$

$$f(n)<n\log_b^a$$

So, $\theta(n)$

Correct option is (a)

5. The recurrence relation that arises with the complexity of binary search is: **[JNUEE-2008]**

(a) $T(n)=T\left(\frac{n}{2}\right)+n$ (b) $T(n)=2T\left(\frac{n}{2}\right)+k$

(c) $T(n)=T\left(\frac{n}{2}\right)+\log n$ (d) $T(n)=T\left(\frac{n}{2}+k\right)$

Soln. For binary search if we have *n* number of elements. So, first it sorts the array in ascending or descending order then divide the array in half of its number and repeat it till we find out specified number.

$$T(n)=2T\left(\frac{n}{2}\right)+k$$

Correct option is (b)

6. Consider the following c-code.

```
int j, n;
j = 1;
while (j < n)
j = j*2;
```

The numbers of comparision made in the execution of the loop for any n > 0 is **[GATE 2007]**

(a) $\lceil \log n \rceil + 1$ (b) n (c) $\lceil \log n \rceil$ (d) $\lceil \log n \rceil + r$

Soln.

```
int j, n;
    j = 1;
    while (j < n)
    j = j*2;
```

For n = 1, J = 1

J = 2

For n = 2, J = 2
For J = 1, n = 3, J = 2
For J = 2, J = 1; out of loop
n = 4, J = 1 for J = 2
For J = 2, J = 4; out of loop
Similarly for n = 100
It will execute for 11 steps.
So, option **(a)** is true.

7. How to find the complexity of a program

Soln. Complexity is basic rough estimate complexity is depend on number of loops

```
for (i =0; i < n; i++) n
        for (j = 0; j < n; j++) n
```

$\theta\left(n^2\right)$

8. Find out the complexity of a program.

```
i = n;
        while (i > 0)
        {  i = i/2;
        }
```

Soln. $i = \frac{n}{2^i} = 1 \quad \Rightarrow \quad n = 2^i \quad \therefore \quad i = \log_2 n$

$\log_2^{2i} = \log_2^n = i\log_2^2 = \log n$

$i = \log n \quad \Rightarrow \theta\left(\log n\right)$

9. Find a solution to the following recurrence equation $T(n) = T\left(\frac{n}{2}\right) + n, \quad T(1) = 1$

[GATE-1987 : 2 Marks]

Soln. By using substitution method we get following series:

$\Rightarrow \quad T\left(\frac{n}{2^k}\right) + \frac{n}{2^{k-1}} + \frac{n}{2^{k-2}} \cdots \frac{n}{2^2} + \frac{n}{2^n} + n$

$\Rightarrow$ So put $k = \log_2 n$ then we get

$$T\left(\frac{n}{2^{\log_2 n}}\right) + n\left(\frac{1}{2^0} + \frac{1}{2^1} + \frac{1}{2^2} \cdots \frac{1}{2^{\log_2 n - 1}}\right) = O(n)$$

10. The recurrence relation

$T(1) = 2$
$T(n) = 3T(n/4) + n$
Has the solution T(n) equal to
(a) O(n) (b) O(log n) (c) $O(n^{3/4})$ (d) none of these

[GATE-1996 : 2 Marks]

Ans. (a)
Soln. Using Master Theorem
We get $n^{\log_4 3} < n$

11. Let T(n) be the function defined by $T(1) = 1,\ T(n) = 2T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + \sqrt{n}$ for $n \geq 2$. Which of the following statements is true ?

(a) $T(n) = O(\sqrt{n})$ (b) $T(n) = O(n)$ (c) $T(n) = O(\log n)$ (d) none of these

[GATE-1997 : 2 Marks]

Ans. (b)

Soln. Use Master Theorem $n^{\log_2 2} \geq \sqrt{n}$.

12. In the following C functions, let $n \geq m$.

```
int gcd (n, m) {
    if (n%m = = 0} return m;
    n = n%m;
    return gcd (m, n); }
```

How many recursive calls are made by this function?

(a) $\Theta(\log_2 n)$ (b) $\Omega(n)$ (c) $\Theta(\log_2 \log_2 n)$ (d) $\Theta\left(\sqrt{n}\right)$

[GATE-2007 : 2 Marks]

Ans. (a)

Soln. Let T(m, n) be the total number of steps. So T(m, 0) = 0, T(m, n) = T(n, m mod n) on average

$$T_n = \frac{1}{n}\sum_{0 \leq k \leq n} T(k, n);\ T_n \approx 1 + \frac{1}{n}(T_0\, T_1 + \cdots + T_{n-1})$$

$$T_n \approx S_n$$

$$S_n = 1 + \frac{1}{n}(S_0\, S_1 + \cdots + S_{n-1})$$

$$S_n = 1 + \frac{1}{n+1}(S_0\, S_1 + \cdots + S_n) = 1 + \frac{1}{n+1}(n(S_{n-1}) + S_n) = 1 + \frac{1}{n+1} = S_n + \frac{1}{n+1}$$

So $T_n \approx \Theta(\log_2 n) + 0(1)$

$T \approx \Theta(\log_2 n)$

13. Which one of the following is the recurrence equation for the worst case time complexity of the Quicksort algorithm for sorting n(≥2) numbers? In the recurrence equations given in the option below, c is a constant.

(a) $T(n) = 2T(n/2) + cn$ (b) $T(n) = T(n-1) + T(1) + cn$

(c) $T(n) = 2T(n-1) + cn$ (d) $T(n) = T(n/2) + cn$

[GATE-2015 (Set-1) : 1 Mark]

Ans. (b)

Soln. In worst case the pivot element position may be either first or last. In that case the elements are always divided in $1 : (n-1)$ proportion.

The recurrence relation for such a proportional division would be $T(n) = T(1) + T(n-1) + O(n) = O(n^2)$

(a) $T(n) = 2T\ (n/2) + cn = O(n \log n)$ (b) $T(n) = T(n-1) + T(1) + cn = O(n^2)$

(c) $T(n) = 2T\ (n-1) + cn = O(2^n)$ (d) $T(n) = T(n/2) + cn = O(n)$

PRACTICE SET

1. Consider the following recursive function fun(x, y). What is the value of fun(4, 3)

```
int fun(int x, int y)
{
        if(x == 0) return y;
        return fun(x -1,  x + y);
}
```

(a) 13 (b) 12 (c) 9 (d) 15

2. What does the following function do?

```
int fun(int x, int y)
{   if(y == 0)   return 0;
return(x + fun(x, y-1));
}
```

(a) x+y (b) x*y (c) x+x*y (d) x^y

3. Output of following program?

```
#include<stdio.h>
void print(int n)
{
        if(n > 4000)        return;
        printf("%d ", n);
        print(2*n);    printf("%d ", n);
}
        int main()
        {
        print(1000);
        getchar();
        return 0;
        }
```

(a) 1000 2000 4000 (b) 1000 2000 4000 4000 2000 1000
(c) 1000 2000 4000 2000 1000 (d) 1000 2000 2000 1000

4. Consider the same recursive C function that takes two arguments

```
unsigned  int foo(unsigned int n, unsigned int r)
{
        if(n  > 0) return(n%r +  foo (n/r, r ));
        else return 0;
}
```

(a) 9 (b) 8 (c) 5 (d) 2

5. Consider the following function

```
double f(double x)
{
        if(abs(x*x - 3) < 0.01) return x;
        else return f(x/2 + 1.5/x);
}
```

Give a value q (to 2 decimals) such that f(q) will return q:_____.
(a) 1.73 (b) 2.24 (c) 4.22 (d) 3.42

6. $T(1) = 1$
$T(n) = 2T(n-1) + n, n \geq 2$
evaluates to
(a) $2^{n+1} - n - 2$ (b) $2^n - n$
(c) $2^{n+1} - 2n - 2$ (d) $2^n - n$

7. Consider the following three claims
1. $(n + k)^m = \Theta(n^m)$, where k and m are constants
2. $2^{n+1} = O(2^n)$
3. $2^{2n+1} = O(2^n)$
Which of these claims are correct ?
(a) 1 and 2 (b) 1 and 3 (c) 2 and 3 (d) 1, 2 and 3

8. What is the worst case time complexity of following implementation of subset sum problem.

```
// Returns true if there is a subset of set[] with sum equal to given sum
boolan SubsetSum(int set[], int n, int sum)
{  // Base Cases  if(sum == 0)     return true;
if(n == 0 && sum != 0)     return false;
// If last element is greater than sum, then ignore it  if(set[n-1] > sum)
return isSubsetSum(set, n-1, sum);
 /* else, check if sum can be obtained by any of the following
 (a) including the last element
(b) excluding the last element
*/  return isSubsetSum(set, n-1, sum) || isSubsetSum(set, n-1, sum-set[n-1]);}
```

(a) O(n * 2^n) (b) O(n^2) (c) O(n^2 * 2^n) (d) O(2^n)

ANSWER KEY

1. (a) 2. (b) 3. (b) 4. (d) 5. (a) 6. (a) 7. (a)

8. (d)