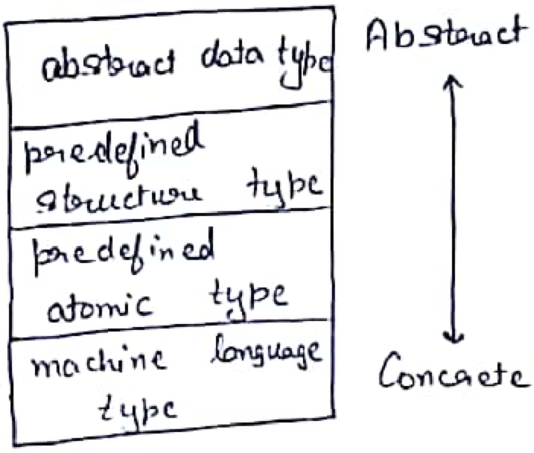
Unit - 1
DAA

Elementary data Structures.

1. Abstract Data Type
2. Stack
3. Queue
4. Linked - List
5. Binary Tree

→ Abstract Data Type - ADT consists of a data type together with a set of operations, which define how the type may be manipulated.



Virtual data type exist on a virtual processor such as a programming language.

Physical data type exist on a physical processor such as the machine level of computer.

Advantage -

1. modularity
2. precise specification
3. information hiding
4. simplicity
5. integrity
6. implementation independence

An ADT has two parts:

1. Value definition
2. Operation definition

Value definition is again divided into 2 parts:

(a) Definition clause states the contents of the data type

(b) Condition clause defines any condition that applies to data type.

Operational definition is divided into 3 parts:

(a) Function

(b) Pre condition

(c) Post condition

→ Stack -

A stack is a temparary abstract data type and data structure based on the principle of Last ~~In~~ In First Out (LIFO).

Stack is a container of nodes and has two basic operations: push and pop

Push adds a given node to the top of the stack

Pop removes and return the current top node of the stack.

If an empty stack is popped, we say the stack underflows.

If top of stack exceeds n, the stack overflows.

Infix Notation - In airthmetic operations, the operator symbol is placed between the operands.

Eg - A + BC - DE * FG/H

Prefix Notation - In airthmetic operations, the operator comes before the operands.

Eg - + A * B C

Postfix Notation - In airthmetic operations, the operator comes after the operands.

Eg - ABC.*+

→ Queue - Queue works on the principle of first - in - first - out (FIFO). In queue insertion can take place only at one end called rear and deletion of element can take place at another end called front. We call the insert operation on a queue ENQUEUE and we call the delete operation on a queue DEQUEUE.

Types of Queue

1. Deque - A deque is a linear list in which elements can be added or removed at either end but not in the middle. The term deque is a contraction of name double - ended queue.

   Two kinds of deque :

   - Input restricted deque - deque which allow insertion at one end but allow deletion at both end
   - Output restricted deque - deque which allow deletion at one end but allow insertion at both end.

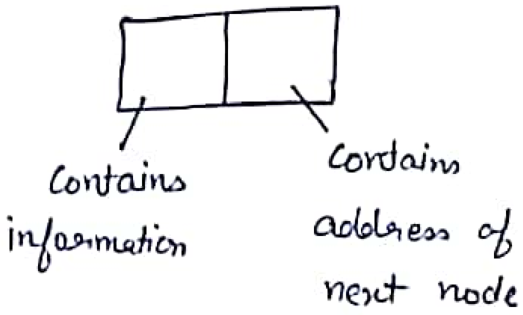
2. Priority Queue - A priority queue is a collection of elements such that each element has been assigned a priority and such that order in which elements are deleted.

→ Linked list -

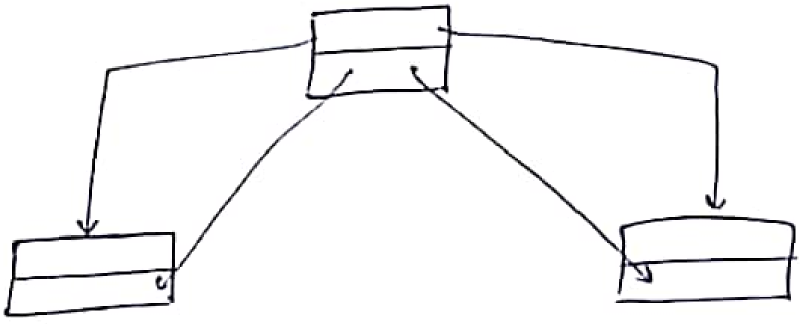
Linked list store collection of ~~stay~~ data. Linked list allocate memory for each element seperately and only when necessary.

A linked list ~~is~~ or one way ~~list~~ list is a ~~coll~~ linear collection of data each

called nodes. Each node is divided into two parts: the first part contains the information of the element and second part called the link field which contains the address of next node in the list.



→ Binary Tree - We represent each node of tree by an object. As in linked list, we assume that each node contains a key field. The remaining field contains pointes to other node, and they vary according to type of tree.



The root of entire tree T is ~~organised~~ pointed to by the attribute root [T]. If root[T] = NIL, then the tree is empty.

Algorithm and its complexity

An algorithm is a set of rules for ~~carrying~~ carrying out calculation either by hand or on a machine. It is a sequence of computational steps that transform the input into output. It is a sequence of operations performed on data that have to be organized in data structures. A finite set of instruction that specify a sequence of operation to be carried out in order to solve a specific problem is called ~~a~~ an algorithm.

Complexity - It is very convenient to classify algorithm based on the relative amount of time or ~~a~~ relative amount of space they require and specify the growth of time/ space requirement as a function of the input size.

Time Complexity - Running time of the program as a function of size of input.

Space Complexity - Space complexity of an algorithm based on how much space an algorithm needs to complete ~~a~~ its task.

⇒ Worst case running time - The behaviour of the algorithm with respect to the worst possible case of the input instance. The worst - case running time of an algorithm is an upper bound on the running time for any input.

⇒ Average case running time - The expected behaviour when the input is randomly drawn from a given distribution. The average - case running time of an algorithm is an estimate of the running time for an "average" input.

⇒ Amortized running time - Here the time required to perform a sequence of operations is averaged over all the operations performed. Amortized analysis can be used to show that the average cost of an operation is small.

Eg - • Quick Sort

Worst case - $O(n^2)$

Average case - $O(n \log n)$

• Merge Sort, Heap Sort -

Worst case - $O(n^2)$

Average case - $O(n^2)$

Analyzing algorithm

Algorithm analysis is an important part of computational complexity theory, which provide ~~theorit~~ theoretical estimation for the required resources of an algorithm to solve a specific computational problem.

The analysis of algorithm provide information that gives us a general idea of how long an algorithm will take for a given problem set.

Analysis of algorithm is determined as

1. Space analysis
2. Time analysis

Space analysis - The space complexity of an algorithm ~~it~~ is the amount of space it needs to run to ~~compilation~~. compilation. An analysis of space means how many cells of a turing machine tape would be needed to solve ~~the~~ ~~problem~~ a given problem. It requires a ~~st~~ set of rules to determine how operations are to be counted.

The space needed for an algorithm is the sum of following component:

1. A fixed part that is independent of the characteristics of inputs and outputs.

```
Eg -    Algo  Add (a, b, c, m,n)
        {
            for (i =1  — m)                              - m
            {
                for (j =1 — n)                           - mn
                {
                    c[i, j] = a{i,j] + b[i,j] ;              - mn
                }
            }
      }
```

Complexity - ~~[illegible]~~ $O(2mn+m)$

→ Step Table method.

	s/e	Frequency	Total
Algo sum(a, n)	0	-	-
{	0	-	-
sum = 0;	1	1	1
for (i = 1 - n)	1	n	n
{	0	-	-
sum = sum + a[i];	1	n	n
}	0	-	-
return sum;	1	1	1
}	0	-	-
			2n+2

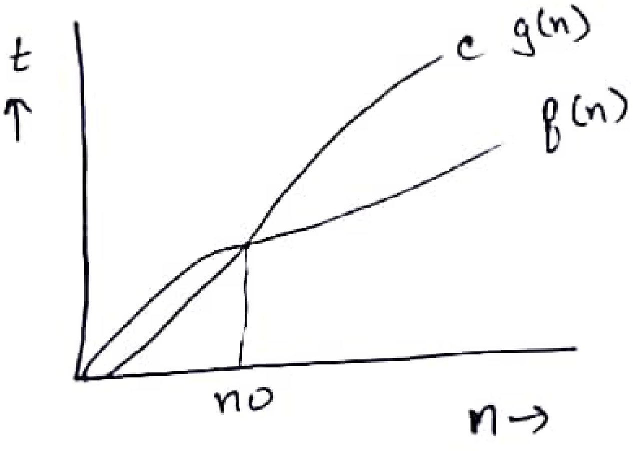
Complexity = O(2n+2)

Total = s/e * frequency

a positive integer.

Asymptotic Notations

1. Big - O notation - It is the formal method of expressing the upper bound of an algorithm's running time. It is measure of the longest amount of time it would take to complete the algorithm. It is used for worst case analysis.

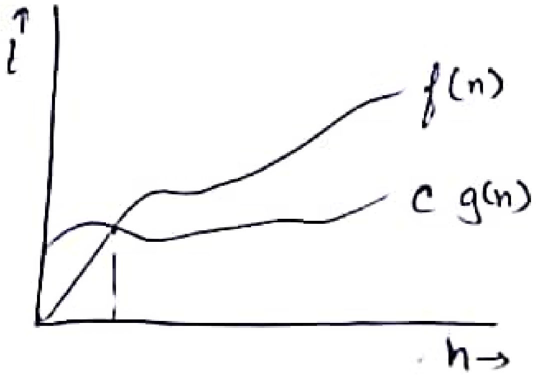


$$0 \leq f(n) \leq c\, g(n)$$

for $c > 0$
$n \geq n_0$

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

2. Big Omega (Ω) notation - For non-negative functions $f(n)$ and $g(n)$, if there exist an integer n_0 and a constant $c > 0$ such that for all integer $n > n_0$, $f(n) \geq c\, g(n)$, then $f(n)$ is big omega function of $g(n)$. It is used for best case analysis.

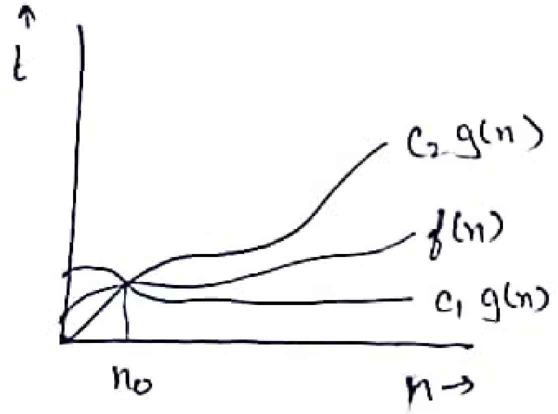


$$f(n) \geq c\, g(n)$$

$c > 0$
$n > n_0$

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

3. Theta notation - The lower bound and upper bound of the function f is provided by the theta ~~notata~~ notation.



$$\beta(n) = \theta(g(n))$$

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

$c_1, c_2 > 0$
$n \geq n_0$

4. Small o - notation -

$$f(n) < c\, g(n)$$

5. Small ω notation -

$$f(n) > c\, g(n)$$

6. Small θ notation -

$$c_1\, g(n) < f(n) < c_2\, g(n)$$

Priority Queue

Priority queue is a variant of queue in data structure in which insertion is performed in order of arrival and deletion is performed on the basis of priority.

It is of two type :

(i) Max Priority Queue.

(ii) Min Priority Queue.

Max priority queue supports :

(i) Insert (ii) Maximum (iii) Extract max

(iv) Increase key

Min priority queue ~~supports~~ supports :

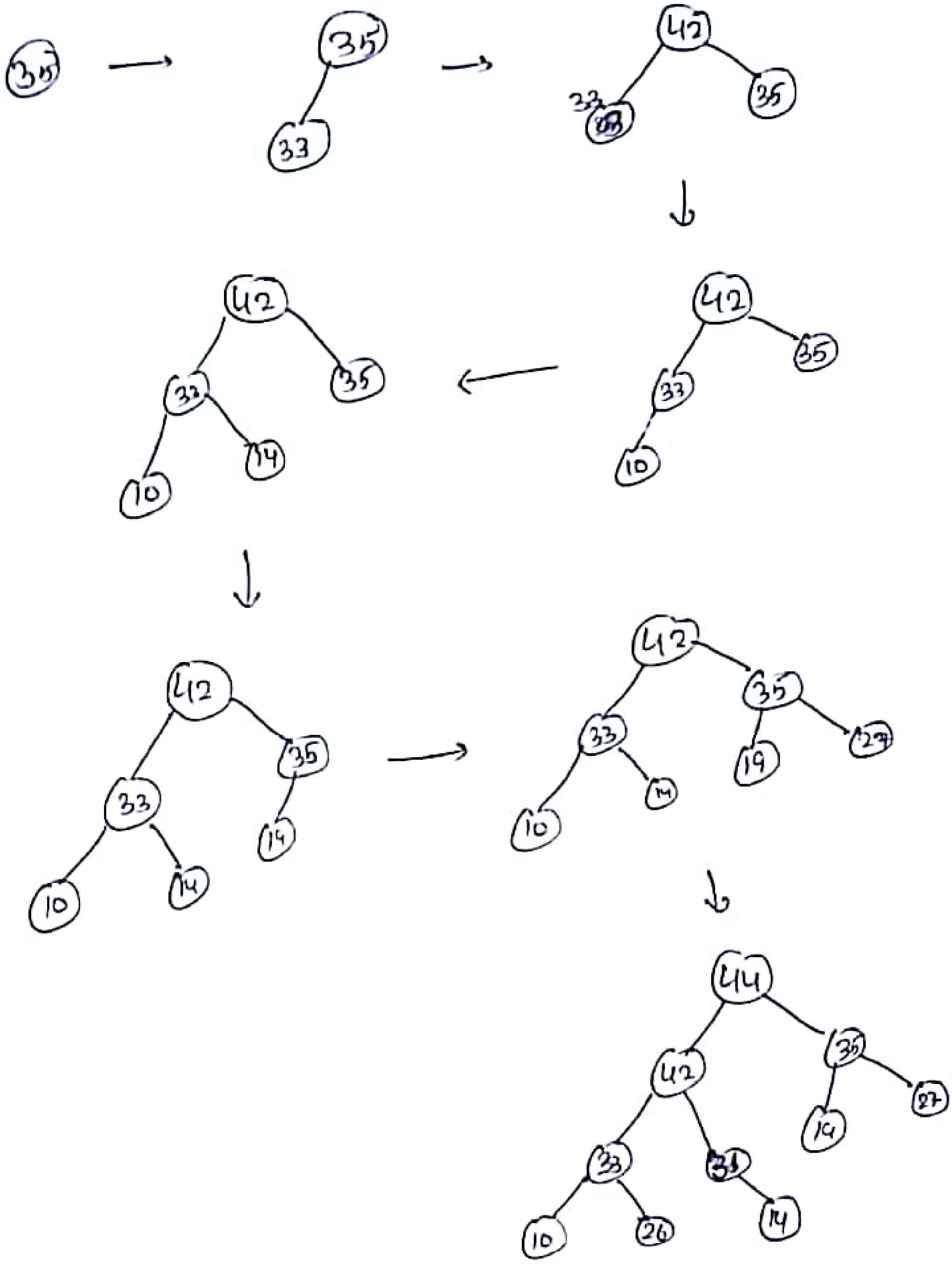
(i) Insert (ii) Minimum (iii) Extract min

(iv) Decrease key

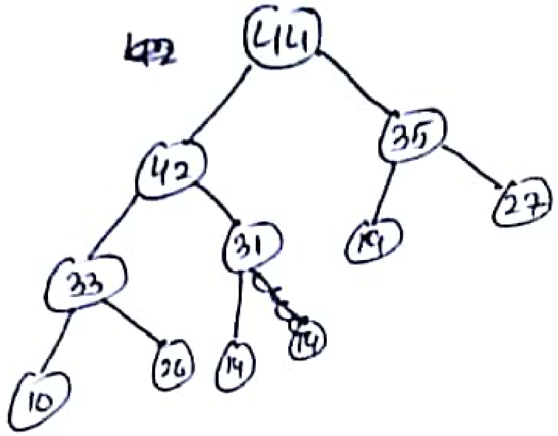
Implementation

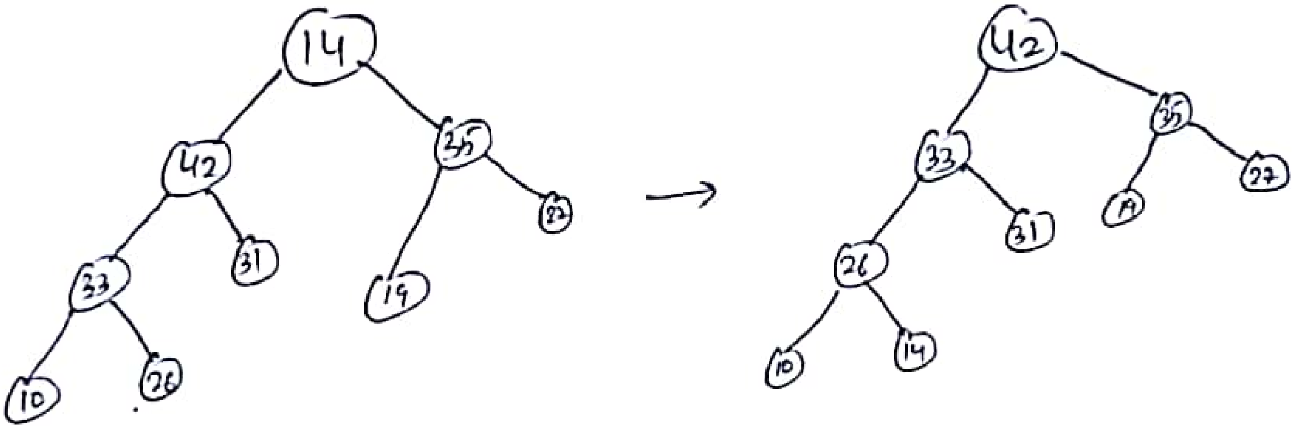
Eg - 35, 33, 42, 10, 14, 19, 27, 44, 26, 31

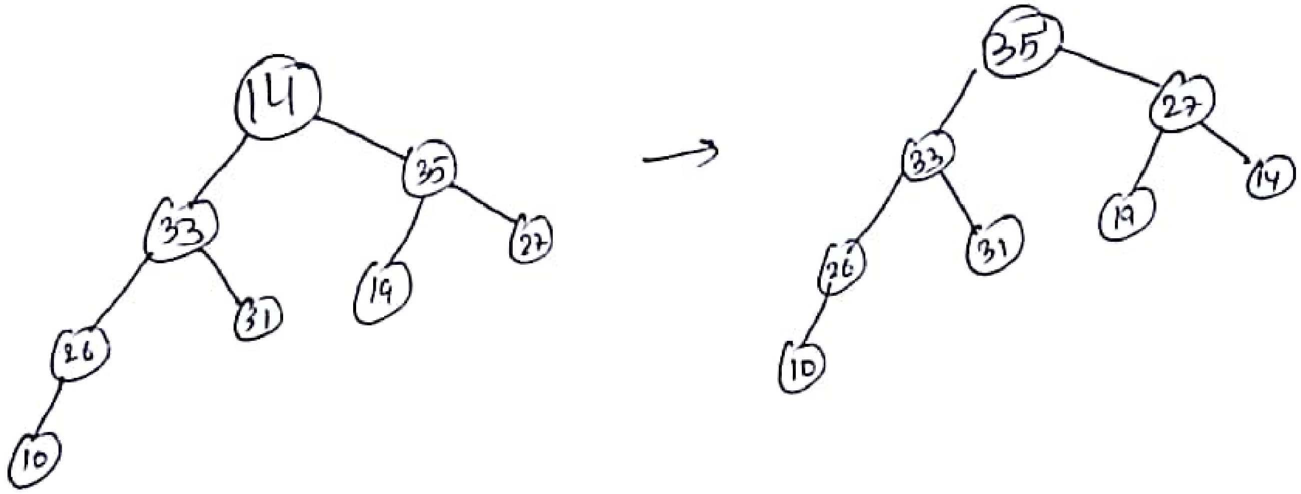
Insertion.



Deletion







and so — on

Quick Sort

Quick Sort is a divide and conquer algorithm. It picks an element as pivot and partitions
~~around~~ ~~the~~ ~~picked~~
~~the~~ ~~given~~ ~~element~~ ~~as~~ ~~pivot~~
the given array around the picked pivot. There are mony different versions of quicksort that picks pivot in different ways:

- → Always pick first element as pivot
- → Always pick last element as pivot
- → Pick random element as ~~po~~ pivot
- → Pick medium element as pivot.

Target of partition is to put all smaller elements of an array before pivot and all greater element after pivot.

Quick Sort

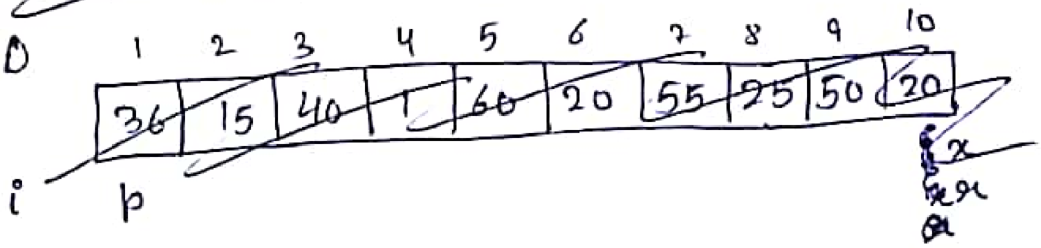
```
Quick - Sort (A, p, r)

Step 1    If p < r then
Step 2        q ← Partition (A, p, r)
Step 3        Quick - Sort (A, p, q-1)
Step 4        Quick - Sort (A, q+1, r)
```

```
Partition (A, p, r)

Step 1    x = A[r]
Step 2    i = p - 1
Step 3    for j = p to r - 1
Step 4        if A[j] ≤ x
Step 5            i = i + 1
Step 6            exchange A[i] with A[j]
Step 7    exchange A[i+1] with A[r]
Step 8    return i + 1
```

Eg ~~36, 15, 40, 1, 60, 20, 55, 25~~, 50, 20



0	1	2	3	4	5	6	7	8	9	10
	36	15	40	1	60	20	55	25	50	20

Eg - 36, 15, 40, 1, 60, 20, 55, 25, 50, 20

0	1	2	3	4	5	6	7	8	9	10
	36	15	40	1	60	20	55	25	50	20
										x
i	p									r
	j									

$x = A[10] = 20$

$i = p-1 = 1-1 = 0$

$j = 1$ to 9

$j=1$ and $i=0$

$A[j] \leq x$

$A[1] = 36 \nleq 20$

$j = 2$ and $i = 0$

$A[j] \leq x$

$A[2] = 15$ ~~≰~~ ≤ 20

$i = 0+1 = 1$ $A[1] \longleftrightarrow A[2]$

	1	2	3	4	5	6	7	8	9	10
	~~36~~ 15	36	40	1	60	20	55	25	50	20
										x
i			j							r
p										

$j = 3$ and $i=1$

$A[j] = A[3] = 40 \nleq 20$

$j = 4$ and $i=1$

$A[j] = A[4] = 1 \leq 20$

$i = 1+1 = 2$

$A[2] \longleftrightarrow A[4]$

1	2	3	4	5	6	7	8	9	10
15	1	40	36	60	20	55	25	50	20
p	i			j					r x

j = 5 and i = 2

A[5] = 60 $\nleq$ 20

j = 6 and i = 2

A[6] = 20 ≤ 20

i = 2+1 = 3 A[3] ⟷ A[6]

1	2	3	4	5	6	7	8	9	10
15	1	20	36	60	40	55	25	50	20
p		i				j			r x

j = 7 and i = 3

A[7] = 55 $\nleq$ 20

j = 8 and i = 3

A[8] = 25 $\nleq$ 20

j = 9 and i = 3

A[9] = 50 $\nleq$ 20

A[3+1] ⟷ A[10]

A[4] ⟷ A[10]

~~15~~	~~1~~	~~26~~	~~20~~	~~36~~	~~60~~	~~40~~	~~55~~	~~25~~	~~50~~

15	1	20	20	60	40	55	25	50	36

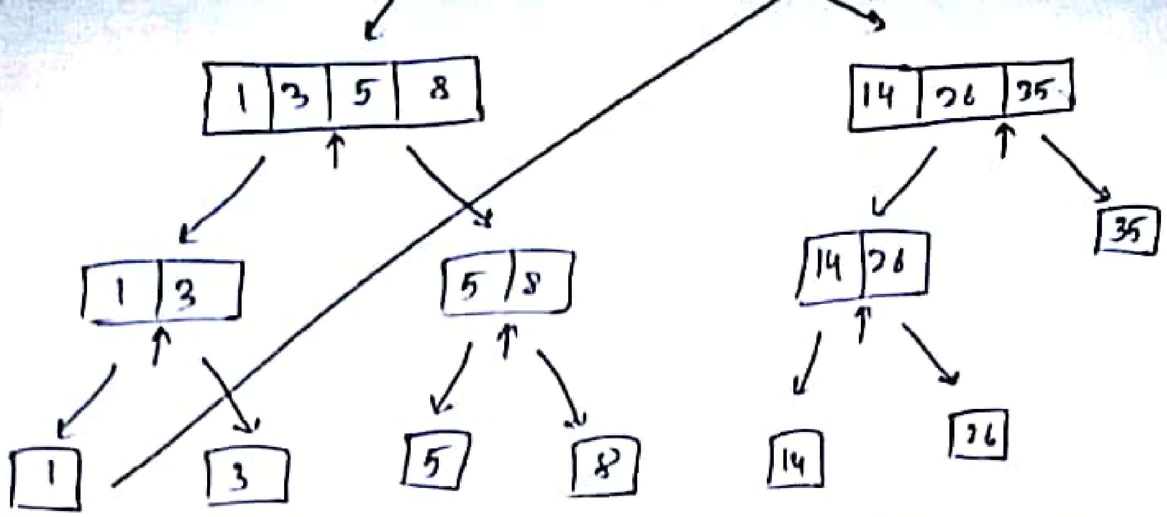
Merge Sort

It is a divide and conquer algorithm. It divide input arrays int in 2 halves, calls itself for two halves and then merges the two sorted halves.

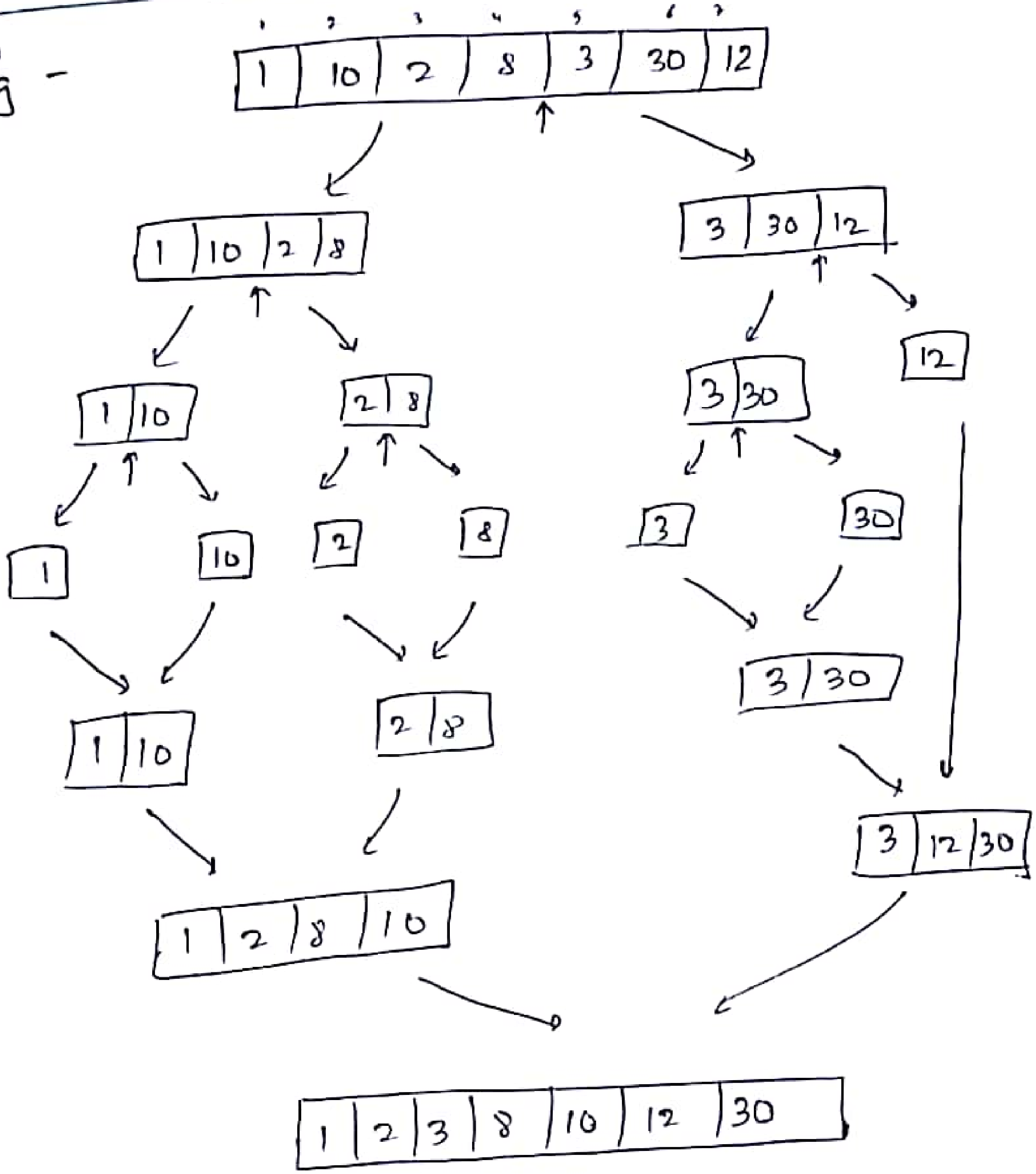
In merge sort the array is recursively divided into two halves till the size becomes 1. Once the size of array becomes 1, the merge processes come into action and start merging sorted. array back till the complete array is merged.

Merge Sort

```
Merge (A, p, q, r)
    $n_1 \leftarrow q - p + 1$
    $n_2 \leftarrow r - q$
    create arrays $L[1..n_1+1]$ and
                  $R[1..n_2+1]$
    for $i \leftarrow 1$ to $n_1$
        do $L[i] \leftarrow A[p+i-1]$
    for $j \leftarrow 1$ to $n_2$
        do $R[j] \leftarrow A[q+j]$
    $L[n_1+1] \leftarrow \infty$
    $R[n_2+1] \leftarrow \infty$
    $i \leftarrow 1$
    $j \leftarrow 1$
    for $k \leftarrow p$ to $r$
        do if $L[i] \leq R[j]$
            then $A[k] \leftarrow L[i]$
                 $i \leftarrow i + 1$
            else $A[k] \leftarrow R[j]$
                 $j \leftarrow j + 1$
```


1 | 3 | 5 | 8
14 | 26 | 35
1 | 3
5 | 8
14 | 26
35
1
3
5
8
14
26

Eg -


1 2 3 4 5 6 7
1 | 10 | 2 | 8 | 3 | 30 | 12
1 | 10 | 2 | 8
3 | 30 | 12
1 | 10
2 | 8
3 | 30
12
1
10
2
8
3
30
1 | 10
2 | 8
3 | 30
3 | 12 | 30
1 | 2 | 8 | 10
1 | 2 | 3 | 8 | 10 | 12 | 30

Recurrence

Recurrence is an equation that describes a function in terms of its value on smaller inputs.

3 methods :

(a) Substitution method

(b) Recursion method

(c) Master method

Substitution - It involves guessing the form of the solution and then using mathematical induction to find the constants.

Eg - $T(n) = 2T(\frac{n}{2}) + n$

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

$T(n) \leq c \; n \log n$

$$T(n) \leq 2\left(c \frac{n}{2} \log \frac{n}{2}\right) + n$$

$$\leq \cancel{2} \; c \; n \log \frac{n}{2} + n$$

$$\leq c \; n (\log n - \log 2) + n$$

$$\leq c n \log n - c n \log 2 + n$$

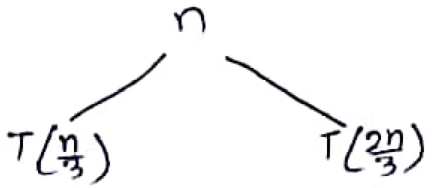
$$T(n) \leq cn \log n - cn + n$$

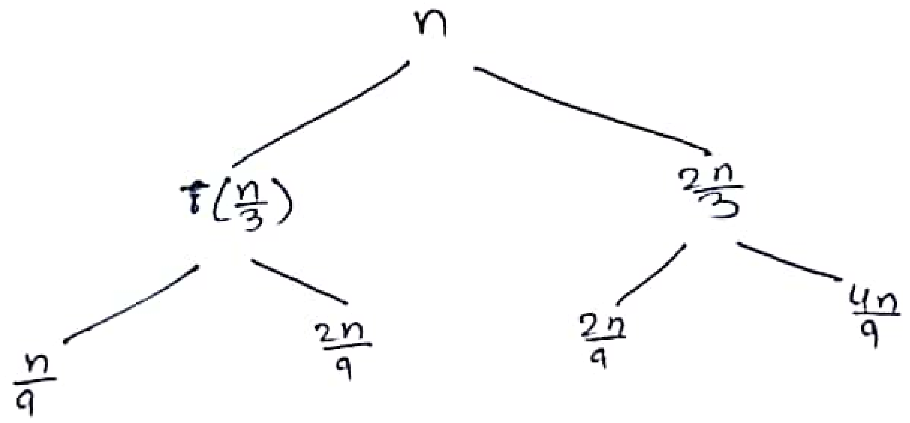
$$c = 1$$

$$T(n) \leq n \log n$$

Recursion - Recursion tree method is a pictorial representation of an iteration method, Which is in the form of tree, where at each level nodes are expanded.

Eg - $T(n) = T(\frac{n}{3}) + T(\frac{2n}{3}) + n$





$$n \longrightarrow \frac{2}{3}n \longrightarrow (\frac{2}{3})^2 n \longrightarrow \dots 1$$

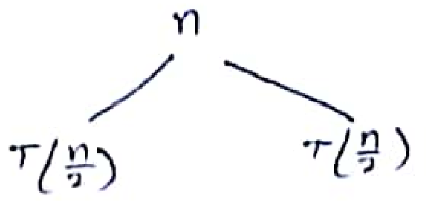
$$\frac{2}{3}n = 1$$

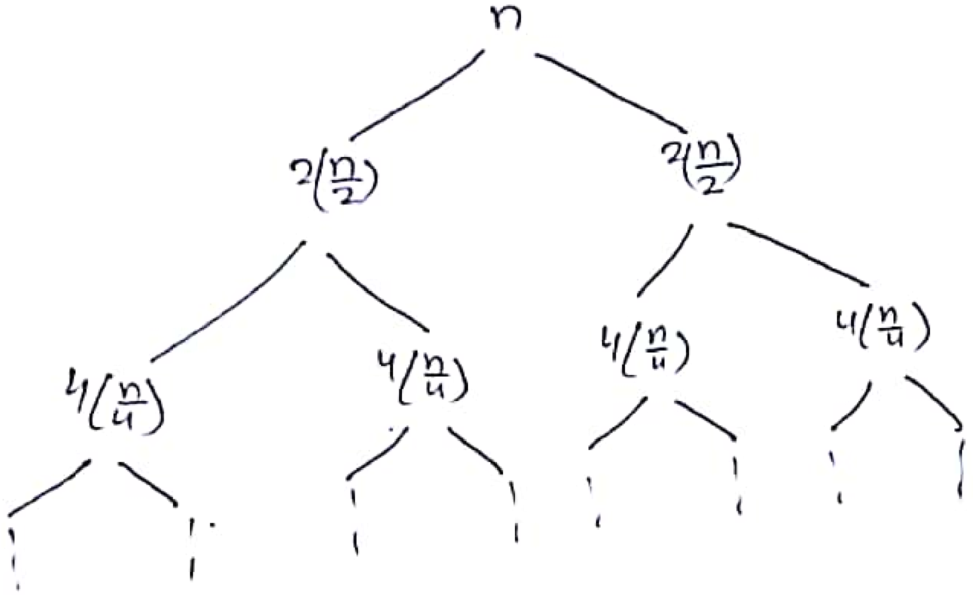
$$i = \log_{3/2} n$$

$$T(n) = n + n + n + \dots \text{ for } \log_{3/2} n$$

$$= \Theta(n \log n)$$

Eg - $T(n) = 4T\left(\frac{n}{2}\right) + n$





We have

$$n + 2n + 4n + \cdots \log_2 n$$

$$= n\,(1 + 2 + 4 + \cdots \log_2 n)$$

$$= n\,\frac{(2^{\log_2 n} - 1)}{2-1} \,.\, \frac{n\,(n-1)}{21} = n^2 - n$$

$$T(n) = \theta(n^2)$$

Master - The master method is used for solving the following type of recurrence

$$T(n) = aT\left(\frac{n}{b}\right) + f(n) \qquad a \geq 1$$
$$b > 1$$

1. If $f(n) = O(n^{\log_b a - \varepsilon})$

for some constant $\varepsilon > 0$

then $T(n) = \Theta(n^{\log_b a})$

2. If $f(n) = \Theta(n^{\log_b a})$

then $T(n) = \Theta(n^{\log_b a} \lg n)$

3. If $f(n) = \Omega(n^{\log_b a + \varepsilon})$

for some constant $\varepsilon > 0$

and if $a f(\frac{n}{b}) \leq c f(n)$

for some constant $c < 1$

then $T(n) = \Theta(f(n))$

Eg - $T(n) = 4T(\frac{n}{2}) + n$

$a = 4 \quad b = 2 \quad f(n) = n$

$n^{\log_2 4} = n^2$

$f(n) = O(n^{\log_b a - \varepsilon})$ if $\varepsilon = 1$

$= O(n^{2-1})$

$n = n$

$T(n) = \Theta(n^2)$

Eg -

$$T(n) = 4T\left(\frac{n}{2}\right) + n^2$$

$a = 4$ $b = 2$ $f(n) = n^2$

$$n^{\log_2 4} = n^2$$

$$f(n) = \Theta(n^{\log_b a})$$

$$n^2 = n^2$$

$$T(n) = \Theta(n^2 \lg n)$$

Chapter 2

Recurrence Relation

2.1 Elementary Data Structure

Q.1. What do you mean by Recurrence? What are different methods for solving recurrences.

Ans. *A recurrence is an equation or inequality that describes a function in term of its values on smaller inputs.* To solve a recurrence relation means to obtain a function defined on the natural numbers that satisfies the recurrence. For example, the worst-case running time $T(n)$ of the MERGE-SORT procedure is described by the recurrence.

$$T(n) = \begin{cases} \theta(1) & \text{if } n = 1 \\ 2T(n/2) + \theta(n) & \text{if } n > 1 \end{cases}$$

There are three methods for solving this:

- **Substitution method,** we guess a bound and then use mathematical induction to prove our guess correct.
- **Iteration Method,** converts the recurrence into a summation and then relies on techniques for bounding summation to solve the recurrence.
- **Master Method,** provides bounds for recurrences of the form.

$T(n) = a\,T(n/b) + f(n)$, where $a \geq 1, b > 1$ *and* $f(n)$ is a given function.

1. Substitution Method

The substitution method for solving recurrences entails two steps:

- Guess the form of the solution.
- Use mathematical induction to find the constant and show that solution works.

It can apply either upper or lower bound on recurrences. Let us apply on recurrences.

Example 1.

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

....(1)

We guess that the solution is

$$T(n) = O\,(n \lg n)$$

Our method is to prove

$T(n) \le Cn \lg n$ for an appropriate choice of $C > 0$.

We start that this bound holds for $n/2$

$$T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) \le C\left(\left\lfloor \frac{n}{2} \right\rfloor \lg \cdot \left\lfloor \frac{n}{2} \right\rfloor\right)$$

Substituting in equation (1) above, we get

$$T(n) = 2\left(C\left\lfloor \frac{n}{2} \right\rfloor \lg \cdot \left\lfloor \frac{n}{2} \right\rfloor\right) + n$$

$$= Cn \lg (n/2) + n$$

$$= Cn \lg n - Cn \log 2 + n$$

$$= Cn \log n - Cn + n$$

$$\le Cn \log n$$

where the last step holds as long as $C \ge 1$.

Example 2. Consider the recurrence:

$$T(n) = 2T\left(\left\lfloor \frac{n}{2} \right\rfloor + 16\right) + n$$

We have to show that it is asymptotically bound by $O\,(n \log n)$

Solution. For $T\,(n) = O\,(n \log n)$, we have to show that for some constant C.

$$T(n) \le Cn \log n$$

Put this in the given recurrence equation

$$T(n) \le 2\left[C\left(\left\lfloor \frac{n}{2} \right\rfloor + 16\right) \log\left(\left\lfloor \frac{n}{2} \right\rfloor + 16\right)\right] + n$$

$$= Cn \log (n/2) + 32 + n$$

$$= Cn \log n - Cn \log 2 + 32 + n$$

$$= Cn \log n - Cn + 32 + n$$

$$= Cn \log n - (C-1)n + 32$$

$$\le Cn \log n \text{ (for } C \ge 1)$$

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

Example 3.: Consider the recurrence

$$T(n) = T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + 1$$

We have to show that it is asymptotically bound by O (log n).

Solution. For T(n) = O (log n)

We have to show that for some constant C.

$$T(n) \leq C \log n.$$

Put this in the given recurrence equation.

$$T(n) \leq C \log\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + 1$$

$$\leq C \log\left(\frac{n}{2}\right) + 1 = C\log n - C\log_2 2 + 1$$

$$\leq C \log n \text{ for } C \geq 1.$$

Thus, T(n) = O (log n)

Making a good guess

1. If a recurrence is similar to one we have seen before, then guessing a similar solution is reasonable. For example,

$$T(n) = 2T\left(\left\lfloor \frac{n}{2} \right\rfloor + 17\right) + n$$

which looks difficult because of added "17" in the argument to T, but this additional term cannot substantially affect the solution to the recurrence. Because, when n is large, the difference between

$$T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) \text{ and } T\left(\left\lfloor \frac{n}{2} \right\rfloor + 17\right)$$

is not large and both cut n nearly evenly in half. Hence, we make the guess that T(n) = O (log n).

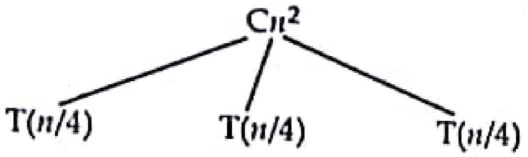
2. Another way to make a good guess is to prove loose upper and lower bounds on the recurrence and then reduces the range of uncertainty.
3. There are times when we can correctly guess at an asymptotic bound on the solution of a recurrence, but show how the math does not seem to work out in the induction. When we hit such a situation revising the guess by subtracting a lower order term often permits the math to go through.
4. Changing variables. Some times, a little algebraic manipulation can make an unknown recurrence similar to one we have seen before.

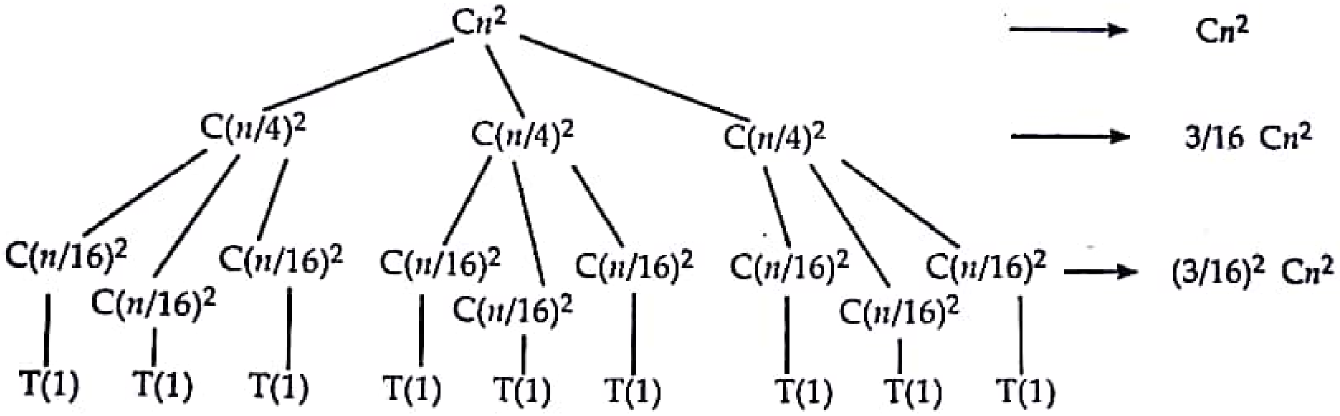
2. Recursion Tree Method

In a recursion tree, each node represents the cost of a single subproblem. Somewhere in the set of recursive function invocations. We sum the costs within each level of the tree to obtain a set of per level costs, and then we sum all the per-level costs to determine the total cost of all levels of the recursion.

Example 1. $T(n) = 3T\left(\left\lfloor \frac{n}{4} \right\rfloor + 0\left(n^2\right)\right)$

Solution.





Depth of tree is given by $\frac{n}{4^i} = 1$

So, $i = \log_4 n$.

Number of levels, $i + 1 = \log_4 n + 1$

$0, 1, 2, 3, \ldots\ldots\ldots, \log_4 n + 1$

Cost of each node = $C\left(\frac{n}{4}i\right)^2$

$$= 3^i C\left(\frac{n}{4}i\right)^2 = \left(\frac{3}{16}\right)^i C\, n^2$$

Cost at last level i.e. at depth $\log_4 n$

$$3^{\log_n n} = n \log_4 3$$

Each contributing cost T(1) for a total cost of $n \log_4 3$, which is $\theta(n \log_4 3)$.

Now we add up the cost

$$T(n) = Cn^2 + \frac{3}{16}Cn^2 + \left(\frac{3}{16}\right)^2 Cn^2 + \ldots\ldots\ldots + \left(\frac{3}{16}\right)^{\log_4 n-1} Cn^2 + 0\left(n \log_4 3\right)$$

$$\Rightarrow \quad \sum_{i=0}^{\log_4 n-1} \left(\frac{3}{16}\right)^i Cn^2 + \theta(n\log_4 3)$$

Now using geometric series

$$\because \quad \sum_{k=0}^{n} x^k = \frac{x^{n+1}-1}{x-1}$$

We have

$$\frac{\left(\frac{3}{16}\right)^{\log_4 n} - 1}{\left(\frac{3}{16}\right) - 1} \cdot Cn^2 + \theta(n\log_4 3)$$

For infinity decreasing geometric series as an upper bound, we have

$$\sum_{i=0}^{\infty} \left(\frac{3}{16}\right)^i \cdot Cn^2 + \theta(n\log_4 3)$$

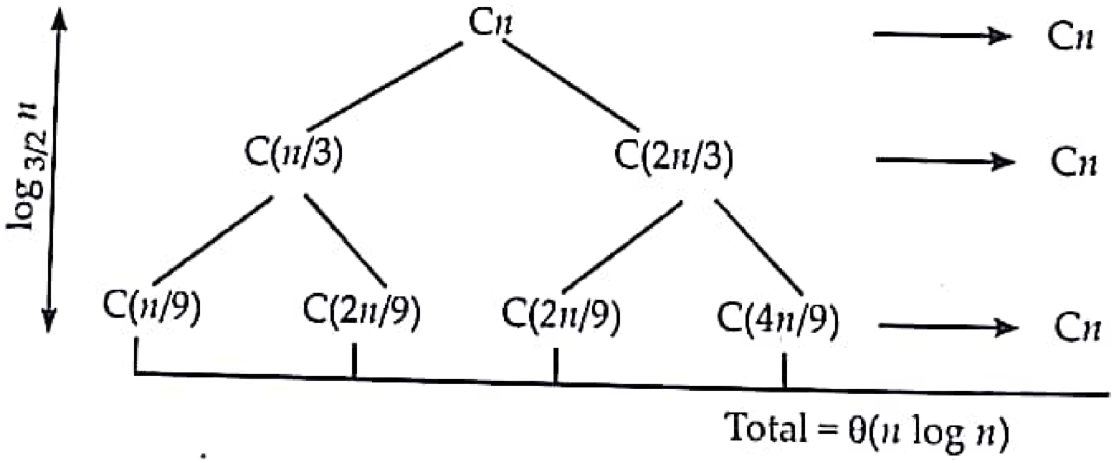
$$\Rightarrow \quad \frac{1}{1-\frac{3}{16}} \cdot Cn^2 + \theta(n\log_4 3)$$

$$\Rightarrow \quad \frac{16}{13} \cdot Cn^2 + \theta(n\log_4 3)$$

$$\Rightarrow \quad \theta(n^2)$$

Example 2. Consider the following recurrence:

Solution. $T(n) = T(n/3) + T(2n/3) + n$



When we add the values across the levels of the recursion tree, we get a value of n for every level. The longest path from the root to a leaf is

$$n \rightarrow \frac{2}{3}n \rightarrow \left(\frac{2}{3}\right)^2 n \rightarrow ....1$$

Since $\left(\frac{2}{3}\right)^i n = 1$ when $i = \log_{3/2} n$

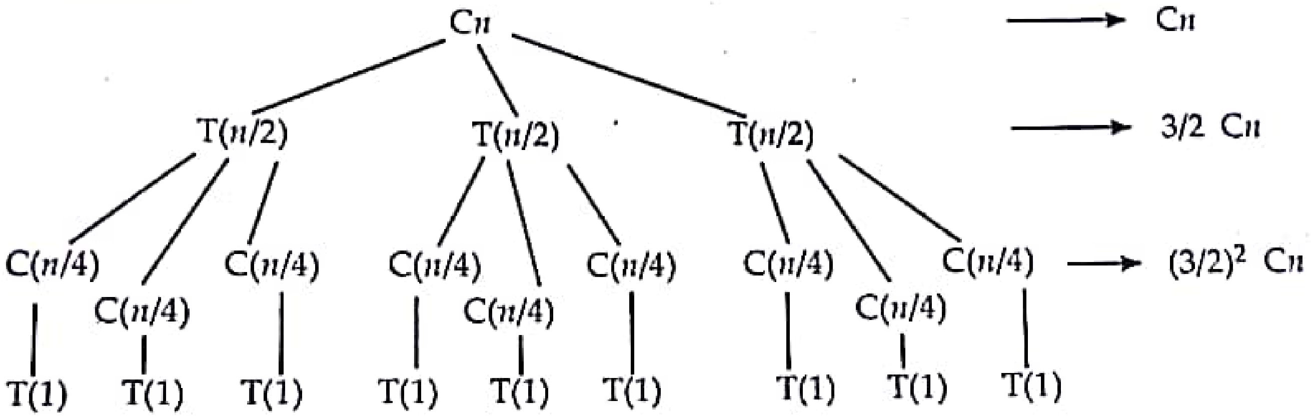
Thus the height of the tree is $\log_{3/2} n$

So, $T(n) = n + n + n + \ldots\ldots + \log_{3/2} n$ times

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

Example 3. $T(n) = 3T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + n$

Solution.



Depth of tree is given by $\frac{n}{2^i} = 1$

So, $i = \log_2 n$.

Total Nodes $3 \log_2 . n = n \log_2 3$

Cost at depth $\log_2 n = \theta(n \log_2 3)$

Now total cost

$$T(n) = Cn + \frac{3}{2}Cn + \left(\frac{3}{2}\right)^2 Cn + \ldots\ldots + \theta\left(n \log_2 3\right)$$

$$\Rightarrow \quad Cn \sum_{i=0}^{\log 2n-1} \left(\frac{3}{2}\right)^i + \theta\left(n \log_2 3\right)$$

$$\Rightarrow \quad Cn \frac{(3/2)^{\log_2 n}}{(3/2)-1} + \theta\left(n \log_2 3\right)$$

For infinity decreasing Geometric Series

$$T(n) = \sum_{i=0}^{\infty} (3/2)^i C_n + 0(n \log_2 3)$$

$$= \frac{1}{1-(3/2)} Cn + 0(n \log_2 3) = -2C_n + 0(n \log_2 3)$$

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

3. Master Method or Recurrence Method or 'Cook-book' Method

The master method is used for solving the following type of recurrence:

$$T(n) = a\,T(n/b) + f(n) \text{ with } a \geq 1 \text{ and } b > 1$$

In this problem is divided into 'a' subproblems, each of size n/b where a and b are positive constants. The cost of dividing the problem and combining the result of the subproblem is described by the function $f(n)$.

Master Theorem

Let T(n) be defined on the non-negative integers by the recurrence.

$$T(n) = a\,T(n/b) + f(n) \text{ with } a \geq 1 \text{ and } b > 1$$

be constant and $f(n)$ be a function.

Then T(n) can be bound asymptotically as follows:

(i) If $f(n) = 0(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \theta(n^{\log_b a})$

(ii) If $f(n) = \theta(n^{\log_b a})$, then $T(n) = \theta(n^{\log_b a} \lg n)$

(iii) If $f(n) = \Omega(n^{\log_b a^{a+\epsilon}})$ for some constant $\epsilon > 0$, and if a $f(n/b) \leq C f(n)$ for some $C < 1$ and all sufficiently large n, then $T(n) = \theta(f(n))$, a $f(n/b) \leq C f(n)$ is called the "regularity" condition.

Example 1. Solve the recurrence using master method:

$$T(n) = 9\,T(n/3) + n$$

Solution. Compare, the given recurrence with the

$$T(n) = a\,T(n/b) + f(n)$$

We get $a = 9, \quad b = 3, \quad f(n) = n$

Now, $n^{\log_b a} = n^{\log_3 9} = n^{\log_3 3^2}$

$$= n^2 = \theta(n^2)$$

$n^{\log_b a} > f(n)$ or $n^2 > f(n)$

So we apply case (i).

$$f(n) = O(n \log_3{}^{9-\epsilon}).$$

Let $\epsilon = 1$.

$$f(n) = O(n^{2-1})$$

So solution is $\theta(n^{\log_3 9}) = \theta(n^2)$

Example 2. Solve $T(n) = T\left(\frac{2n}{3}\right) + 1$ by master method. (KUK Dec. 2016)

Solution. Here $a = 1$, $b = 3/2, f(n) = 1$

Now, $n^{\log_b a} = n^{\log_{3/2} 1} = n^0 = 1$

$$f(n) = n^{\log_b a}$$

Hence case (ii) applies. Thus solution is

$$T(n) = \theta(n^{\log_b a} \lg n)$$

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

Example 3. Solve the recurrence by master method:

$$T(n) = 3T\left(\frac{n}{4}\right) + n \lg n$$

Solution. Here $a = 3$, $b = 4$, $f(n) = n \lg n$

$$n^{\log_b a} = n^{\log_4 3} = O(n^{0.793})$$

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

Case (iii) applies here.

$$f(n) = \Omega(n^{\log_4 3 + \epsilon}), \quad \text{where } \epsilon = 0.2$$

$$f(n) = n^{0.793+\epsilon}$$

$$\epsilon = 1 - 0.8 = 0.2$$

We check regularity condition, i.e.

$$a f(n/b) \leq C f(n) \qquad ....(1)$$

$$3 f(n/4) = 3 (n/4) \lg n/4$$

$$= (3/4) \ n \lg n - \lg 4$$

$$= (3/4) \ n \lg n$$

Put it in equation (1), we get

$$(3/4) \ n \lg n \leq C n \lg n$$

$$C = 3/4$$

Therefore, the solution is $T(n) = \theta(n \lg n)$.

Example 4. Solve $T(n) = 16\,T(n/4) + n^3$ by master method.

Solution. Here $a = 16$, $b = 4$, $f(n) = n^3$

$n^{\log_b a} = n^{\log_4 16} = n^2$

$f(n) = n^3$

$n^{\log_b a} < f(n)$ So, case (iii) applies here.

$f(n) = n^3 = n^{\log_b a + \epsilon} = n^{2+1}$, So $\epsilon = 1$

Now regularity condition, i.e.

$$16 f\left(\frac{n^3}{4^3}\right) \le C f\left(n^3\right)$$

$$16\frac{n^3}{64} \le Cn^3 \text{ for } C = \frac{1}{4}$$

So regularity condition holds. Hence by case (iii), the solution is $T(n) = \theta(f(n)) = \theta(n^3)$.

Example 5. Solve $T(n) = 7\,T(n/2) + n^2$.

Solution. Here $a = 7$, $b = 2$, $f(n) = n^2$

$n^{\log_b a} = n^{\log 7} = n^{2.8}$

$f(n) = O(n^{\log 7 - \epsilon})$, where $\epsilon \approx 0.8$

So, $T(n) = \theta(n^{\log 7})$

Example 6. Solve $T(n) = \begin{cases} T(1) & n = 1 \\ T(n/2) + C & n > 1 \end{cases}$

Solution. Here $a = 1$, $b = 2$, $f(n) = C$

$n^{\log_b a} = n^{\log_2 1} = 1$

$f(n) = C$

$n^{\log_b a} = 1$

Since two functions are of same. So, solution is find by case (ii).

$T(n) = \theta(n^{\log_b a} \lg n)$

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

$n^2 \cdot n^2$

$T(n) = \Theta(n^2 \lg n)$

DAA

Strassen's Matrix Multiplication

$$\left[\begin{array}{cc|cc} a_{11} & a_{12} & a_{13} & a_{14} \\ \multicolumn{2}{c|}{\textcircled{A}} & \multicolumn{2}{c}{\textcircled{B}} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ \hline a_{31} & a_{32} & a_{32} & a_{34} \\ \multicolumn{2}{c|}{\textcircled{C}} & \multicolumn{2}{c}{\textcircled{D}} \\ a_{41} & a_{34} & a_{24} & a_{44} \end{array}\right] \times \left[\begin{array}{cc|cc} b_{11} & b_{12} & b_{13} & b_{14} \\ \multicolumn{2}{c|}{\textcircled{E}} & \multicolumn{2}{c}{\textcircled{F}} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ \hline b_{31} & b_{32} & b_{33} & b_{34} \\ \multicolumn{2}{c|}{\textcircled{G}} & \multicolumn{2}{c}{\textcircled{H}} \\ b_{41} & b_{42} & b_{43} & b_{44} \end{array}\right]$$

$$= \begin{bmatrix} AE + BG & AF + BH \\ CE + DG & CF + DH \end{bmatrix}$$

$P_1 = A(F - H)$

$P_2 = (A + B)H$

$P_3 = (C + D)E$

$P_4 = D(G - E)$

$P_5 = (A + D)(E + H)$

$P_6 = (B - D)(G + H)$

$P_7 = (A - C)(E + F)$

$AE + BG = P_5 + P_4 - P_2 + P_6$

$AF + BH = P_1 + P_2$

$CE + DG = P_3 + P_4$

$CF + DH = P_5 + P_3 + P_1 + P_7$

It is an application of familiar design technique " divide and conquer".

Suppose we wish to product $C = AB$ where A, B and C are $n \times n$ matrices. Assuming that n is an exact power of 2, we divide A, B, and C into four $\frac{n}{2} \times \frac{n}{2}$ matrices.

$$C = AB$$

$$\begin{bmatrix} r & g \\ t & u \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} e & g \\ f & h \end{bmatrix}$$

$$r = ae + bf$$

$$g = ag + bh$$

$$t = ce + df$$

$$u = cg + dh$$

~~Strassen's method has 4 steps:~~

~~Divide the input matrices A & B into $\frac{n}{2} \times \frac{n}{2}$ sub matrices.~~