(H) (Theta Notation) $0 \leq c_2 g(n) \leq f(n) \leq c_1 g(n)$

O (Big oh) $0 \leq f(n) \leq c_1 g(n)$

$\Omega$ (omega) $0 \leq c_2 g(n) \leq f(n)$

o (Little oh) $0 \leq f(n) < c_1 g(n)$

$\omega$ (Little omega) $0 \leq c_2 g(n) < f(n)$

# Algorithm Analysis + Design (AAD)

## Algorithms

It is any well defined computational procedure that takes some value or a set of values as input and produces some value or a set of values as output

Algorithm is a tool for solving well specified computational problem.

eg. we might need to sort a sequence of numbers into nondecreasing order. Here is how we formally define the sorting problem:-

Input: A sequence of n numbers $(a_1, a_2 \ldots a_n)$

SORT //

Output: A permutation (reordering) $(a_1', a_2', \ldots a_n')$ of input sequence such that

$a_1' \leq a_2' \leq \ldots \leq a_n'$

Instance of a problem:- consists of input needed to compute a solution to the problem.

eg. input sequence (31, 41, 59, 26, 41, 58), a sorting algo returns as output the sequence (26, 31, 41, 41, 58, 59). Such an input sequence is called an instance of sorting problem.

An algorithm is said to be correct if for every input instance, it halts with the correct output.

An incorrect algorithm might not halt at all on some input instances or it might halt with an answer other than desired one.

## Growth of Functions

The order of growth of running time of an algorithm gives a simple characterization of algorithm's efficiency and also allows us to compare the relative performance of alternative algorithms.



## Asymptotic notation

The notation we use to describe the asymptotic running time of an algo are defined in terms of functions whose domains are set of natural numbers, $N = \{0, 1, 2 - -\}$.

Asymptotic Tight Bound

### $\Theta$-notation (Theta notation)

Let $f(n)$ + $g(n)$ be functions with domain $\{1, 2, 3 \_\_\_\_\}$ we write $f(n) = \Theta(g(n))$ and say that $f(n)$ is of order $g(n)$ if there exists positive constant $c_1$ + $c_2$ and no such that

$$0 \le c_2 g(n) \le f(n) \le c_1 g(n) \quad \text{for } n \ge n_0$$

eg. $f(n) = 5n^2 + 6n + 7$

$$0 \le 5n^2 \le f(n) \le 18n^2 \quad \text{for } n \ge 1$$

$$\Rightarrow f(n) = \Theta(n^2)$$

Let there be $n!$

$n! = n * n-1 * n-2 * \dots * 1$

$\leq n * n * n * n \dots$

$\leq n^n$

$\leq 1 \cdot (n^n)$

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

$= O(n^n)$

Lets take an example of $n = 11$

since $11 \leq 11$ → this is the lowest value from which the condition of '$\leq$' starts getting true hence is big Oh

$\leq 12$

$\leq 13$

$\leq 14$

$\leq 15$

$\leq \dots$

$\leq 99$

Now we know big O goes to upper bound.

$1$

$n^1$

$n^2$ → for lower bound

$O(n^3)$ $\quad$ $\Omega(n^3)$ → where they both matches is called Θ

$n^4$

$n^5$

$n^6$

$\vdots$

Eg $f(n) = 5n^3 + 6$

$\leq 5n + 6n$ $\quad n \geq 1$

$\leq 11n$

$= O(n)$

but this is considered oo !

$\leq 5n^2 + 6n^2$

$\leq 11n^2$

$= O(n^2)$

Similarly $O(n^3)$, $O(n^4)$

## Asymptotic upper bound :-

**Big-Oh Notation** :- Let $f(n)$ & $g(n)$ be functions with domain $\{1, 2, 3, \dots\}$ we write $f(n) = O(g(n))$ and say that $f(n)$ is of order almost $g(n)$ if there exists +ve constants $c$ and $n_0$ s.t.

$0 \leq f(n) \leq c_1 g(n)$ for $n \geq n_0$

NOTE

eg. $f(n) = 5n^2 + 6n + 7$

$f(n) \leq 5n^2 + 6n^2 + 7n^2$ for $n \geq 1$

$f(n) \leq 18n^2$

$\Rightarrow f(n) = O(n^2)$ $\quad c = 18$, $n_0 = 1$

eg. $f(n) = 5n^2 + 6n + 7$

$f(n) \leq 5n^3 + 6n^3 + 7n^3$

$f(n) \leq 18n^3$ $\quad (n \geq 1)$

$\Rightarrow f(n) = O(n^3)$ $\quad n_0 = 1$, $c = 18$

$f(n) = O(n^2), O(n^3), O(n^4), O(n^5) \dots$

$f(n) = 16 * 1$

$f(n) = O(1)$ $\quad n_0 = 0$, $c = 16$

$f(n) = 16 n^0$

$f(n) \leq 16$ $\quad n \geq 0$

$f(n) \neq O(n \log n)$

$f(n) \neq O(n)$

$f(n) \neq O(1)$ → ?

## Asymptotic lower bound :-

**Omega Notation** :- Let $f(n)$ & $g(n)$ be two function with domain $\{1, 2, 3 \dots\}$

if $0 \leq c_2 g(n) \leq f(n)$ for $n \geq$ some $c_2$ and $n_0$ are +ve constants, then we write $f(n) = \Omega g(n)$ and say that $f(n)$ is of order atleast $g(n)$

$f(n) = 5n^2 + 6n + 7$

NOTE

$f(n) \geq 5n^2$

$f(n) = \Omega(n^2)$ $\quad (n \geq 1)$

$f(n) \geq 6n$

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

$f(n) \geq 7 * 1$ $\quad n_0 = 0$

$f(n) = \Omega(1)$ $\quad c = 7$

$0 \le f(n) \le c\, g(n)$ big O

6

Little-Oh-Notation:- Let $f(n)$ & $g(n)$ be 2 functions. If $0 \le f(n) < c_1 g(n)$ for $n \ge n_0$ where $c_1$ & $n_0$ are +ve constants then we write $f(n) = o(g(n))$ and say little-oh notation for $f(n)$ is $g(n)$.

Little-Omega-Notation:- Let $f(n)$ & $g(n)$ be two functions. If $0 \le c_2 g(n) < f(n)$ for $n \ge n_0$ where $c_2$ and $n_0$ are +ve constants, then we write $f(n) = \omega(g(n))$ and say $f(n)$ is of order little omega $g(n)$

eg. $f(n) = 5n^2 + 6n + 7$

$f(n) = O(n^3), O(n^4), O(n^5)$

$f(n) = \omega(n), \omega(1)$

(i) $f(n) \le 5n^2 + 6n^2 + 7n^2$ for $n \ge 1$

(ii) $f(n) < 5n^3 + 6n^3 + 7n^3$

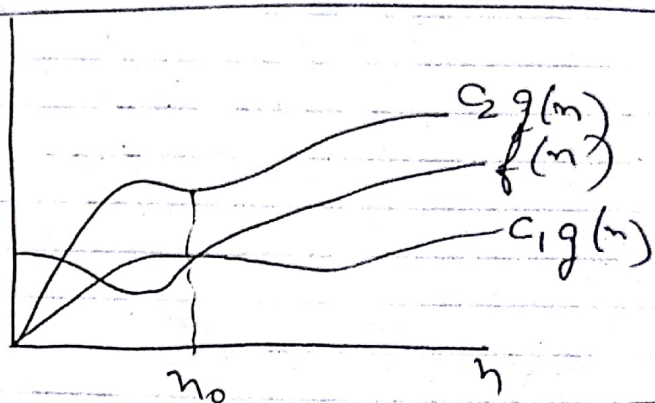
Note:-

(1.) (Big-oh notation of $f(n)$) ∩ (Omega notation of $f(n)$) = Theta notation of $f(n)$

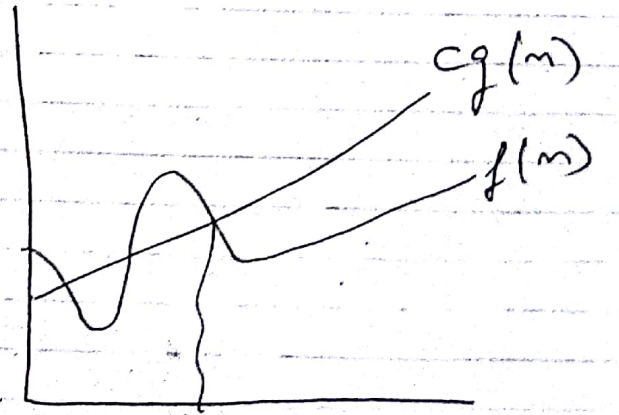
7

(2.) (Big-oh notation) − (Theta notation) = (Little-oh-notation)

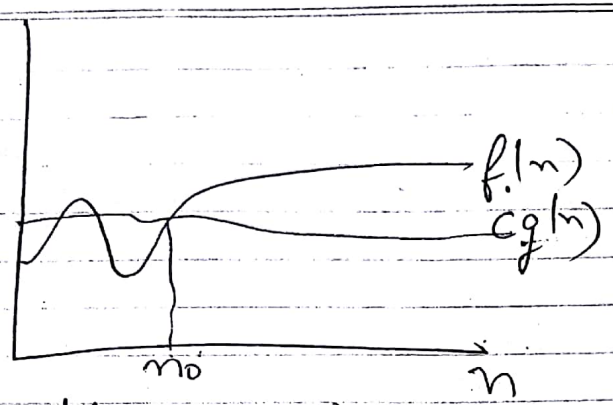
(3.) (Omega notation) − (Theta notation) = (Little-omega-notation)



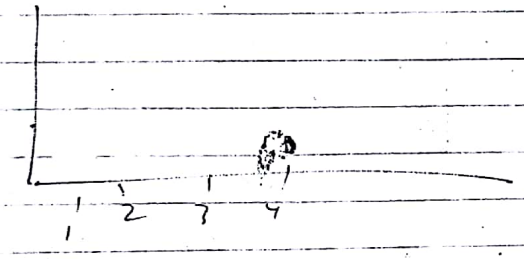

$f(n) = \Theta(g(n))$




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




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




for $n \geq 3$



Transitivity

$f(n) = \Theta(g(n)) + g(n) = \Theta(h(n))$

$f(n) = \Theta(h(n))$

$O$
$\Omega$
$o$
$\omega$

Reflexivity

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

$O$
$\Omega$

Symmetry

$f(n) = \Theta g(n)$ iff $g g(n) = \Theta(f(n))$

Transpose symmetry

$f(n) = O g(n)$ iff $g(n) = \omega f(n)$

$o$

## Recurrences

A recurrence is an equation or inequality that describes a function in terms of its value on smaller inputs.

Methods of solving recurrences (i.e. for obtaining asymptotic bounds on solution):-

(1) Substitution method :- We guess a bound and then use mathematical induction to prove our guess correct.

(2) Recursion tree method :- converts the recurrence into tree whose nodes represents costs incurred at various level of recursion.

(3) Master method :- provides bounds for recurrences of the form:-
$T(n) = aT(n/b) + f(n)$
where $a \geq 1$, $b > 1$ and $f(n)$ is a given function.



Note: when we solve recurrence, we omit floor, ceilings + boundary conditions

Go to master method

## (2) Substitution method

Substitution method for solving recurrences consists of 2 steps:-

(1) Guess the form of solution.-
(2) Use mathematical induction to find constants and show that solution works.

eg. Determine an upper bound on the foll$^n$ recurrence:-
$T(n) = 2T(\lfloor n/2 \rfloor) + n$

We guess sol$^n$ to be
$\Rightarrow T(n) = O(n \lg n)$
$\Rightarrow$ ie $T(n) \leq cn \lg n$ for some $c > 0$ and $n \geq n_0$

We will prove it by induction:-

Basic step :-

$T \leq$ [illegible] $T_1 \leq 0$

If $n \geq 1$

$T(1) \leq c \cdot 1 \cdot \lg 1$

$\leq 0$

If we take $n = 1$, then $\lg 1 = 0$

i.e. $T(1) \leq 0$

So, taking $n = 2$

$T(n) = 2T(n/2) + n$

$T(2) = 2T(1) + 2$ Let $T(1) = c_1$

$\geq 2c_1 + 2$

We have to prove:-

$T(2) \leq c\, 2 \lg 2 = 2c$

$\lg 2 = 1$

$2c_1 + 2 \leq 2c$

$\Rightarrow c \geq c_1 + 1$

i.e. for $n = 2$, $T(2) \leq c \cdot 2 \lg 2$

for $c \geq c_1 + 1$

Inductive step

Let us assume it true for $n/2$

i.e. $T(n/2) \leq c \frac{n}{2} \lg n/2$

Let us check for $n$ i.e.

we have to prove:-

$T(n) \leq c\, n \lg n$

$T(n) = 2T(n/2) + n$

$\Rightarrow T(n) \leq 2c \frac{n}{2} \lg (n/2) + n$

$\Rightarrow T(n) \leq cn \lg (n/2) + n$

$= cn \lg n - cn \lg 2 + n$

$= cn \lg n - cn + n$

$\leq cn \lg n$ if $-cn + n \leq 0$

i.e. $T(n) \leq cn \lg n$

if $cn - n \geq 0$

$cn \geq n$

$c \geq 1$

Using M.I, it is proved.

---

Q: Solve

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

Let us guess the sol to be:- $O(\lg n)$

i.e. we need to prove

$T(n) \leq c \lg n$ for some $c > 0$

& $n \geq n_0$

We will prove it by induction

Basic step

If we take $n \geq 1$, we need to prove

$T(1) \leq c \times \lg 1$ i.e. $T(1) \leq 0$

not true

so take $n = 2$, Let $T(1) = c$

$T(2) = T(1) + 1 = c_1 + 1$

We need to prove
$T(2) \le c \lg 2$
$c_1 + 1 \le c$

ie for $n=2$ $T(2) \le c \lg 2$
for $c \ge 1+1$

Inductive step.
Let us assume it true for $n/2$
ie $T(n/2) \le c \lg n/2$

Let us check for $n$
ie we have to prove
$T(n) \le c \lg n$

$T(n) = T(n/2) + 1$
$\le c \lg n/2 + 1$
$= c(\lg n - \lg 2) + 1$
$= c \lg n - c + 1$
$\ge c \lg n - (c-1)$

ie $T(n) \le c \lg n$
if $c \ge 1$ ($c-1 \ge 0$, $c \ge 1$)

So, it is true for $n$
∴ using induction, it is proved.

$n^{\log_b a}$
$n^{\log_2 2} = n^0 = 1$

(4) Solving Recurrence relation using Master method

Let $T(n) = aT(n/b) + f(n)$
where $a \ge 1$, $b > 1$.
Let $f(n)$ be function

(1) If $f(n)$ is polynomially smaller than $n^{\log_b a}$.
i.e. $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$.
Then $T(n) = \Theta(n^{\log_b a})$

(2) If $f(n) = \Theta(n^{\log_b a})$
Then $T(n) = \Theta(n^{\log_b a} \cdot \lg n)$

(3) If $f(n)$ is polynomially larger than $n^{\log_b a}$ i.e.
$f(n) = \Omega(n^{\log_b a + \epsilon})$
for some constant $\epsilon > 0$
and if $af(n/b) \le cf(n)$
for some constant $c < 1$
and all sufficiently large $n$, then $T(n) = \Theta(f(n))$

Master method does not cover all possible cases:-
It is not applicable if :-

(i) $f(n) < n^{\log_b a}$ but not polynomially smaller.

(ii) $f(n) > n^{\log_b a}$ but not polynomially larger

(iii) Condition $a f(n/b) \leq c f(n)$ for some constant $c<1$ fails to hold.

eg. $T(n) = 2T(n/2) + \Theta(n)$

$a=2, b=2, f(n) = \Theta(n)$

$n^{\log_b a} = n^{\log_2 2} = n$

~~$f(n) = \Theta(n) = \Theta(n)$~~

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

$\therefore$ 2nd case is applicable

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

eg. $T(n) = 9T(n/3) + n$

$a=9, b=3$

$f(n) = n$

$n^{\log_b a} = n^{\log_3 9}$

$= n^{\log_3 3^2}$

$= n^{2\log_3 3}$

$= n^2$ ($\because \log_3 3 = 1$)

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

$= O(n)$ for $\epsilon = 1$

As $n < n^2$, so $f(n)$ is smaller than $n^{\log_b a}$

so, 1st case is applicable

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

$a f(n/b) \leq c f(n)$

$c < 1$

eg. $T(n) = 4T(n/2) + n^3$

$a = 4, \quad b = 2, \quad f(n) = n^3$

$n^{\log_b a} = n^{\log_2 4}$

$= n^{2\log_2 2}$

$= n^2 \quad (\text{as } \log_2 2 = 1)$

$f(n) = n^3 > n^2$ i.e. $f(n) > n^{\log_b a}$,

$\therefore$ 3rd case is applicable.

$\Rightarrow a f(n/b) \le c f(n)$ for some $c < 1$ and $n \ge n_0$

$\Rightarrow 4 f(n/2) = 4\left(\frac{n^3}{8}\right)$

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

$\Rightarrow \frac{n^3}{2} \le cn^3$

? for $c = \frac{3}{4}$

$\Rightarrow c \ge \frac{1}{2}$

then $a f(n/b) \le c f(n)$ for $n \ge 0$

$\therefore T(n) = \Theta(n^3)$

eg. $T(n) = 3T(n/4) + n \lg n$ (1)

$a = 3$

$b = 4$

$n^{\log_b a} = n^{\log_4 3}$

$\log_4 3 = \frac{\log_{10} 3}{\log_{10} 4} = \frac{0.477}{0.602}$

$= 0.792 < 1$

eg. $n = 2$

$n \lg n = 2$

$n^{\log_4 3} = 2^{0.792} = 1.731$

$\epsilon \simeq 0.2$

eg. $n = 4$

$n \lg n = 2 \times 2 = 4$

$n^{\log_4 3} = 4^{0.792} = 2.99$

$4 \lg 4$

$4 \lg 2^2$

$4 \cdot 2 \cdot 1$

$f(n) = n \lg n$ is larger than $n^{\log_4 3}$

So, 3rd case is applicable

$af(n/b) \leq cf(n)$ for $c<1$ and $n \geq n_0$

$\Rightarrow 3f(n/4)$

$\Rightarrow 3(n/4 \log n/4)$

$\Rightarrow \frac{3n}{4}\lg n - \frac{3n}{4}\lg 4$

$\leq \frac{3n\lg n}{4}$

$\leq cf(n)$

if $c = \frac{3}{4} < 1$

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

$= \Theta(n \lg n)$



eg. $T(n) = 4T(n/2) + n^2/\lg n$

$a=4,\ b=2$

$f(n) = n^2/\lg n$

$n^{\log_b a} = n^{\log_2 2^2} = n^2$

$f(n) = \frac{n^2}{\lg n}$ is smaller than $n^2$, but not polynomially smaller so we cannot apply master method.

$n^{\log_2 2^{2^k}}$

$= n^{2^k}$

→ Iteration method

Time — no. of elements

eq. $T(n) = 2T(n/2) + n$ → there are 2 parts and to solve we merge the final answer → Time taken to partition 'n'

$\Rightarrow 2[2T(n/4) + n/2] + n$

$\Rightarrow 4T(n/4) + n + n$

$\Rightarrow 2^2 T(n/2^2) + 2n$

$\Rightarrow 2^2[2T(n/2^3) + n/4] + 2n$

$\Rightarrow 2^3 T(n/2^3) + 3n$

$\Rightarrow 2^m T(n/2^m) + mn$

$\Rightarrow 2^m T(1) + mn$

$\frac{n}{2^m} = 1$

$n = 2^m$

$m = \log_2 n$

and $T(1) = 1$

there element one time $T(1)$ it is

$\Rightarrow 2^m + mn$

$\Rightarrow n + \lg n \cdot n$

$\Rightarrow n \lg n + n$

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

$T(n) \le n \lg n + n \lg n$

$T(n) \le 2n \lg n$

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

---

eq. $T(n) = 4T(n/2) + n^2$

$= 4[4T(n/4) + n^2/4] + n^2$

$= 4^2 T(n/2^2) + 2n^2$

$= 4^2[4T(n/2^3) + n^2/4] + 2n^2$

$= 4^3 T(n/2^3) + 3n^2$

$= (2^m)^2 T(n/2^m) + mn^2$

$= (2^m)^2 T(1) + mn^2$

$\frac{n}{2^m} = 1$

$n = 2^m$

$m = \log_2 n$

and $T(1) = 1$

n

$n^2$

$(n/4)^2$

$n^2/16$

$\Rightarrow n^2 \cdot 1 + n^2 \lg n$

$T(n) \geq n^2 + n^2 \lg n$

$T(n) \left( \begin{array}{l} \leq n^2 \lg n + n^2 \lg n \\ \leq 2n^2 \lg n \end{array} \right)$

$T(n) \geq O(n^2 \lg n)$

2nd order recurrence relation

Q: $a_n = \begin{cases} 5a_{n-1} - 6a_{n-2} & n > 1 \\ 16 & n = 1 \\ 7 & n = 0 \end{cases}$

$a_n = 5a_{n-1} - 6a_{n-2}$

Characteristic equation is:-

$x^n - 5x^{n-1} + 6x^{n-2} = 0$

Divide by lowest degree term $x^{n-2}$

$x^2 - 5x + 6 = 0$

$\Rightarrow x^2 - 3x - 2x + 6 = 0$

$\Rightarrow x(x-3) - 2(x-3) = 0$

$\Rightarrow (x-3)(x-2)$

$\Rightarrow x = 3, 2$

25

Solution is of form

$a_n = \alpha \cdot 3^n + \beta \cdot 2^n$

$n = 0$

$\Rightarrow \alpha + \beta = 7 \quad \ldots (1)$

$n = 1$

$\Rightarrow 3\alpha + 2\beta = 16$

$\Rightarrow 2\alpha + 2\beta = 14$

$\Rightarrow 3\alpha + 2\beta = 16$

$(-) \quad (-) \quad (-)$

$-\alpha = -2$

$\beta = 5$

$a_n = 2 \cdot 3^n + 5 \cdot 2^n$

$\left( \begin{array}{l} \leq 2 \cdot 3^n + 5 \cdot 3^n \\ \leq 3^n (2+5) \\ \leq 7 \cdot 3^n \end{array} \right)$

$a_n = O(3^n)$

## Change of Variables

$T(n) = 2T(\sqrt{n}) + \lg n$

Let $n = 2^m$

$\Rightarrow m = \lg n$

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

$\Rightarrow$ Let $T(2^m) = S_m$

$\Rightarrow S_m = 2S_{m/2} + m$

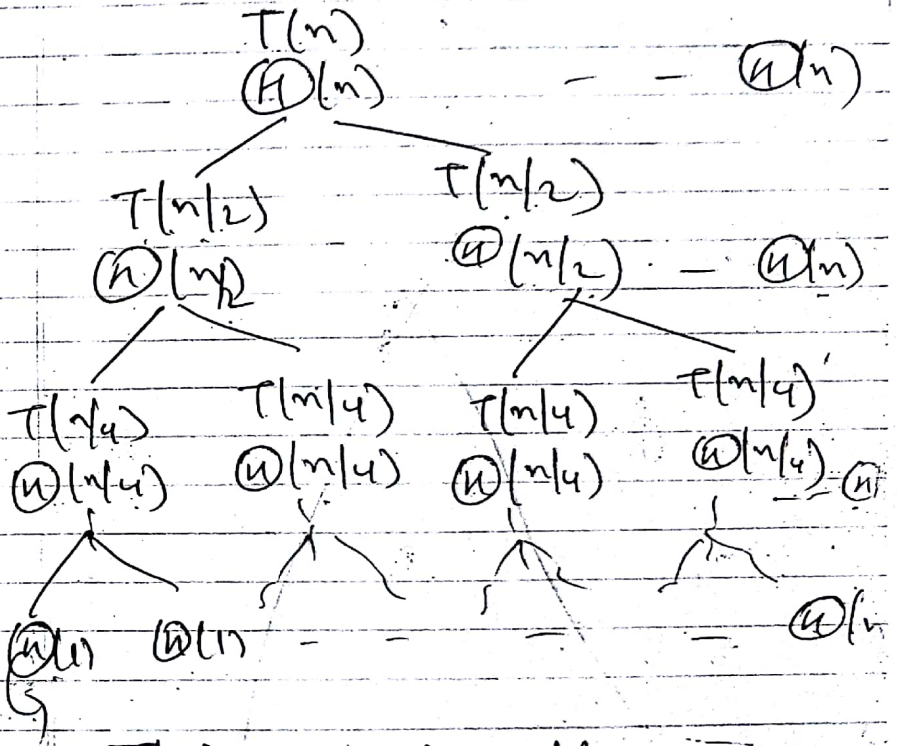
$\Rightarrow S_m = \Theta(m \lg m)$ (merge sort or master method $\Rightarrow \Theta(n \lg n)$)

$\Rightarrow T(n) = \Theta(\lg n \cdot \lg \lg n)$

$n = 2^m$

## Recursion tree

$T(n) = 2T(n/2) + \Theta(n)$




For, $T(n)$ = Total no of levels × running time of each level

level $i$ should become $T(1)$

$\frac{n}{2^i} = 1$

$n = 2^i$

$i = \lg n$

$= (\lg n + 1)\,\Theta(n)$

$= \Theta(n \lg n) + \Theta(n)$

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

$i = 0, 1, \ldots, \lg n \Rightarrow$ Total $= \lg n + 1$ levels

No. of levels for a tree with root $n$

$\Rightarrow \lg n + 1$

eg. 8

4 4

2 2 2 2

1 1 1 1 1 1 1 1

$= \log_2 8 + 1$

$= \log_2 2^3 + 1$

$= 3 \lg 2 + 1$

$= 3 + 1 = 4$

at last level $= T(1)$

last level $= 1 + \lg n$

$\frac{n}{2^i} = 1$

$n = 2^i$

$i = \lg n$

$n^{\log_b a} = n^{\log_2 2} = n$ ← leaves

Q: $T(n) = 2T(n/2) + n^2$ $\Theta(n)$

$T(1) = \Theta(1)$

$T(n) = n^2 + T(n/2) + T(n/2)$

$T(n)$ — $n^2$

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

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

$T(n) = n^2 + \frac{n^2}{2} + \frac{n^2}{4} + \cdots + \frac{n^2}{2^{\lg n - 1}} + \Theta(n)$

$T(n) = n^2\left[1 + \frac{1}{2} + \frac{1}{4} + \cdots + \frac{1}{2^{\lg n - 1}}\right] + \Theta(n)$

last $i$ = $\lg n + 1$, but start with zero

$\lg n$ terms

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

$$T(n) = n^2 \cdot 1 \left(1 - \left(\frac{1}{2}\right)^{\log n}\right) + \Theta(n)$$

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

$$T(n) = 2n^2\left(1 - \frac{1}{n}\right) + \Theta(n)$$

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

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

$\frac{a(1-r^n)}{1-r}$

$$\frac{n^2 \cdot 1\left(1 - \left(\frac{1}{2}\right)^{\log n}\right)}{1 - \frac{1}{2}}$$

$1 - a^r$

/(



eg. $T(n) = 3T(n/4) + n$

$\Rightarrow T(n) = 3\left[3T\left(n/16\right) + \frac{n}{4}\right] + n$

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

$\Rightarrow T(n) = 3^2\left[3T\left(\frac{n}{64}\right) + \frac{n}{16}\right] + n + \frac{3n}{4}$

$\Rightarrow T(n) = 3^3 T\left(\frac{n}{4^3}\right) + n + \frac{3n}{4} + \frac{9n}{16} + \cdots$

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

$\Rightarrow T(n) = 3^m T\left(\frac{n}{4^m}\right) + n\left[\frac{1 \cdot \left(1 - \frac{3}{4}\right)^m}{1 - \frac{3}{4}}\right]$

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

$\Rightarrow \left(\frac{n}{4^m} = 1\right.$

$n = 4^m$

$m = \log_4 n$

$\left.\text{and } T(1) = 1\right)$

$\Rightarrow T(n) = 3^{\log_4 n} \cdot 1 + 4n\left[\frac{1 - 3^{\log_4 n}}{1 - 4^{\log_4 n}}\right]$

As $n^{\log_b a} = a^{\log_b n}$

$\Rightarrow T(n) = n^{\log_4 3} + 4n\left[1 - \frac{n^{\log_4 3}}{n^{\log_4 4}}\right]$

$\Rightarrow T(n) = n^{\log_4 3} + 4n\left[\frac{n - n^{\log_4 3}}{n}\right]$

$\Rightarrow T(n) = n^{\log_4 3} + 4n - 4n^{\log_4 3}$

$\Rightarrow T(n) = 4n - 3n^{\log_4 3} \quad = \frac{\log 3}{\log 4}$ ✓

$\Rightarrow T(n) \leq 4n$

$\Rightarrow T(n) = O(n)$ ✓

$= a - b$

$\leq a$

Q: $t_n = \begin{cases} n & \text{if } n = 0, 1, 2 \\ 5t_{n-1} - 8t_{n-2} + 4t_{n-3} & \text{otherwise} \end{cases}$

$t_n - 5t_{n-1} + 8t_{n-2} - 4t_{n-3} = 0$

$x^n - 5x^{n-1} + 8x^{n-2} - 4x^{n-3} = 0$

$x^3 - 5x^2 + 8x - 4 = 0$

$\Rightarrow (x-1)(x^2 - 4x + 4)$

$\Rightarrow x = 1, 2, 2$

$$
\begin{array}{r|l}
x-1 & x^3 - 5x^2 + 8x - 4 \\
 & x^3 - x^2 \quad (-)\ (+) \\
\hline
 & -4x^2 + 8x - 4 \\
 & -4x^2 + 4x \quad (+)\ (-) \\
\hline
 & 4x - 4 \\
 & 4x - 4 \quad (-)\ (+) \\
\hline
 & \times
\end{array}
$$

Solution is of the form:-
(if 2 roots are same)
$t_n = \alpha \cdot 1^n + \beta \cdot 2^n + r \cdot n \cdot 2^n$

**Note:** If 3 roots are same:
eg. if 2, 2, 2
$t_n = \alpha \cdot 2^n + \beta \cdot n \cdot 2^n + r \cdot n^2 2^n$

$\Rightarrow t_0 = \alpha + \beta = 0$
$\Rightarrow t_1 = \alpha + 2\beta + 2r = 1$
$\Rightarrow t_2 = \alpha + 4\beta + 8r = 2$

$(-\beta) + 2\beta + 2r = 1$
$\beta + 2r = 1 \quad ] \times 3$
$3\beta + 8r = 2$

$\Rightarrow 3\beta + 6r = 3$
$\Rightarrow 3\beta + 8r = 2$
$(-) \quad (-) \quad (-)$
$-2r = 1$

$r = \frac{-1}{2}$

~~Loop~~
$\beta + 2r = 1$
$\beta + 2\left(\frac{-1}{2}\right) = 1$
$\beta - 1 = 1$
$\beta = 2$

$\alpha = -\beta$
$\alpha = -2$

$\alpha = -2, \beta = 2, r = -\frac{1}{2}$ 35

Solution is:
$t_n = -2 \cdot 1^n + 2 \cdot 2^n - \frac{1}{2} \cdot n \cdot 2^n$

$t_n = (4)(n \cdot 2^n)$

---

Q: $t_n - 2t_{n-1} = 3^n$ ①

Multiply equation ① by ③

$\Rightarrow 3t_n - 6t_{n-1} = 3^{n+1}$

Put $n = n-1$

$\Rightarrow 3t_{n-1} - 6t_{n-2} = 3^n$ ②

From ① & ②

$\Rightarrow t_n - 2t_{n-1} = 3t_{n-1} - 6t_{n-2}$

$\Rightarrow t_n - 5t_{n-1} + 6t_{n-2} = 0$

$\Rightarrow x^n - 5x^{n-1} + 6x^{n-2} = 0$

$\Rightarrow x^2 - 5x + 6 = 0$

$\Rightarrow x^2 - 3x - 2x + 6 = 0$
$\Rightarrow x(x-3) - 2(x-3) = 0$
$\Rightarrow (x-3)(x-2)$
$\Rightarrow x = 2, 3$

$t_n = \alpha \cdot 2^n + \beta \cdot 3^n$

From (1)

$\Rightarrow t_1 - 2t_0 = 3$
$\Rightarrow t_1 = 3 + 2t_0$

$\Rightarrow t_0 = \alpha + \beta$
$\Rightarrow t_1 = 2\alpha + 3\beta = 3 + 2t_0$

$\Rightarrow t_0 - \alpha = \beta$

$\Rightarrow 2\alpha + 3(t_0 - \alpha) = 3 + 2t_0$
$\Rightarrow 2\alpha + 3t_0 - 3\alpha = 3 + 2t_0$
$\Rightarrow 3t_0 - \alpha = 3 + 2t_0$
$\Rightarrow -\alpha = 3 + 2t_0 - 3t_0$
$\Rightarrow -\alpha = 3 - t_0$
$\Rightarrow \alpha = t_0 - 3$

$\Rightarrow \alpha + \beta = t_0$
$\Rightarrow t_0 - 3 + \beta = t_0$
$\Rightarrow \beta = 3$

$\alpha = t_0 - 3$
$\beta = 3$

$\Rightarrow t_n = (t_0 - 3) \cdot 2^n + 3^{n+1}$

$\Rightarrow t_n = \Theta(3^n)$

## Divide & Conquer Technique

In this technique:

1. Divide the problem into a no. of subproblems.
2. Conquer the subproblems by solving them recursively till we get a straight forward solution to subproblem.
3. Combine the solutions to the subproblems into the solution for the original problem.

eg. Merge sort

1. In this algorithm, divide the n-element sequence into two subsequences of $n/2$ elements each.
2. Sort the 2 subsequences recursively using merge sort.
3. Combine (merge) the two sorted subsequences to produce the sorted list.



## Algo for merge sort

```
Merge sort (A, p, r)
1. if p < r                                  } Constant time
2. then q ← L(p+r)/2]                        } Θ(1) executed only once
3. Merge sort (A, p, q)                      } T(n/2) each
4. Merge sort (A, q+1, r)                    } if merge sort takes T(n) time
5. Merge (A, p, q, r)  → Θ(n)
```

Index of 1st element → p
" " last " → r
" " middle " → q

Input: A[p -- r]

A[p -- q] ---- A[q+1 -- r]

A[p --- r]

i = p
j = q+1
k = p

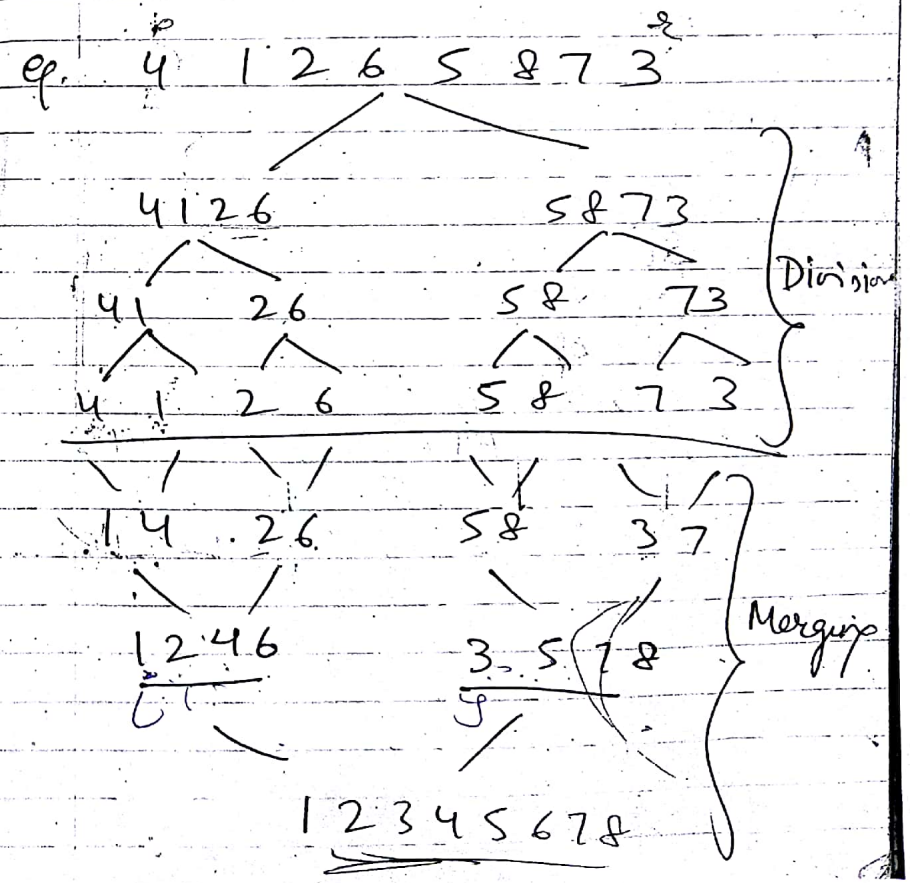
## Algo for merge procedure

```
Merge(A, p, q, r)
▷ C[p--r] stores sorted list.
  i ← p
  j ← q+1
  k ← p
▷ copy smaller of A[i] and A[j]
while i ≤ q and j ≤ r
do if A[i] < A[j]
then C[k] ← A[i]
     i ← i+1, k ← k+1
else C[k] ← A[j]
     j ← j+1, k ← k+1

▷ copy remainder of 1st sequence
while i ≤ q
do C[k] ← A[i]
   i ← i+1
   k ← k+1

▷ copy remainder of second sequence
while j ≤ r
   do C[k] ← A[j]
      j ← j+1
      k ← k+1
```



```
▷ Copy C[p--r] to A[p--r]
   l ← p
   while l ≤ r
   do A[l] ← C[l]
      l ← l+1
```

Merge procedure takes $T(n) = \Theta(n)$ where n is the no. of elements in array A ie $n = r - p + 1$



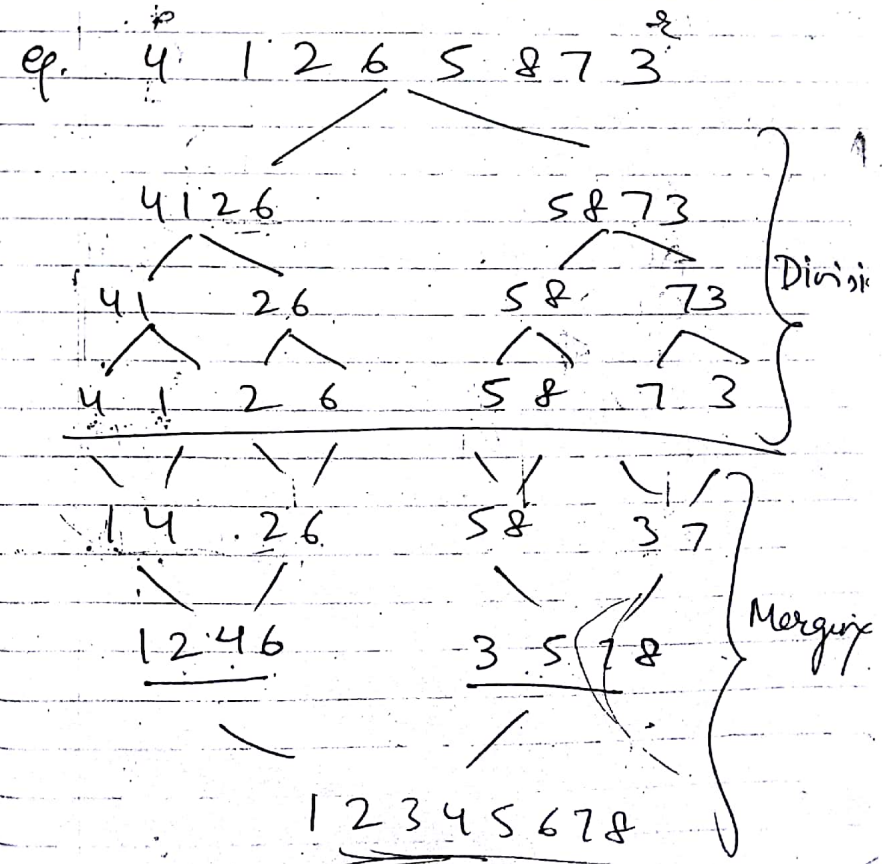
## Algo for merge procedure

```
Merge (A, p, q, r)
▷ C[p..r] stores sorted list.
  i ← p
  j ← q+1
  k ← p
▷ copy smaller of A[i] and A[j]
while i ≤ q and j ≤ r
do if A[i] < A[j]
then C[k] ← A[i]
     i ← i+1, k ← k+1
else C[k] ← A[j]
     j ← j+1, k ← k+1

▷ copy remainder of 1st sequence
while i ≤ q
do C[k] ← A[i]
   i ← i+1
   k ← k+1

▷ copy remainder of second sequence
while j ≤ r
do C[k] ← A[j]
   j ← j+1
   k ← k+1
```



```
▷ Copy C[p..r] to A[p..r]
  l ← p
  while l ≤ r
  do A[l] ← C[l]
     l ← l+1
```

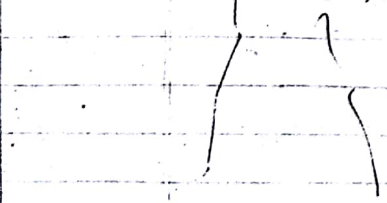
Merge procedure takes $T(n) = \Theta(n)$ where n is the no. of elements in array A ie $n = r - p + 1$



## Merge procedure

1 2 4 6 3 5 7 8

p=1
q=4
r=8
i←1
j←5
k←1
A[1] < A[5] — C
C[1] ← A[1] ie 1 — 1
i←2, k←2
A[2] < A[5]
C[2] ← A[2] ie 2 — 1 2
i←3, k←3
A[3] < A[5]
C[3] ← A[5] ie 3 — 1 2 3
j←6, k←4
A[3] < A[6]
C[4] ← A[3] ie 4 — 1 2 3 4
i←4 k←5
A[4] < A[6]
C[5] ← A[6] ie 5 — 1 2 3 4 5
j←7, k←6
C[6] ← A[4] ie 6 — 1 2 3 4 5 6
i←5, k←7

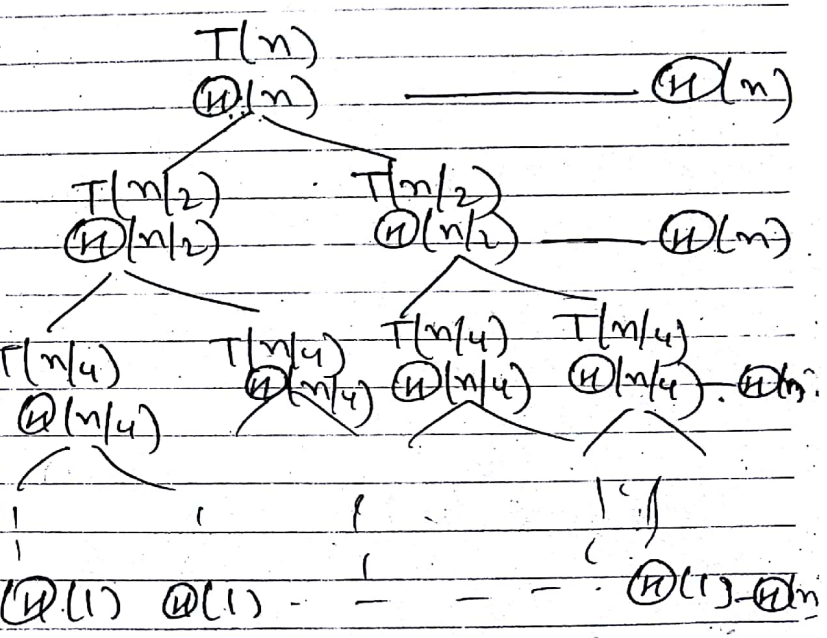




Running time for merge sort

$T(n) = \Theta(1) + T(n/2) + T(n/2) + \Theta(n)$
$T(1) = \Theta(1)$

$\rightarrow T(n) = 2T(n/2) + \Theta(n)$
$T(1) = \Theta(1)$




No. of levels = $\lg n + 1$

$T(n) = (\lg n + 1)\,\Theta(n)$
$T(n) = \Theta(n \lg n) + \Theta(n)$
$T(n) = \Theta(n \lg n)$

## Quick sort

* We use divide and conquer technique
* Array is partitioned into 2 subarray such that elements in left subarray are less than or equal to elements in right subarray.
* Two subarrays are recursively sorted using quicksort.
* The final result is a completely sorted array so no subsequent merge procedure is necessary.

### Algorithm for Quicksort

```
Quicksort (A, p, r)
  if p < r
then q ← partition (A, p, r)
  Quicksort (A, p, q)
  Quicksort (A, q+1, r)
```

Intial call to quicksort is
Quicksort (A, 1, n)

A[p -- r]

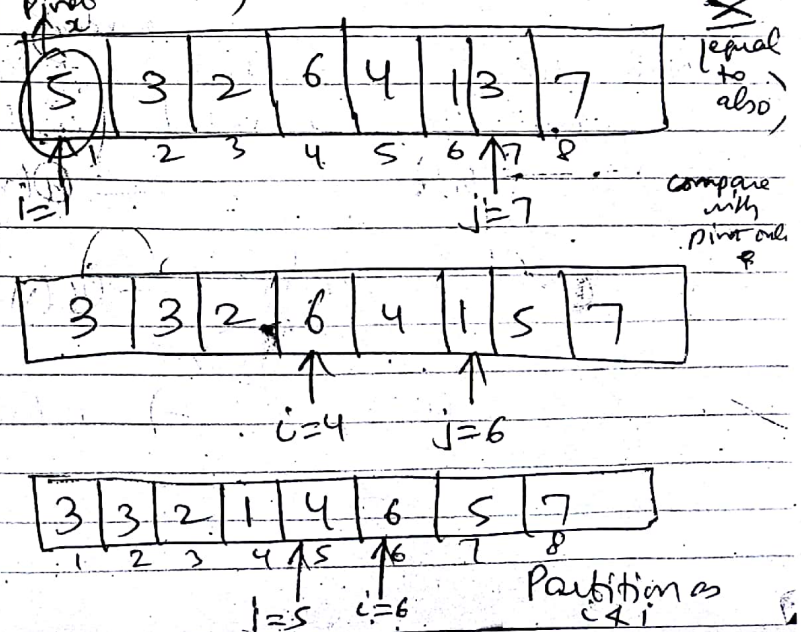
A[p -- q]  A[q+1, -- r]

partition procedure returns it

### Algorithm for partition procedure

```
Partition (A, p, r)
  x ← A[p]
  i ← p - 1
  j ← r + 1
while true
do repeat j ← j - 1
     until A[j] ≤ x
do repeat i ← i + 1
     until A[i] ≥ x
     if i < j
then exchange A[i] ↔ A[j]
     else return j
```

Running time for partition procedure is Θ(n).



A: j=5

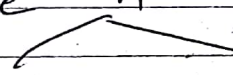
3 3 2 1 4 6 5 7

Running time for quick sort

Worst-case

Array will be partitioned into 2 subarrays, one with (n-1) elements and other with a single element.

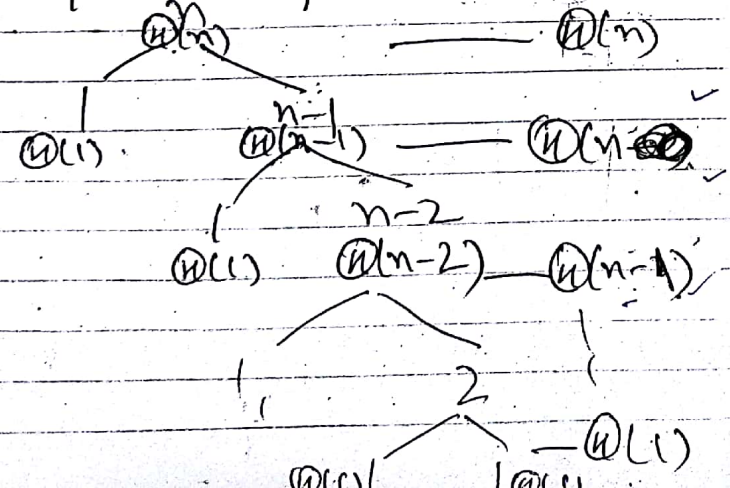
i.e.




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

($\Theta(n)$: Partition procedure)

Note: This occurs when array is already sorted.




① 4 7 9 11

11 9 7 4 1

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

$T(n) = \Theta(n) + \Theta(n-1) + \Theta(n-2) + \cdots + \Theta(1)$

$T(n) = \Theta\left(\sum_{i=1}^{n} i\right)$

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

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

Eg.

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

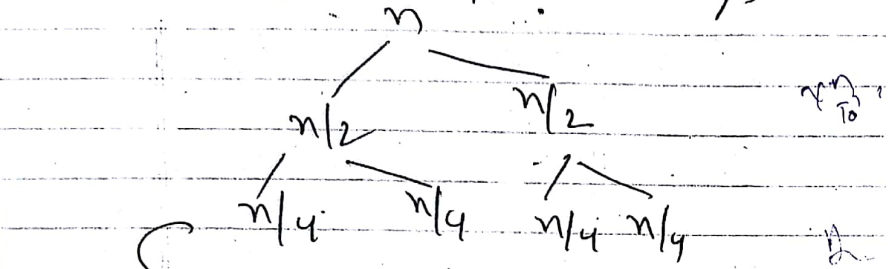
$i \geq 1$

$i = 1$

| 1 | 2 3 4 5 6 7 8 |

**Best case :-**

If partition procedure partitions array in 2 equal subarrays




same as merge sort

$$T(n) = 2T(n/2) + \Theta(n)$$

$2T(n/2)$ → for equal partitions

$\Theta(n)$ → for partition procedure

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

---

**Average case :** It is closer to best case. ie

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

eg. if partition procedure produces 9 to 1 proportional split.

$T(n) \leq (\log_{10/9} n)\, n$

$\Rightarrow T(n) \leq n \log_{10/9} n$

$\Rightarrow T(n) \geq n \lg n$ — (1)

as $\log_{10/9} > k_2$

Shortest path is:-

$n \to \frac{n}{10} \to \frac{n}{100} \dots \to \left(\frac{1}{10}\right)^i n$

$\to 1$

It terminates when

$\left(\frac{1}{10}\right)^i n = 1$

$\Rightarrow n = 10^i$

$\Rightarrow i = \log_{10} n = \frac{\log_2 n}{\log_2 10} = \frac{\lg n}{k_2}$

$\Rightarrow T(n) \geq n \log_{10} n$

$\Rightarrow T(n) \leq n \lg n$ — (2)

as $\log_{10} <$ $\lg_2$

From (1) & (2)

$\Rightarrow T(n) = \Theta(n \lg n)$

$\Rightarrow T(n) \geq n \left(\frac{\lg n}{k_2} + 1\right)$

$\Rightarrow T(n) \geq \frac{1}{k_2} n \lg n + n$

$\Rightarrow T(n) = \Omega(n \lg n)$ — (2)

## Algo for Randomized Quicksort

```
Randomized QuickSort (A, p, r)
  if p < r {
  q ← Randomized Partition (A, p, r)
  Randomized QuickSort (A, p, q)
  Randomized QuickSort (A, q+1, r) }
```

## Algo for Randomized Partition

```
Randomized Partition (A, p, r)
  i ← Random (p, r)
  exchange A[p] ↔ A[i]
  return Partition (A, p, r)
```

$\Rightarrow T(n) \leq n \left(\frac{\lg n}{k_1} + 1\right)$

$\leq \frac{1}{k_1} n \lg n + n$

$T(n) \geq O(n \lg n)$ — (1)

From (1) + (2)

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

# Matrix Multiplication

We are given 2 $n \times n$ matrices A and B, we have to multiply A + B and store result in matrix C.

```
Matrix_Multiply (A, B, n)
  for i ← 1 to n
  do for j ← 1 to n
  do Cij ← 0
  for k ← 1 to n
  do Cij ← Cij + aik · bkj
  return C
```

Running time :-

$T(n) = \Theta(n^3)$

(As there are 3 nested for loops running till 'n'

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} i & j \\ k & \ell \end{bmatrix}$$

A B C

$i = ae + bg$

$j = af + bh$

$k = ce + dg$

$\ell = cf + dh$

No. of multiplications $= n^3$ $(2^3 = 8)$

No. of additions $= (n^2)$ $(2^2 = 4)$

Using divide and conquer technique

If we use divide and conquer technique, we divide each of A, B and C matrices into four $n/2 \times n/2$ matrices and rewriting the equation $C = AB$ as follows:-

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

where:

$r = ae + bf$

$s = ag + bh$

$t = ce + df$

$u = cg + dh$

$T(n/2)$

$n = 2 = w$

Each of four equations specifies 2 multiplications of $n/2 \times n/2$ matrices and addition of their $n/2 \times n/2$ products.

In this case, we derive the following recurrence:-

$T(n) = 8\,T(n/2) + \Theta(n^2)$

$n^{\log_b a}$ → $8$ multiplications of $n/2 \times n/2$ matrices

$\Rightarrow n^{\log_2 8}$

$\Rightarrow n^3$ (1st case)

$\Theta(n^2)$ for four additions

of $n/2 \times n/2$ matrices and $\Theta(n^2)$ scalar

$T(n) = \Theta(n^3)$

## Strassen's Matrix Multiplication

This algo requires only 7 recursive multiplications (to multiply two $2 \times 2$ matrices. We divide each of A, B and C matrices into four $n/2 \times n/2$ matrices as:

We rewrite $C = AB$ as:-

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

It yields the recurrence:

$T(n) = 7T(n/2) + \Theta(n^2)$

$n^{\log_2 7} = n^{\log_2 7}$

$= n^{2.81}$ (1st case)

$T(n) = \Theta(n^{\log 7})$

$T(n) = \Theta(n^{2.81})$

additions and subtractions. $n = 2^k$

Strassen's method has 4 steps:

1. Divide the input matrices A and B into $n/2 \times n/2$ submatrices.
2. Compute 14 $n/2 \times n/2$ matrices $A_1, B_1, A_2, B_2 \dots A_7, B_7$.
3. Recursively compute the seven matrix products $P_i = A_i B_i$ for $i = 1, 2 \dots 7$
4. Compute the desired submatrices $r, s, t, u$ of result matrix C by adding and/or subtracting various combinations of the $P_i$ matrices.

Determining the submatrix products

It is not clear exactly how Strasen discovered the submatrix products that are key to making this algo work.

Let us guess that each matrix product $P_i$ can be written in the form:-

$P_i = A_i B_i$

$= (\alpha_{i1} a + \alpha_{i2} b + \alpha_{i3} c + \alpha_{i4} d)$

$\times (\beta_{i1} e + \beta_{i2} f + \beta_{i3} g + \beta_{i4} h)$

where coefficients $\alpha_{ij}$, $\beta_{ij}$ are all drawn from set $\{-1, 0, 1\}$.

We shall use 4×4 matrices to represent linear combination of (P_i) products of submatrices, where each product combines one submatrix of A (A_i) with one submatrix of B.

eg. $s = ae + bf$

$$= \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} e & f & g & h \end{array} \\ \begin{pmatrix} + & . & . & . \\ . & + & . & . \\ . & . & . & . \\ . & . & . & . \end{pmatrix} \end{array}$$

"+" $\Rightarrow +1$

"." $\Rightarrow 0$

"−" $\Rightarrow -1$

$s = ag + bh$

$$= \begin{pmatrix} . & . & + & . \\ . & . & . & + \\ . & . & . & . \\ . & . & . & . \end{pmatrix}$$

$$P_3 = A_3 B_3$$
$$= (c+d) \cdot e$$
$$= ce + de$$

$$= \begin{pmatrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ + & \cdot & \cdot & \cdot \\ + & \cdot & \cdot & \cdot \end{pmatrix}$$

and

$$P_4 = A_4 B_4$$
$$= d \cdot (f - e)$$
$$= df - de$$

$$= \begin{pmatrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ - & + & \cdot & \cdot \end{pmatrix}$$

(i) $\Theta(n^3)$ (ii) $\Theta(n^{2.7})$

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

So, we have:-

$r = P_5 + P_4 - P_2 + P_6$

$s = P_1 + P_2$

$t = P_3 + P_4$

$u = P_5 + P_1 - P_3 - P_7$

where,

$P_1 = a(g-h)$

$P_2 = (a+b).h$

$P_3 = (c+d).e$

$P_4 = d(f-e)$

$P_5 = (a+d).(e+h)$

$P_6 = (b-d).(f+h)$

$P_7 = (a-c).(e+g)$

$$\begin{bmatrix} \cdot & \cdot \\ \cdot & \cdot \end{bmatrix} \begin{bmatrix} \cdot & \cdot \\ \cdot & \cdot \end{bmatrix} = \begin{bmatrix} r & s \\ t & u \end{bmatrix}$$

$P_7 = A_7 B_7$

$= (a-c).(e+g)$

$= ae + ag - ce - cg$

$$= \begin{pmatrix} + & \cdot & + & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ - & \cdot & - & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{pmatrix}$$

$P_5 + P_1 - P_3 = ae + ag - ce + de$

$$= \begin{pmatrix} + & \cdot & + & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ - & \cdot & - & \cdot \\ \cdot & \cdot & \cdot & + \end{pmatrix}$$

$u = P_5 + P_1 - P_3 - P_7$

$= ae + ag - ce + dh - ae - ag + ce + cg$

$= cg + dh$

$$= \begin{pmatrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & + & \cdot \\ \cdot & \cdot & \cdot & + \end{pmatrix}$$

Strassen's algo is often not the method of choice for matrix multiplication for 4 reasons:-

(1) Constant factor hidden in running time of Strassen's algo is larger than the constant factor in naive $\Theta(n^3)$ method.

(2) When matrices are sparse, method tailored for sparse matrices are faster.

(3) Strassen's algo is not quite as numerically stable as the naive method.

(4) Submatrices formed at the levels of recursion consume space.

Q: $\begin{pmatrix} 1 & 3 \\ 5 & 7 \end{pmatrix} \begin{pmatrix} 8 & 4 \\ 6 & 2 \end{pmatrix}$

## Medians and Order Statistics

The $i^{th}$ order statistic of a set of $n$ distinct elements is the $i^{th}$ smallest element.

Eg. minimum of a set of elements is the first order statistic ($i=1$) and the maximum is the $n^{th}$ order statistic ($i=n$).

Median: is the halfway point of the set.

If $n$ is odd, median is unique at $i = \left(\frac{n+1}{2}\right)$

If $n$ is even, there are 2 medians at $i = n/2$ and $i = n/2+1$

## Minimum

Algo to find min$^{n}$ in array

```
Minimum (A, n)
1 min ← A[1]
2 for i ← 2 to n
3     do if min > A[i]
4        then min ← A[i]
5 return min
```

No. of comparisons = $n-1$

Algo to find max$^{m}$

```
Maximum (A, n)
1 max ← A[1]
2 for i ← 2 to n
3     do if A[i] > max
        then max ← A[i]
```

No. of comparisons = $n-1$

01

We compare pairs of element from the input first with each other and then compare the ~~s~~maller to current minimum and the larger to current maximum.

```
MaxMinPair (A, n)
    max ← A[1]
    min ← A[1]
for i ← 1 to n/2
    do if A[2i] > A[2i-1]
    then mint ← A[2i-1]
         maxt ← A[2i]
    else mint ← A[2i]
         maxt ← A[2i-1]
if maxt > max
    then max ← maxt
if min > mint
    then min ← mint
```

Total comparison $= \frac{3n}{2}$

(To divide n elements into $\frac{n}{2}$ pairs and 3 comparisons are compared in each
① Pairs
② Smaller to current min
③ Larger to current max)

Randomized-Select(A, p, r, i)

1. if p = r
2. then return A[p]
3. q ← Randomized-Partition(A, p, r)
4. k ← q - p + 1
5. if i ≤ k
6. then return Randomized-Select(A, p, q, i)
7. else return Randomized-Select(A, q+1, r, i - k)

Worst case running time
-> = Θ(n²)

To find min, we could be unlucky and partition around largest remaining element

Working

A[p -- r]

A[p -- q]  A[q+1 -- r]

such that each element of A[p -- q] is less than each element of A[q+1 -- r]

② k → no. of elements in left subarray A[p -- q]

③ Determine in which 2 subarrays A[p -- q] and A[q+1 -- r], the iᵗʰ smallest element lies.