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

↳ comparison

↳ Recursively divide n to two.

$b = 2, a = 1.$

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

Exactly equal $\varepsilon = 0$

$$\theta\left(n^{\log_b a} (\log_2 n)^1\right) = \theta(\log_2 n)$$

Powering a number

```
power(num,pow)

if pow == 0
    return 1

else
    return num x power(num, pow-1);
```

5^3

$5 \times power(5,2) \longrightarrow 5 \times 5 \times 5$

$= 125.$

$5 \times power(5,1) \longrightarrow 5 \times 5$

$5 \times power(5,0) \longrightarrow 5 \times 1 = 5$

5×1

How many recursions. n recursions.

(Powers)

$a^n = \theta(n)$

```
Ans = 1;
for (i=0; i<n; i++)
    Ans = Ans * num;
```

$\theta(n)$

$$2^{100} = 2^{50} \times 2^{50}$$

$$a^n = \begin{cases} a^{n/2} \cdot a^{n/2} & \text{if } n \text{ is even} \\ a^{(n-1)/2} \cdot a^{(n-1)/2} \cdot a & \text{if } n \text{ is odd.} \end{cases}$$

a^{10} ⟶ $a^5 \cdot a^5$

$a^2\ a^2\ a \quad a^2\ a^2\ a$

$a\ a\ aa \quad a\ a\ a\ a\ a\ a$

* $2T\left(\frac{n}{2}\right) + \theta(1)$

↳ multiplication steps

$T\left(\frac{n}{2}\right) + \theta(1)$ ×

∴ once calculated (recursion) – we don't need it any more'

$= \theta(\log_2 n)$

'Assuming integers'

Problem with floating – Round off involved. (more computation)

Divide & conquer

$\theta(n) = n$

Fibonacci numbers

0, 1, 1, 2, 3, 5, 8, 13, ...

$F_0\ F_1\ F_2\ F_3\ F_4\ F_5$ ....

```
fibo(n)
    if n==0
        return 0
    elif n==1
        return 1
    else
        return (fibo(n-1) + fibo(n-2))
```

$$F_n = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ F_{n-1} + F_{n-2} & \text{if } n \geq 2 \end{cases}$$

Naive method: Straight from base

↳ $\theta(n)$ [Linear time algorithm]

$$F_n = \frac{\phi^n}{\sqrt{5}}, \text{ where } \phi = \frac{1+\sqrt{5}}{2}$$

↓

Golden ratio.

Problem: powering a real number - not so simple as powering an integer.

using

$$a^n = \begin{cases} a^{n/2} \cdot a^{n/2} & \text{even}(n) \\ a^{\left(\frac{n-1}{2}\right)} \, a^{\left(\frac{n-1}{2}\right)} \, a & \text{odd}(n) \end{cases}$$

Not a log time algorithm - Fixing takes time.

Any Formula - not involving - powering.

Theorem on Fibonacci numbers:

$$\begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n, \quad n \geq 1$$

(∵ Fibonacci starts from F_0)

Base case: when $n = 1$

$$\begin{pmatrix} F_2 & F_1 \\ F_1 & F_0 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

(I.H)

NO: F_{n-1}

Induction Hypothesis: Assume $\begin{pmatrix} F_{K+1} & F_K \\ F_K & F_{K-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$

is true for K.

To prove : True for $K+1$.

$$\begin{pmatrix} F_{K+1} & F_K \\ F_K & F_{K-1} \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^K \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

$$\begin{pmatrix} F_{K+1}+F_K & F_{K+1} \\ F_K + F_{K-1} & F_K \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{K+1}$$

$$\boxed{\begin{pmatrix} F_{K+2} & F_{K+1} \\ F_{K+1} & F_K \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{K+1}}$$

$$F_{K+2} = F_K + F_{K+1}$$

$$F_{K+1} = F_K + F_{K-1}$$

True for $n = K+1$.

Find F_n

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n = \begin{pmatrix} a & b \\ b & c \end{pmatrix} = \begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix}$$

Gives $\boxed{F_n = b}$

Our problem is reduced to powering a 2×2 matrix.
not $n \times n$

Now use

$$A^n = \begin{cases} A^{n/2} \; A^{n/2} & \text{even } (n) \\ A^{\left(\frac{n-1}{2}\right)} \; A^{\left(\frac{n-1}{2}\right)} \; A & \text{odd } (n) \end{cases}$$

$$A = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

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

↳ "multiplication"

$$= \Theta(\log_2 n)$$

$$A^n = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n = \begin{pmatrix} a & b \\ b & c \end{pmatrix}$$

$$A = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ a_{21} & & \vdots \\ \vdots & & \vdots \\ a_{m1} & \cdots & a_{mn} \end{pmatrix}, \quad B = \begin{pmatrix} b_{11} & \cdots & b_{1m} \\ \vdots & & \vdots \\ b_{n1} & \cdots & b_{nm} \end{pmatrix}$$

$(m \times n)$ $\quad (n \times m) \quad = C_{m \times m}$

$(m \times n)\ (n \times p) = (m \times p)$

Solution :

$$\sum_{K=1}^{n} a_{ij} * b_{jk}$$

$$c_{11} = (a_{11} * b_{11}) + (a_{12} * b_{21}) + (a_{13} * b_{31}) + \cdots + (a_{1n} * b_{n1})$$

$$c_{12} = (a_{11} * b_{12}) + (a_{12} * b_{22}) + (a_{13} * b_{32}) + \cdots + (a_{1n} * b_{n2})$$

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} =$$

$$c_{11} = (a_{11} * b_{11}) + (a_{12} * b_{21})$$

$$c_{12} = (a_{11} * b_{12}) + (a_{12} * b_{22})$$

$$c_{21} = (a_{21} * b_{11}) + (a_{22} * b_{21})$$

$$c_{22} = (a_{21} * b_{12}) + (a_{22} * b_{22})$$

$$c_{ij} = \sum_{K=1}^{n} a_{iK} * b_{Kj}$$ → Std. Matrix multiplication Algorithm.

$T(n) = \Theta(n^3)$ → 3 loops.

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

$* (a_{00} + 0 + 0)$ $\quad * (a_{00} + (1 * 2) + 0)$

$* (a_{00} + 0 + 1)$ $\quad * (a_{00} + (1 * 3) + 1)$

$* (a_{00} + 0 + 2)$ $\quad * (a_{00} + (1 * 3) + 2)$

$* (a_{00} + (2 \times 3) + 0)$

$* (a_{00} + (2 \times 3) + 1)$

$* (a_{00} + (2 \times 3) + 2)$

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}$$

Solution: *(arr1 + (i * column_count) + k)

* *(arr2 (k * column_count) + j))

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} 2 & 1 \\ 1 & 1 \end{pmatrix}$$

$$\begin{pmatrix} 2 & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} 3 & 2 \\ 2 & 1 \end{pmatrix}$$

$$\begin{pmatrix} 3 & 2 \\ 2 & 1 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} 5 & 3 \\ 3 & 2 \end{pmatrix}$$

Precaution: a[2][2] → assign fully!

Strassen's algorithm.

Divide & Conquer:

1) (Sub problems)
2) Conquer
3) Combine.

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \quad B = \begin{pmatrix} e & f \\ g & h \end{pmatrix}$$

Partition — Solve 2×2 matrix.

$$\begin{pmatrix} A & B \\ C & D \end{pmatrix}$$

$$\begin{bmatrix} ae+bg & af+bh \\ ce+dg & cf+dh \end{bmatrix}$$

$$A \times B = C_{n \times n} = \left(\begin{array}{c|c} r & s \\ \hline t & u \end{array}\right)$$

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

W A B

$r = ae + bg$
$s = af + bh$
$t = ce + dg$
$u = cf + dh$

For 2×2 $\begin{pmatrix} a & b \\ c & d \end{pmatrix}_{2\times 2}$

Now

$r = ae + bg$

$\left(\frac{n}{2}\right)\left(\frac{n}{2}\right)\left(\frac{n}{2}\right)\left(\frac{n}{2}\right)$

$\frac{n}{2} \times \frac{n}{2}$

Now

8 matrix multiplication
4 addition

Merge sort: $T(n) = 2T\left(\frac{n}{2}\right) + \theta(n)$

2 → # of subproblems

$\theta(n)$ → Divide & combine.

$$T(n) = 8T\left(\frac{n}{2}\right) + \theta(n^2)$$

addition

$C_{11} = A_{11} + B_{11}$

$C_{12} = A_{12} + B_{12}$

4 additions (n^2)

$n \times n \rightarrow \frac{n}{2} \times \frac{n}{2}$

∴ solving, $b = 2,\ a = 8,\ f(n) = \theta(n^2)$

$$n^{\log_b a} = n^3 > n^2$$

$\theta(n^3)$ → solution

Not better than standard multiplication?

make it better - Strassen's

Idea:

$$\begin{pmatrix} r & s \\ t & u \end{pmatrix}_{2\times 2} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} e & f \\ g & h \end{pmatrix}$$

$r = ae + bg$
$s = af + bh$
$t = ce + dg$
$u = cf + dh$

Reduce multiplications, do more additions. Some

$n^{\log_b a}$

$n^{\log_2 8} = n^3$

$n^{\log_2 7} = n^{2.8}$

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

$P_1 = a \times (f - h)$
$P_2 = (a+b) \times h$
$P_3 = (c+d) \times e$
$P_4 = d \times (g - e)$
$P_5 = (a+d)(e+h)$
$P_6 = (b-d)(g+h)$
$P_7 = (a-c)(e+f)$

Intermediate terms.

$P_5 + P_4 - P_2 + P_6$	$P_1 + P_2$
$P_3 + P_4$	$P_1 + P_5 - P_3 - P_7$

$r = P_5 + P_4 - P_2 + P_6$

$s = P_1 + P_2$

$t = P_3 + P_4$

$u = P_5 + P_1 - P_3 - P_7$

$P_1 + P_2 = af - ah + ah + bh$
$= af + bh$
$= s$

'18 addition'

'7 multiplication'

$$T(n) = 7T\left(\frac{n}{2}\right) + \theta(n^2)$$

$n^{\log_b a} = n^{\log_2 7} = n^{2.81}$

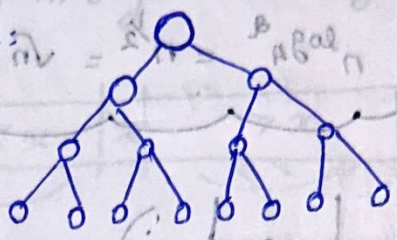
$$T(n) = \theta(n^{2.81})$$

→ Better than n^3.

VLSI layout

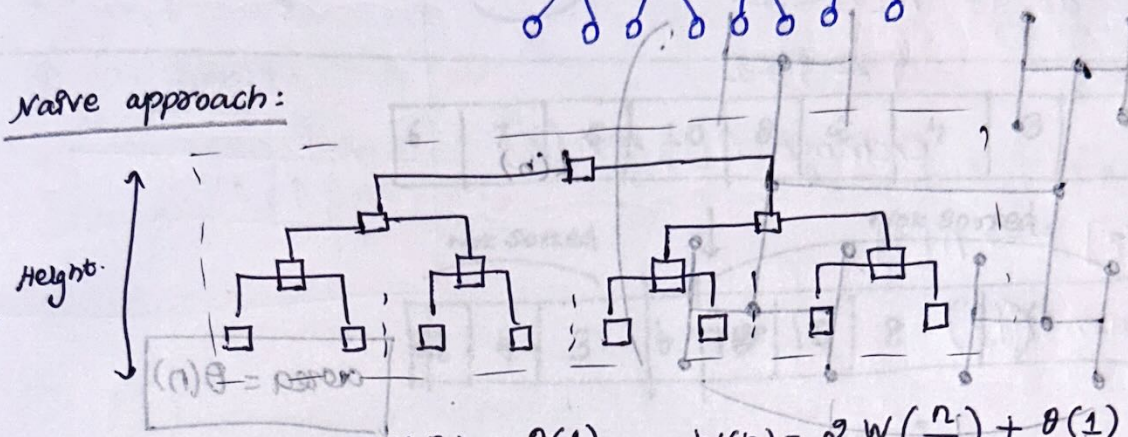
Problem: Embedd a Complete binary tree with n leaves in a grid using minimum area - rectangular area.

eg:



'Binary tree'

Naive approach:



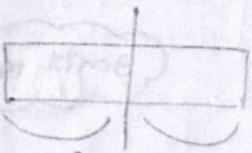
Height = $H(n) = H\left(\frac{n}{2}\right) + \theta(1)$, $W(n) = 2\,W\left(\frac{n}{2}\right) + \theta(1)$

$= \theta(\log_2 n)$, $W(n) = \theta(n)$

Height - Even 2 branches they are $\parallel^{el}$ (Same height))

$\theta(1)$ → comparison (Binary search tree)

$$\text{Area} = n \log_2 n$$



Make it order eg n - H tree embedding

height = $\sqrt{n}$, width = $\sqrt{n}$, Area = n

when $n^{1/2}$,

$$n^{\log_b a} = n^{1/2}$$

$$\log_2^a = 1/2$$

$$2 \log_2 a = 1$$

$$a^2 = 1$$

$$a = \frac{1}{\sqrt{2}} \quad : a = \frac{1}{\sqrt{2}}.$$

$$\log_2 2 = 1$$

(or)

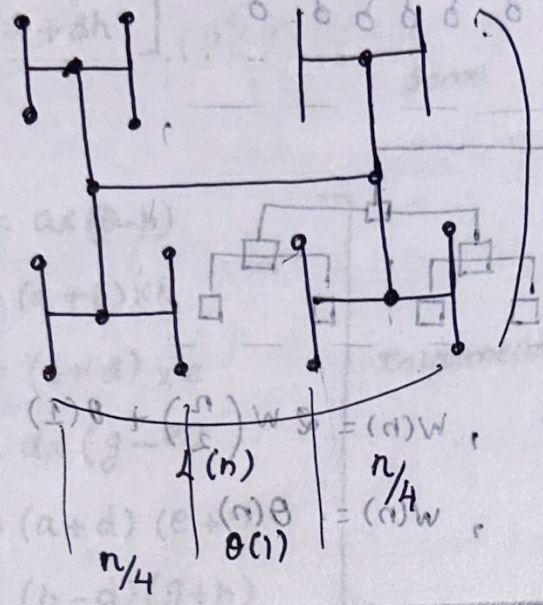
$$\log_4 2 = \frac{1}{2} \log_4 4 = \frac{1}{2}$$

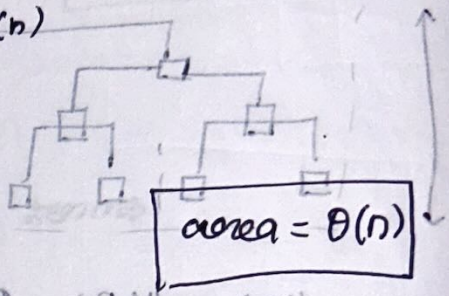
"H-embedding"

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

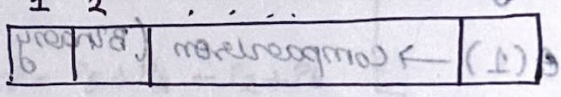
$a = 2, b = 4$

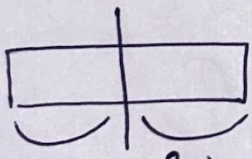
$n^{\log_4 2} = n^{1/2} = \sqrt{n} \Rightarrow \theta(\sqrt{n})$.





Quick sort → median pivot (efficient)
→ Random pivot (easy)





Divide → ①

Sort, Sort → Conquer → ②

merge → merge → ③ (Crucial — than divide step)

★ Divide step - Straight forward

★ Combine step - merge (merge sort : Uses extra array).

Quicksort - Inplace sorting algorithm.

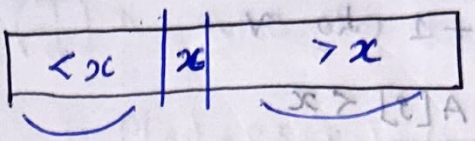
★ Partition: Partition array based on the pivot P, to sub arrays.

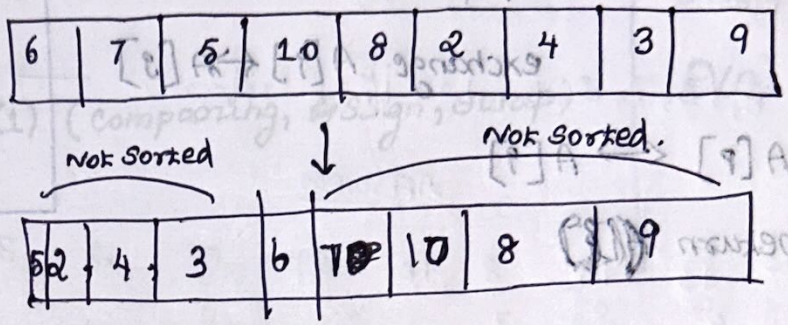
"many versions of pivot Choosing"

* Basically anyone could be a pivot.

Partition ('All the elements in left subarray are $\leq x$
All the elements in right subarray are $\geq x$ '

$x \leftarrow$ pivot





'After partition'

* Pivot - in its correct position

Conquer: Recursively sort both the subarrays.

Combine: Nothing - we are not using extra arrays'

Idea: Have subroutine in linear time

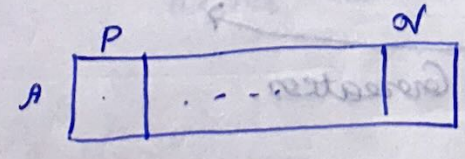
In merge sort

↓

* Divide step: trivial.
* Conquer: Sort
* Combine: Merge

* In merge sort, combine step is more trivial
* In quick - no combine step (divide step - crucial).

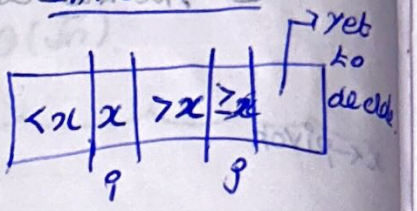
Partition $A[p, \ldots, r]$



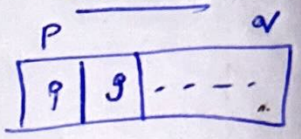
"First element as pivot"

1) $x \leftarrow A[p]$ (pivot as first element)
2) $q \leftarrow p$
3) for $j \leftarrow p+1$ to r
4) do if $A[j] < x$
5) $q \leftarrow q+1$
6) exchange $A[q] \leftrightarrow A[j]$
7) $A[p] \leftrightarrow A[q]$
8) return

Invariant



Initial



* If $A[j] \geq x$, just move by 1 (everything is in position) – Cool.
* If $A[j] < x$, Problem! Do something!

"Exchange with somebody who is $> x$.

∴ we know $A[q+1]$ is greater than x

---

(6) 10, 13, 5, 8, 3, 2, 11

(1) pivot, j, q, q, j

6 5 13 10 8 3 2 11
pivot q j

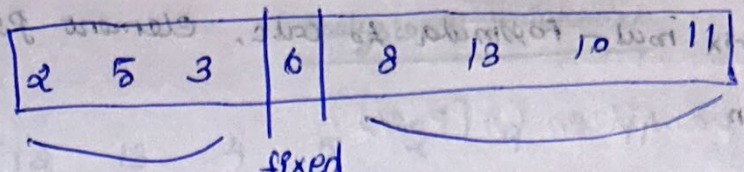
6 5 3 10 8 13 2 11
pivot q j

6 5 3 2 8 13 10 11
pivot q j

Swap

2 5 3 6 8 13 10 11

Smaller | Greater



① $\theta(1)$

② $\theta(1)$

③
④ loop
⑤ $\theta(1)$ (comparing, assign, swap)
⑥

overall $\theta(n)$

⑦ $\theta(1)$

⑧ $\theta(1)$

Assignment-2

1) Masters theorem - Solving recurrence

2) matrix multiplication - Divide & conquer - $O(n^3)$

3) median eg three partioning method

8 1 4 9 6 3 5 2 8 7 10

↳ center

$\frac{0+9}{2} = 4$

4) Binary search $T\left(\frac{n}{2}\right)$ split $\longrightarrow$ 2

5) $a_n \rightarrow$ no. eg n-bit strings - don't containing 2 consecutive 1's

recurrence relation = ?

$$a_n = a_{n-1} + a_{n-2}$$

---

6) 62 15 21 77 79 112 61 89

i) 77 can't be a pivot (61 → on right side)

ii) 112 could be a pivot (small - Right side

But small - left side)

7) Strassens matrix mul - Formula to calc. element present in II row, I Column

$$c = P_1 + P_7$$

8) master method: only solve $a\left(\frac{T}{b}\right) + f(n)$ form

9) (13), 18, 8, 10, 21, 7, 2, 32, 6, 19.
pivot

9) 8, 10, 7, 2, 6, 13, 18, 21, 32, 19 → No pivot

Answer

pivot=13 | 13 | 18 | 8 | 10 | 21 | 7 | 2 | 32 | 6 | 19

13	8	18	10	21	7	2	32	6	19
pivot									
13	8	10	18	21	7	2	32	6	19
13	8	10	7	21	18	2	32	6	19
13	8	10	7	2	18	21	32	6	19
13	8	10	7	2	6	21	32	18	19
6	8	10	7	2	(13)	21	32	18	19

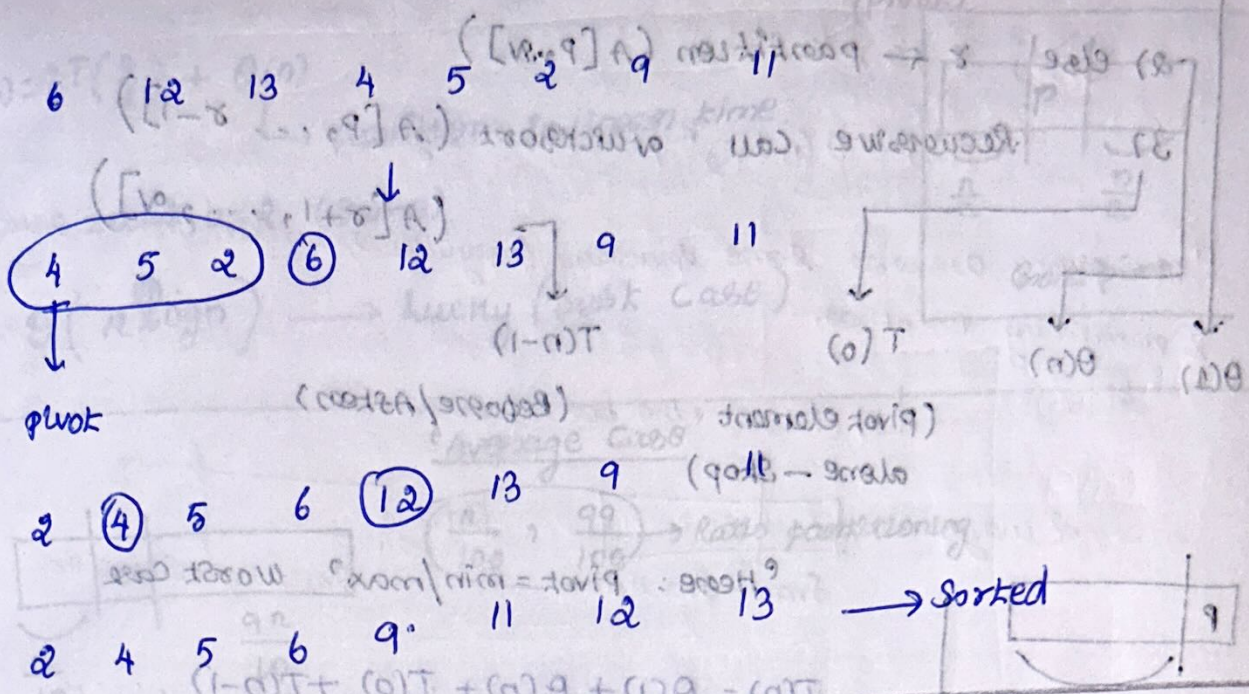
---

10) $T(n) = a\,T\left(\frac{n}{b}\right) + f(n)$

$f(n) = n^c$ (1st case) $\left(n^{\log_b a} + \varepsilon\right) > n^c$

$\therefore O\left(n^{\log_b a}\right)$

Analysis of quicksort



Runtime of quick sort – worst case

I/p sorted or reverse sorted: (pivot: minimum element) – Tes case

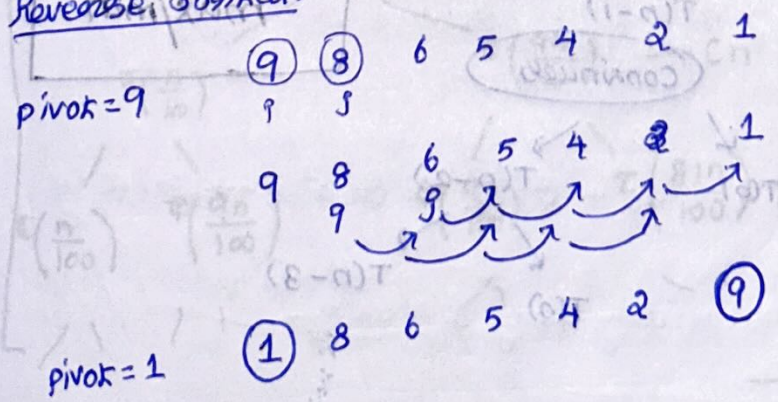
i jumps up to end, continues up to end (Just jumping).

'No use'

Partition: 0 to n-1 partition.

'only partition happens' – No change (comparison) waste

Reverse sorted:



Continues (only exchange when partition loop ends)

when pivot = min/max → Takes more time

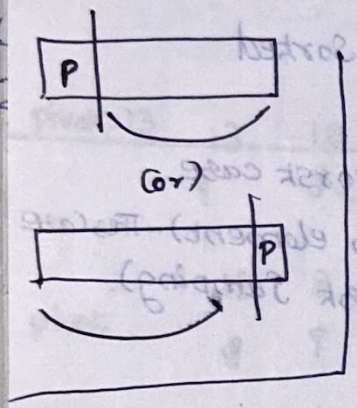
Recursion

Stop (place fixed) – partition stops when it's the single element.

Quicksort

1) If $p = q$ then done
2) else $r \leftarrow$ partition $(A[p..q])$
3) Recursive call quicksort $(A[p, \ldots, r-1])$ $(A[r+1, \ldots, q])$

$\theta(1)$ $\theta(n)$ $T(0)$ (pivot element alone — Stop) $T(n-1)$ (Before/After)



Here: pivot = min/max → worst case

$$T(n) = \theta(1) + \theta(n) + T(0) + T(n-1)$$

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

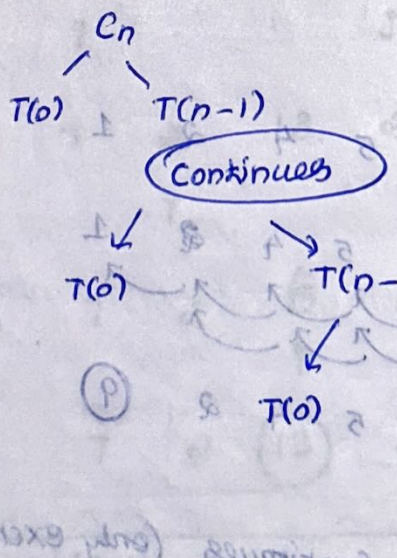
$$T(n) = T(n-1) + \theta(n)$$

→ can't solved by master method.

$$T(n) = T(n-1) + \theta(n) + c_n$$

'Bad pivot'
max/min

$T(n) =$



$$\therefore c_n + c(n-1) + c(n-2) + \ldots c(1)$$

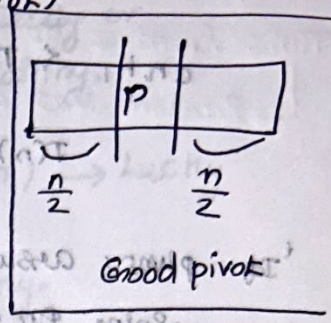
$$c \frac{n(n-1)}{2} = \theta(n^2)$$

'worst case'

'unlucky case'

Best case (partitioning array into half)

(pivot)



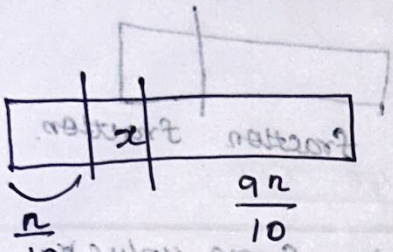
$T(n) = 2T\left(\frac{n}{2}\right) + \theta(n)$

↳ partition in linear time.

$a = 2, b = 2$ (Same)

$\theta(n \log n)$ ⟶ Lucky (best case)

Average case



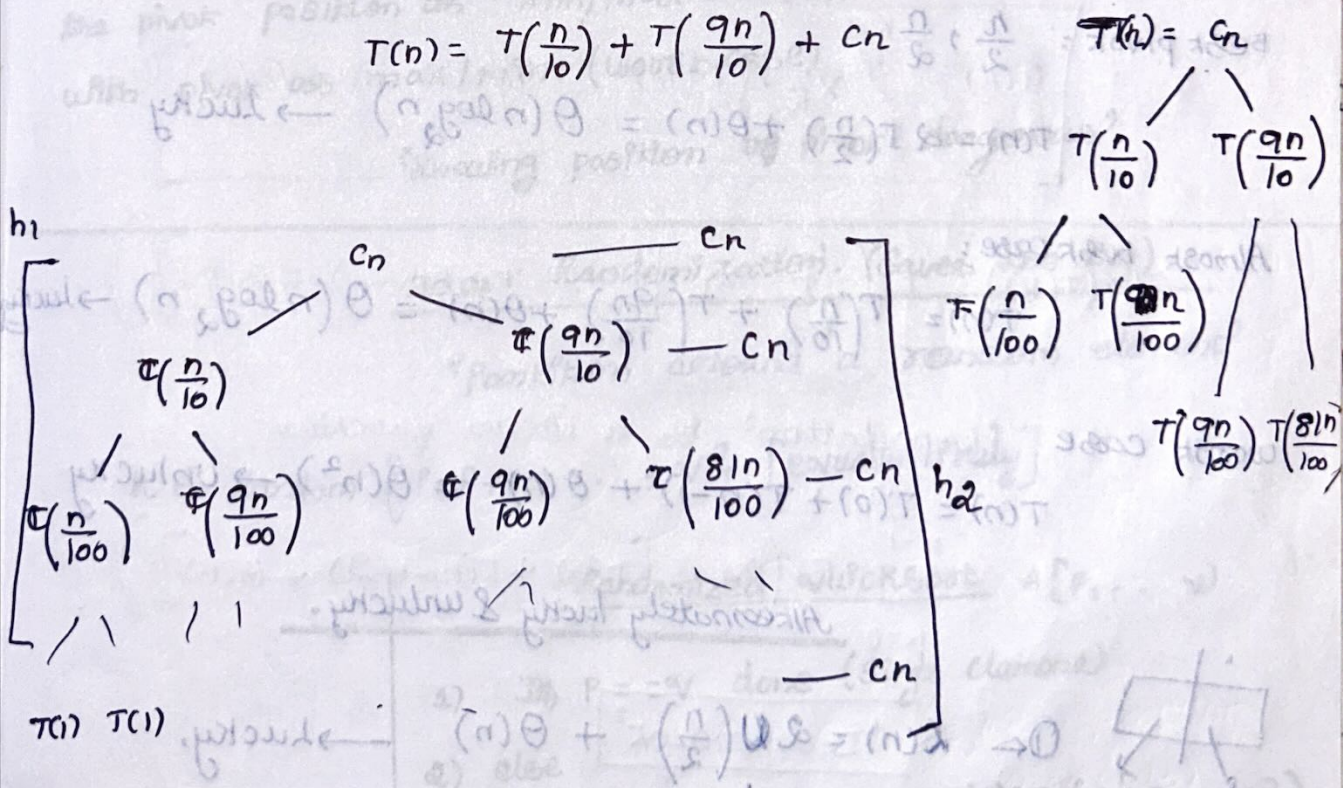
$\frac{n}{100}, \frac{99}{100}$ → Ratio partitioning.

⟶ (Assume) → Just in case.

$n = 20$, (2, 17)

partitioned.

$$T(n) = T\left(\frac{n}{10}\right) + T\left(\frac{9n}{10}\right) + cn$$



height = h_1, h_2

$$\frac{n}{10^{h_1}} = 1$$

$$h_1 = \log_{10} n$$

$$= \log_2 n \cdot \log_{10} 2$$

$$\frac{n}{10^{h_2}} = 1$$

$$h_2 = \log_{10} n$$

$$= \log_2 n - \log_{10} 2$$

Complete tree

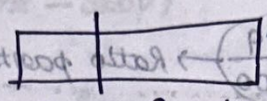
$c_1 n h_1 < T(n) < c_2 n h_2$

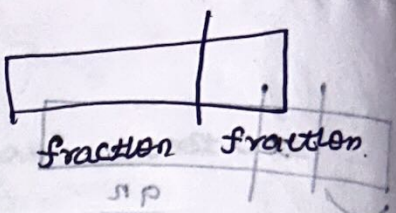
$T(n) = \theta(n \log_2 n)$ → Lucky

If pivot: assumed some fraction (may little fraction in one side) & remaining on other

↳ This is the best case.

∴ 2 ways





---

* 50-50 lucky, unlucky → Some stages - lucky, Some unlucky
* overall - better one

Randomized quick sort

Best pivot: $\frac{n}{2}, \frac{n}{2}$

$T(n) = 2T(\frac{n}{2}) + \theta(n) = \theta(n \log_2 n)$ → Lucky

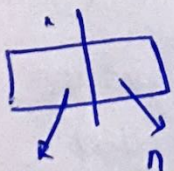
Almost best case:

$T(n) = T(\frac{n}{10}) + T(\frac{9n}{10}) + \theta(n) = \theta(n \log_2 n)$ → Lucky

Worst case

$T(n) = T(0) + T(n-1) + \theta(n) = \theta(n^2)$ → Unlucky

Alternately lucky & unlucky.



① ← $L(n) = 2U(\frac{n}{2}) + \theta(n)$ → Lucky.

$\frac{n}{2}$ $\frac{n}{2}$

↓

next time unlucky

② ← $U(n) = L(n-1) + \theta(n)$ → Unlucky

Combine ① & ②

$$L(n) = 2\left(L\left(\frac{n}{2}-1\right) + \theta\left(\frac{n}{2}\right)\right) + \theta(n)$$

→ Lower Ceiling or upper Ceiling.

$$= 2L\left(\frac{n}{2}-1\right) + \theta(n)$$

$$= 2L\left(\frac{n}{2}\right) + \theta(n) = \theta(n \log n) \rightarrow \text{Lucky}$$

"even Lucky, unlucky" → finally we are lucky.

"Its giving the idea of Randomization"

A[k]th element as pivot

$x \leftarrow A[k]$

$A[p] \leftrightarrow A[k]$

partition

Drawback: This way one can always construct an input which with pivot as max/min (worst case). the pivot position as min/max element. one can come with an i/p

"Knowing position of pivot: dangerous".

Idea: Randomization. (Saves the day).

"Partition around a random element"

$k \in \text{Random}\{p, p+1, \ldots, q\}$ [equally likely]

Randomized quicksort $A[p, \ldots q]$

1) If $p == q$ done (Single element)
2) else

$r \leftarrow$ Random partition $(A[p, \ldots q])$

(or)

Choose k randomly,

$A[p] \leftrightarrow A[k]$

$r \leftarrow$ partition $(A[p \ldots q])$

Time complexity

$T(n)$ = Runtime of randomized quicksort with n element.

↳ Random variable.

$$T(n) = \begin{cases} T(0) + T(n-1) + \theta(n) & 0 : n-1 \text{ split} \\ T(1) + T(n-2) + \theta(n) & 0 : n-2 \text{ split} \\ T(K) + T(n-K-1) + \theta(n) & K : n-K-1 \text{ split} \\ T(0) + T(n-1) + \theta(n) & \text{or } n-1 : 0 \text{ split.} \end{cases}$$

we know any one will occur randomly.

Algebra → Take expectation

'Indicator random variable'

$$X_K = \begin{cases} 1 & \text{if } K : n-K-1 \\ 0 & \text{otherwise} \end{cases}$$

$$T(K) = \sum_{K=0}^{n-1} X_K \cdot \left[T(K) + T(n-K-1) + \theta(n) \right]$$

$$E(T(K)) = E\left(\sum_{K=0}^{n-1} X_K \cdot \left[T(K) + T(n-K-1) + \theta(n) \right] \right)$$

'Expectation' is a linear function

$$= \sum_{K=0}^{n-1} E\left(X_K \cdot \left[T(K) + T(n-K-1) + \theta(n) \right] \right)$$

'Independent'

∴ For every step we are choosing a random no. for pivot.

$$= \sum_{}^{n-1} E(X_K)\, E\left(\left[T(K) + T(n-K-1) + \theta(n) \right] \right)$$

$$P(X_K = 1) = \frac{1}{n} \quad (n \text{ possible split})$$

(equally likely) $P(X_K = 0) = \frac{1}{n} = \frac{1}{n} = \frac{1}{n}$

$\therefore E(X_K) = 1 \times \frac{1}{n} + 0\left(1 - \frac{1}{n}\right)$

$$\boxed{E(X_K) = \frac{1}{n}}$$

$$E(T(K)) = \frac{1}{n} \sum_{K=0}^{n-1} \left(E\left[T(K) + T(n-K-1) + \theta(n)\right] \right)$$

$$= \frac{1}{n} \sum_{K=0}^{n-1} \left(E\left[T(K)\right] + E\left[T(n-K-1)\right] + E\left[\theta(n)\right] \right)$$

Sum: $K \rightarrow 0$ to $n-1$

$n-K-1 \rightarrow 0$ to $n-1$ → Reverse

'Same'

$$= \frac{2}{n} \sum_{K=0}^{n-1} E(T(K)) + \theta(n)$$

$$= \frac{2}{n} \sum_{K=0}^{n-1} E(T(K)) + \theta(n)$$

'Inequality'

Substitution method

$$E(T(n)) = \theta(n \log_2 n)$$

$$E(T(K)) \leq a K \log_2 K \qquad \forall K < n \quad [\text{Assume}]$$

$$\leq \frac{2}{n} \sum a K \log_2 K + \theta(n)$$

Fact:

$$\sum_{2}^{n-1} K \log_2 K \leq \frac{1}{2} n^2 \log_2 n - \frac{1}{8} n^2$$

$$\leq \frac{2a}{n}\left[\frac{1}{2}n^2\log_2 n - \frac{1}{8}n^2\right] + \theta(n)$$

$$\leq a\,n\log_2 n$$

we can prove this is true for n^2 — True for all.

Expected runtime: $n\log_2 n$ (average case analyze)

worst case: $O(n^2)$ → Always min/max pivot (even randomized)

quicksort: Inplace Algorithm Sorting.

---

(9)	1	8	2	7
9	1	8	2	7
9	1	8	2	7
(9)	1	8	2	9
9	1	8	2	9
7	1	8	2	9
7	1	8	2	9
7	1	8	2	9
2	1	8	7	9

→ Step 2

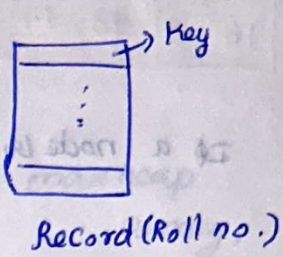
(2)	1	8
2	1	8

Heap Sort

* Heap – priority queue implementation.

A data structure implementing a set S, each set associated with a key, supporting the following operations

dynamic set : S



*Insert (S, x)
↳ key.

*max (S) → return the element whose key value is maximum

*extract - Max (S) → Return the element (maximum key value S) and remove it from S.

*Increase key (S, x, K) → Replace key value with a new key value

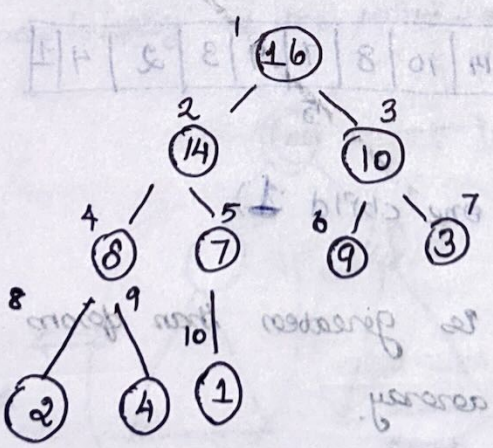
we need a data structure for S' → Heap.

Heap - An array - viewed as a complete binary tree. (nearly)

---

* Heap is used to implement a priority queue
* An array - visualized / viewed as nearly Binary tree (complete)

	1	2	3	4	5	6	7	8	9	10
A	16	14	10	8	7	9	3	2	4	1



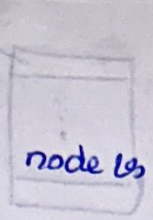
→ This visualization 'Heap'

'visualize' as a tree.

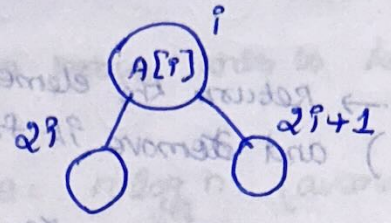
In c → we will do

struct node
{ int key;
struct node * left * right * parent
};

Here: No pointer is required : Advantage!



If a node is i then the childs are at $A[2i+1]$



left (i) = $2i$

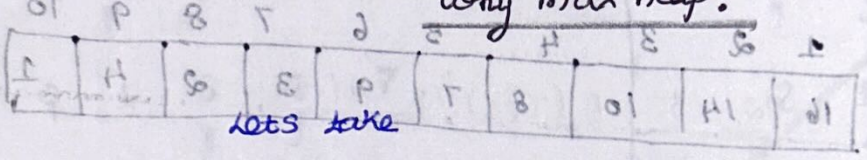
Right (i) = $2i+1$

parent (i) = $\left(\frac{i}{2}\right)$

height = $\Theta(\log_2 n)$ → good (balanced tree)

balanced tree : height is $\log_2 n$

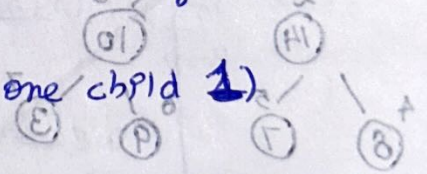
why max heap?



16	14	10	8	7	9	3	2	4	1

i = 5 → $2i+1 = 11$

→ $2i = 10$ (only one child 1)



If each the element is greater than from its child - max array.

8 > 2 & 4

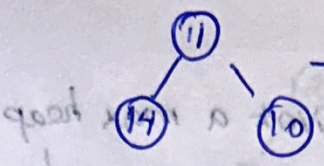
16 > 14 & 10

) parent > than childeren.

$A[i] \geq \{\max\{A[2i], A[2i+1]\}\ \forall i$

↓

max heap



→ Not a max heap array.

$max(S) = root \geq 16$ → Id array is max heap array.

Similarly: we can have min heap array

In general: how we make the array to max heap array?

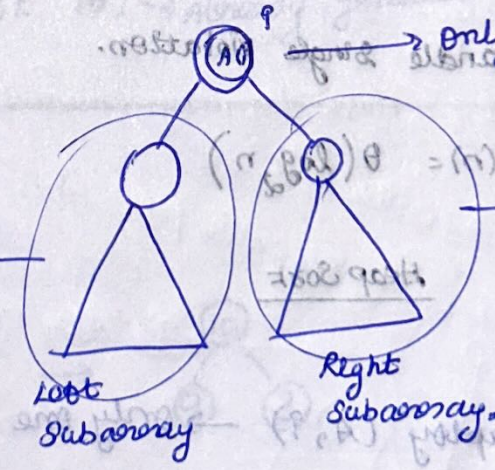
"Build max heap"

1) Produce a max heap array from an array. (unordered)

* build_max_heap → function / subroutine (take any i/p array – give max heap array).

* max heapify → Correct single violation of heap from in a subtree at its root.

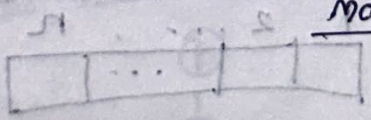
we call max heapify at the point →



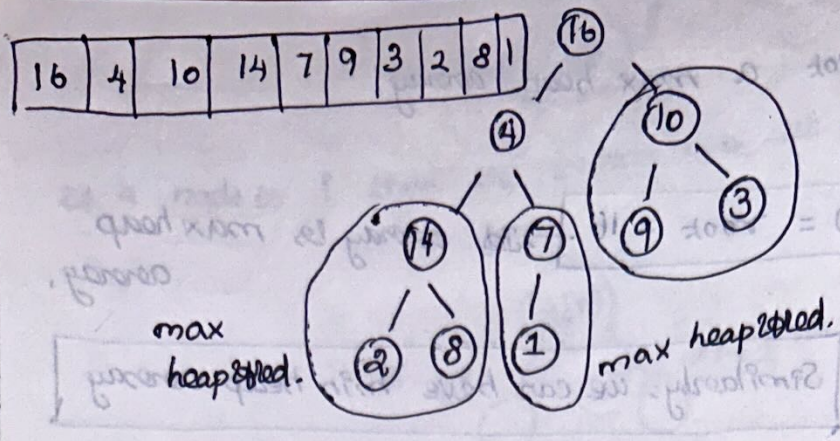
→ only need to tackle this.

Assume everything is OK with the two subarrays, we need call the max heapify at only A[1], which handles single violation.

Max heapify



16	4	10	14	7	9	3	2	8	1



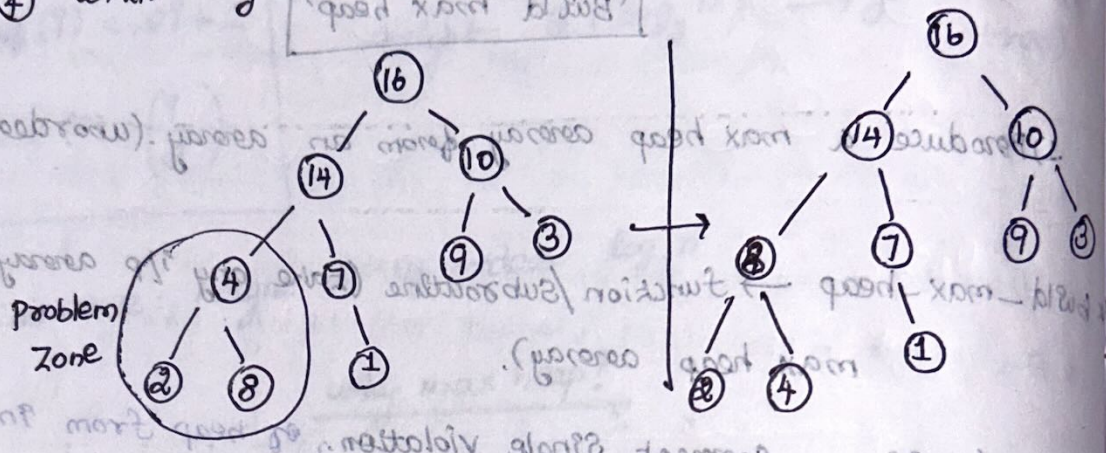
→ Not a max heap array!

max heapified.

max heapified.

we need to call max heapify at the point 4 : (Problem Spot)

Replace (4) with any one of child — maximum of.



max heapify:

we keep on doing until we reach the leaf. / everything is fixed.

Note: only handle single violation.

$$T(n) = \theta(\log_2 n)$$

Heap Sort

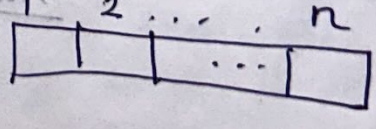
Build-max-heap

max-heapify (A, i) → only one violation

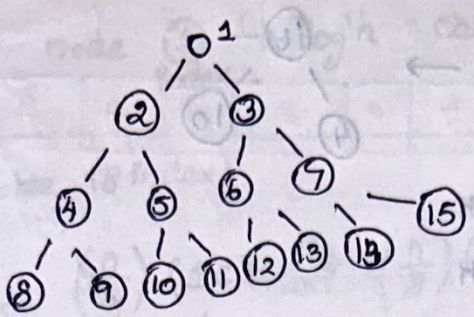
Build max heap.

1) For i ← $\frac{n}{2}$ down to 1

2) max-heapify (A, i)



why $\frac{n}{2}$ to 1



Leaves are basically

$$\left[\frac{n}{2}+1, \ldots n\right] = [8, 9, 10, \ldots 15]$$

"No child" → Leaves.

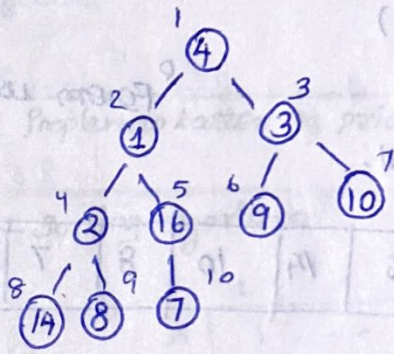
"They are already max heapified" — No child.

Where violation is possible? From $\frac{n}{2}$ to 1. (parents)

---

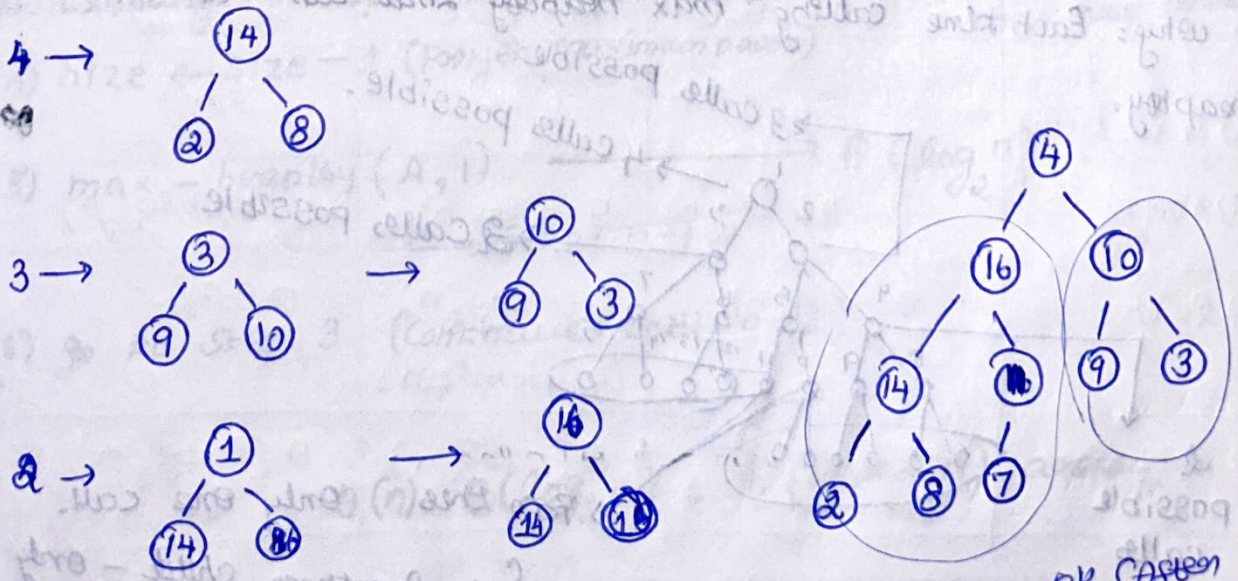
example

4	1	3	2	16	9	10	14	8	7



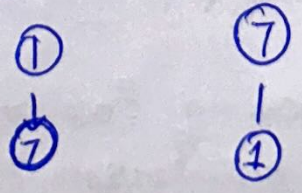
$[5+1, 7, 8, 9, 10]$ → children.

Take $\frac{n}{2}$ to 1 : (At 16) → Already generated (max heapified)

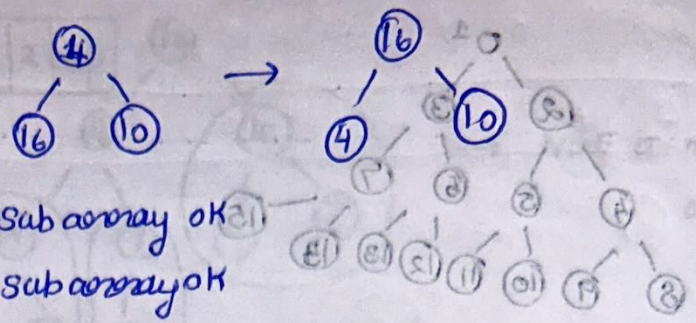


OK Case

Call (1) to max heapify:

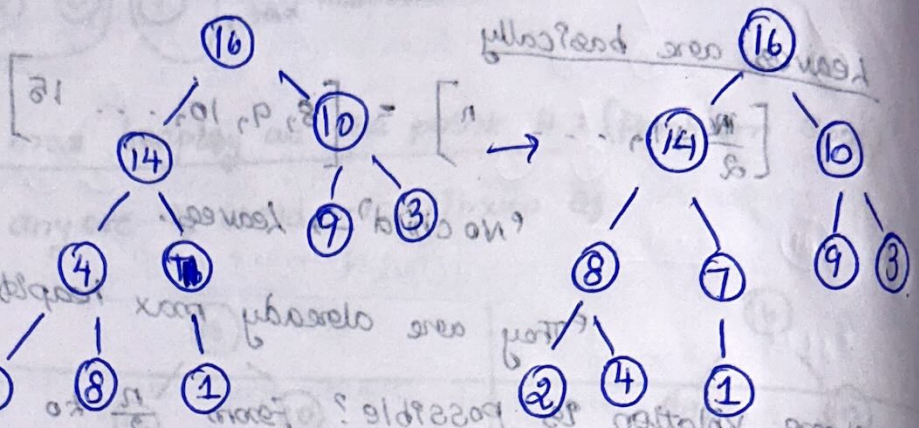


Finally



Now: Left subarray ok

Right subarray ok



max heapify(A,1)

// (A,2)

// (A,4)

No more call.

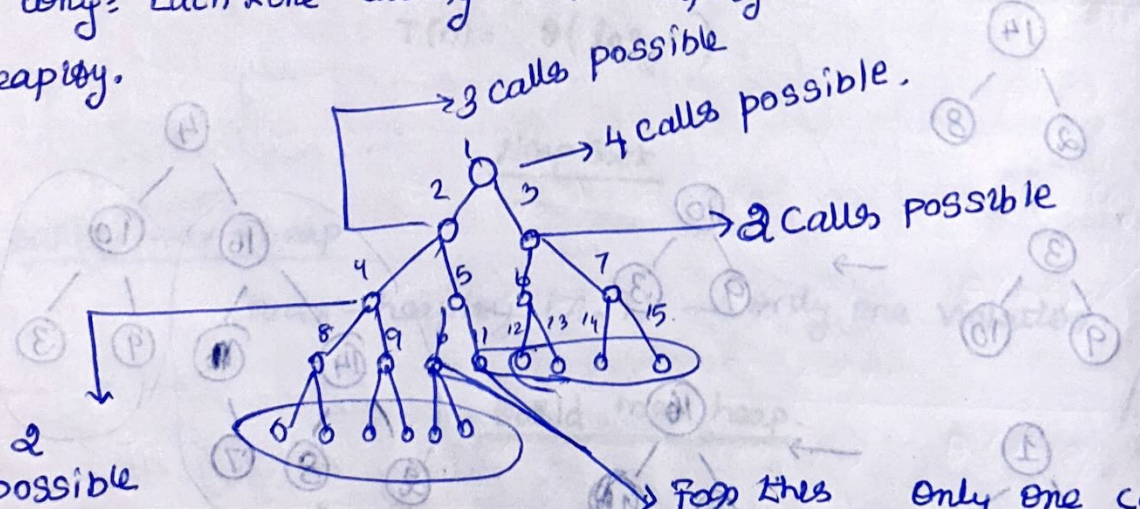
From whose you exchanged? → Call max heapify.

16	14	10	8	7	9	3	2	4	1

O/p

Intuitively: $\theta(n \log n)$

Why: Each time calling max heapify that can call max heapify.



① → child

② → grandchild

For node (1) → log n calls possible (not for all nodes)
index
log n is possible.

1 level above (8 index)

$= \left(\frac{n}{4}\right)(1c) + \left(\frac{n}{8}\right)(2\text{calls}) + \left(\log_2 \frac{n}{2}\right)(c)$

↳ 1 call ← calls

$= \left(\frac{n}{4}\right)c + \left(\frac{n}{8}\right)2c + \cdots + (\log_2 n)\,c$

$= c\,2^K \left(\frac{1}{2^0} + \frac{2}{2^1} + \frac{3}{2^2} + \cdots + \frac{(K+1)}{2^K} \right)$ → Bounded (Finite)

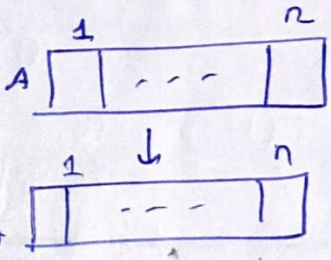
$\leq c\,n = \Theta(n)$

'Linear time'

---

* Heap data structure: used as a implementation of priority queue.

max heaping — Sorting algorithm.



max heap array.

1) size ← n

2) Build max heap

3) Exchange A[1] ⟺ A[n]

4) Size ← size − 1 (Forget maximum part)

5) max-heapify (A, 1) → $\Theta(\log_2 n)$
↳ size: n−1

6) go to step 3 (Continue until n = 1)

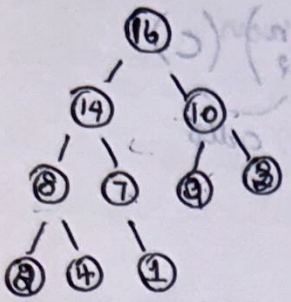
$\Theta(n) \cdot \Theta(\log_2 n) = \Theta(n \log_2 n)$ array.

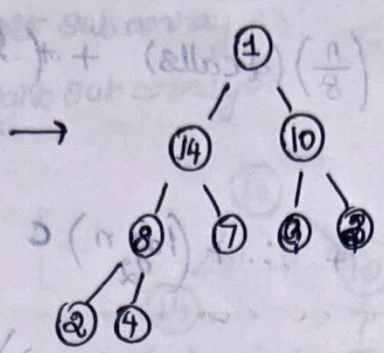
'In place Sorting Algorithm'

Given 4 1 3 2 16 9 10 14 8 7

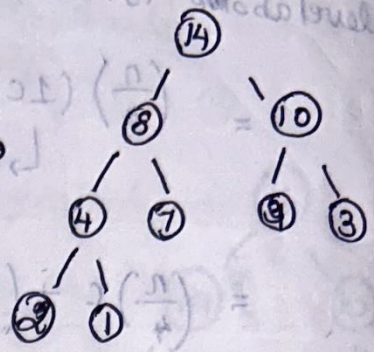
Build max heap :

16	14	10	8	7	9	3	2	4	1





Build →



Forget nth element

After exchanging

A[1] ↔ A[n]

Exchange (14) & (1)

'continue this'

Decision tree

Comparison based Sorting algorithm

- ↳ Insertion
- ↳ Merge
- ↳ Quick
  - ↳ Randomized
- ↳ Heap Sort

comparison based.

	Best case	Avg case	worst case
1) Insertion	$\theta(n)$	$\theta(n^2)$	$\theta(n^2)$
2) Merge	$\theta(n \log_2 n)$	$\theta(n \log_2 n)$	$\theta(n \log_2 n)$
3) Quick	$\theta(n \log_2 n)$	$\theta(n \log_2 n)$ (Randomized)	$\theta(n^2)$
4) Heap Sort	$\theta(n \log_2 n)$	$\theta(n \log_2 n)$	$\theta(n \log_2 n)$

'How fast at worst case'?

$n \log_2 n$ Best way.

'Is this the best way? or any possible optimization?'