10.10.2022

Data structures + algorithms = program

Complexity of algorithm : resource requirement

↳ analyze complexity before implementation

apriori : mathematical analysis (system independent)

↳ posteriori: behavioural analysis

- depends on the implementation (language)
- depends on the machine used to execute [configuration of the system, system dependent]

* Machine for implementation: Random Access Machine (RAM)
(theoretical model)

Linear Search:

```
L [1 ... n]
for (i=1 ; i≤n ; i++)        // assignment, condition, increment
    if (L[i] == key)
        return i
    if L[i] == (n+1)
        return 0
```

Step	count	cost (machine dependent)
i=1	1	c_1
i ≤ n	n+1	c_2
i++	n	c_3
indexing + comparison	n	c_4
return	1	c_5
comparison + add	1	c_6
return	1	c_7

n = i/p size

Size = L[1 ... n]

magnitude = k

most freq op [basic op of algo]

Exact Time Complexity of Algo $E(n) = c_1(1) + c_2(n+1) + c_3(n) + c_4(n) + c_5(1) + c_6(1) + c_7(1)$

RAM Model:

To make it machine independent, assume:

- All the basic operations take same basic constant amount of time.
- No memory hierarchy
- Only 1 processor
- Steps are executed sequentially [Sequential control flow]

* Most frequent operation is called the basic operation of algo
  Eg: (n+1) in linear search.

Input size:

consider an array L [1 ... n] with n elements ⇒ i/p size = n

OR

i/p size is also the magnitude.
Eg: to check element K is prime or not
Here, K = i/p size

* Theoretical definition: number of symbols required to encode the i/p in a Turing machine.

$\Sigma = \{1\}$ $\Sigma_2 = \{0, 1\}$ $\Sigma_3 = \{0, 1, 2\}$

2	11	0	
3	111	10	

$\log_{\text{base}} n$

* ~~not reasonable Exactly co~~

* Exact complexity isn't used as it's not reasonable for encoding, space complexity is high so asymptotic complexity is used

Asymptotic Complexity — something which is sufficiently large

approximation [looking at growth rate of function]

Eg: $E(n) = 3n^3 + 4n + 3$

$4n + 3$ → can be avoided since its negligibly small

∴ $T(n) = O(n^3)$ or $\Omega(n^3)$ or $\theta(n^3)$

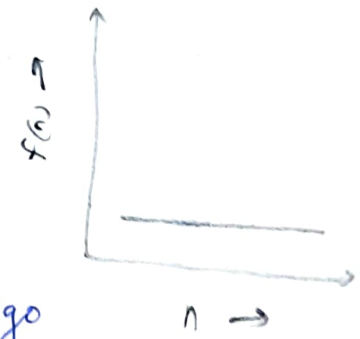
12.10.2022

Rate of Growth of Functions

* $f(n) = 5$ [Complexity of algo is independent of i/p size]

∴ $f(n) = c$

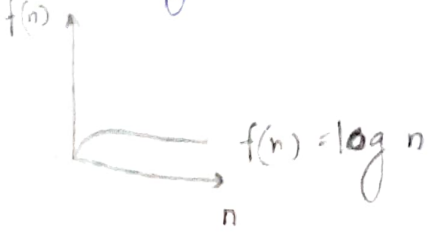
↳ Constant function algo → rate of growth of fn is constant.



Q Given a list of 'n' distinct nums, write an algo to find a number which is neither the smallest nor the largest.

→ take the 1st 3 nums in the list, the answer is the middle element of those 3.

* $f(n) = \log n$



Binary Search

L [1 . . . . n]

1 2 3 5 7 9 11 20 25

→ Searching element = 5

- find midpoint of array = 7
- check if 5 is small / large than midpoint 7 ⇒ smaller

↳ thus take lower half : 1 2 3 5

- find midpoint = 3
- check if 5 is small / large than midpoint 3 ⇒ larger

↳ thus, take upper half, we will get 5.

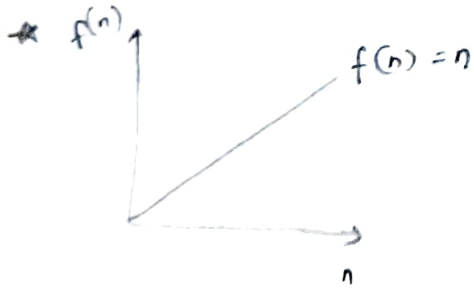
1	$n/2$
2	$n/4$
3	$n/8$
⋮	⋮
k	$n/2^k$

∴ Complexity = $\log n$

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

$2^k = n$

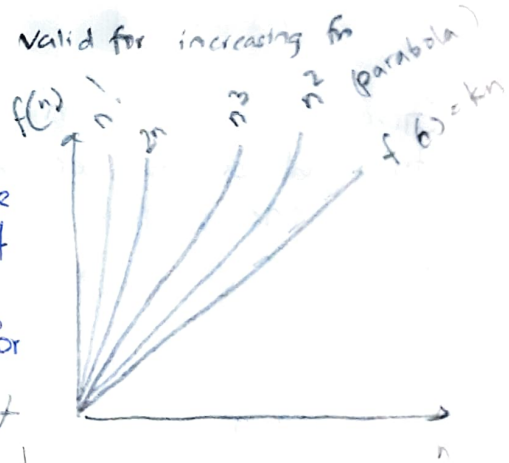
$k = \log n$



Basic Efficiency classes valid for increasing fn

constant, linear, n^3, $n!$, etc... are because in most algos the complexity will be any of these

→ also its valid only for +ve fn not in -ve



* Linear search → $n \log n$

5 4 1 3 7 6 9

To search 5, perform 1 time: best case i/p → takes min work

To search 20, has to perform n time: worst case i/p → takes max work

Avg case: expected num of operations.

* In binary search: best case = middle element

worst case = if element isn't present

or

2^{nd} element.

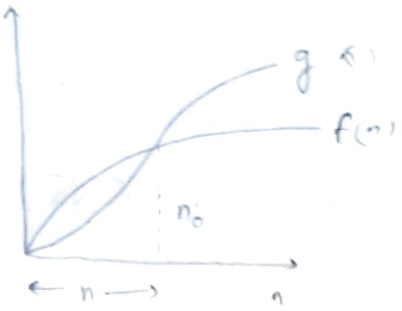
1. c
2. $\log n$
3. n
4. $n \log n$
5. n^2
6. n^3
7. 2^n
8. $n!$

Asymptotic Notations

Big O → worst case asymptotic complexity
Big Ω → best case
Big θ → tight bound

★ $f(n) = O(g(n))$, if there exists a +ve cost c and a n_0 such that
$f(n) \leq c(g(n))$, $\forall\ n \geq n_0$

⇒ for sufficiently large n, $f(n) \leq g(n)$
→ if it satisfies, then $f(n) = O(g(n))$



★ $f(n) = \Omega(g(n))$, if there exists a +ve cost c and an n_0 such that:
$f(n) \geq c(g(n))$ $\forall\ n \geq n_0$

26.10.2022

Q. S.T $n^2 + 2n + 5 = O(n^2)$. Yes

Let $f(n) = n^2 + 2n + 5$ and $g(n) = n^2$
We need to find a +ve constant c, s.t $f(n) \leq c \cdot g(n)$ $\forall\ n \geq n_0$
(basically we are trying to remove 2nd)

$\forall\ n \geq 5$, $2n + 5 \leq 2n + n$
$\leq 3n$

$\Rightarrow n^2 + 2n + 5 \leq n^2 + 3n$ [We added n^2 on both sides to compare]

$\forall\ n^2 > 3n$, $n^2 + 2n + 5 \leq n^2 + n^2$
$\leq 2n^2$

$n > 3$, $n^2 + 2n + 5 \leq 2n^2$

i.e. $n > 5$ and $n > 3$ $\Rightarrow n > 5$

if $n_0 = 6$, $n^2 + 2n + 5 \leq 2n^2$, where $c = 2$

Q. Is $2^{n+1} = O(2^n)$? Yes

$2^{n+1} = 2 \cdot 2^n$

Let $f(n) = 2^{n+1}$

$g(n) = 2^n$

$f(n) \leq c \cdot g(n), \forall n \geq n_0$

$2^{n+1} = 2 \cdot 2^n$, ~~which is of the form~~ where $c = 2$

Q. Is $2^{2n} = O(2^n)$? No

$(2^m)^n = 2^{mn}$

We can write: $2^{2n} = 2^n \cdot 2^n$

$2^{2n} = (2^n)^2$

But c is a constant, so $c \neq 2^n$.

Q S.T $\log n = O(\sqrt{n})$

★ We can use the derivative test:

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = \begin{cases} 0, & f(n) = O(g(n)) \rightarrow \text{rate of growth of } g \text{ is higher} \\ \infty, & \text{rate of growth of } f(n) \text{ is higher} \\ c > 0, & \text{rog of both fns is same} \end{cases}$$

Using the above:

$$\lim_{n \to \infty} \frac{\log n}{\sqrt{n}} = \frac{\infty}{\infty} \text{ form so applying 1st derivative rule:}$$

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = \lim_{n \to \infty} \frac{f'(n)}{g'(n)}$$

$$= \lim_{n \to \infty} \frac{1/n}{\frac{1}{2\sqrt{n}}}$$

$$= \lim_{n \to \infty} \frac{2\sqrt{n}}{n}$$

$$= 2 \lim_{n \to \infty} \frac{1}{\sqrt{n}} = \underline{0}$$

∴ rog of $g(n)$ is ~~equal~~ greater ⇒ true.

★ $f(n) = O(g(n))$ and $g(n) = O(h(n))$, then:

$$f(n) = O(h(n))$$

↳ called transitivity (applicable to all 3)

★ $f(n) = O(f(n))$
$f(n) = \Theta(f(n))$
$f(n) = \Omega(f(n))$
} reflexive.

★ $f(n) = O(g(n))$, $g(n) = O(f(n))$? No } ⇒ not symmetric.
$f(n) = \Omega(g(n))$, $g(n) = \Omega(f(n))$? No }
$f(n) = \Theta(g(n))$, $g(n) = \Theta(f(n))$? Yes ⇒ symmetric.

Q. Let $f(n)$ and $g(n)$ be two non-negative functions.
ST. $\max(f(n), g(n)) = \Theta(f(n) + g(n))$

Let $f(n) = \max(f(n), g(n))$
$g(n) = f(n) + g(n)$

To prove: $c_1(f(n) + g(n)) \leq \max(f(n), g(n)) \leq c_2(f(n) + g(n))$

this will give 1 of the 2 fns

we know that the result will be $\leq f(n) + g(n)$, so we can take $c_2 = 1$

We can also say that:

$$f(n) + g(n) \leq 2 \cdot \max(f(n), g(n))$$
$$\frac{1}{2}(f(n) + g(n)) \leq \max(f(n), g(n))$$

$$\therefore \frac{1}{2}(f(n) + g(n)) \leq \max(f(n), g(n)) \leq 1(f(n) + g(n))$$

Insertion Sort

5 9 3 6 2 7 8

5 | 9 3 6 2 7 8

5 9 | 3 6 2 7 8

3 5 9 | 6 2 7 8

3 5 6 9 | 2 7 8

2 3 5 6 9 | 7 8

2 3 5 6 7 9 | 8

2 3 5 6 7 8 9

```
for (i = 1 to n)
{
    ind = i
    for (j = i-1 to 0)
    {
        if (L[ind] < L[j])
        {swap
         ind = j
        }
    }
}
```

Algorithm:

i/p: A list $L[1 \ldots n]$

o/p: Sorted list L

1	2	3	4	5	6	7
5	9	3	6	2	7	8

for $j = 2$ to n

$x_j = L[j]$, $i = j-1$

while $x_j < x_i$ and $i > 0$

$x_{i+1} = x_i$

$i = i - 1$

$x_{i+1} = x_j$

→ Time complexity:

in the j^{th} iteration of outer loop, no. of data movements $= 2 + d_j$, where d_j is the total # of data movements in the while loop

$$T(n) = (2+d_2) + (2+d_3) + \cdots + (2+d_n) = \sum_{j=2}^{n} (2+d_j)$$

⇒ Best case: when all the elements are in the sorted order $d_j = 0$

$$T_{Best}(n) = \sum_{j=2}^{n} (2) = 2(n-1) = O(n)$$

adding 2 (n-1) times

⇒ ~~Average~~ Worst case: when all the elements are sorted in the reverse order, i.e. d_j is a max value.

$$T_{worst}(n) = \sum_{j=2}^{n} 2 + (j-1) = \underbrace{\sum_{j=2}^{n} 2}_{2(n-1)} + \underbrace{\sum_{j=2}^{n} (j-1)}_{\frac{n(n-1)}{2}} = 2(n-1) + \frac{n(n-1)}{2}$$

$$= O(n^2)$$

⇒ Average case:

- On an average, how many data movements occur in the while loop?
- jth iteration, prob that x_j > all elements in the sorted part = $\frac{1}{j}$

↳ jth iteration, prob that x_j is the j^{th} largest is $\frac{1}{j}$

- jth iteration $\frac{1}{j}(2 + j - 1)$

$$\sum_{j=2}^{n} \frac{1}{j}(2 + (j-1)) = \frac{3}{j} + \frac{4}{j} + \cdots + \frac{2+(n-1)}{j}$$

$$= \frac{1}{j}[3 + 4 + \cdots + (n+1)]$$

$1^{st} = \frac{1}{j}(2+0)$

$2^{nd} = \frac{1}{j}(2+1)$

$\vdots$

$j^{th} = \frac{1}{j}(2+j-1)$

$$\sum_{i=1}^{j} \frac{1}{j}(2 + i - 1) = \frac{2}{j} + \frac{3}{j} + \cdots + \frac{2(j-1)}{j}$$

$$= \frac{2}{j} + \frac{3}{j} + \cdots + \frac{j+1}{j}$$

$$= \sum_{i=1}^{j} \frac{i+1}{j}$$

$$= \frac{1}{j}\left(\sum_{i=1}^{j} i + j\right)$$

$$= \frac{1}{j}\left[\frac{j(j+1)}{2} + j\right]$$

$$= \frac{1}{j}\left[\frac{j(j+1) + 2j}{2}\right]$$

$$= \frac{1}{j}\left[\frac{j(j+1+2)}{2}\right]$$

$$= \frac{j+3}{2}$$

$$T_{avg}(n) = \sum_{j=2}^{n}\left(\frac{j+3}{2}\right) = \frac{1}{2}\left[\underbrace{\sum_{j=2}^{n} j}_{O(n^2)} + \underbrace{\sum_{j=2}^{n} 3}_{O(3(n-1))}\right] = O(n^2)$$

Avg num of swaps required to place $x_j = \sum_{i=1}^{j}(2+i-1)$

27.10.2022

Binary Search:

Best case = $O(1)$

Worst = $O(\log n)$

Average case:

# Comparisons	# elements	prob
1	1	$1/n$
2	2	$2/n$
3	4	$4/n$
4	8	$8/n$
⋮	⋮	⋮
i	2^{i-1}	$2^{i-1}/n$

$$\sum_{i=1}^{\log n} i \cdot 2^{i-1} = \underbrace{2^k(k-1)+1}_{\text{an assumption that needs to be proved}}, \text{ where } k = \log n$$

Base case: $i=1$

Assume $k=m$, $m>1$: $\sum_{i=1}^{m} i \cdot 2^{i-1} = 2^m(m-1)+1$

To show: $k = m+1$ $\quad \sum_{i=1}^{m+1} i \cdot 2^{i-1} = 2^{m+1}(m+1-1)-1 = 2^{m+1}\cdot(m+1)$

$$\downarrow$$

$$\sum_{i=1}^{m} i \cdot 2^{i-1} + (m+1)\,2^{m+1-1} = \sum_{i=1}^{m} i \cdot 2^{i-1} + (m+1)\,2^{m}$$

$$2^m(m-1) + 1 + (m+1)2^m$$

$$2^m(m-1 + m+1) + 1$$

$$2^m \cdot 2m + 1 = 2^{m+1} \cdot m + 1$$

Assume n is a power of 2:

$$2^{\log n}(\log n - 1) + 1 = n(\log n - 1) + 1 = n \log n - n + 1$$

$\therefore$ Average case = $O(\log n)$

Recursive Algorithms

Eg: $$fact(n) = \begin{cases} 1, & \text{if } n = 0 \quad \text{base case} \\ n \times fact(n-1), & \text{otherwise} \quad \text{recursive base} \end{cases}$$

$$fib(n) = \begin{cases} 0, & \text{if } n = 1 \\ 1, & \text{if } n = 2 \end{cases} \} \text{ base case}$$
$$fib(n-1) + fib(n-2), \quad \text{otherwise} \quad \text{recursive case}$$

- Base case: case where the recursion stops
- Recursive case:
- Progress: every recursion moves/progresses towards base case

} properties of recursion

Eg: $n = 5$

$fact(5) = 5 \times fact(4)$

$\downarrow$

$4 \times fact(3)$

$\downarrow$

$3 \times fact(2)$

$\downarrow$

$2 \times fact(1)$

$\downarrow$

$1 \times fact(0)$

$\downarrow$

1

Recursive Binary Search:

```
{
if (s<e) ⟶ here the base case is implicit, i.e. s>e
  {
    m = (s+e)/2
    if A[m] == key
        return 1
    else if A[m] < key
        s = m+1
        rbs(s,e)
    else
        e = m-1
        rbs(m,e)
  }
}
```

Recursive Factorial

```
i/p: n
o/p: n!

if n == 0  return 1
else return n * RecursiveFactorial(n-1)
```

Let T(n) be the complexity of the above function.

Then: ~~T(n) = T(n-1)+1~~ ~~This is called.~~

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

} Recurrance Relation.

★ Unrolling method

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

By the time we reach T(0), we will get 'n' ones

$\Rightarrow O(n)$

2.11.2022

Q. Algo $R_1(n)$

```
if (n > 0)
    for (i = 1 to n)
        s = s + i
    return R1(n/2)
```

return ↑ for loop ↓

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

$T(1) = c$

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

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

$= T(n/8) + n/4 + n/2 + n$

n

$= 2n$

$= O(n)$

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

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

Then it will be complex

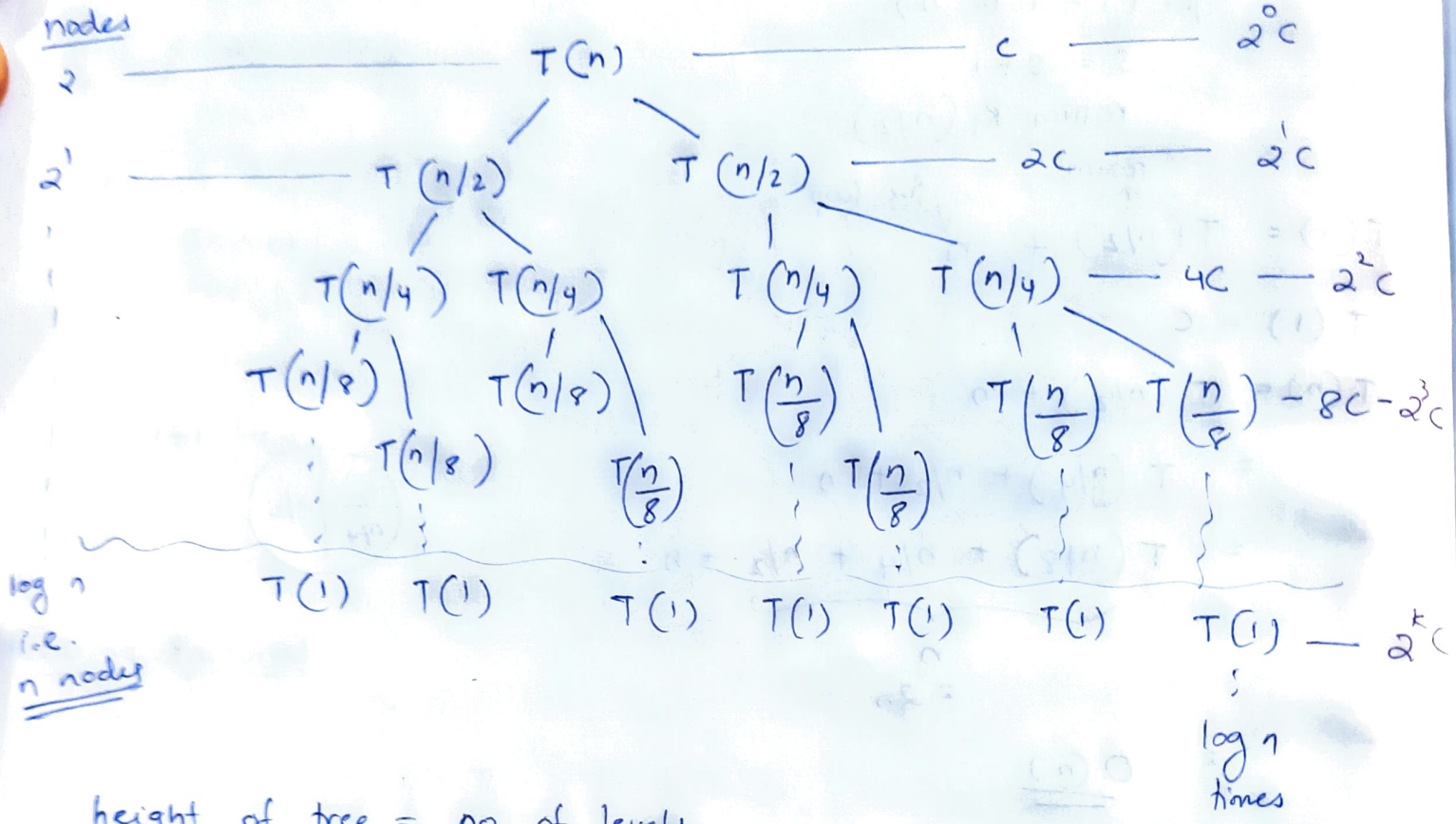
∴ we will use the Recursion Tree Method

Recursion Tree Method

TC = IC + LC → we get exact

A tree that shows how the recursion progresses.

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



height of tree = no. of levels

$k = \log_2 n$

$2^k c = 2^{\log n} c$

$= n^{\log_2 2} c$

$= nc$

Work done in leaf node = constant

* Complexity of internal node

$= c(1 + 2 + 4 + 8 + \ldots n)$

$\Rightarrow \frac{ar^{n-1}}{r-1} = c\left[\frac{1 \cdot 2^{\log_2 n} - 1}{2-1}\right] = c \cdot \frac{n-1}{1} = O(n)$

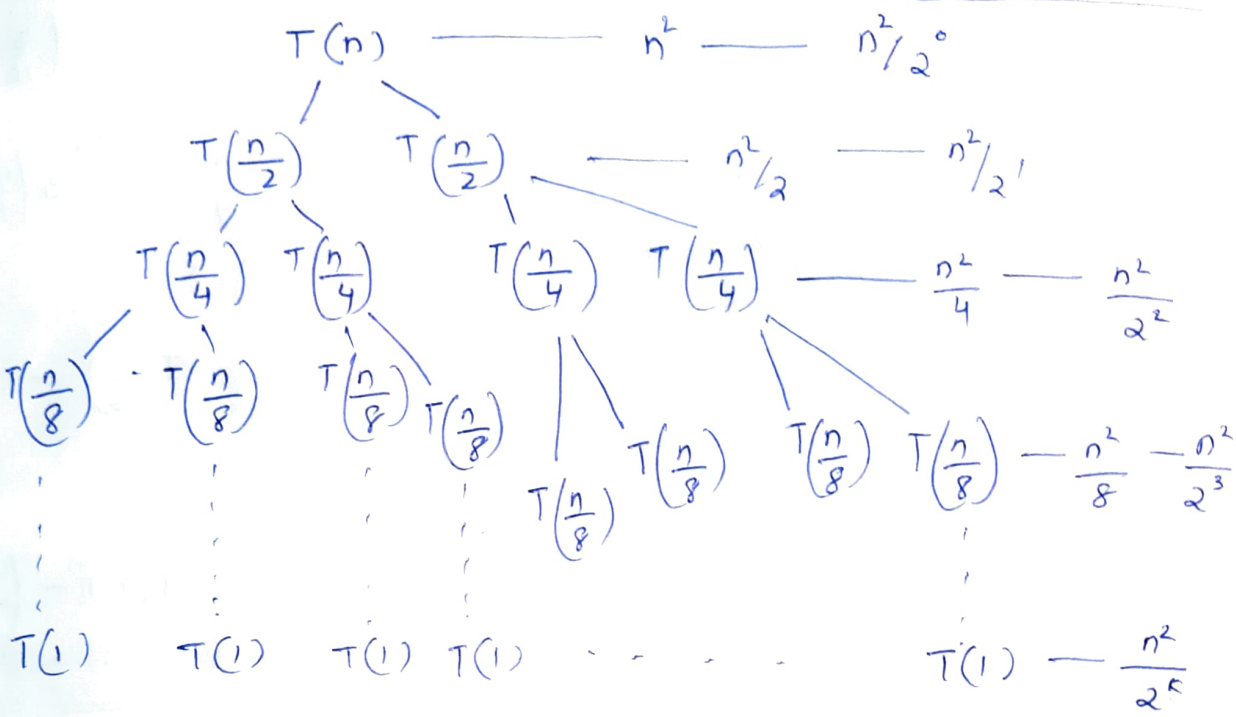
$\frac{ar^{n-1}}{r-1}$

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

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

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

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



complexity of internal node :

$$= n^2 + \frac{n^2}{2} + \frac{n^2}{4} + \cdots \frac{n^2}{2^k}$$

$$= n^2 \left(1 + \frac{1}{2} + \frac{1}{4} + \cdots \frac{1}{2^k}\right)$$

$$= n^2 \left(1 + \frac{1}{2} + \left(\frac{1}{2}\right)^2 + \left(\frac{1}{2}\right)^3 + \cdots\right)$$

$1/2 < 1$

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

$a = 1$

$r = 1/2 \Rightarrow \frac{ar^n - 1}{r-1} = \frac{1}{1-\frac{1}{2}} = \underline{\underline{2n^2}}$

$LC = n \cdot T(1)$

$= n\,\Theta(1)$

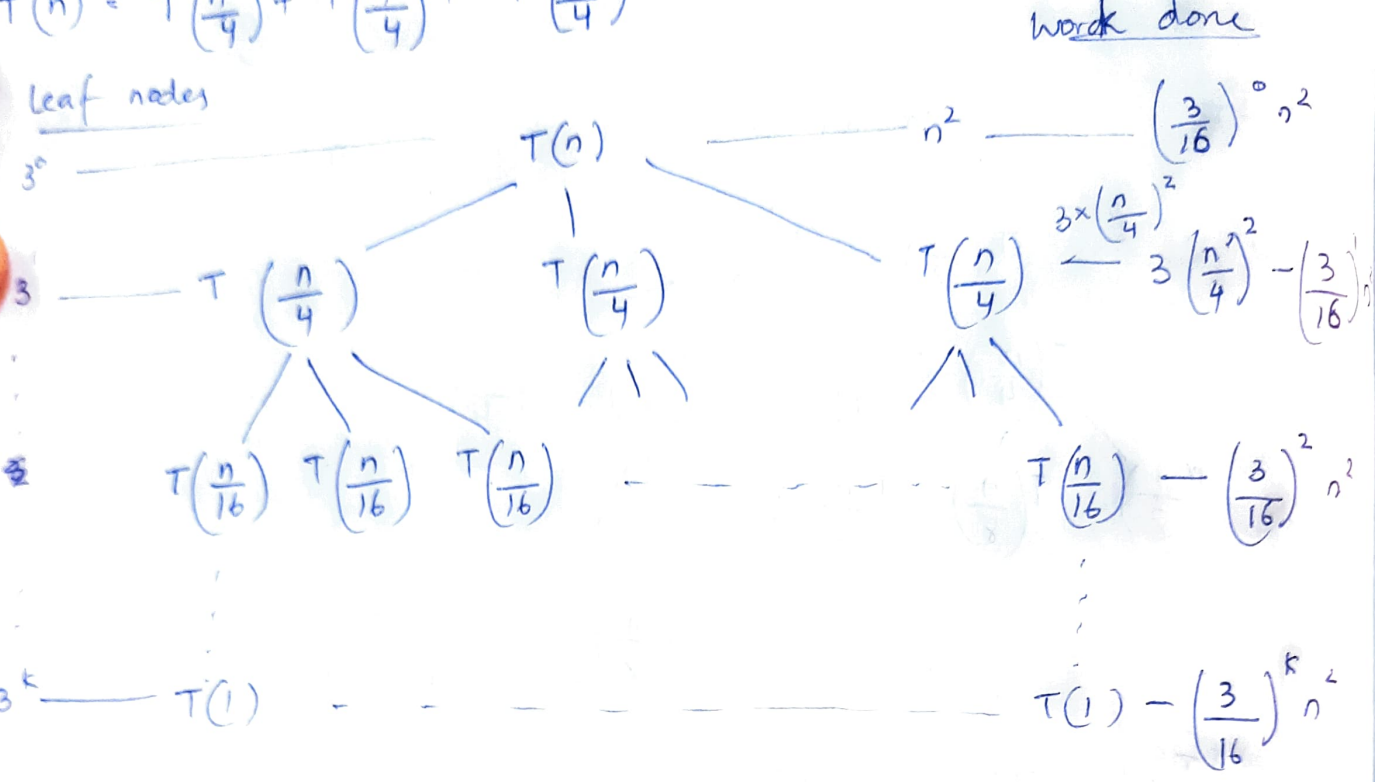
$= \underline{\underline{n}}$

$LC \rightarrow 2^{\log_2 n}$

∴ Total complexity = ~~$2n^2 + n$~~ $2n^2 + n = \underline{\underline{O(n^2)}}$

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

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



Total internal complexity (IC)

$= n^2 \left[\left(\frac{3}{16}\right)^0 + \left(\frac{3}{16}\right)^1 + \left(\frac{3}{16}\right)^2 + \cdots \right]$

$= n^2 \left[1 + \frac{3}{16} + \left(\frac{3}{16}\right)^2 + \cdots \right]$

$= n^2 \left(\frac{16}{13}\right)$

$= O(n^2)$

GP : $a = 1$

$r = 3/16$

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

$= \frac{1}{1 - \frac{3}{16}}$

$= \frac{1}{13/16}$

$= 16/13$

Complexity of leaf nodes

leaf nodes $= \log_4 n = n^2$

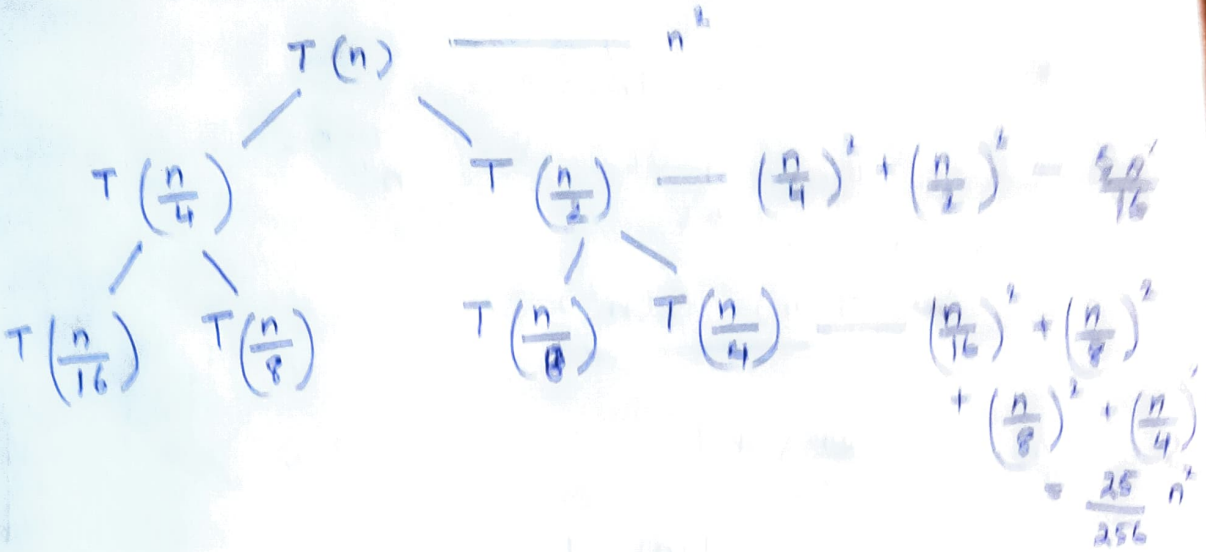
LC $= n^2 \; T(1) = n^2 \; \Theta(1) = O(n^2)$

BST
Binary Tree Traversal

Binary Tree BST Tree Traversal

3.11.2022

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



* if the tree was balanced, then the work for CN would be $\frac{5^k}{16^k} n^2$.

$\therefore T(n) \leq O(n^2)$

Master Method: extension of recursion tree

1. if $f(n) = O(n^{\log_b a - \varepsilon})$ for some $\varepsilon > 0$, then
   $T(n) = \Theta(n^{\log_b a})$
2. if $f(n) = \Theta(n^{\log_b a})$ then $T(n) = \Theta(n^{\log_b a} \cdot \log n)$
3. if $f(n) = \Omega(n^{\log_b a + \varepsilon})$ and if $af(n/b) \leq c \cdot f(n)$, then
   $T(n) = \Theta(f(n))$

↓ regularity condition

* This method is applicable if in the form: $T(n) = aT(n/b) + f(n)$ the recursion is

* $f(n) = O(n^{\log_b a - \varepsilon})$ → even if we reduce a small quantity ε from the power, $f(n)$ is still smaller than $n^{\log_b a}$

if there is atleast 1 ε value that satisfies then its true.

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

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

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

Let $\epsilon = 0.5$, then $n^{\log_b a - \epsilon} = n^{2-0.5}$

↳ $n \leq n^{2-0.5}$

$\Rightarrow T(n) = \underline{\theta(n^2)}$

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

$\left|\frac{2n}{3} = \frac{n}{3/2}\right.$

$a = 1,\ b = 3/2,\ f(n) = 1$

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

$\therefore f(n) = \theta(n^{\log_b a}) \Rightarrow T(n) = \theta(1 \cdot \log n) = \underline{\theta(\log n)}$

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

$a = 3$

$b = 4$

$f(n) = n \log n$

$n^{\log_b a} = n^{\log_4 3} = n^{0 \leq x < 1}$

We know that $n \log n > n^{0<x<1}$, so we can find some +ve ϵ such that $f(n) > n^{\log_b a + \epsilon}$

$4^x = 3$

$4^{1/2} = 2$

$4^1 = 4$

b/w 1/2 and 1

Applying case 3: checking the regularity condition.

just check if this is correct

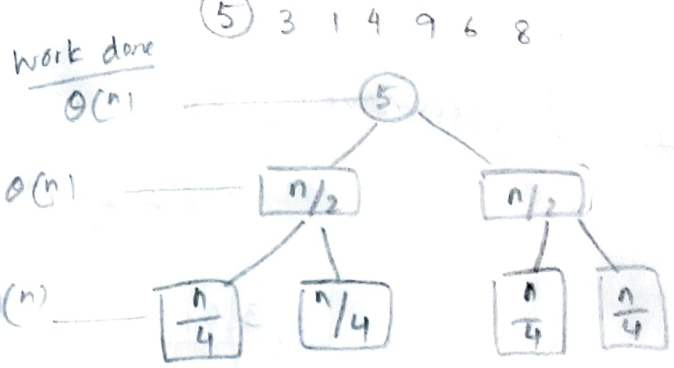
$a\,f(n/b) \leq c\,f(n)$ ~~$3 \cdot f(n/4) = 3$~~

$3 f(n/4) \leq c \cdot f(n)$

$3 \cdot \frac{n}{4} \log \frac{n}{4} \leq c \cdot n \log n \Rightarrow \frac{3}{4} n \log \frac{n}{4} \leq c \cdot n \log n$

which is true.

∴ $T(n) = \underline{\Theta(n \log n)}$



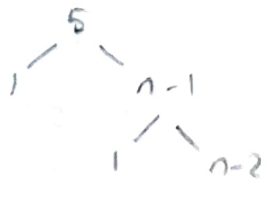
Quicksort

Algo. Quicksort (L, S, E)
P = Partition ( ) — n
QS (L, S, P-1) — i
QS (L, P+1, E) — n-i

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

★ If partition ~~produces~~ proccdure divides the array into 2 halves always, then height of tree = log n
Work done in each level = Θ (n)
∴ total complexity = n log n (best case.) } this happens when the array is already sorted.

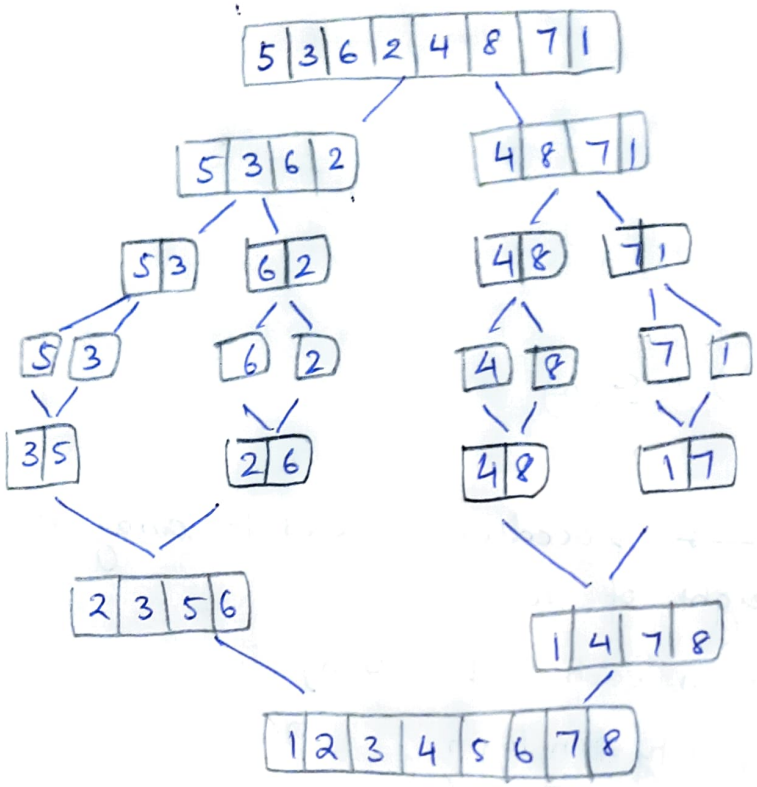
★ If the partition procedure divides the array into 2 unequal parts such that :
- 1 half has 1 element
- other half has n-1 elements.



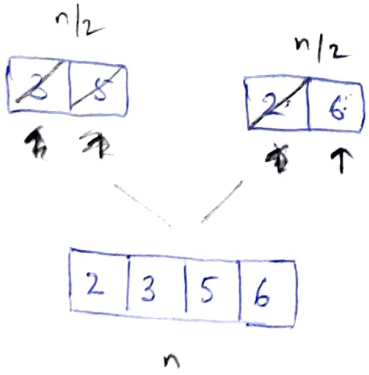
and work done at each level = Θ (n)
∴ total complexity = n × n = n^2 (worst case) } happens when the array is reversely sorted

7.11.2022

Merge sort (Recursive)



Merge procedure: used to merge two sorted arrays into a single sorted array



2 < 3

3 < 6

5 < 6

requires $\Theta(n)$ comparisons

the base op is comparison

★ Write the algorithm ★
for merge procedure

Recursive Merge sort (A, s, e):

$$m = \left\lceil \frac{s+e}{2} \right\rceil$$

LA = RMS (A, s, m-1)

RA = RMS (A, m, e)

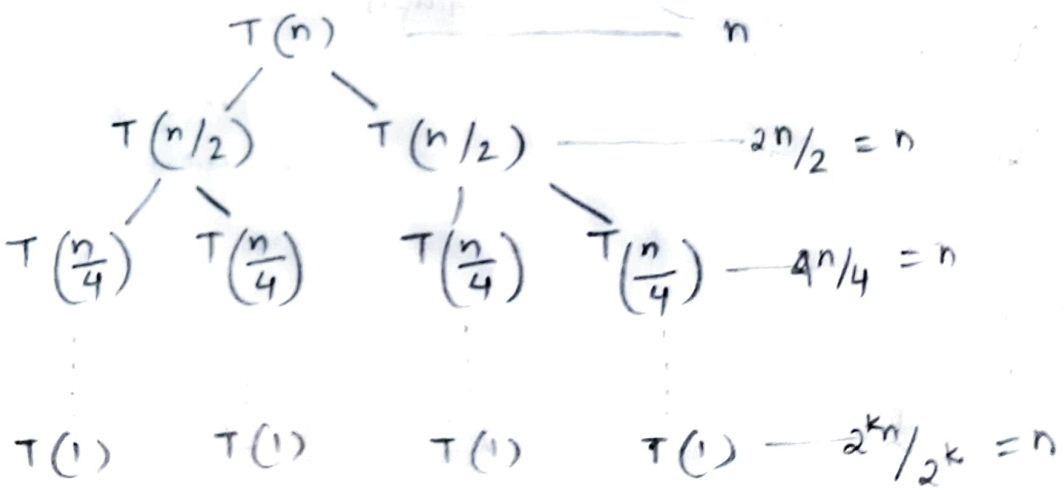
return merge (LA, RA)

★ $T(n) = 2T(n/2) + \Theta(n)$

using recursion tree method

division by 2 ⇒ log n levels

each level, n work done



∴ TC = O(n log n)

levels = log n

at each level, work done = n.

★ Theorem: merge sort is optimal

Thm 2 :

Jean Paul & Trem

Data Structures

how we organize data atoms in the memory efficiently

data atoms ↓ basic unit of storage

We don't specify how the DS is ↑ implemented

★ Abstract Data type (ADT) : blueprint of data structure

insert, delete, lookup, sort → basic operations

★ Types of DS : Linear, Non-Linear

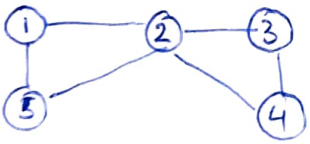
- Linear
  - 1-D array
  - List
  - Tuple
  - data is arranged in a seq manner
- Non-Linear
  - Tree
  - Graph
  - usually there's some kind of grouping

Graph

A collection of vertices and edges. $G = \{V, E\}$ or $G = \{V, E, \ldots\}$

w() assigns some cost

w()



$V = \{1, 2, 3, 4, 5\}$

$E = \{(1,2)\ (2,3)\ (2,4)\ (3,4)\ (3,5)\ (1,5)\}$

Weight function is defined as: $w() : E \rightarrow R$

- undirected graph: no direction associated with the edges
- directed graph: direction associated with edges
  - ↳ each edge will be an ordered pair

$E = \{(1,2)\ (2,3)\ (3,4)\ (4,2)\ (3,5)\ (5,1)\}$