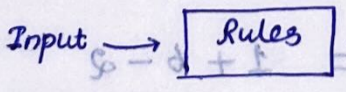
Intro to Algorithms & analysis – 12 weeks – 3 credits

1) Sorting problem, time complexity & asymptotic analysis
2) Solving recurrence, Divide & conquer
3) Quicksort, Heap sort & decision tree.
4) Linear time sorting, order statistics
5) Hash function, BST Sort
6) Randomly build BST, Red black tree, Augmentation of DS
7) Van Emde Boas, Amortized analysis, computational Geometry
8) Dynamic prog., Graphs, prim's algorithms
9) BFS & DFS, Shortest path problem, Dijkstra, Bellman ford
10) All pairs shortest path, Floyd – Warshall, Johnson Algorithm
11) Mosse amortized analysis, set DS (disjoint)
12) Network flow, computational complexity.

Algorithm

A set of rules / instructions – followed to obtain a desired o/p from a given i/p.



Analysis of Algorithms

The theoretical study of compututer program performance & resource usage.

what's important than performance?

- Modularity
- Correctness
- maintainability
- Functionability
- user friendliness
- programmers time
- simplicity
- Extensibility
- Reliability
- Robustness.

Why study algorithms & performance.

* Algorithms help us to understand scalability.
* performance often draws the line b/w what is feasible & impossible
* Algorithmic math provides a language for talking about program behaviour
* Speed.

---

week-0

$44 \rightarrow (101100)_2$

$64 \rightarrow$ Decimal

2	44
2	22 → 0
2	11 → 0
2	5 → 1
2	2 → 1
	1 → 0

ASCII value eg 1 ⟶ 49

Inserting an element in stack → push

Stack – First in Last out

Algorithm – A procedure for solving problem

Complexity of an algorithm – depends upon both size & time

Infix

$1 + 2 \times 3 - 2 = 1 + 6 - 2$

$= 7 - 2 = 5$

$(1\,0\,1\,0\,1\,1\,1\,0\,1\,1)_2 = 1 + 2 + 2^3 + 2^4 + 2^5 + 2^7 + 2^9$

$= 1 + 2 + 8 + 16 + 32 + 128 + 512$

$= 27 + 160 + 512$

$= (699)_{10}$

8	699
8	87 → 3
8	10 → 7
	1 → 2

$(1273)_8$

160
512
27
699

10) Algorithms can't be represented as syntax.

week - 1 - Insertion Sort

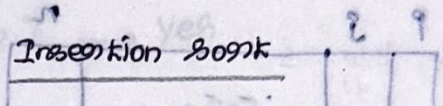
I/p : $a_1, a_2, \ldots a_n$

O/p : a permutation $a_{q_1}, a_{q_2}, \ldots, a_{q_n}$ such that $a_{q_1} \leq a_{q_2} \leq a_{q_3} \ldots \leq a_{q_n}$

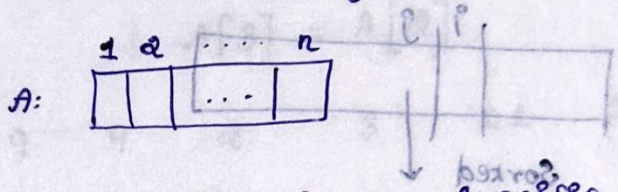
(Ascending order)

eg: 6, 3, 2, 9, 7, 10, 11 → I/P

2, 3, 6, 7, 9, 10, 11 → O/P



* A[1 . . . n] → Array of n numbers



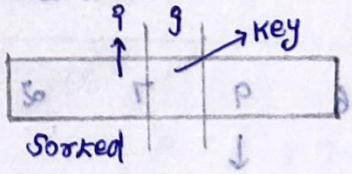
* Not dealing with any specific language.

Pseudocode (English / compact way) → pseudocode.

1) For j (starts with 2) to n

key (starts with A[j])

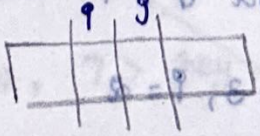
i ← j - 1 (starts with)



* Aspect some time'

After:

key > a[i] → Just increase i by 1



key = A[j]

A[i+1] ← A[i]

i ← i - 1

for $j \leftarrow 2$ to n

Key $\leftarrow A[j]$

$i \leftarrow j-1$

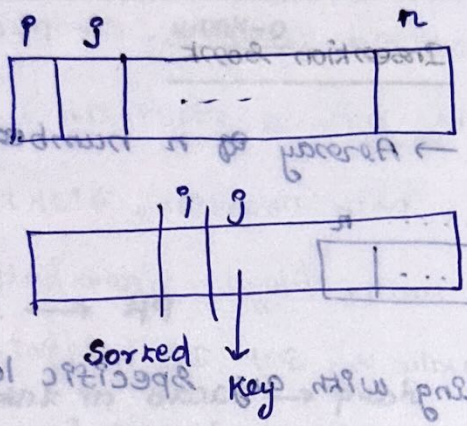
while ((i > 0) and (Key > A[i]))

$A[i+1] \leftarrow A[i]$

$i = i-1$

$A[i+1] \leftarrow$ Key.

eg:



* If Key > A[i] → No need to do anything
* If Key < A[i] → shift i by one position
check the key with this. (Keep on shifting until i>0 & A[i] > Key.

6 9 7 2 3 11 10 1

j

Key=9, j=2, i=1

* while : i>0 and A[i]>9 → No

Key=9, j=3, i=2

* while i>0, 9>9 → NO

Key=9, j=4, i=3

6 9 7 2 3 11 10 1

① $j = 2,\ i = 1,\ key = 9$

while → $i > 0$ and $6 > 9$ → No

$A[i+1] = A[2] = key;$

② 6 9 7 2 3 11 10 1

$j = 3,\ i = 2,\ key = 7$

while $i > 0$ and $9 > 7$ → yes

$A[i+1] = A[i]$

$A[3] = A[2]$

6 9 9 2 3 11 10 1

$i = 1$

while $i > 0$ and $6 > 9$ → No

$A[i+1] = key$

$A[2] = 7$

6 7 9 2 3 11 10 1

③ $j = 4,\ i = 3,\ key = 2$

while $i > 0,\ 9 > key$

6 7 9 9 3 11 10 1

while $2 > 0,\ 7 > key$

6 7 7 9 3 11 10 1

while $1 > 0$ $6 > key$

$i = 0$

6 6 7 9 3 11 10 1

$A[i+1] = key$

2 6 7 9 3 11 10 1

'continues'

① 5 4 3 2

↓ ↓

i j

Key = 4

while $1 > 0$, $5 > 4$

5 5 3 2

'fail' ⟶ 4 5 3 2

② $j = 3$, $i = 2$, Key = 3

while $i > 0$, $5 > 3$

4 5 5 2

while $i > 0$, $4 > 3$

4 4 5 2

'fail' ⟶ 3 4 5 2

③ $j = 4$, $i = 3$, Key = 2

while $i > 0$, $5 > 2$

3 4 5 5

while $i > 0$, $4 > 2$

3 4 4 5

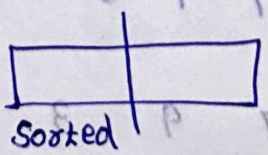
while $i > 0$, $3 > 2$

3 3 4 5

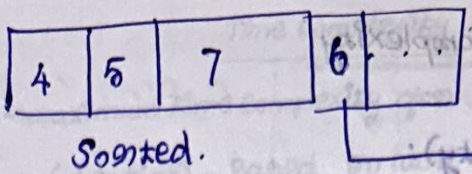
'fail' ⟶ 2 3 4 5.

Idea

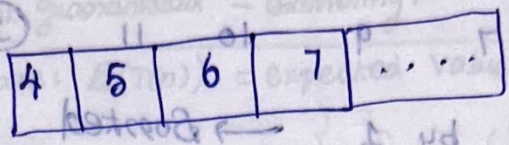
'Key - we are going to add it in corresp. position'



(At any position)

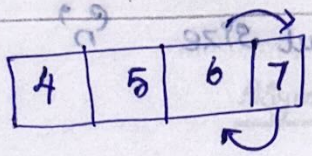


① "where to add"

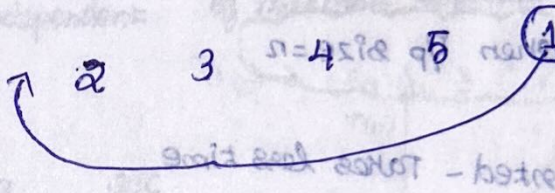


* ∴ First compare to 7 – if 7 is less → so in sorted order
* If 7 is greater, 7 need to move right.
(5 is greater, 5 need to move right.)

↳ Else 6 need to (key) be after 5)



eg:



1 2 3 4 5.

* '2, 3, 4, 5' need to shift right by 1.
* 1st place need to be occupied by 1.

Each main loop

→		6	5	4	3	2	1
① →	5	6	4	3	2	1
② →	4	5	6	3	2	1
③ →	3	4	5	6	2	1
④ →	2	3	4	5	6	1
⑤ →	1	2	3	4	5	6

Complexity

Runtime: (time complexity):

2 3 6 7 9 11 (10) 1

2 3 6 7 . . . 9 10 11 (1)

'10' only moves by 1 → Sorted

1 2 3 6 7 9 10 11

'Here 1 moves a long way to index 1' → Reverse Sorted

Runtime

'Depends on * Input size 'n'

*

T(n) = Runtime when i/p size = n

Sorted - Takes less time (See above cases)

Reverse sorted - Takes more time.

4 6 7 9 10 12

'Already sorted'

Best case

10 9 6 5 3 2 1

'more time'

'Everybody need to come forward'

worst case.

'Average case' - Expected pattern

Time Complexity

* worst case T(n) – maximum time complexity of any n sized I/P (reverse sort)
* Best case Bocus (cheating) – Based on luck (I/P is already sorted)
  (Just – one comparing occurs).
  ↳ "Not guaranteed" – Gambling.

Average case: $E(T(n))$ = expected value of T(n) – Average.

worst case:

n = 100; (Reverse sorted way)! [Speed of PC]

We can't Judge: ∴ Same laptop, same situation, environment – Fair speed, other programs not running – "Not possible"

↓

Need a better method!

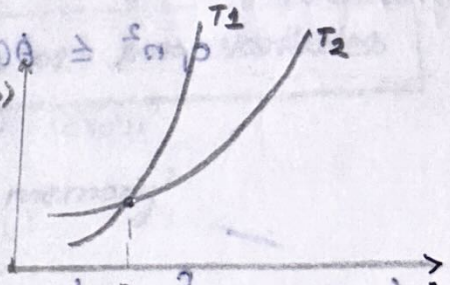
Asymptotic notation.

* Independent of machine dependency.

Let,

$T_1(n) = 2n^3 + 5n + 3$ sec

$T_2(n) = 100n^2 + 50n + 9$ sec



when n is small, T_1 better

n is large, T_2 better

"Ignore low orders & coeff"

$T_1(n) = \theta(n^3)$

$T_2(n) = \theta(n^2)$

As $n \to \infty$ (we get rid of coeff & other low order terms)

"Ignoring architecture, adders etc"... (Not possible to give Ideal situation)

Asymptotic notation

↙ Big O | ↓ Big omega | ↘ Big theta

Definition:

$\Theta(f(n)) \rightarrow$ Set of all function $\left\{ g(n) \;\middle|\; c_1 f(n) \leq g(n) \leq c_2 f(n) \;\forall\; n \geq n_0 \right\}$

c_1, c_2 two +ve Constants

$$\Theta(n^2) = \left\{ g(n) \;\middle|\; c_1 n^2 \leq g(n) \; c_2 n^2 \right\}$$

$10n^2 + 5 \in \Theta(n^2)$ — Just Ignore

$10n^2 - 5n + 3 \in \Theta(n^2)$

Asymptotic Notation

$T(n) = 5n^2 - 10n + 2 = \Theta(n^2)$

$c_1 n^2 \leq \Theta(n^2) \leq c_2 n^2$

Big O

$$O(f(n)) = \left\{ g(n) \;\middle|\; g(n) \leq c\, f(n) \;\forall\; n \geq n_0 \right\}$$

"upper bound"

c — +ve Constant

$$O(n^2) = \left\{ g(n) \;\middle|\; g(n) \leq c\, n^2 \right\}$$

$10n^2 + 5n + 2 \leq c\, n^2$ → upper bound

$$\Omega(f(n)) = \left\{ g(n) \;\middle|\; g(n) \geq c f(n) \;\forall\; n \geq n_0 \right\}$$

$$\Omega(n^2) = \left\{ g(n) \;\middle|\; g(n) \geq c n^2 \right\}$$

$\theta(f(n)) = \{ g(n) \mid c_1 f(n) \leq g(n) \leq c_2 f(n) \}$

$\Omega(f(n))$ $\qquad$ $O(f(n))$

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

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

"we usually go for upper bound - ≥ 5 → not tells much info.

Less than 10 sec → we can have much more info.

"Big O notation"

Analysis of - Insertion Sort

Space complexity: Here - we are not creating any extra arrays

"In place sort".

Key, i, j → one or two variables

"we are not taking any extra memory"

---

worst case T(n) analysis

* "I/p is reverse sorted"

1) for $j \leftarrow 2$ to n
2) $\quad$ key $\leftarrow A[j]$ $\qquad$ → $\theta(1)$
$\quad$ $i \leftarrow j - 1$ $\qquad$ → $\theta(1)$
3) $\quad$ while $(i > 0)$ and $(key < A[i])$
4) $\quad\quad$ $A[i+1] \leftarrow A[i]$ $\qquad$ → $\theta(1)$
5) $\quad\quad$ $i \leftarrow i - 1$ $\qquad$ → $\theta(1)$
6)
7) $\quad$ $A[i+1] \leftarrow key$. $\qquad$ → $\theta(1)$

$T(n) = \sum_{j=2}^{n} \theta(j)$

↳ j times $\theta(1)$

↓

For loop.

* Long times (Reverse Sort)
* Everytime need to reach beginning

$= \sum_{j=2}^{n} j\,\theta(1) = 1+2+\dots+n$

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

Why?

4 elements

1 loop → 3 steps
2 loop → 2 steps
3 loop → 1 step

$3+2+1 = \frac{n(n+1)}{2}$

'worst case run time: $\theta(n^2)$'

↓

'we don't want exact time' – Asymptotic Analysis

Best case.

Sorted – Ascending order

3 4 6 7 9 10 11

'1 comparison each time'

$\sum_{j=2}^{n} \theta(1) = n$

$= \theta(n)$

'Best case'

"Average case"

"Randomized input"

Assume: $\sum_{}^{n} \theta\left(\frac{j}{2}\right)$ → half way (Intuitive analysis)

Best case: Just one compare

worst case: All the way to the beginning.

For accurate: distributive analyze & take E(x)

$= n^2/2$

$\approx \theta(n^2)$

Best case: $\theta(n)$

Worst case: $\theta(n^2)$

Average case: $\theta(n^2)$

merge sort

"worst case = better than best case"

"Divide & conquer"

1) $n = 1$ → done

2) Recursively sort two subarrays $A\left[1 \ldots \left(\frac{n}{2}\right)\right]$ and $A\left[\left(\frac{n}{2}\right)+1, \ldots n\right]$

3) merge

Recurrence for merge sort

* Split subarray - recursively

* "Extra array needed" — For merge — To be linear.

Merge → $\theta(n)$

"we can't make sorted array in linear time without extra array"

↓

Lot of swapping.

'Not a Inplace Sorting algorithm'

* $\theta(1) \rightarrow n = 1$
* else sort $A[1 \ldots \frac{n}{2}]$ and $A[\frac{n}{2}+1 \ldots n] \rightarrow 2T(\frac{n}{2})$ (2 calls of $\frac{n}{2}$)
* $\theta(n) \rightarrow$ merge

$$T(n) = \theta(1) + 2T(\tfrac{n}{2}) + \theta(n)$$

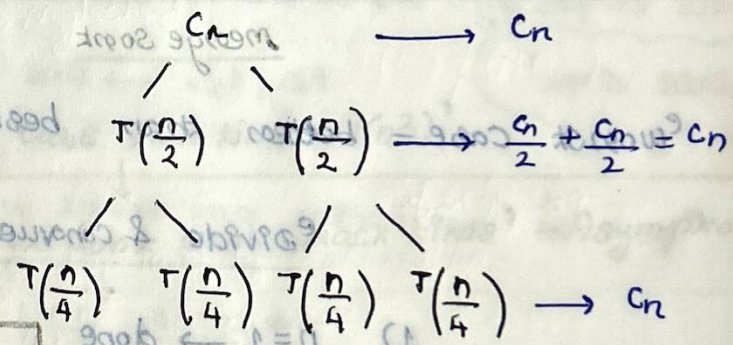
$$\boxed{T(n) = 2T(\tfrac{n}{2}) + \theta(n)}$$

Recurrence

Solve recurrence

Recursive tree method. → Recurrence tree

$T(n) =$



when $T = 1 \rightarrow$ no

$T = 2 \rightarrow 2$

$T = 4 \rightarrow$ 3 stages

unit size as 1

$T(1)$ → base case

$\theta(1)$

Height = height × c_n

$\frac{n}{2}, \frac{n}{2^2}, \frac{n}{2^3}, \frac{n}{2^4}$

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

$\frac{n}{2^h}$

$2^h = n$

$$\boxed{h = \log_2(n)}$$

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

Recursive method – Intuitive ('ith level sum = cn)

what's the guarantee?'

Substitution method (Induction method)

↓

Proof for Recursive tree method.

Best case of merge sort

what even i/p → divide & merge (didn't bother about i/p)

For any i/p ⇒ $T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n)$

$= \Theta(n \log n)$ → Best

↓ worst

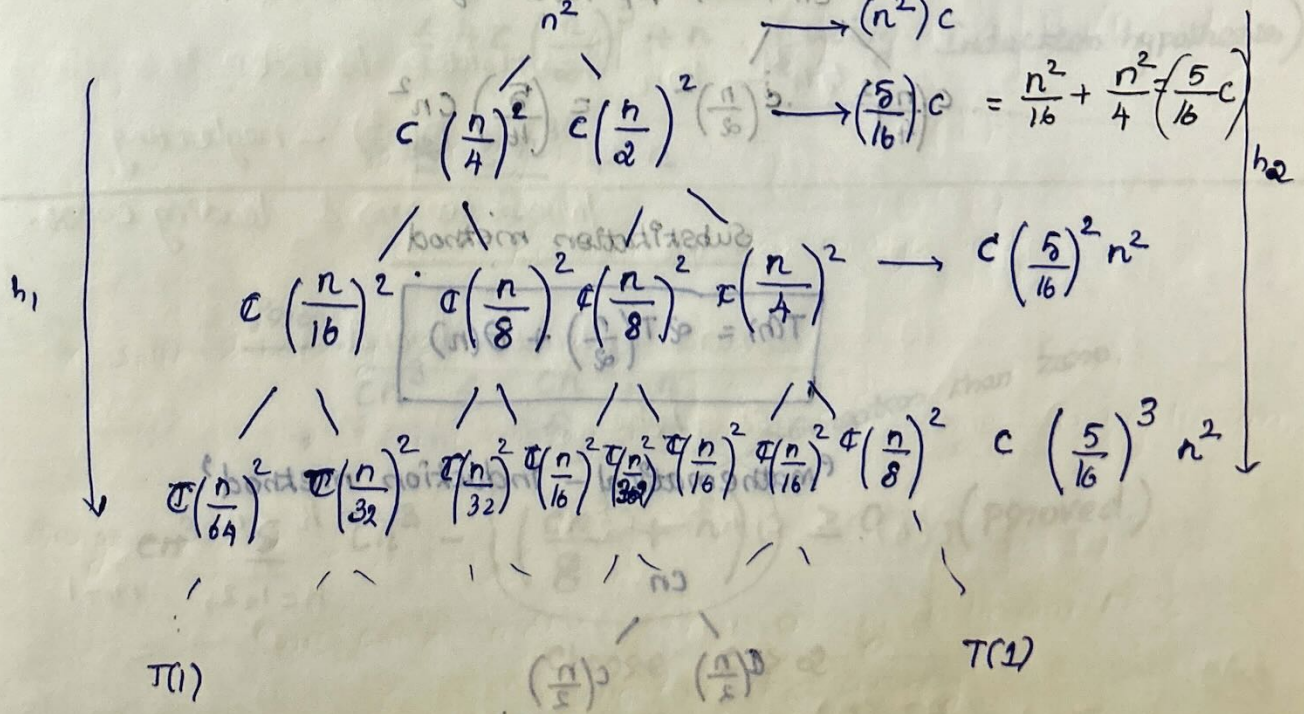
↓ Average

Doesn't bother about any i/p'

Adv with Insertion sort: In place sort

Merge sort: Not in place.

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



* $h_1 \rightarrow$ ending easily $\rightarrow$ up to h_1 (complete binary tree)

* up to $h_2 \rightarrow$ (Not Complete binary tree)

$$\boxed{h_1 = ?}$$

$$\frac{n}{4^{h_1}} = 1 \qquad \Big| \qquad \frac{n}{2^{h_2}} = 1$$

$$h_1 = \log_4 n \qquad \Big| \qquad h_2 = \log_2 (n)$$

$$= c n^2 \left[1 + \frac{5}{16} + \left(\frac{5}{16}\right)^2 + \ldots\right]$$

$$x < 1$$

$$= c n^2 \left(\frac{1}{1-x}\right)$$

$$= c n^2 \left(\frac{1}{1-\frac{5}{16}}\right) \rightarrow \text{Neglected.}$$

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

Ref: Pg: 89.

$cn^2 \rightarrow$ Cost at the top level of recursion.

c_2 arrays' $\rightarrow cn^2$

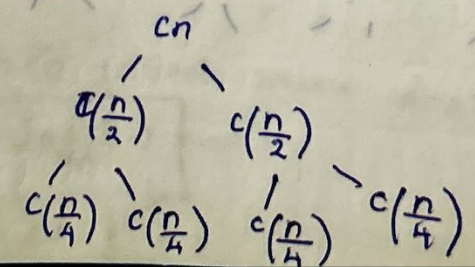
$cn^2 \rightarrow$ Top level of recursion.

$$c\left(\frac{n}{4}\right)^2 \qquad c\left(\frac{n}{2}\right)^2 \qquad \left(\frac{5}{16}\right) cn^2$$

Substitution method

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

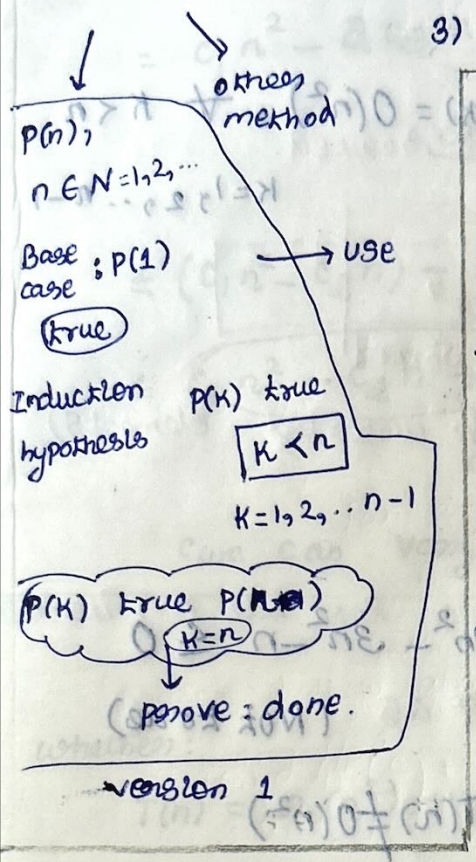
Mathematical - Induction method'



More specific analysis:

1) Guess the solution
2) Verify the Guess (method of Induction)
3) Proved.

Mathematical Induction



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

"By substitution method"

Solu:

Guess: $T(n) = O(n^3)$

Assume (Induction hypothesis)

$T(K) = O(K^3)$ $(\forall\ K = 1, 2, .. n-1)$

$\therefore T(K) = cK^3 \quad \forall\ K < n$

Proof: $T(n) \leq cn^3$

$T(n) = O(n^3) \rightarrow$ proved.

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

$$\leq 4c\left(\frac{n}{2}\right)^3 + n \quad \text{(using Induction hypothesis)}$$

$$\leq \frac{cn^3}{2} + n$$

Proof:

$$cn^3 \leq \frac{cn^3}{8} + n$$

$$cn^3 \geq cn^3 - \left(\frac{cn^3}{8} + n\right) \geq 0 \quad \text{(proved)}$$

($\frac{cn^3}{8} + n$ → Greater than zero.)

Choose $c > 2$

$$\frac{8n^3}{8} - \frac{n^3}{8} - n \geq 0$$

$$n^3 - n \geq 0 \longrightarrow \text{True}$$

$T(n) = O(n^3)$

Guess: $T(n) = O(n^2)$

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

$\leq 4c\left(\frac{n}{2}\right)^2 + n$

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

$T(k) = O(n^2), \forall\ k < n$

$k = 1, 2, \ldots n-1$

Now

$cn^2 - \left(4c\left(\frac{n}{2}\right)^2 + n\right) \geq 0$

$cn^2 - \left(\frac{4cn^2}{4}\right) - n \geq 0$

$cn^2 - n \geq 0$

$2n^2 - 3n^2 - n \leq 0$

(Not true)

$\frac{4n^2}{2} - \frac{n^2}{2} - n \geq 0$

$\frac{3n^2}{2} - n \geq 0$ → True.

$T(n) \neq O(n^2)$

failed

$T(k) \leq ck^2$ → we can't achieve

we had tighten bound — Not able to achieve.

$5n^2 - 10n$ → tighten bound

$O(n^2)$ — neglecting lower terms & leading coeff.

Instead of tighten bound

let take

$T(k) \leq c_1 k^2 - c_2 k$ $\forall\ k \leq n$

$k = 1, 2, \ldots n-1$

Solu:

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

$\leq 4(c_1 k^2 - c_2 k) + n$

$\leq 4\left(c_1\left(\frac{n}{2}\right)^2 - c_2\left(\frac{n}{2}\right)\right) + n$ [using I.H]

$= c_1 n^2 - 2 c_2 n + n$

Show: $\boxed{T(n) \leq c_1 n^2 - c_2 n}$

$\leq (c_1 n^2 - c_2 n) - (c_1 n^2 - 2c_2 n + n)$

$\leq c_1 n^2 - c_2 n + n$ $\quad c_2 > 1 \mid c_1 = 0$

greater than 0 (True)

we can vary c_1, c_2 → verify whether we achieve!

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

whether:

$T(n) = \Omega(n^2)$ → ? (Both achieved)

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

In recursive tree - Approximated - Used θ

* Lower bound O
* upper bound Ω

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

$T(k) = \Omega(k^2)$

$T(k) \geq ck^2$

$T(n) \geq cn^2$ → To prove

Assignment 1

Increasing order asymptotic complexity of functions.

$f_1 = n^2$, $f_2 = n!$, $f_3 = n \log n$, $f_4 = 2^n$.

Solu: Let $n = 20$

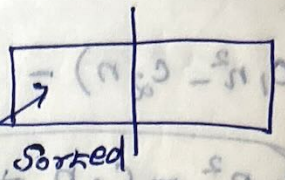
$f_1 = 400$, $f_2 = 2.4 \times 10^{18}$, $f_3 = 26$, $f_4 = 1048576$

$f_3 < f_1 < f_4 < f_2$

2) Best case I/P, the running time of an Insertion sort: Linear. (shift by 1)

3) Worst case of Insertion sort: Reverse sorted.

4) FALSE statement:



True → In Insertion sort (after m steps)

False → m smallest elements in the array

(∵ The smallest element may be in unsorted elements)

True → Insertion sort - Stable, In place.

True → on an array size of n, Insertion sort has n-1 iterations

---

{6, 4, 8, 1, 3} → Each iteration ₹ 25

(n-1) iterations = 4 × 25 = 100.

when 1 reaches first position:

when key = 1' → step 3 (iteration 3)

3 × 25 = ₹ 75.

27, (19), 33, 15, 4

(3 rd step iteration)

1 →	19	27	(33)	15	4
2 →	19	27	33	(15)	4
3 →	15	19	27	33	(4)
4 →	4	15	19	27	33

W(n) → Worst case, A(n) → Average case.

(From n I/P s)

$A(n) = O(W(n))$

$\Omega \rightarrow \geq$

$O \rightarrow \leq$

$\Theta \rightarrow \leq - \leq$

'Best' - with in limit (Both)

'Avg' - we know upper bound.

9) Worst case time complexity of merge sort

$$W(n) = O(n \log n)$$

10) Two sorted lists → m, n (size); No. of comparisons needed in worst case

$$m+n-1.$$

8) Compared to Insertion sort, merge sort will take least time when all i/p is identical.

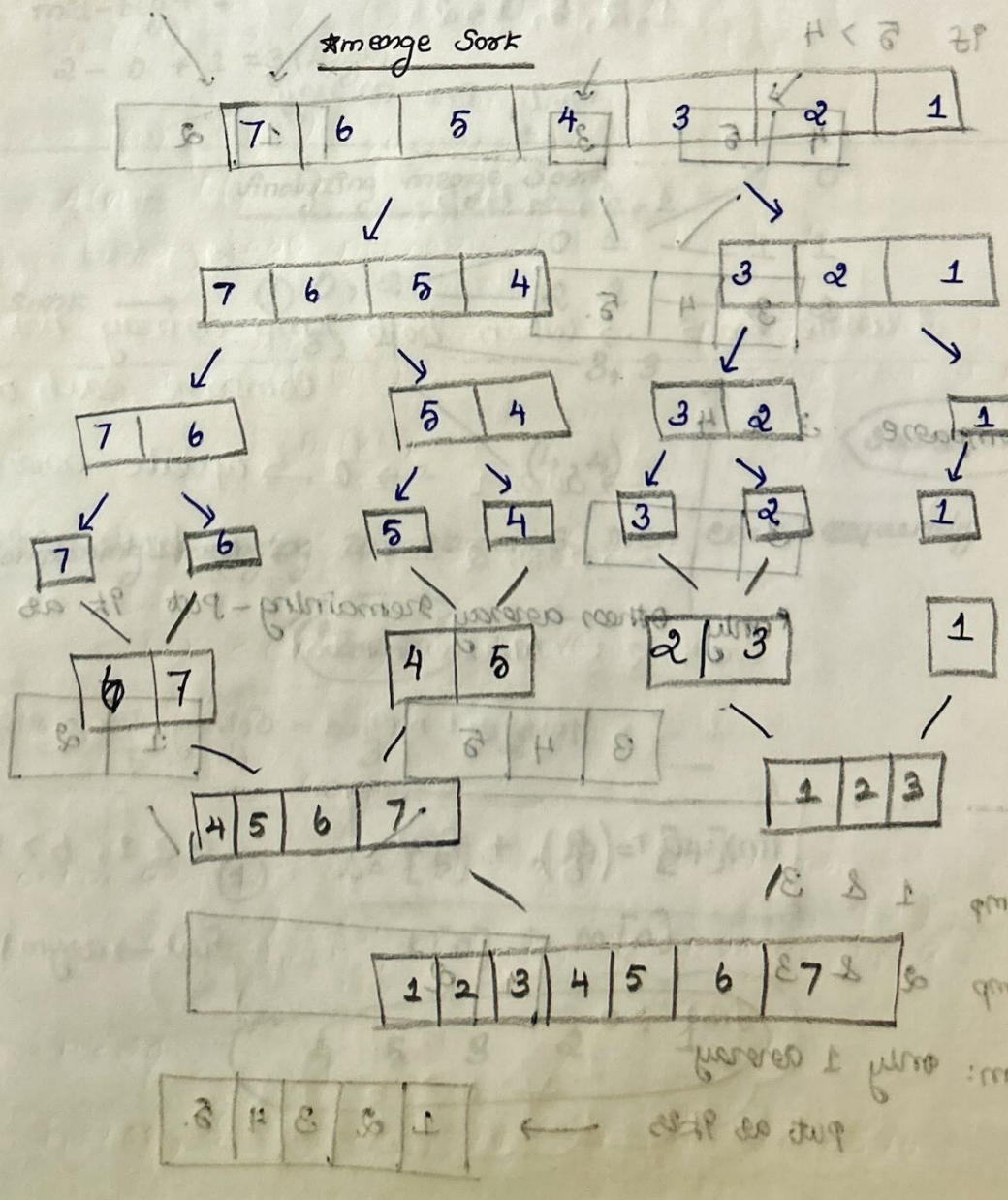
"No" – False.

∴ Merge – Same $n \log n$

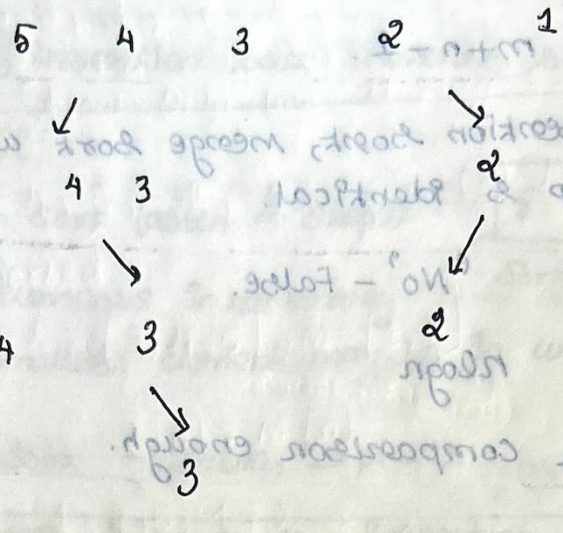
Insertion – $n-1$ comparison enough.

Master method.

* Another powerful method to solve recurrence.

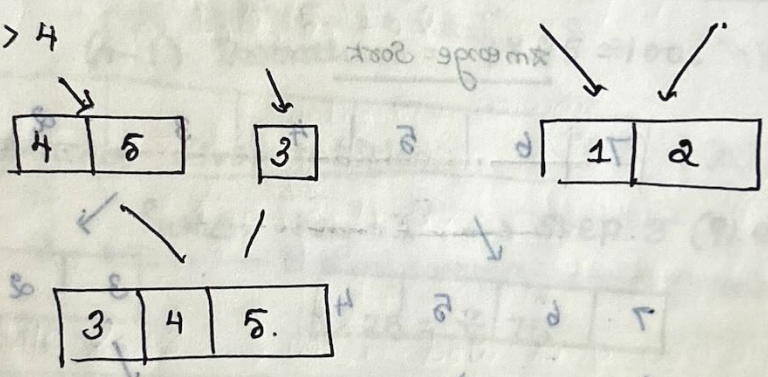


Dig deeper:

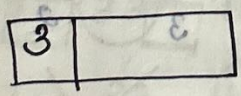


① $l = 5, m = 4$ (index 0) → beg

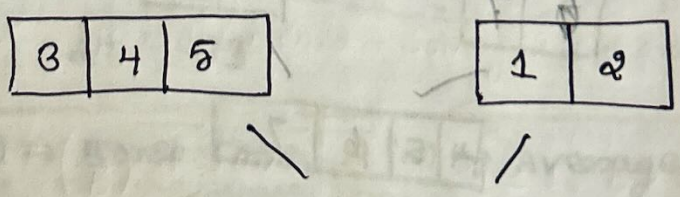
If $5 > 4$



Compare 3 & 4

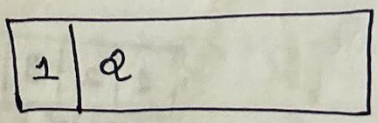


Only other array remaining – put it as it is

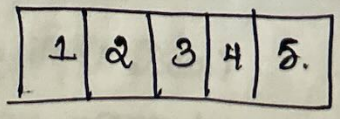


Comp 1 & 3

Comp 2 & 3



Rem: only 1 array

put as it is → 

'split - Not real' → Not going to touch the given array.

2 arrays : continuous (from the diagram clear).

0 to 2 (3 elements) 2 → 1 array, 1 → 2nd array.

(0 to 1) , (2 to 2)

why split → go give the index (make copy, change (sort) that position)

merge_sort → split (cut array in to two) — Imaginary

merge → merge (replace) the two temp array with original array.

1 2 3 4 5

beg mid

mid - beg + 1

2 - 0 + 1 = 3 (length

4 - 2 = 2. (length = 2)

Analyzing merge sort

merge_sort → ① 0, 2 → 0 1 → 0 0 / 1 1

2 2 → 2 2

3, 4 → 3, 3 → (4, 4)

start, end.

we are comparing - merging two elements, then consequently 'index'

5 4 3 2 1

merge ① L = [5] , M = [4]

L[0] > M[0]

4 5 3 2 1

② L = [4, 5], M = [3]

L[0] > M[0]

3, 5, 3, 2 1

'Now only one array' — Just update

3, 4, 5, 2, 1

sorted

③ L = [2], M = [1]

3, 4, 5, 1, 2

That index alone changed

④ L = [3, 4, 5], M = [1, 2]

L[0] > M[0]

1, 4, 5, 1, 2

L[0] > M[1]

1, 2, 5, 1, 2

'only one array'

1, 2, 3, 4, 5.

---

3 while loops → when both temp array not empty

Compare each other

→ arr 1 → 0 → update arr 2

→ arr 2 → 0 → update arr 1

---

week : 2 : Masters method

'powerful method – Solving recurrence'

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

$a \geq 1,\ b > 1$

$f(n)$ → asymptotically +ve.

$f(n) > 0 \quad \forall\ n \geq n_0$

$T(n) = a\,T\left(\frac{n}{b}\right) + f(n)$ → This form can be solved using this

"Think divide & conquer" → Recurrence.

* $f(n)$ → Cost for combining subproblems.

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

↓ 2 subarrays

↳ merge part

$a = 2, b = 2$

$\theta(n) = f(n)$

("merge sort")

Compare:

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

$a\,T\left(\frac{n}{b}\right)$ → Time for solving the problem recursively | $f(n)$ - merging the solution.

Runtime = dominating one

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

↓ merging ↓ Divide 'n' to 'n' sub problems.

Cases:

$A > B$ → $f(n)$

$A = B$

$A < B$ → $n^{\log_b a}$

Case 1: $f(n)$ is polynomially slower than $n^{\log_b a}$

$f(n) << \frac{n^{\log_b a}}{n^{\varepsilon}}$, $\varepsilon > 0$

$= c\,n^{\log_b a - \varepsilon} \leq$

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

Solution: $\theta(n^{\log_b a})$ → Dominating term.

Case 2: Similar

$f(n)$ and $n^{\log_b(a)}$ similar growth rate.

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

$$c_1 n^{\log_b a} \leq f(n) \leq c_2 n^{\log_b a}$$

They could be differ in log factors

$n = 2^{10}$

$\log_2 n = 10$

compared to n, $\log_2(n)$ → small.

multiply by $(\log_2 n)^k$

$$c_1 n^{(\log_b a)} (\log_2 n)^k \leq f(n) (\log_2 n)^k \leq c_2 n^{(\log_b a)} (\log_2 n)^k$$

$$\therefore f(n) = \theta\left(n^{\log_b a} (\log_2 n)^{k+1}\right), \quad k = 0, 1, 2, \cdots$$

when $k = 0$ → Exactly similar

$k = 1$ → Log factors difference

Solution: $T(n) = \theta\left(n^{\log_b a} (\log_2 n)^{k+1}\right)$

Adding (mul) $(\log_2 n)^k$ → For logarithmic difference

Case 3: $f(n)$ is polynomically faster

$f(n) \geq n^{\log_b a} \times n^{\varepsilon}$ (even after multiplying with a polynomial n^{ε}) $f(n)$ is faster.

meaning: Big omega

$$f(n) = \Omega\left(n^{\log_b a + \varepsilon}\right) \qquad \varepsilon > 0$$

Another Condition: Regularity condition

$T(n) =$ $f(n)$

$f(\frac{n}{b})$ $f(\frac{n}{b})$ . . $f(\frac{n}{b})$ $= af(\frac{n}{b})$

Total cost = Sum of each stage cost

we want $f(n)$ should be bigger. (Bigger than next level)

$$\boxed{af(\frac{n}{b}) < cf(n)} \quad 0 < c < 1$$

we want $f(n)$ should be bigger (even than its fraction)

'Regularity condition' where $0 < c < 1$.

Solution $\boxed{T(n) = \theta(f(n))}$

$$T(n) = 4T(\frac{n}{2}) + n \qquad \left(aT(\frac{n}{b}) + f(n)\right)$$

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

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

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

$$= n^2$$

n^2 grows faster than n $\boxed{f(n) = O\left(n^{\log_b a - \varepsilon}\right)}$

$\boxed{\varepsilon = 1}$

$\boxed{T(n) = \text{Solution: } \theta(n^2)}$

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

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

$\boxed{f(n) = n^2}$ → Exactly similar

'No log factor' → $k = 0$

$$T(n) = \Theta\left(n^{\log_b a} (\log_2 n)^{k+1}\right)$$

$$T(n) = 4T\left(\frac{n}{2}\right) + n^2 \log_2 n$$

log factor: $k = 1$

$$T(n) = \Theta\left(n^{\log_b a} (\log_2 n)^{k+1}\right)$$

$$= \Theta\left(n^2 (\log_2 n)^2\right)$$

Merge sort recurrence

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

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

Case: $a, k = 0$

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

$$T(n) = 4T\left(\frac{n}{2}\right) + n^3, \quad a = 4, b = 2$$

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

$$f(n) = n^3$$

$\Theta(n^3)$ → It satisfies regularity condition

$k = 3$

$$f(n) = \Omega\left(n^{\log_b a + \varepsilon}\right)$$

Here $\varepsilon = 1 \quad (\because n^3)$

$$af\left(\frac{n}{b}\right) < c\, f(n)$$

$$4f\left(\frac{n}{2}\right) < c\, n^3$$

$4\left(\frac{n}{2}\right)^3 < cn^3 \longrightarrow \frac{n^3}{2} < cn^3$

$\frac{4n^3}{8} < cn^3$ when $\boxed{c > 0.5}$

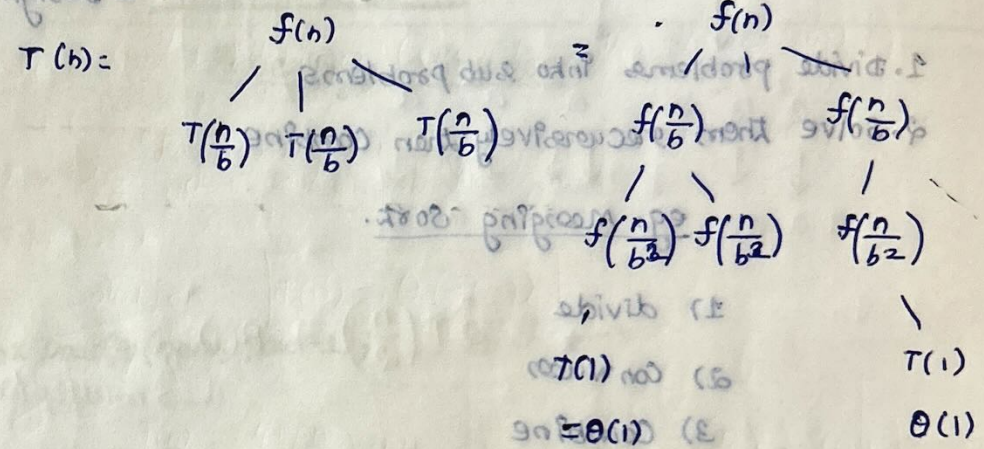
$c \in \left(\frac{1}{2}, 1\right)$

∴ Satisfied

Solution: $T(n) = \theta(f(n))$

$= \theta(n^3)$

$T(n) = aT\left(\frac{n}{b}\right) + f(n)$ Proof by masters method

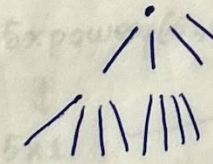


$\frac{n}{b^h} = 1$

$b^h = n$

$\boxed{h = \log_b n}$

$\boxed{\text{No. of Leaves} = a^h = a^{\log_b n}}$



$a = 16$ (each 4)

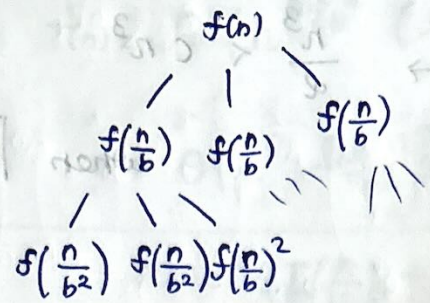
$a = 64$

up to height.

when height = 2, $4^2 = 16$ (branches)

Leaves.

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



$\longrightarrow$ Sum = $a\, f\left(\frac{n}{b}\right)$

$\longrightarrow$ Sum = $a^2 f\left(\frac{n}{b^2}\right)$

Ref: Pg: 56 (Log formulae)

$n^{\log_b a}\, f\left(\frac{n}{b^n}\right)$

Divide & Conquer – Design paradigm

1. Divide problems into sub problems
2. Solve them recursively then combine

eg: Merging sort.

1) divide
2) Conquer
3) Combine

Subproblems = ? (Size of each subproblem $= \frac{n}{2}$) → Merge Sort.

recursively.

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

↳ Divide & Conquer

quicksort: divide is more expensive than combine.

↳ Two subarrays

$\theta(n)$.

- 'divide' → $\theta(1)$ Just assignments & Calls.
- 'merge' → $\theta(n)$ [Replace elements] n steps maximum.

Binary Search.

'Recursion'