- CLRS
- Kleinberg

Algorithm :- set of well defined instructions to solve a computational problem.

Ex:- Bunch of nos - Sorting
Road map - from origin to destinat^{n}.
Scheduling
Routing Protocols in communication networks - shortest path algorithm
Public key crypto - . . .
Computer Graphics (CG) - Geometric Algo
Database indices - Balanced Search Tree
Computational biology - Dynamic Prog
Google - Page ranking algo

Resources - Time, space, random bits
(trade off)

- lower bound of all comparision based Algo - nlogn

→ Linear time Sorting Algorithm — Counting
— Radix

→ Performance
- Correctness
- Scalability
- User friendliness
- Robustness?

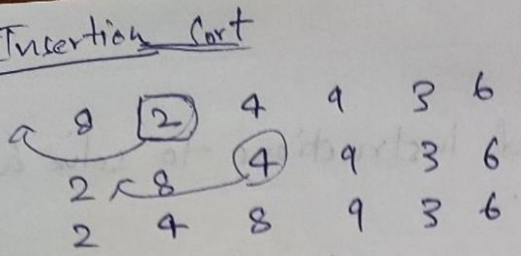
Problem of Sorting

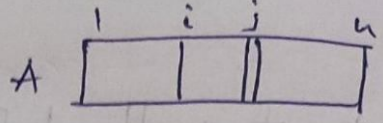
i/p Sequence of nos $\langle a_1, a_2, \ldots, a_n \rangle$

o/p Permutations of the seq $\langle a_1', a_2', \ldots, a_n' \rangle$

02/20/2023 11:09

Insertion Sort






```
Ins_sort (A, n)
  for j ← 2 to n
    do key ← A[j]
       i ← j-1
       while i>0 and A[i] >key
         do
            A[i+1] ← A[i]
            i ← i-1
       A[i+1] ← key
```

Correctness :-

Loop invariant :- A[1 . . . j-1] ~~are the~~ original elements and sorted.

The loop invariance in inner loop is not considered here

Running time :-

Worst - case

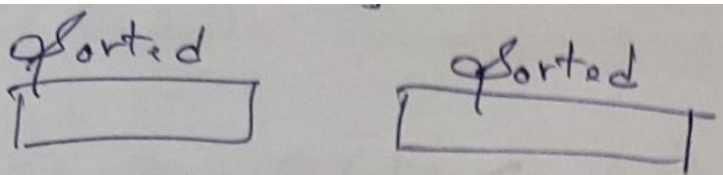
T(n) = max time algo on any i/p of size n

Avg case :- ~~The~~ In avg case, for "while loop" above, we take half of them are greater than key & half are less i.e every A[j] goes to $A[\frac{j}{2}]$

Best case :- X

~~In place algo??~~

We use RAM model



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

tion Sort – In place

e)

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

and Conquer :-

-Merge-Sort $O(n \log n)$

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

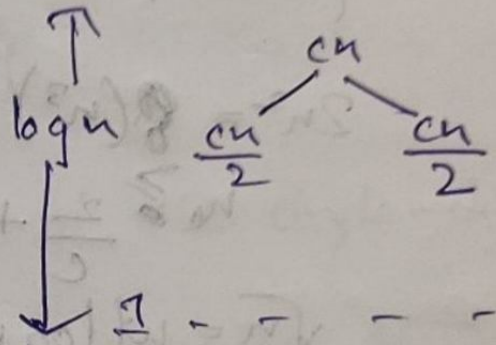
totic Notation :-

notation (upper bound)

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

$\exists n_0 \Rightarrow 0 \le f(n) \le cg(n) \quad \forall n \ge n_0$

dependent
of "n"



This is recursion tree method — y have to guess it later. This is not proof technique.

$2n^2 \in O(n^3)$

$c = 1, n_0 = 2$

$\rightarrow f(n) = n^3 + O(n^2)$

$f(n) = n^3 + h(n)$ for some $h(n) \in O(n^2)$

$\rightarrow n^2 + O(n) = O(n^2)$ means

$\nexists f(n) \in O(n)$

$\exists g(n) \in O(n^2) \Rightarrow n^2 + f(n) = g(n)$

Why not $\forall g(n) \in O(n^2)$

Ω notation (lower)

$\Omega(g(n)) = \{ f(n) \mid \exists\, c, n_0 > 0 \Rightarrow 0 \le cg(n) \le f(n) \ \forall\ n \ge n_0 \}$

Ex:- $n = \Omega(\log n)$

Θ-notation (Tighter)

$\Theta(g(n)) = O(g(n)) \cap \Omega(g(n))$

Ex:- $\frac{1}{2}n^2 - 2n = \Theta(n^2)$

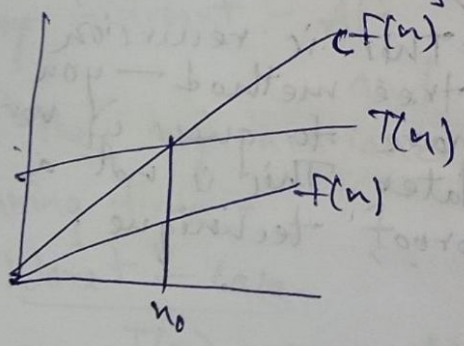
o-notation and ω-not

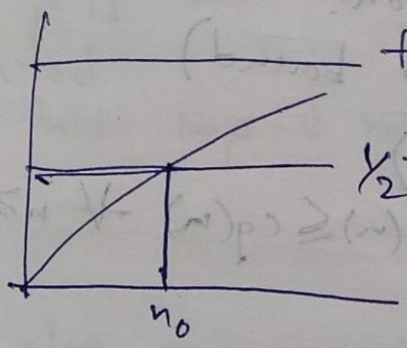
$o(g(n)) = \{ f(n) \mid \forall\, c > 0, \exists\, n_0 > 0 \Rightarrow 0 \le f(n) < cg(n) \ \forall\ n \ge n_0 \}$

$2n^2 = o(n^3)$

$n_0 \Rightarrow \frac{2}{c} + 1$

$\sqrt{n} = \omega(\log n)$





1) $T(n) = a_k n^k + a_{k-1} n^{k-1} + \dots + a_1 n + a_0$ then

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

Pf:- $c = |a_k| + |a_{k-1}| + \dots + |a_0|$

$n_0 = 1$

2) P.T $\forall\ k \ge 1$, n^k is not $O(n^{k-1})$

$n^k \le cn^{k-1}$

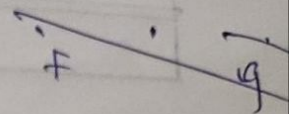
02/20/2023 11:10

3) $2^{n+10} = O(2^n)$

$c = 2^{10}$, n_0 is anything

4) $2^{10} \neq O(2^n)$

5) $\max\{f, g\} = \Theta(f(n) + g(n))$



Pf: $\text{avg} \leq \max \leq \text{sum}$

Integer multiplication

i/p: 2 n-digit nos x & y

o/p :- x * y

primitive operation: add/multiplication of 2 single-digit nos

Grade-school : $O(n^2)$

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

$x = a\,10^{n/2} + b$

$y = c\,10^{n/2} + d$

$xy = 10^n(ac) + 10^{n/2}(ad + bc) + bd$

$4 > 2^1$

$T(n) = O(n^2) = O(b\,n^{\log_2 4})$ $\quad a > b^d$

$n^{\log_b a}$

$n^{\log_2 4}$

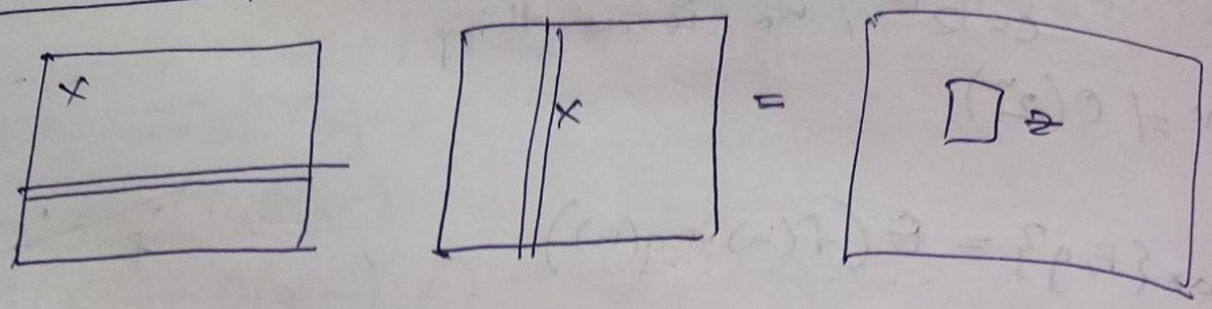
Karatsuba Multiplication

1. recursively compute ac
2. " bd
3. " $(a+c)(b+d) = ac + bd + ad + bc$

$(3) - (1) - (2) = ad + bc$

Gauss $O(n^{\log_2 3})$

Matrix Multiplication



$O(n^3)$

$$X = \begin{pmatrix} A & B \\ C & D \end{pmatrix} \quad Y = \begin{pmatrix} E & F \\ G & H \end{pmatrix}$$

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

$$O(n^{\log_2 8}) = O(n^3)$$

Strassen's algo

$$X = \begin{pmatrix} A & B \\ C & D \end{pmatrix} \quad Y = \begin{pmatrix} E & F \\ G & H \end{pmatrix}$$

$$P_1 = A(F - H)$$
$$P_2 = (A + B)H$$
$$P_3 = (C + D)E$$
$$P_4 = D(G - E)$$
$$P_5 = (A + D)(E + H)$$
$$P_6 = (B - D)(G + H)$$
$$P_7 = (A - C)(E + F)$$

$$X \cdot Y = \begin{pmatrix} P_5 + P_4 - P_2 - P_6 & P_1 + P_2 \\ P_3 + P_4 & P_1 + P_5 - P_3 - P_7 \end{pmatrix}$$

02/20/2023 11:10

Divide & Conquer

- merge sort — $O(n \log n)$
- Karatsuba Algo — $T(n) = 3T(\frac{n}{2}) + O(n)$
- Strassen's Algo — $T(n) = 7T(\frac{n}{2}) + O(n^2)$

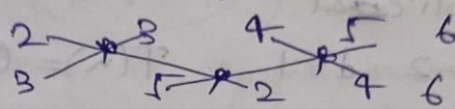
#inversions in an Array

i/p :- Array of n nos

d/p :- A pair (i, j) is an inversion if $A(i) > A(j)$ for $i < j$

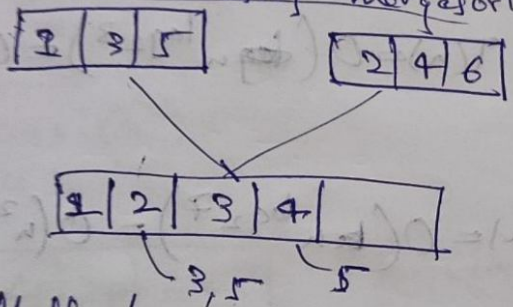
Brute Force : $O(n^2)$

Ex :- (1 3 5 2 4 6)



the no of intersections = # inversions

Finding inversions using mergesort



When you add 2nd element, see the elements left in the 1st array.

Master Method

Assumptions :- All subproblems are of same size (unlike a problem where $T(n) = aT(\frac{n}{10}) + bT(\frac{9n}{10})$)

Recurrence format :-

1) Base case :- $T(n) \leq$ a constant for all sufficiently smaller n

2) For larger n,

$$T(n) \leq aT(\frac{n}{b}) + O(n^d)$$

- a = # recursive calls
- b = i/p shrinkage
- d = exponent in the running time of the combined step independent of n

02/20/2023 11:10

$$T(n) = \begin{cases} O(n^d \log n) & \text{if } a = b^d \\ O(n^d) & \text{if } a < b^d \\ O(n^{\log_b a}) & \text{if } a > b^d \end{cases}$$

Ex:- 1. Merge sort $T(n) \le 2T\left(\frac{n}{2}\right) + O(n)$

$a=2 \quad b=2 \quad d=1 \quad T(n) = O(n^d \log n)$

$= O(n^1 \log n)$

2. Binary search

$T(n) \le T\left(\frac{n}{2}\right) + O(1)$

$a=1 \quad b=2 \quad d=0 \quad T(n) = O(\log n)$

3. Integer multiplication

$a=4 \quad b=2 \quad d=1 \quad T(n) = O(n^{\log_2 4}) = O(n^2)$

4. Karatsuba

$a=3, \quad b=2, \quad d=1 \quad T(n) = O(n^{\log_2 3}) = O(n^{1.59})$

5. Strassen

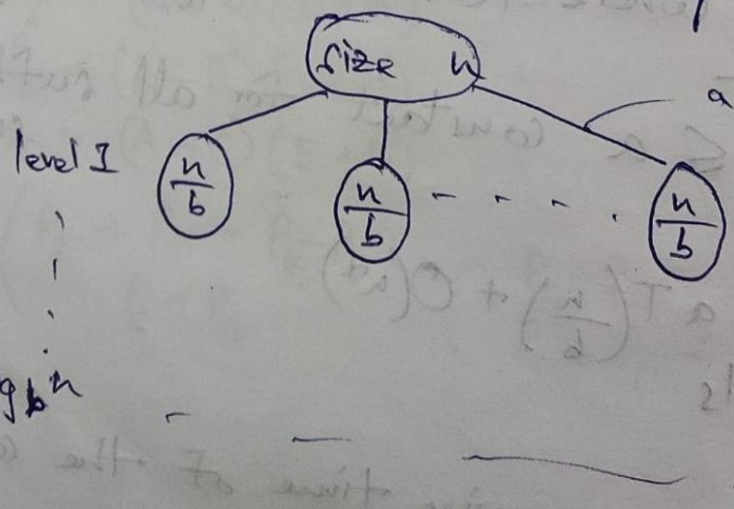
$a=7, \quad b=2, \quad d=2$

$T(n) = O(n^{\log_2 7}) = O(n^{2.81})$

Pf

1. $T(1) \le c$

$T(n) = aT\left(\frac{n}{b}\right) + cn^d \quad n$ is a power of "b".



02/20/2023 11:10

Total work at level $j \le a^j c\left(\frac{n}{b^j}\right)^d = c n^d \left(\frac{a}{b^d}\right)^j$

(the time taken to combine the solved sub problems)

Total work $\le c n^d \sum_{j=0}^{\log_b n} \left(\frac{a}{b^d}\right)^j$

$\le c n^d \log_b n$

$1 + r + r^2 + \dots + r^k = \frac{r^{k+1} - 1}{r-1}$

$r < 1$ RHS $\le \frac{1}{1-r} = \text{const}$

$r > 1$ RHS $\le r^k \left(1 + \frac{1}{r-1}\right)$

Case 2:- $a < b^d$

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

Case 3:- $a > b^d$

$T(n) = O\left(n^d \left(\frac{a}{b^d}\right)^{\log_b n}\right) = O(n^{\log_b a})$

(In the 3rd case, time is spent more at upper levels (smaller i)

— 2nd " " — — — lesser i — — —)

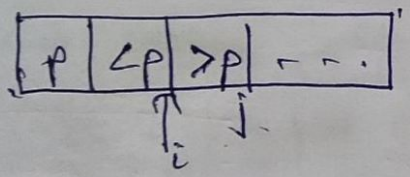
$n^d \left(\frac{a}{b^d}\right)^{\log_b n} = n^{\log_b a}$

$\log_b n \log_b a = \log_b a$

Quicksort

Worst case: $O(n^2)$

Inplace

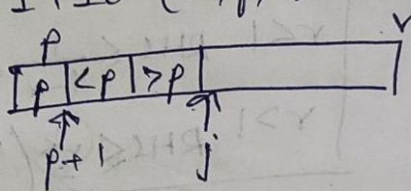



```
Partition(A, p, q)
    x ← A[p]
    i ← p
    for j ← p+1 to q
    do
        if A[j] ≤ x
        then i ← i+1
             swap A[i] ↔ A[j]
    swap A[p] ↔ A[i]
    return i
```

12/1/23

Quicksort's constants are smaller so it is considered ~~the~~ a faster than mergesort (See ~~Compare~~ their best, avg. & worst cases)

PARTITION(A, p, r)

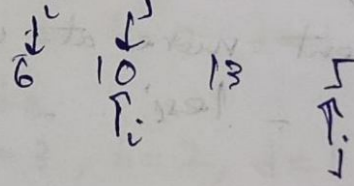


Claim :- The loop maintains the invariants

i) $A[p+1] \ldots A[i-1]$ are all less than pivot

ii) $A[i] \ldots A[j-1]$ " greater " "

Pf :-



QUICKSORT (A, p, r)

if p<r

q ← PARTITION (A, p, r)

QUICKSORT (A, p, q)

" (A, q+1, r)

In Quicksort, we spend more time in solving sub problems

Mergesort, Combining the sol^s of sub problems

02/20/2023 11:10

P(n) :- QUICKSORT correctly sorts every i/p of length n

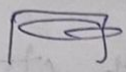
Claim :- P(n) is true for all $n \geq 1$

Strong induction.

P(1) is true

P(k) is true $\forall k < n$

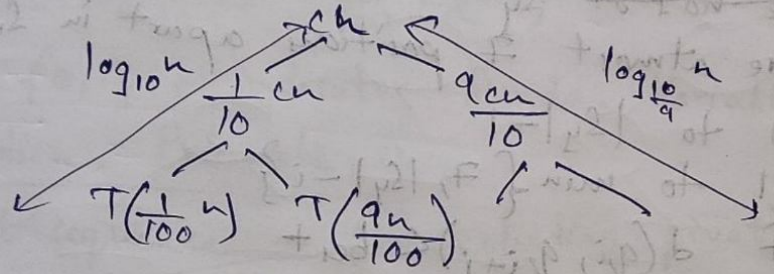
T.P :- P(n) is true



→ $T(n) = 2T(\frac{n}{2}) + O(n)$

$= O(n \log n)$ (Just like merge sort)

→ $T(n) = T(\frac{n}{10}) + T(\frac{9n}{10}) + O(n)$



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

Lucky case

Unlucky case

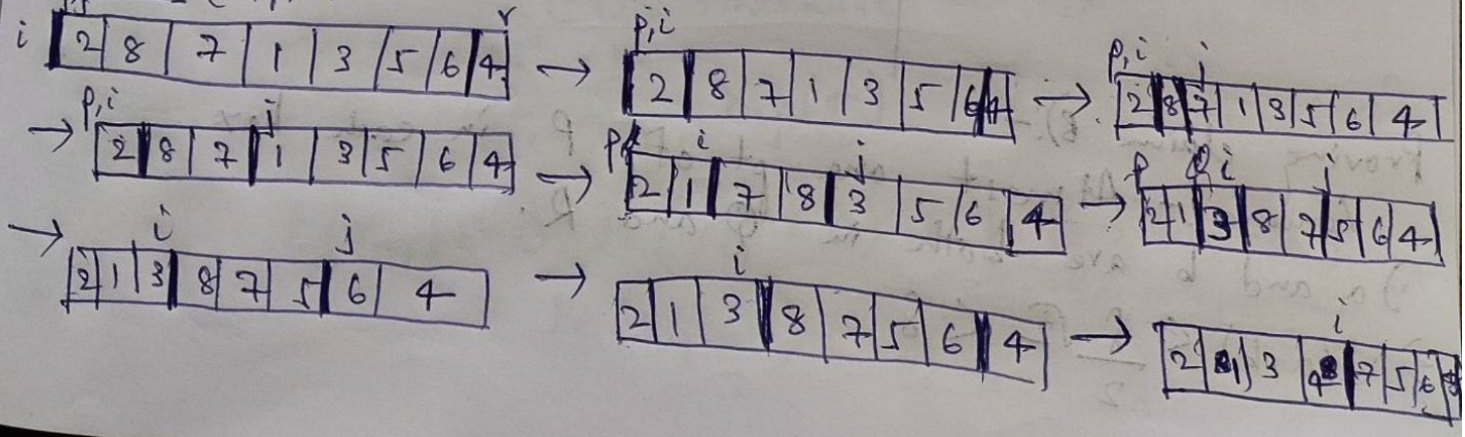
$L(n) = 2U(\frac{n}{2}) + \Theta(n)$

$U(n) = L(n-1) + \Theta(n)$

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

$= \Theta(n \log n)$

Partition (A, p, r)



Closest Pair problem (Computational Geometry)

i/p :- a set $P = \{p_1, p_2, \ldots, p_n\}$ of n pts in $\mathbb{R}^2$

o/p :- a pair $p^*, q^* \in P$ of distinct pts that minimize $d(p,q)$ over p, q in P.

Algorithm :-

P → P_x Q_x, Q_y → S_y – 2δ length strip → $O(n)$

P_y R_x, R_y

$O(n)$

Claim :- Let $p \in Q, q \in R$ be a split pair with $d(p,q) < \delta$

Then

A) p & q ~~are not of~~ $\in S_y$

B) p & q are atmost 7 positions apart in S_y

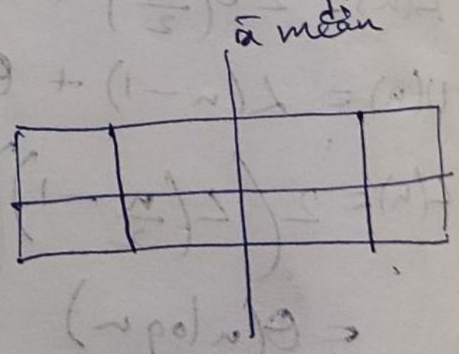
```
for i=1 to |Sy|-1
    for j=1 to min{7, |Sy|-i}
        if d(qi, qi+j) < best
            best = d(qi, qi+j)
            best pair = (qi, qi+j)
```

13/1/23

Q | R

Q_x, Q_y | R_x, R_y

p_1, q_1 | p_2, q_2



Proving claim B) :-

Lemma :- Atmost one pt of P in each box

i) a and b are both in Q and R.

ii) $d(a,b) < \frac{\delta}{2}\sqrt{2} < \delta$

Randomized Quicksort

- Sample Space (Ω) — all possible outcomes (finite – in algo)
- Events (S) $S \subseteq \Omega$
- Random variable $X : \Omega \to R$
- Expectation

$$E(X) = \sum_{i \in \Omega} X(i) \cdot p(i)$$

Ex: When X maps array to the size of left array

form $E(X) = \sum_{i=0}^{n-1} i \cdot \frac{1}{n} = \frac{n-1}{2}$

$X_1, X_2, \ldots X_n$ – random variable in Ω

$$E\left(\sum_{i=1}^{n} X_i\right) = \sum_{i=1}^{n} E(X_i)$$

$X : \Omega \to \{0,1\}$ – indicator Random Variable

Decomposition Principle

σ – Pivot sequence (excluding pivot)

An element will either go to the left array or right so it is not compared twice.

$C(\sigma)$ = # comparisions

$z_i, z_{i+1}, \ldots, z_j$

z_i – i^{th} smallest element

$$X_{ij}(\sigma) = \begin{cases} 1 & \text{if } z_i \text{ \& } z_j \text{ get compared} \\ 0 & \text{otherwise} \end{cases}$$

$$C(\sigma) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}(\sigma)$$

$$E(X_{ij}) = 0 \cdot Pr(X_{ij} = 0) + 1 \cdot Pr(X_{ij} = 1)$$

$$= Pr(X_{ij} = 1) = \frac{2}{j - i + 1}$$

(z_i is chosen 1^{st} + z_j is chosen 1^{st}.)

$$E(x_{ij}) = 2 \sum_{i=1}^{n-1} \sum_{j=i+1} \frac{1}{j-i+1}$$

$$\leq 2n \sum_{k=2}^{n} \frac{1}{k} \leq 2 \int_{1}^{n} \frac{1}{x} \cdot dx$$

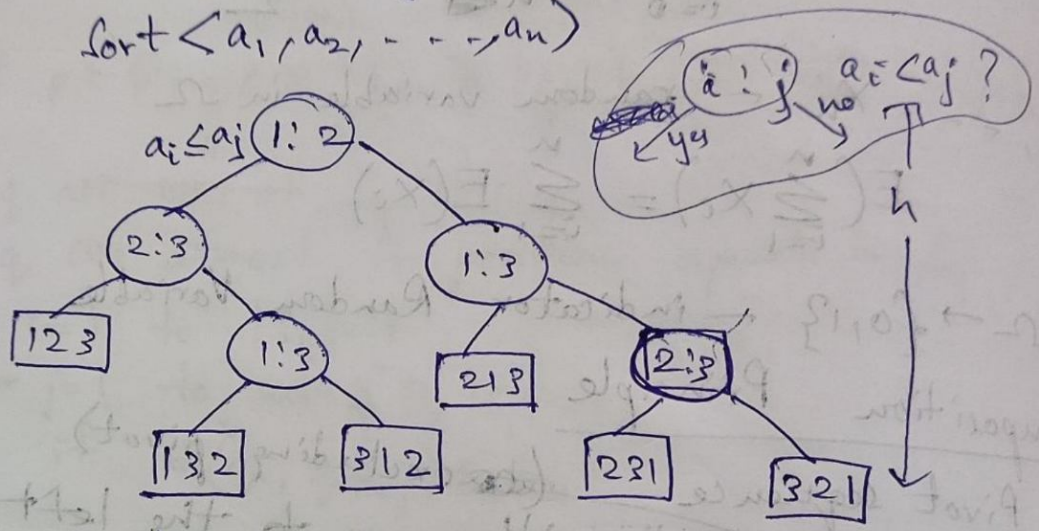
18/1/23

Ω – Pivot seq

$\sigma \in \Omega$ $\quad c(\sigma) = \#$ comparisions

The best comparision based algorithm is $O(n \log n)$.

Pf:- Sort $\langle a_1, a_2, \ldots, a_n \rangle$



Running time is the length of the path

Worst case running time = height of the tree

$n! \leq \left(\frac{n}{e}\right)^n$ — Stirling approximat^n $\quad n! \leq 2^h$ — The leaves are the $n!$ permns of the ~~order~~ array

$\log(n!) \leq h$

$n \log n - n \log e \leq h$

$$\boxed{h = \Omega(n \log n)}$$

Sorting in linear time

No comparision b^n elements

i/p :- $A[1 \ldots n]$ where $A[j] \in \{1, 2, \ldots k\}$

o/p :- $B[1 \ldots n]$ sorted

Auxilary storage :- $C[1 \ldots k]$

02/20/2023 11:11

A | 4 | 1 | 3 | 4 | 3 |

B

C | 1 | 0 | 2 | 2 |

C` – Cumulative of C | 1 | 1 | 3 | 5 |

Now scan A from back ← see 3rd

Assume 1-indexing

see C`[3rd] = 3

C`[3] – –

B[C[3rd]] = 3rd

B- | | | | | |

C[k] = {0, 0, ...}

```
for (int i=0, i<n, i++)
{   C[a[i]]++;   ⟶ O(n)
}
```

The runtime of counting sort is O(n+k)

```
for (int i = 2 → k).
{   C[i] = C[i] + C[i-1];   ⟶ O(k)
}
for (int j = n → 1)
{   B[C[A[j]]] ← A[j];
    C[A[j]] ← C[A[j]] - 1;
}
```

Stable Sorting :- It preserves the input order among equal elements

Find the stable & unstable sorts.

Quicksort, heapsort, selection sort are unstable

Mergesort, Counting sort, Insertion sort, Bubble sort are stable.

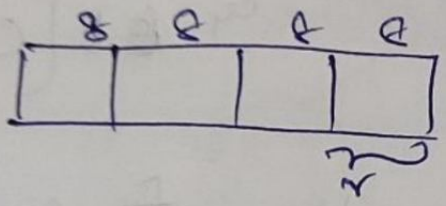
02/20/2023 11:11

3 2	9
4 5	7
5	7
3	9
3	6
2	0
5	5

→ Radix sort on

-bit word

-bit word is made in $\frac{b}{r}$ r-bit words



$$T(n,b) = \Theta\left(\frac{b}{r}(n+2^r)\right)$$

Choose r to minimise $T(n,b)$.

$r = \log n$. $\quad T(n,b) = \Theta\left(\frac{bn}{\log n}\right)$

nor 0 to $n^d - 1$

this is not the global m but is close to

$b = d \log n$

running time $\Theta(nd)$

ion :-

er statistics :- i^{th} – i^{th} min no

min – 1st order statistics

max – n^{th} " "

median – $\left\lceil \frac{n}{2} \right\rceil^{th}$ " "

" – $\left\lceil \frac{n}{2} \right\rceil + 1$ " "

23/1/23

Median :-

~~i th order statistic~~

Finding min, max

(min, max) — $3\lfloor \frac{n}{2} \rfloor$ comparisons

$1 + \frac{3}{2}(n-2) = \frac{3n}{2} - 2$ if n is even

$\frac{3(n-1)}{2} = \lfloor \frac{3n}{2} \rfloor$

$T(n) \leq T(\frac{n}{2}) + O(n)$

Linear selexn

```
selexn( int i, int a[a+size])
{   if
    partition (a[a+size]) <= i
        selexn( i,
}
```

```
selexn(int i, a, a+size)
{   int j = partition(a, a+size)
    if j < i
        selexn( ,a, a+j
```

$n \log(2n)(\log n)$

$= 2n$

$2^r \log 2$

$\frac{n}{r} - \frac{2^r}{r} + 2^r \log 2 = 0 = \frac{n + 2^r}{}$

02/20/2023 11:11

Randomized selexn

Different pivot selexns in the partition gives different running times for different i^{th} order statistics.

RSelect (A, n, i)

1) if $n=1$ return $A[1]$
2) choose pivot p from A uniformly at random
3) Partition around p, j = index of p.
4) if $j=i$, return p.

if $j>i$, return RSelect (1st part of A, $j-1$, i)

if $j<i$, " " (2nd " " " , $n-j$, $i-j$)

RSelect uses $\leq cn$ operations outside of recursive call (for some $c>0$)

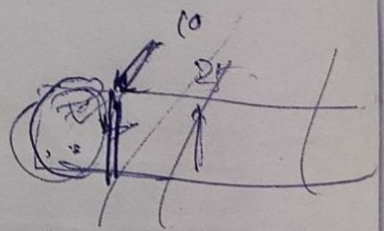
RSelect is in "phase j" if the size of the sub array is in $b^n \left(\frac{3}{4}\right)^{j+1} n$ & $\left(\frac{3}{4}\right)^{j} n$.

Phase no "j" – # times we made 25% progress w.r.t "n"

X_j = # recursive calls during phase "j" –

running time $\leq \sum_{\text{phases } j} X_j \cdot c \cdot \left(\frac{3}{4}\right)^j n$

$\leq \sum_{\text{phases } j} c \left(\frac{3}{4}\right)^j \left(\frac{3}{4}\right)^j$



Good pivot – 25-75 split or better

$E(X_j)$ = expected no of times you need to flip to get "H"

#N = No of coin flips until you get H

02/20/2023 11:11

$E(N) = 1 + \frac{1}{2} E(N)$ Ref: Algo design, Tardos, Kleinberg for this algo.

$E(N) = 2$

$E(\text{no. of } T) \le E\left(cn \sum \left(\frac{3}{4}\right)^j X_j\right)$

$= 2cn \sum_{\text{phase } j} \left(\frac{3}{4}\right)^j \le 8cn = O(n)$

24/1/23

Deterministic algo

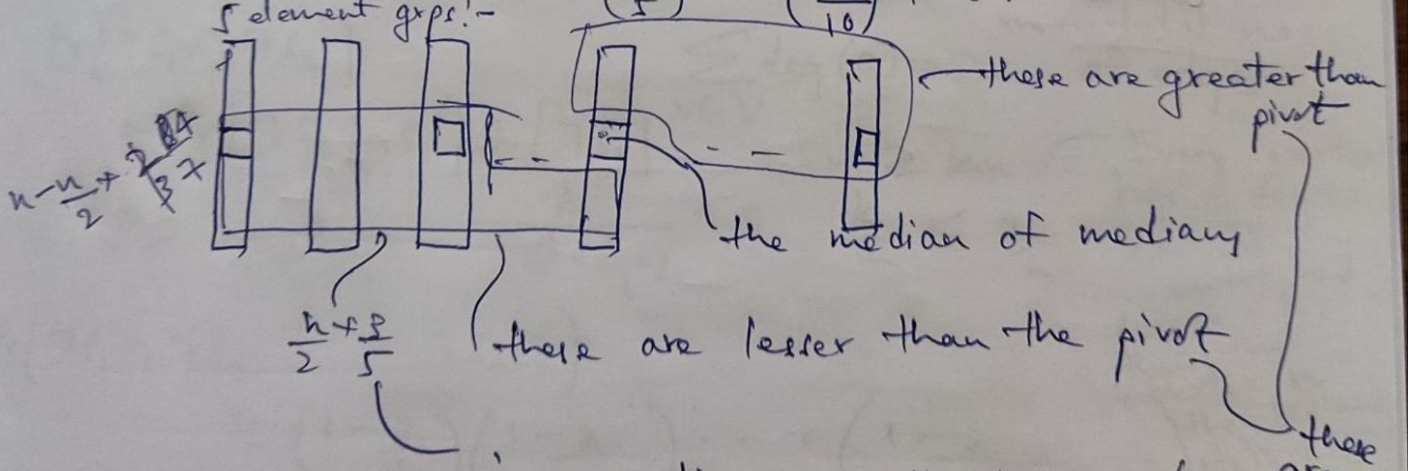
"Median of medians":- $T(n) \le T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + O(n)$

$n \log n$ $\frac{n}{3} \log n + \frac{2n}{3} \log n$

Dselect(A, n, i)

1. Break A into group of 5. Sort each group — $O(n)$
2. $c = n/5$ middle elements
3. p = Dselect$\left(C, \frac{n}{5}, \frac{n}{10}\right) = T\left(\frac{n}{5}\right)$
4. j = Partition A around p.
5. If $j = i$, return p. ~~Dselect(1st part A, j−1, i)~~
6. If $j < i$, return Dselect(1st part A, j−1, i)
7. If $j > i$, " " (2nd " ", n−j, i−j)

$T(n) \le T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10}\right) + O(n)$



This is not an inplace algo.

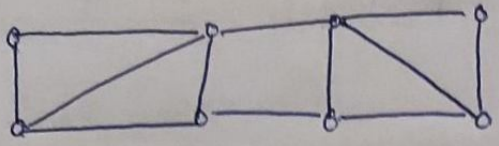
The constants are better in

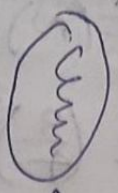
Group of 7? — $O(n)$ like here

Group of 3? — $O(n \log n)$

Minimum cut problem

Ex:-





i/p :- indirect graph $G_1=(V, E)$

o/p :- Compute a cut with fewest # of edges

Here, (as in general) we use adjacency list.

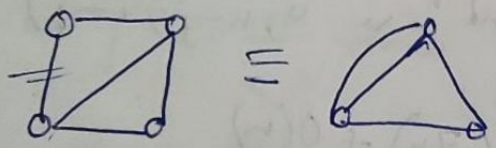
See the brute force algo.

Karger algo

while there are more than 2 vertices :-

- pick a remaining edge (u, v) uniformly at random
- merge "u" & "v" into a single vertex (contraxn)
- remove self-loops.

Ex:-



25/1/23

2 7 12 13 17 | 8 20 4 6 3 | 19 1 9 5 16 | 10 15 18 14 11

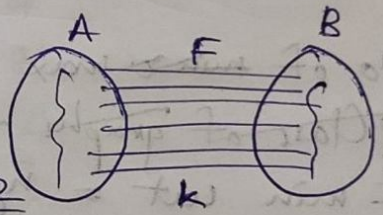
2 7 (12) 13 17 | 3 4 (6) 8 20 | 1 5 (9) 16 19 | 10 11 (14) 15 18

Two types of randomized algos.
1) Correctness depends on randomness — Karger algo
2) Running time " " " .

Karger's algo

Why prob of correct ans $= 1/n^2$?

S_i = event that an edge of F contracted in iteration i



$Pr[\neg S_1 \wedge \neg S_2 \cdots \neg S_{n-2}]$

- $Pr[S_1] = \frac{k}{m} \leq \frac{2}{n}$ $Pr[\neg S_1] > 1 - \frac{2}{n}$

m — total no of edges

$Pr[\neg S_1 \wedge \neg S_2] = Pr[\neg S_2 | \neg S_1] Pr[\neg S_1]$

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

$\sum_{v \in V} deg(v) = 2m$

$2m \geq kn$

$\frac{k}{m} \leq \frac{2}{n}$

$\frac{k}{\#edges}$

Every vertex has a degree $\geq k$. If a vertex had degree $< k$, then that vertex (& remaining the pts) make the min cut

$Pr(\neg S_1 \wedge \neg S_2 \cdots \neg S_{n-2})$

$\geq \left(1 - \frac{2}{n}\right)\left(1 - \frac{2}{n-1}\right) \cdots \left(1 - \frac{2}{n-(n-3)}\right)$

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

02/20/2023 11:12

Repeated Trials

- Run the algo a large no of N times remember the smallest cut formed

Q. Trials?

Let T_i = event that the cut (A, B) is formed on the i^{th} try (independent)

$$\Pr[\text{all } N \text{ trails fail}] = \Pr(\neg T_1 \wedge \neg T_2 \cdots \wedge \neg T_N)$$

$$= \prod_{i=1}^{N} \Pr(\neg T_i) \leq \left(1 - \frac{1}{n^2}\right)^N$$

As $1 + x \leq e^x \ \forall$ real x

$N = n^2$, $\Pr[\text{All fail}] \leq \left(e^{-\frac{1}{n^2}}\right)^{n^2} = \frac{1}{e}$

$N = n^2 \log n$, $\Pr[\ " \ ] \leq \left(\frac{1}{e}\right)^{\log n} = \frac{1}{n}$

Lower bound of running time $- \Omega(n^2 m)$

No of ~~min~~ max mincuts possible $- n-1$ (for a tr

~~Q. Class of graphs for which~~

min cut $= {}^nC_2$ (for cycles) — lower bound

Upper bound :-

$(A_1, B_1) \cdots (A_t, B_t)$ - min cuts of G with "n" ver

$\Pr[\text{output} = (A_i, B_i)] \geq \frac{2}{n(n-1)} = \frac{1}{{}^nC_2}$ for trees

$\{i\}$ disjoint

Post Quiz

Class missed

30/1/23

Binary counter :- k-bit binary counter

A[0 - - - k-1]

LSB MSB

Value of the counter = $\sum_{i=0}^{k-1} A[i]2^i$

Increment by 1 (mod 2^k) :- Aggregate Analysis

k=3 A

Counter value	2	1	0	Cost (Cummulative)
0	0	0	0	0
1	0	0	1	1
2	0	1	0	3
3	0	1	1	4
4	1	0	0	7
				8
7				10
				11
0				14
1				15

Cost of increment = Θ(# bits flipped)

= $O(kn)$

n → no of increment

flips = $\sum_{i=0}^{k-1} \left\lfloor \frac{n}{2^i} \right\rfloor$

= $2n$

Cost per operat^n – $O(1)$

02/20/2023 11:12

Accounting method :-

Charge $2 – for setting a bit to 1

- $1 – flipping
- $1 – prepayment for later flipping to 0

$1 – for every 1

Amortized cost $\leq \$2$

Potential Method

$D_{i-1} \xrightarrow[\hat{c}_i]{c_i} D_i$

$\hat{c}_i = c_i + \Delta\phi(D_i)$ D_i is the no of 1's.

$\phi(D_0) = 0$ $\phi = b_i = \#$ 1's after i^{th} increment

i^{th} operat^n – reset t_i bits

$c_i \leq t_i + 1$

$b_i = 0$ $b_{i-1} = t_i = k$

$b_i = b_{i-1} - t_i$

$b_i > 0$ $b_i = b_{i-1} - t_i + 1$

$\boxed{b_i \leq b_{i-1} - t_i + 1}$

$\Delta\phi(D_i) \leq (b_{i-1} - t_i + 1) - b_{i-1}$

$= 1 - t_i$

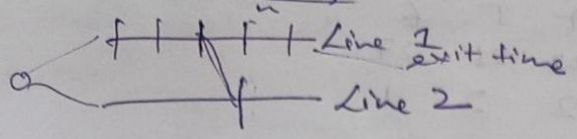
Dynamic Programming

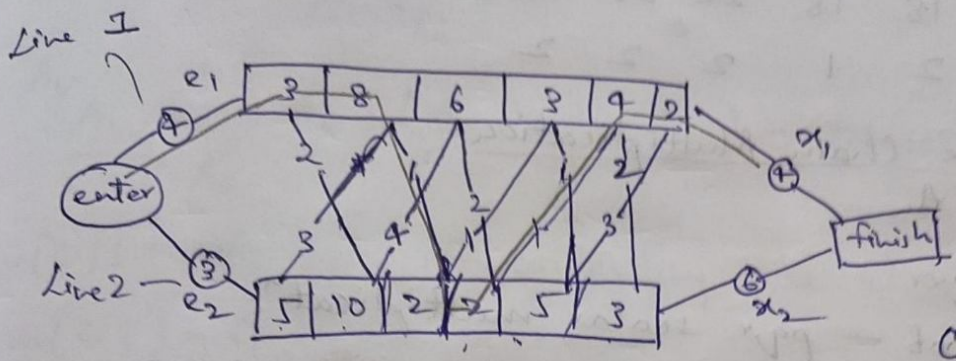
Not only solving subproblems by remembering, we try to find overlap b^w sub problems.

- Overlapping subproblems
- Optimal substructure

02/20/2023 11:12

Assembly line scheduling





Brute force :- Solve 2^n sub problems ($O(2^n)$)

$O(n)$ – dp sol^n

$S_{i,j}$ – j^{th} stat^n in i^{th} time

$j = \{1, 2, 3, \ldots n\}$ $\quad i = \{1, 2\}$

e_i – entry time

x_i – exit time

$t_{i,j}$ – time to move from $S_{i,j}$ to $S_{3-i,j+1}$

$a_{i,j}$ – time req for $S_{i,j}$

f^* – fastest time for the entire assembly

$f_i[j]$ – fastest possible time to state^n j in i^{th} time

$f^* = \min \{ f_1[n] + x_1, f_2[n] + x_2 \}$

~~$f_i[j]$~~

$f_1[1] = e_1 + a_{1,1}$

$f_2[1] = e_2 + a_{2,1}$

for all j, $2 \le j \le n$

$f_1[j] = \min \{ f_1[j-1] + a_{1,j}, f_2[j-1] + t_{2,j-1} + a_{1,j} \}$

$f_2[j] =$ . . . sly or 1

j	1	2	3	4	5	6
$f_1[j]$	7	15	21	22	25	27
$l_1[j]$		1	1	2	2	1
$f_2[j]$	8	18	18	20	25	28
$l_2[j]$		2	1	2	2	2

2/2/23 **Matrix Chain Multiplication**

$M = A_1 \dots A_n$

$(A)_{p \times q}$ $(B)_{q \times r}$

For $A \times B$ — pqr scalar multiplicat^{n}s

~~ABC~~

$A_{10\times100}$ $B_{100\times10}$ $C_{10\times100}$ — $(A)(BC)$ — 2,00,000 multipl

$(AB)(C)$ — 20,000 "

A chain of matrices is fully parenthesized if either a single matrix or the product of two fully parenthesized matrices.

ABCD - How many fully parenthesizat^{n}s?

Sol) $(A)(B(CD))$ $(A)((BC)D)$

$((A(BC))D)$ $((AB)(CD))$ $(((AB)C)D)$

$P(n)$ = # distinct fully parenthesizat^{n}s of a chain of n matrices

$$P(n) = \begin{cases} 1 & \text{if } n = 1 \\ \sum_{k=1}^{n-1} P(k)P(n-k) & \text{if } n \geq 2 \end{cases}$$

$P(n) = C(n-1)$

$C(n) = \frac{1}{n+1}\binom{2n}{n} = \Omega\left(\frac{4^n}{n^{3/2}}\right)$

02/20/2023 11:12

Redefine :- $p = (p_0, p_1, \ldots, p_n)$ +ve integers.
Compute the cost of full parenthesization of a chain of matrices

$$A_1 \ldots A_n$$

$(A_i)_{p_{i-1} * p_i}$

$A_1 \ldots A_n$

$(A_1 \ldots A_k)(A_{k+1} \ldots A_n)$:- Opt cost of $A(k)$ + Opt cost of $B(k)$ + $p_0 p_k p_n$

($A_1 \ldots A_k$ = A_k, $A_{k+1} \ldots A_n$ = B_k)

$m[i,j]$ = opt cost of computing the chain $A_i \ldots A_j$

$m[i,i] = 0$

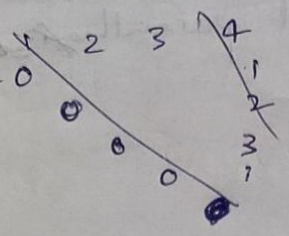
$$m[i,j] = \min_{i \le k \le j-1} \{ m[i,k] + m[k+1,j] + p_{i-1} p_k p_i \}$$

~~MC(A, p, n, i, j)~~

```
{
    for (i ← 1 to n) do m[i,i] ← 0
    for (l ← 2 to n) do
        for i ← 1 to n-l+1 do
        {   j ← i+l-1
            m[i,j] ← +∞
            for k ← i to j-1 do {
                y ← m[i,k] + m[k+1,j] + p_{i-1} p_k p_i
                if y < m[i,j]
                then
                    m[i,j] ← y
                    s[i,j] ← k
```

(4, 3, 5, 2, 2)

$A_{4\times3}$ $B_{3\times5}$ $C_{5\times2}$ $D_{2\times2}$



1	2	3	4	
0	60	54	66	1
	0	30	42	2
		0	20	3
			0	4

$$m[1,5] \min \begin{cases} m[1,1] + m[2,3] + p_0 p_1 p_3 \\ m[1,2] + m[3,5] + p_0 p_2 p_3 \end{cases}$$

$\min[1,4]$

$(A((B\ C)\ D))$

Memoization

$MC(p, n, i, j, m)$ $m = \{0\}$

```
{ If i=j;
    return 0
  for( t = 1 ⟶
```

Largest Common Subsequence :

6/2/23

Given 2 sequences $x[1 \ldots m]$ & $y[1 \ldots n]$. Find a longest common subsequence.

x: A B C B D A B — 2^m

y: B D C A B A

Brute force :- $O(2^m \cdot n)$

LCS(x, y) = B C B A

|c|, $c[i,j] = |LCS(x[1 \ldots i], y[1 \ldots j])|$

$c(m,n) = |LCS(x,y)|$

$X_i = prefix\{x_1 \ldots x_i\}$ $Y_j = prefix\{y_1 \ldots y_j\}$

Thm :- $Z = \{z_1 \ldots z_k\}$ be any LCS of X & Y.

1. If $x_m = y_n$, then $z_k = x_m = y_n$ & Z_{k-1} is an LCS of X_{m-1} & Y_{n-1}

2. If $x_m \neq y_n$, then $z_k \neq x_m \Rightarrow Z$ is an LCS of X_{m-1} & Y.

3. If $x_m \neq y_n$, then $z_k \neq y_n \Rightarrow Z$ is an LCS of X & Y_{n-1}

Pf :- T.P $z_k = x_m = y_n$

Suppose not, then $\langle z_1 \ldots z_k, x_m \rangle$ is a CS of X & Y whose length is $k+1 \Rightarrow\Leftarrow$ Z is an LCS

T.P z_{k-1} is an LCS of X_{m-1} & Y_{n-1}

Clearly z_{k-1} is a CS.

W is another CS, $|W| \geq k$.

$W x_m$ is a CS of X & Y whose length $> |Z|$

Let $z_k \neq x_m$.

Suppose W is a subseq of X_{m-1} & Y with length k

W is a CS of X & Y $\Rightarrow\Leftarrow$ Z is LCS

2. If $x_m \neq y_n$, then $z_k \neq x_m \Rightarrow$ Z is an LCS of X_{m-1} and Y_n

$$c[i,j] = \begin{cases} c[i-1,j-1]+1 & \text{if } x[i]=y[j] \\ \max\{c[i-1,j], c[i,j-1]\} & \text{o.w} \end{cases}$$

		A	B	C	B	D	A	B
	0	0	0	0	0	0	0	0
B	0	0	1	1	1	1	1	1
D	0	0	1	1	1	2	2	2
C	0	0	1	2	2	2	2	2
A	0	1	1	2	2	2	3	3
B	0	1	2	2	3	3	3	4
A	0	1	2	2	3	3	4	4

There are 2 optimal sol^{n}s

Optimal BST

Given (a seq $\langle k_1, \ldots, k_n \rangle$ of 'n' distinct keys sorted $k_1 < k_2 \ldots < k_n$

Want to build a BST from the keys.

p_i – prob that the search is for k_i.

Want BST with min expected search cost.

Actual cost = # of items examined

For key k_i, cost = depth(k_i) + 1

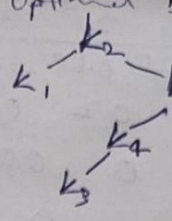
$$E[\text{search cost in } T] = \sum_{i=1}^{n} (\text{depth}_T(k_i)+1)\, p_i$$

$$= \sum_{i=1}^{n} \text{depth}_T(k_i)\, p_i + 1$$

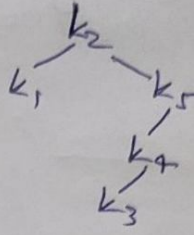
Assumption:- All searches are successful

i	1	2	3	4	5
p_i	0.25	0.2	0.05	0.2	0.3

Optimal BST:-



E( ) = 2.1



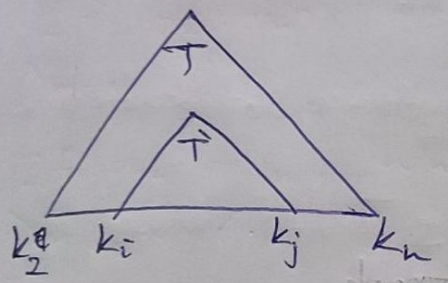
E( ) = 2.15

8/2/23

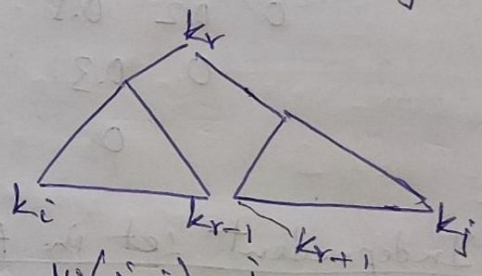
→ The most searched element need not be closed to the root.

→ The tree need not be with the shortest height.

i	depth(k_i)	depth(k_i).p_i
1	1	0.25
2	0	0
3	2	0.1
4	1	0.2
5	2	0.6



Given key $k_i - - - k_j$



$$w(i,j) = \sum_{l=i}^{j} p_l$$

(It is not l, it's e)

$l[i,j]$ = exp. search cost of opt BST for $k_i - - - k_j$

$j = i-1$, the tree is empty

$l[i,j] = 0$ for $j = i-1$

$j \geq i$

$$l[i,j] = p_r + l[i,r-1] + l[r+1,j] + w(i,r-1) + w(r+1,j)$$

$$= \min_{i \leq r \leq j} \{ l[i,r-1] + l[r+1,j] + w(i,j) \}$$

e \ i	0	1	2	3	4	5
1	0	0.25	0.65	0.8	1.25	2.1
2		0	0.2	0.3	0.75	1.35
3			0	0.05	0.3	0.85
4				0	0.2	0.7
5					0	0.3
6						0

root	1	2	3	4	5
1	1	1	1	2	2
2		2	2	2	4
3			3	4	5
4				4	5
5					5

w \	0	1	2	3	4	5
1	0	0.25	0.45	0.5	0.7	1.0
2		0	0.2	0.25	0.45	0.75
3			0	0.05	0.25	0.55
4				0	0.2	0.5
5					0	0.3
6						0

Weighted Independent set in Path Graph

$G = (V, E)$ – path Graph with non -ve weights on vertices

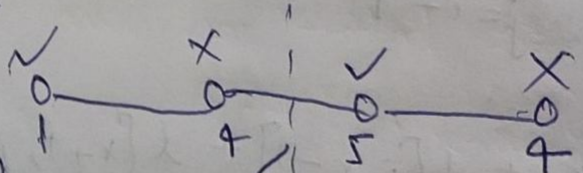
Independent set – non adjacent set of vertices

(IS)

~~Subset of non adjacent vertices~~ –

IS max weight

Ex:-



Think why divide & conquer is wrong.

4 & 5 will come, as any but u can't consider

02/20/2023 11:13

v_n – n^{th} vertex

S – Max weighted (MW) IS.

$v_n \notin S$, $G' = G \setminus v_n$. ~~S is a max IS of G'~~

$v_n \in S$, $G'' = G' - \{v_{n-1}\}$

16/2/23

Greedy won't work: See Ex!

$S \subseteq V$ max I-S. v_n is last vertex in this path.

Case 1:-

$v_n \notin S$ $G' = G \setminus \{v_n\}$

S is a max I-S of G'.

Case 2:-

$v_n \in S$ $G'' = G' - \{v_{n-1}\}$.

$S \setminus \{v_n\}$ is a max IS of G''.

A max weighted IS must be either

i) a max weighted IS of G' (or) S_1

ii) v_n + a max-weighted IS of G'' — S_2

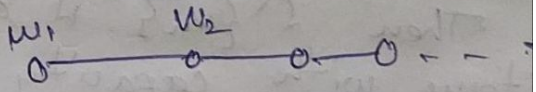
– recursively calculate S_1

– " " S_2

– return S_1 or $S_2 \cup \{v_n\}$ whichever is better.

Running time:- exponential

(The given graph is a path)



G_i = 1^{st} i vertices of G

A[i] = value of max weight IS of G_i

$A[0] = 0$ $A[1] = w_1$

For $i = 2, 3, \ldots, n$

$A[i] = \max\{A[i-1], A[i-2] + w_i\}$

Running time: $O(n)$

$S = \phi$

while $i \geq 1$

if $A[i-1] \geq A[i-2] + w_i$

$i--;$

else

add v_i to S

$i = i-2$

Knapsack Problem (0-1 knapsack)

Input:- n items

- v_i value (non-ve)
- size w_i (non-ve integer) — some significance for this.
- capacity W (")

Output:- A subset $S \subseteq \{1, 2, \ldots, n\}$ that maximizes $\sum_{i \in S} v_i$ subject to $\sum_{i \in S} w_i \leq W$.

Pf:- S = max-value soln to an instance of knapsack

Case 1:- $n \notin S$

S must be optimal with the 1st $n-1$ items

Case 2:- $n \in S$

Then $S \setminus \{v_n\}$ is an optimal soln w.r.t first $(n-1)$ items with capacity ~~$\{w_n\}$~~ $W - w_n$

Let $V_{i,x}$ = value of the best soln that has 1st "i" items and has total size $\leq W$

$= \max \{ V_{i-1,x}, v_i + V_{i-1,x-w_i} \}$

If $w_i > x$, $V_{i,x} = V_{i-1,x}$

Running time: $O(nW)$

02/20/2023 11:13

Example :- $n=4,\ W=6$

$v_1 = 3 \quad w_1 = 4$

$v_2 = 2 \quad w_2 = 3$

$v_3 = 4 \quad w_3 = 2$

$v_4 = 4 \quad w_4 = 3$

6	0				
5	0				
4	0				
3	0				
2	0				
1	0				
0	0	0	0	0	0
	0	1	2	3	4

20/2/23

Sequence Alignment :-

Needleman Wunsch - Score

A G G G C T

A G G — C A

Total penalty = $\alpha_{gap} + \alpha_{AT}$

Input :- Strings $X = x_1 \ldots x_m$, $Y = y_1 \ldots y_n$ } over A G C T

- penalty α_{gap} for inserting gap,

for matching "a" and "b" — α_{ab}

$[\alpha_{ab} = 0 \text{ for } a = b]$

Goal :- Alignment with min possible total penalty.

3 cases :-

Case 1 :- x_m, y_n matched

Case 2 :- x_m matched with a gap

Case 3 :- y_n is matched with a gap.

$X' = X - x_m$, $Y' = Y - y_n$

02/20/2023 11:13

$X_i = x_1 \ldots x_i$

$Y_j = y_1 \ldots y_j$

p_{ij} = penalty for optimal alignment of X_i & Y_j

$$p_{ij} = \begin{cases} p_{i-1,j-1} + \alpha_{x_i y_i} \\ p_{i-1,j} + \alpha_{gap} \\ p_{i,j-1} + \alpha_{gap} \end{cases}$$

$p_{i,0} = i \cdot \alpha_{gap}$

$p_{0,j} = j \cdot \alpha_{gap}$

Running time :- $O(mn)$ time algo
$O(m+n)$ time to construct the sequences from table.

Edit distance
Longest increasing subsequence common

Greedy Algorithms

- Make a locally optimal choice in hope of getting a globally optimal soln.

$O(nW)$ — pseudo-polytime algo

Activity Selexn :

n activities — exclusive use of common resource

$S = \{a_1, \ldots, a_n\}$

$a_i = [s_i, f_i)$ $\quad f_i < f_j$ for $i < j$

Goal :- Select the largest possible set of non-overlapping activities.

i	1	2	3	4	5	6	7	8	9
s_i	1	2	4	1	5	8	9	11	13
f_i	3	5	7	8	9	10	11	14	16

$a_1\ a_5\ a_6\ a_8$ $\quad a_2\ a_5\ a_7\ a_9$ $\quad$ give all answers

$a_2\ a_5\ a_7\ a_8$

02/20/2023 11:13

- Greedy choice Property :- A.T Greedy choice is part of optimal sol^n.

$S_{ij} = \{a_k / f_i \le s_k < f_k \le s_j\}$

Thm :- Let $S_{ij} \neq \phi$, Let a_m be the activity in S_{ij} with the earliest finish time.

$$f_m = \min\{f_k / a_k \in S_{ij}\}$$

1) a_m is used in some max-size of mutually compatible activities of S_{ij}.

2) $S_{im} = \phi$, So choosing a_m leaves S_{mj} as only non-empty subproblem.

i	1	2	3
v_i	60	100	120
w_i	10	20	30
$\frac{v_i}{w_i}$	6	5	4

$W = 50$

The optimal sol^n is not greedy -