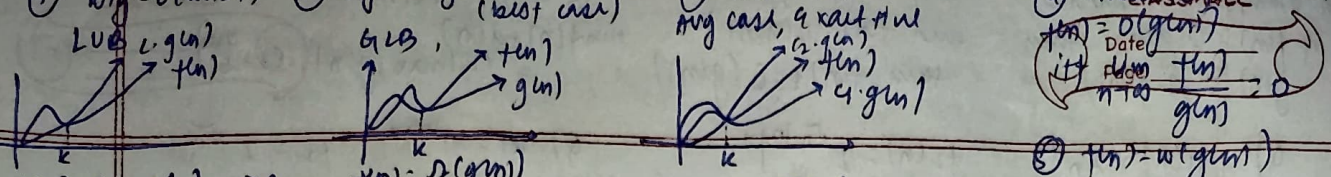
Asymptotic Notation :- mathematical way of representing the time complexity

① Big-O (worst) ② big-omega (Ω) (best case) ③ Theta (Θ) Avg case, exact time ④ small-o (strictly)



$f(n) \le c \cdot g(n)$, $c > 0$ for $n \ge k$ & $k > 0$

$f(n) = 2n^2 + n$

$f(n) = O(?)$

~~$f(n)$~~

$2n^2 + n \le 3n^2$

$\therefore c = 2$

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

$f(n) \ge c \cdot g(n)$

$c > 0, n \ge k \quad k > 0$

$f(n) = 2n^2 + n$ ✓

$c \cdot g(n) = n^2$

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

$c_1 \cdot g(n) \le f(n) \le c_2 \cdot g(n)$

$n^2 \le 2n^2 + n \le 3n^2$

⑤ $f(n) = \omega(g(n))$

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

Properties of Asymptotic Notation

① Reflexive

a) Big(O): $a \le b$ then $a \le a$ yes

b) Big(Ω): $a \ge b$ then $a \ge a$ yes

c) (Θ): $f(n) = g(n)$ then $f(n) = f(n)$ yes

d) small(o): $f(n) < c \cdot g(n)$, $a < b$, $a < a$ No

e) small(ω): $f(n) > c \cdot g(n)$, $a > b$, $a > a$ No

② Symmetric

a) Big(O): $a \le b$, $b \le a$, No

b) Big(Ω): $a \ge b$, $b \ge a$, ~~no~~ No

c) Theta(Θ): $a = b$, $b = a$, yes

d) small(o): $a < b$, $b < a$, No

e) small(ω); $a > b$, No, as $b < a$

③ Transitive

a) Big(O): $a \le b$, $b \le c$, $a \le c$ yes

b) Big(Ω): $a \ge b$, $b \ge c$, $a \ge c$ yes

c) Theta(Θ): $a = b$, $b = c$, $a = c$, yes

d) small(o): $f(n) < g(n)$, $g(n) < h(n)$, $f(n) < h(n)$ yes

e) small(ω): $f(n) > g(n)$, $g(n) > \frac{1}{c} h(n)$; yes, $f(n) > h(n)$

~~comparison~~ of various time complexity

$O(1) < O(\log\log n) < O(\log n) < O(\sqrt{n}) < O(n) < O(n \log n) < O(n^2) < O(n^3) < O(n^k) < O(2^n) < O(n^n) < O(2^{2^n})$

Q) $f_1(n) = n^2 \log n$, $f_2(n) = n (\log n)^{10}$, which is greater

$n \cdot (n \log n)$ vs $(n \log n) \cdot (\log n)^9$

Taking log both & putting $\log n = y$

$\log n + \log y$ vs $\log y + 9 \log\log n$ as $\log n > 9 \log\log n \quad \therefore f_1 > f_2$

Q) $f_1(n) = 2^n$, $f_2(n) = n^{n/2}$, $f_3(n) = n \log n$, $f_4(n) = n^{\log_2 n}$

clearly $f_1 > f_2$

$f_2 = n^{n/2}$, $f_3 = n \log n$

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

$\frac{1}{2} \log n > \log\log n$

$\therefore f_2 > f_3$

$f_3 = n \log n$, $f_4 = n^{\log n}$

$\log n + \log\log n$ vs $\log n \cdot \log n$

$\log n$ vs $\log n - 1$

$\therefore n^{\log n} > n^{\log n - 1}$

$\log\log n < \log(n-1) \cdot \log n$

$\therefore f_4 > f_3$

$f_2 = n^{n/2}$, $f_4 = n^{\log_2 n}$

$\frac{n}{2} \log n$ vs $\log n \cdot \log n$

$f_4 > f_2$

$f_1 = 2^n$ & $f_4 = n^{\log n}$

n vs $\log n \cdot \log n$

n vs $(\log n)^2$

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

$f_1 > f_4$

Some more properties

- Transpose symmetric: if $f(n) = O(g(n))$ then $g(n) = \Omega(f(n))$
- if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$ then $f(n) + g(n) = \Theta(g(n))$
- if $f(n) = O(g(n))$ and $d(n) = O(e(n))$ then $f(n) + d(n) = O(\max(g(n), e(n)))$

classmate

Q) $f(n) = 3n^{\sqrt{n}}$ $\quad f_2(n) = 2^{\sqrt{n}\log n}$

log on both sides

$\log 3 + \sqrt{n}\log n \quad \sqrt{n}\log n \log 2$

$\log\log n + \log\sqrt{n} \quad \frac{1}{2}\log n + \log\log n + \log\log 2$

$\log\log n > \log\log 2$

$\therefore f_1 > f_2$

Q) $f = n^{\log n} \quad f_2 = 2^{\sqrt{n}}$

$\log n \cdot \log n \quad \sqrt{n}\log 2$

$2\log\log n \quad \frac{1}{2}\log n + \log\log n$

$\log\log n \leq \frac{1}{2}\log n$

$\therefore f_1 < f_2$

Q) $f(n) = n^{\log n} \quad g(n) = n^{\sqrt{n}}$

$\log n \quad \sqrt{n}\log n$

$g(n) > f(n)$

Q) check $\sqrt{\log n} = O(\log\log n)$

$f_1 = \sqrt{\log n} \quad f_2 = \log\log n$

$f_1 > f_2$

$\therefore$ false

Recurrence relations

B.S. (a, i, j, x)

mid = (i+j)/2 if a[mid] == x return mid

else:

if (a[mid] > x) $\quad T(n) = T(n/2)$

BS(a, i, mid-1, x);

else BS(a, mid+1, j, x);

① substitution method

$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ T(n/2) + 1 & \text{if } n > 1 \end{cases}$

$T(n) = T(n/4) + 2 = T(n/8) + 3$

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

$T(n) = T(1) + \log_2 n \quad T(n) = \log_2 n \quad T(n) = O(\log n)$

Q) $T(n) = \begin{cases} 1 & \text{if } n = 1 \\ T(n-1) + n & \text{if } n > 1 \end{cases}$

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

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

putting $k = n-1$

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

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

$T(n) = n^n \quad T(n) = O(n^n)$

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

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

$T(n) = 2\left(2T(n/4) + n/2\right) + n$

$T(n) = 4T(n/4) + 2n \quad T(n) = n + 4\left(2T(n/8) + n/4\right) \quad T(n) = n + 8T(n/8) + 3n$

$T(n) = 8T(n/8) + 3n$

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

$n = 2^k \quad T(n) = 2^k T(1) + k \cdot 2^k \quad T(n) = 2^k + k \cdot 2^k$

$T(n) = n + n\log n \quad T(n) = O(n\log n)$

Q) $T(n) = T(n-1) + \log n \quad T(n-1) = T(n-2) + \log(n-2)$

$T(n) = T(n-2) + \log n + \log(n-2) \quad T(n) = T(n-k) + \log n + \log(n-1) \cdots \log(n-k)$

putting $k = n-1$

$T(n) = 1 + \log n + \log(n-1) \cdots \log(1) \quad T(n) = O(\log n!)$

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

Q) $T(n) = T(\sqrt{n}) + 1 \quad n > 2$

$T(\sqrt{n}) = T(n^{1/4}) + 1$

$T(n^{1/4}) = T(n^{1/8}) + 1$

$T(n) = T(n^{1/2^k}) + k$

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

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

② Recurrence tree

① $T(n) = 2T(n/2) + cn$

work done = height of tree · work at level $= (\log n) \cdot (cn)$

$\therefore T(n) = O(n\log n)$

② $T(n) = 2T(n/4) + cn^2$

$T(n/4) = 2T(n/16) + \frac{cn^2}{16}$

$T(n/16) = 2T(n/64) + \frac{cn^2}{16^2}$

$h = 2 = \log_4 16$

$\therefore h = \log_4 n$

work done $= \log n \left[cn^2 + \frac{3}{16}cn^2 + \left(\frac{3}{16}\right)^2 cn^2 + \left(\frac{3}{16}\right)^3 cn^2 \cdots\right]$

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

$= \log n \cdot cn^2 \cdot \frac{1}{1 - 3/16}$

$= cn^2 \log n \cdot \frac{16}{13}$

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

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

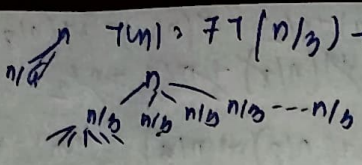
$T(n/4) = 3T(n/16) + \frac{n^2}{16}$

$T(n/3) = 7T(n/9) + \frac{n^2}{9}$

$T(n/4) = 7T(n/27) + \frac{n^2}{9^2}$

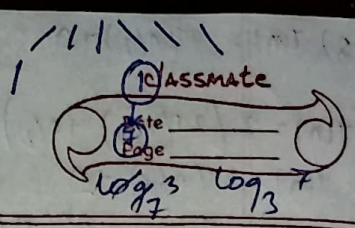
work done: $n^2\left(1 + \frac{7}{9} + \left(\frac{7}{9}\right)^2 \cdots \text{upto}\right)$

$T(n) = 7T(n/3) + n^2$



$n^2 + \frac{7n^2}{9} + \left(\frac{7}{9}\right)^2 n^2$

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



(3) Master's theorem

$T(n) = aT(n/b) + f(n)$ $\quad a \geq 1 \quad b > 1$

$T(n) = n^{\log_b a}\, v(n)$

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

$f(n)$	$v(n)$
$n^k + h(n)$	
n^k, $k > 0$	$O(n^k)$
n^r, $r < 0$	$O(1)$
$(\log n)^i$ $i \geq 0$	$\frac{(\log_2 n)^{i+1}}{i+1}$

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

$T(n) = n^{\log_2 1} v(n)$

$h(n) = \frac{c}{n^{\log_2 1}} = c = c(\log n)^0$

$\therefore v(n) = \frac{(\log_2 n)^{i+1}}{i+1} \Rightarrow \log n$

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

9) $T(n) = T(\sqrt{n}) + \log n \qquad n = 2^m$

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

$S(m) = S(m/2) + m \qquad v(n) = h\,v(n) = \frac{m}{\log_2 1} = m \qquad \therefore h(n) = O(n)$

$S(m) = O(m) \qquad T(2^m) = O(m)$

$m = \log_2 n$

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

using aus:

$T(n) = T(\sqrt{n}) + \log n \qquad T(\sqrt{n}) = T(n^{1/4}) + \frac{1}{2}\log n \qquad T(n^{1/4}) = T(n^{1/8}) + \frac{1}{4}\log n$

$T(n) = T(n^{1/4}) + \log n + \frac{1}{2}\log n \qquad T(n) = T(n^{1/8}) + \log n + \frac{1}{2}\log n + \frac{1}{4}\log n$

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

$k \to \infty \qquad \frac{1}{2^k} \to 0$

$T(n) = T(1) + \log n \frac{1}{1/2} = 1 + 2\log n$

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

Q) $T(n) = 4T(n/6) + n^2 \log n$

$T(n) = n^{\log_6 4} f(n)$

$h(n) = \frac{n^2 \log n}{n^{\log_3 6^c}} = \frac{n^2 \log n}{n^{\log_3 3 + \log_3 2}} = \frac{n^2 \log n}{n \cdot n^{\log_3 2}} = \frac{n \log n}{n}$

$T(n) = a T(n/b) + f(n)$ $\quad f(n) = \Theta(n^k \log^p n)$

a) $\log_b a$ b) k

case 1: if $\log_b a > k$ then $T(n) = \Theta(n^{\log_b a})$

case 2: if $\log_b a = k$

if $p > -1$ $\quad \Theta(n^k \log^{p+1} n) = T(n)$

if $p = -1$ $\quad T(n) = \Theta(n^k \log \log n)$

if $p < -1$ $\quad \Theta T(n) = \Theta(n^k)$

case 3: if $\log_b a < k$ $\quad$ if $p \geq 0$ $\Theta(n^k \log^p n)$

if $p < 0$ $\Theta(n^k)$

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

$f(n) = \Theta(n^0 \log^0 n)$ $\quad \log_a b = 1$ $\quad k = 0$

as $\log_a b > k$ $\therefore T(n) = \Theta(n^{\log_b a}) = \Theta(n)$

Q) $T(n) = 4T(n/2) + n$ $\quad f(n) = \Theta(n^1 \log^0 n)$

$\log_b a = 2$ $\quad k = 1$ $\quad \log_b a > k$ $\therefore T(n) = \Theta(n^2)$

Q) $T(n) = 8T(n/2) + n$ $\quad f(n) = \Theta(n^1 \log^0 n)$

$\log_b a = 3$ $\quad k = 1$ $\quad \therefore T(n) = \Theta(n^3)$

Q) $T(n) = 9T(n/3) + n^2$ $\quad f(n) = \Theta(n^2 \log^0 n)$

$\log_b a = 2$ $\quad k = 2$ $\quad \log_b a = k$ $\quad p = 0$

as $p > -1$ $\quad T(n) = \Theta(n^k \log^{p+1} n) = \Theta(n^2 \log n)$

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

$\log_b a = 2$ $\quad k = 2$ $\quad p > -1$

$T(n) = \Theta(n^2 \log^3 n)$

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

$\log_a b = 2$ $\quad k = 2$ $\quad p > -1$

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

Q) $T(n) = 2T\lceil n/2 \rceil + n/\log n$

$\log_b a = 1$ $\quad k = 1$ $\quad p = -1$

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

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

$\log_b a = k$

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

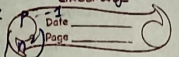
Q) $T(n) = 8T(n/2) + n^3/\log^3 n$

$\log_a b = k$

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

Q) $T(n) = T(n/3) + n^2$
$\log_b a = 0 \quad k = 2 \quad p = 0$
$T(n) = \theta(n^2)$

Q) $T(n) = 27(n/3) + n^3 \log n$
$\log_b a = 1 \quad k = 3 \quad p = 1$
$T(n) = \theta(n^3 \log n)$

Q) $T(n) = 47(n/6) + \frac{n^3}{\log n}$
$\log_a b < k$
$T(n) = \theta($  $)$

Q) $T(n) = 2T(n/2) + n \log n$
$\log_b a = k \quad p = 0$
$T(n) = \theta(n \log^2 n)$

Q) $T(n) = 7T(n/3) + n^2$
$\log_b a < k \quad p = 0$
$\therefore T(n) = \theta(n^2)$

Q) $T(n) = 2T(n/4) + \theta(n^2)$
$\log_4 2 < (k = 2) \quad p = 0 \quad \therefore T(n) = \theta(n^2)$

Q) $T(n) = 64T(n/8) + n^2 \log n$
$\log_8 64 < (k = 2) \quad \therefore T(n) = \theta(n^2 \log n)$

Q) $T(n) = 2T(n/4) + n \log n$
$\log_4 2 < 1$
$T(n) = \theta(n \log n)$

Q) $T(n) = 5T(n/3) + 6 \log n$
$\log_3 5 > 1 \quad T(n) = \theta(n^{\log_3 5})$

Divide & conquer

```
DAC(P) {
  if (small(P)) {
    S(P); }
  else {
    divide P into P1, P2, P3 ... Pk
    Apply DAC(P1), DAC(P2), ... DAC(Pk)
    Combine DAC(P1), DAC(P2), ... DAC(Pk) } }
```

quick sort

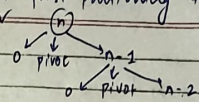
```
void quicksort (int arr[], int lo, int hi) {
  if (lo >= hi) return;
  int mid = (lo + hi)/2; int pivot = a[mid];
  int left = lo; int right = hig;
  while (left <= right) {
    if (arr[left] < pivot) left++;
    if (arr[right] > pivot) right--;
    if (left <= right) {
      swap(arr[left], arr[right]);
      left++; right--; } }
  quicksort (arr, lo, right);
  quicksort (arr, left, hig); }
```

* Not stable

Analysis of quicksort

① If pivot after 1st partioning is placed in the mid, then it is the best case
$T(n) = 2T(n/2) + n \quad T(n) = O(n \log n)$

② worst case; if after first partioning the pivot is either smallest or largest number



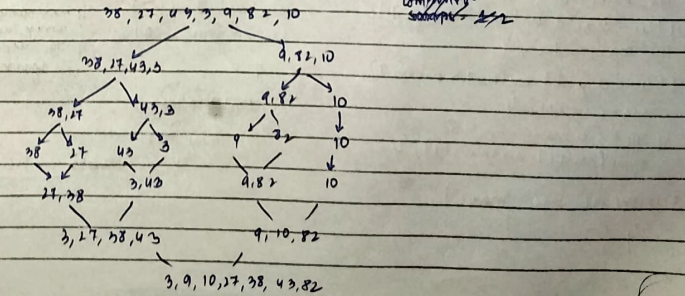
$T(n) = T(n-1) + n$
$T(n) = O(n^2)$

Merge sort

```
void merging (sorted array (int arr[], int left, int right, int mid) {
  int size1 = mid - left + 1;
  int size2 = right - mid;
  int arr1[size1], arr2[size2];
  for (i=0; i<size1; i++) {        for (j=0; j<size2; j++) {
    arr1[i] = arr[left+i];          arr2[j] = arr[mid+1+j]
  int i=0, j=0, k=0;
  while (i<size1 && j<size2) {
    if (arr1[i] < arr2[j]) {
      arr[k++] = arr1[i++]; }
    if (arr1[i] > arr2[j]) {
      arr[k++] = arr2[j++]; }
  if (i == size1)                  if (j == size2) {
    while (j < size2)                while (i < size1)
      arr[k++] = arr2[j++];            arr[k++] = arr1[i++]; }

void mergesort (int arr[], int lo, int hi) {
  if (lo >= hi) return;
  int mid = (lo + hi)/2;
  mergesort (lo, arr, lo, mid);
  mergesort (arr, mid+1, right, hi);
  merging (arr, lo, hi, mid); }
```

$T(n) = 2T(n/2) + n$
Best, Avg, Worst



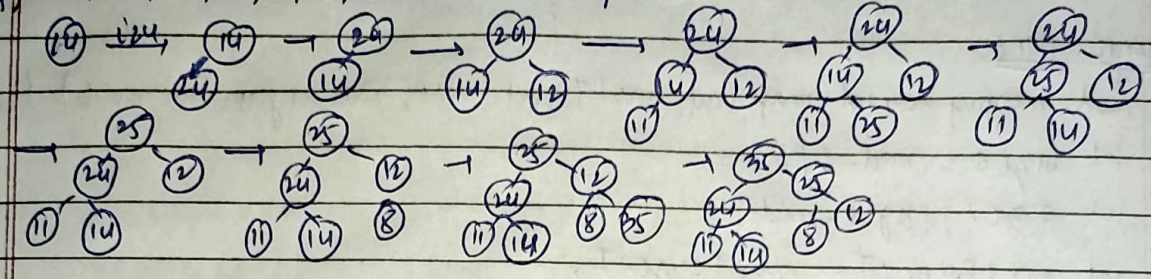
Pros of merge sort
(1) large size list
(2) works on L.L.
(3) external sorting
(4) stable

cons of merge sort
(1) extra memory (not inplace)

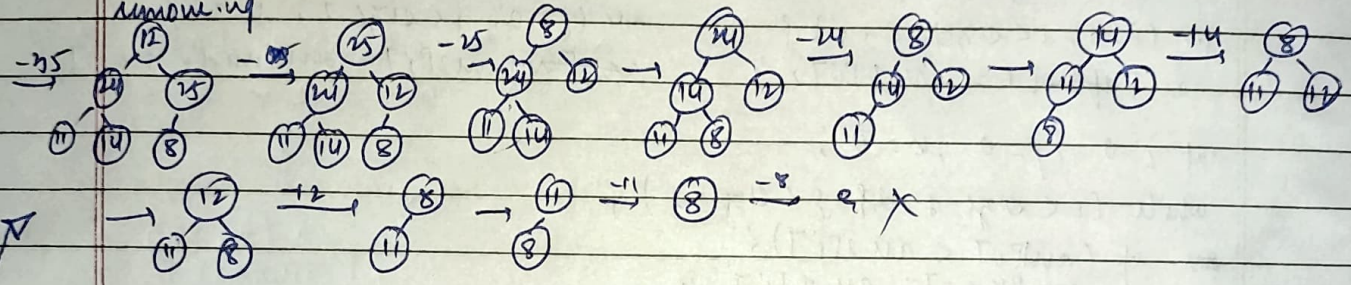
classmate
Date
Page

Introduction to Heaps :- CBT & heap order property

Q) 14, 24, 12, 11, 25, 8, 35 (Max heap)



removing



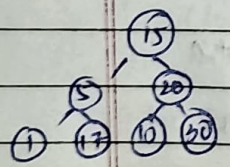
heap tree construction

insert key one by one in the given order (nlogn)

heapify method (O(n))

Heapsort :- building max heap from the array & deleting the max heap, we get in ascending order sorted elements.

initial array :- 15, 5, 20, 1, 17, 10, 30



```
void heapify(int arr[], int size, int pi){
int lci = 2pi+1; int rci = 2pi+2;
int maxi = pi;
if(lci < size && arr[lci] > arr[pi]){
    maxi = lci; }
if(rci < size && arr[rci] > arr[pi]){
    maxi = rci; }
if(pi != maxi){
    swap(arr[pi], arr[maxi]); }
heapify(arr, size, maxi);
}
```

```
void heapsort(int arr[], int n){
for(i=n/2-1; i>=0; i--)
    heapify(arr, n, i);          } building the heap

for(i=n-1; i>=0; i--){           } after deleting the max element from array we need to heapify again
    swap(arr[0], arr[i]);
    heapify(arr, i, 0);
}
```

T(n) = O(nlogn)
best avg worst
⇒ It is stable unstable

Greedy Algorithm :- an algorithm that follow the problem solving by making the local optimal choice at each stage hoping to find a global optimum

It works on :-

① **Greedy choice property** :- It makes locally optimal choice in the hope that this choice leads to globally optimal sol^n

② **optimal substructure** :- optimal solution contains optimal sub solution

problem

```
algo greedy (a, n)        input value
{
  for i:=1 to n do
  {
    x := select(a);
    if feasible(x) then
      solution = solution + x;
  }
}
```

here n is the no. of inputs
→ greedy problem is solved in stages so at each stage we consider one input from a given problem & if that input is feasible then we include it in the set" x", so by including all these inputs we get an optimal sol^n]

Knapsack Problem

weight profit no. of obj size of bag

```
fractional_Knapsack (wt[], pt[], int n, int m) {
  int profit = 0;
  float ratio[n];
  for(i=0; i<n; i++) {                    O(n)
    ratio[i] = Profit[i]/wgt[i]; }
  sort ratio[] in descending order          O(nlogn)
  for(i=0 to n-1) {
    if(M>0 & wt[i]<M) {                     O(n)
      Profit += p[i];
      M = M - wt[i]; }
    else break; }
  if(M>0) {
    profit += (M / wt[i]) * pt[i]; }
  return profit }
```

O(n) + O(nlogn) + O(n) = O(nlogn)

Job sequencing Problem (with deadline)

→ set of n jobs, each has a deadline & profit associated with it, assuming it take unit time to perform a job, find max^m profit

Q)

Job	A	B	C	D	E
Deadline	2	1	2	1	3
Profit	100	19	27	25	15

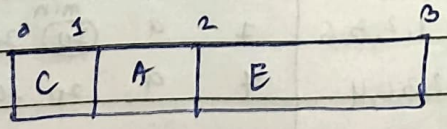
→ Sort in dec order of profits

Job	A	C	D	B	E
Profit	100	27	25	19	15
deadline	2	2	1	1	3
			×	×	

max^m deadline = 3

Try to place the jobs in the chart as far as possible but within the deadline

→ If a job posn is preoccupied look for place before it, if found place the job at that time



8) n=7, (P1 ... P7) = (3, 5, 13, 10, 16, 1, 38), (d1 ... d7) = (1, 3, 3, 4, 1, 2, 1)

Job	7	4	3	5	2	1	6
Profit	38	20	18	6	5	3	1
deadline	1	9	3	1	3	1	2

max deadline = 4

0	1	2	3	4
J7	J5	J3	J4	

38 + 6 + 18 + 20 = 82

Huffmann coding :- data compression algorithm (lossless). The idea is to assign variable-length code to input chars, the length of code are based on the freq of char.

algorithm :- ① Calculate freq. of each char in the string

② Sort the char. in the increasing order of freq, stored in a priority queue.

③ make each unique character a leaf node.

④ Create an empty node z. Assign the min freq to left child of z & second min to right child of z. z value = sum of the child freq.

⑤ Remove these two min freq from Q & add the sum into Q.

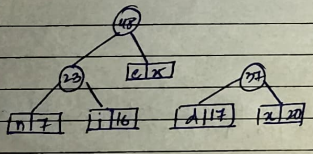
⑥ Insert node z into the tree.

⑦ Repeat ④, to ⑥ for each char.

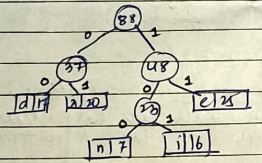
⑧ For each non-leaf, assign 0 to the left edge, 1 to right edge.

9) i - 16, n - 7, d - 17, e - 25, x - 20, code of 'x'

1) 7, 16, 17, 20, 25 2) 17, 20, 23, 25 3) 25, 25, 37

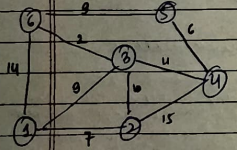


∴ x = 01



Single-source shortest path

① Dijkstra's algorithm



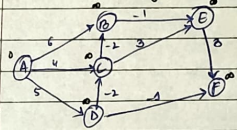
source 1	destination 2	3	4	5	6
	∞	∞	∞	∞	∞
	7 (min)	9	∞	∞	14
1,2	7	9 (min)	22	∞	14
1,2,3	7	9	20	∞	11 (min)
1,2,3,6	7	9	20 (min)	20	11
1,2,3,6,4	7	9	20	20	11

∴ shortest path {1, 2, 3, 6, 4, 5}

Dijkstra Algo (Graph, source) {
  create visited & set Q (min heap)
  for each vertex V in graph
    dist[V] = ∞
    add V to Q (build heap)
  dist[source] = 0
  while Q is not empty
    U = extract-min[Q]
    for each neighbour v of u
      if dist(v) < (graph(u,v) + dist(u)) {
        dist(v) = graph(u,v) + dist(u); }

② Bellman-Ford algorithm : every edge needs to be relaxed (V-1) times.

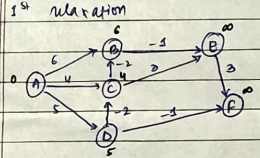
8)



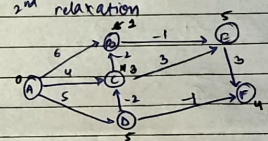
V = 6 relaxations = 5 req.

(AB), (AC), (AD), (BE), (EC), (DC), (DF), (EF), (CD)

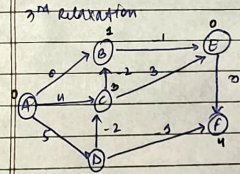
1st relaxation



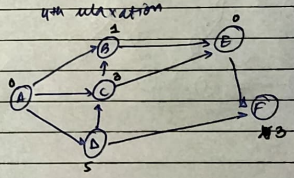
2nd relaxation

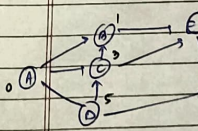


3rd relaxation



4th relaxation





A → 0
B → -1
C → 3
D → 5
E → 0
F → 3

T(n) = O(EV)

e edges need to be relaxed V times

in complete graph E = V²

∴ T(n) = O(n³)

Avg case T(n) = O(n²)

T(n) = O(E log V)

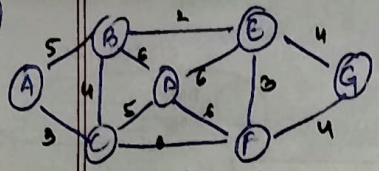
Drawbacks

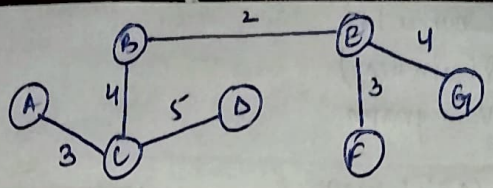
→ doesn't work appropriately for -ve weighted edges

Kruskal algo :- minimum spanning tree

→ pick min^m weight edge, check for cycles, if no cycle, add the edge to the MST

8)





Kruskals (G, w) {

A = ∅

for each vertex v ∈ V[G]

do make-set (v)

sort the edges of E in inc. order by weight w

for every edge (u, v) ∈ E taken in ascending order

do if find-set (u) ≠ find-set v

A = A ∪ {(u, v)}

UNION (u, v)

Return A; }

Prim's Algo :- start from any vertex, from the neighbours, select the min^m edged vertex