28/7/22
Thursday

2. D&C, Greedy Method

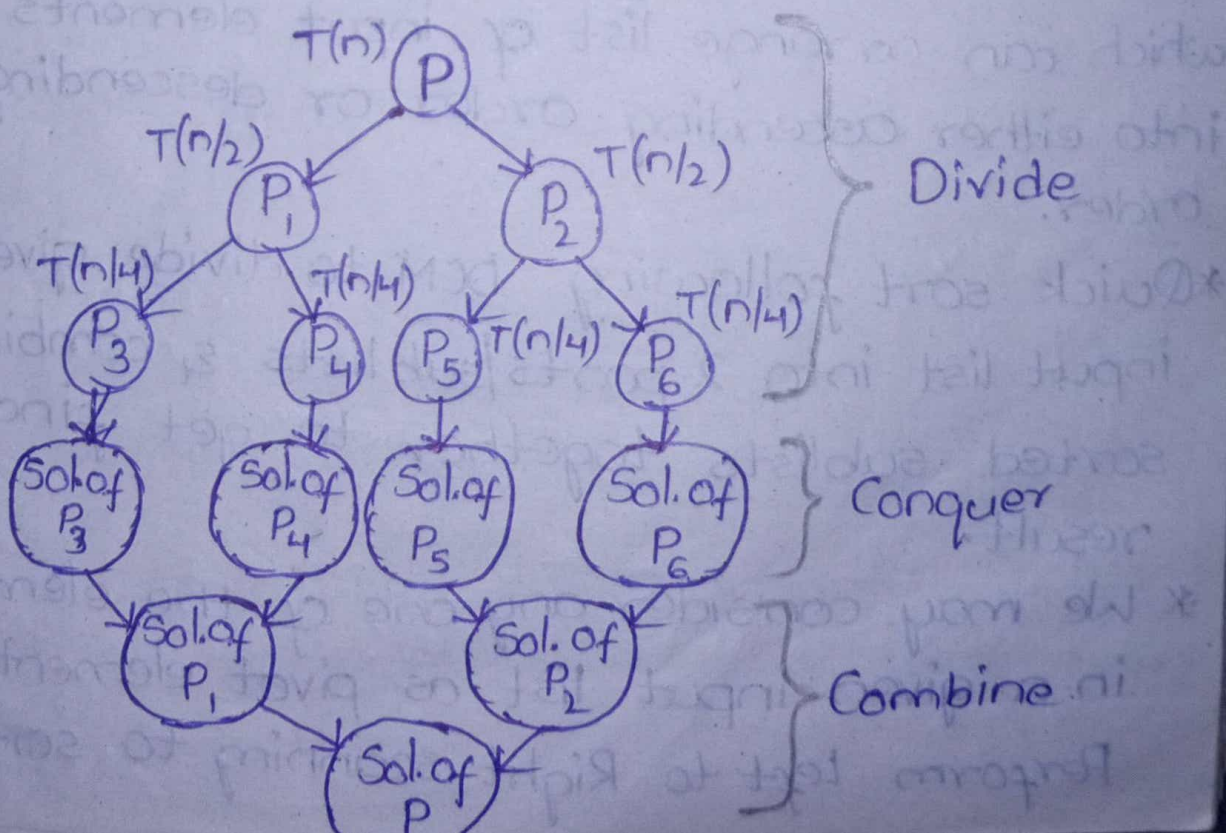
Divide and Conquer Technique/Method:

General method:

DC method is used to solve high complexity problems like P which can be divided into sub-problems like $P_1, P_2, \ldots P_n$, where every sub-problem will be solved individually and combine all the solutions together to give the solution for main problem P.

The following operations need to be implemented in DC method:

1. Divide (Dividing the problem into sub-problems)
2. Conquer (Solving individual sub-problems)
3. Combine (Combining sol^n of sub-problems)



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

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

This is the RR for only two sub-prob general

So, the RR is:

$$T(n) = aT(n/b) + f(n)$$

where a = no. of sub-problems

$T(n/b)$ = Time complexity of each sub-problem.

$f(n)$ = Time required for dividing the problem & combining the sol^{n}s.

Applications of DCM:

- Binary search
- Quick sort
- Merge sort

Quick sort:

* Quick sort is effective sorting technique which can arrange list of input elements into either ascending order or descending order.
* Quick sort following DCM to divide given input list into 2 parts/sublists & combine sorted sublists together to get final result.
* We may consider any one of the element in a given input list as pivot element. Perform Left to Right scanning to sort

or to arrange all the smallest elems than pivot, scan Right to Left to arrange all largest elems than pivot in order.

1. Pivot → any element

2. i L → R } Find smaller elem comp p
   i++

   j ~~R → R~~
   j L ← R } Find larger elem
   j--

3. (i<j) → swap a[i] & a[j]
   else → swap a[j] & pivot

Quicksort partition algorithm:

```
while(i<j)
{
  while(a[i]<=p)
  {
    i++;
  }
  while(a[j]>p)
  {
    j--;
  }
  if(i<j)
    swap a[i] & a[j]
  else
    swap a[j] & pivot. Divide into 2 parts
                          ←P→
}
```

Arrange the following list of elements into ascending order using quick sort & find out its Time complexity. 48, 7, 13, 21, 25

Ans: i=0; j=4, pivot = 13

while(i<j)
while(0<4) ✓
while(a[i]<=p)
a[0]<=p
48<=13 ✗

while(a[j]>p)
a[4]>p
25>13 ✓
j-- ⇒ j=3.
while(a[j]>p)
a[3]>p
21>13 ✓
j-- ⇒ j=2
while(a[j]>p)
a[2]>p
13>13 ✗

Now, i=0, j=2
if (i<j)
0<2 ✓
swap a[i] & a[j]
a[0] & a[2]

So,

13	7	48	21	25

~~while~~ i=0; j=2; p=13
while(i<j)
0<2 ✓

while(a[j]>=p)
a[0]<=p
13<=13 ✓
i++ ⇒ i=1
while(a[i]<=p)
a[1]<=p
7<=13 ✓
i++ ⇒ i=2
while(a[i]<=p)
a[2]<=p
48<=13 ✗

while(a[j]>p
a[2]>p
48>13 ✓
j-- ⇒ j=1
while(a[j]>p)
a[1]>p
7>13 ✗

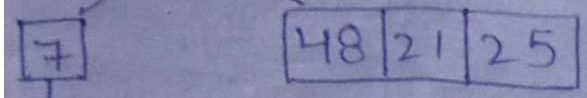
Now i=2, j=1
if(i<j)
2<1 ✗
else:
swap a[j] & pivot &
divide: a[1] & pivot
7 & 13

So,

7	13	48	21	25

Dividing the list:

7 13 48 21 25



only one ele

So, 48 21 25

i=0, p=25, j=2

while (i<j)

0<2 ✓

---

while(a[i]<=p)

a[0]<=p

48<=25 X

---

while(a[j]>p)

a[2]>p

25>25 X

Now i=0, j=2

if (i<j)

0<2 ✓

swap a[i] & a[j]

a[0] & a[2]

So, 25 48 21

---

i=0, p=25, j=2

while(i<j)

0<2 ✓

---

while(a[i]<=p)

a[0]<=p

25<=25 ✓

i++ ⇒ i=1

while(a[i]<=p)

a[1]<=p

48<=25 X

---

while(a[j]>p)

a[2]>p

21>25 X

Now, i=1, j=2

if (i<j)

1<2 ✓

swap a[i] & a[j]

a[1] & a[2]

So, 25 21 48

---

i=1, j=2, p=25

while(i<j) ⇒ 1<2 ✓

---

a[i]<=p ⇒ a[1]<=p

21<=25 ✓

i++ ⇒ i=2

a[i]<=p ⇒ a[2]<=p

48<=25 X

---

a[j]>p ⇒ a[2]>p

48>25 ✓ j--

a[j]>p ⇒ a[1]>p ⇒ 21>25 X

Now i=2, j=1

i<j X swap a[j] & pivot

21 25 48

∴ 7 13 21 25 48 ✓

QuickSort Worst Case Analysis:

- Worst case analysis of quick sort depends on the arrangement of elements.
- If all the eles. are in sorted order & if consider last element has pivot element recursively then we need to perform (n-1) no. of elements execution in level-1, (n-2) no. of executions in level-2, - - - - - -

  n, n-1, n-2, n-3, - - - - 1
  ↓
  Total no. of eles.

Eg: 51, 81, 86, 98, 100

n [51 | 81 | 86 | 98] [100] Pivot — We ignore it.
sublist-1 sublist-2

n-1 [51, 81, 86] [98]
pivot

n-2 [51, 81] [86]
pivot

[51] [81]
pivot

1 [51]
pivot

$\therefore T(n) = T(n-1) + n$
↳ no. of divisions/integrations of sols.

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

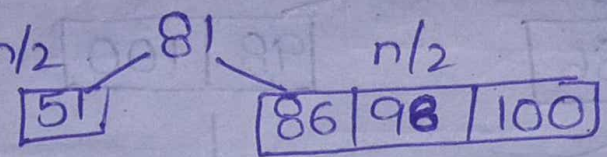
Randomised Quicksort:

Randomised Quicksort considers some random number to identify pivot elements position in the given input. Random number

should be other than last index of the given input.

51 81 86 98 100

P



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

$= O(\log_2 n)$

Used to overcome the drawback in normalized worst-case quicksort.

Min-Max Algorithm using Divide & Conquer Technique:

Procedure:

Step-1: Identify total no. of elements in the given input & assign that to a variable i.e. 'n'.

Step-2: Find mid value to divide the list into two sublists based on low & high values.

mid = (low+high)/2.

Step-3: Use comparision operator to compare min. & max. values in each sublist and update in the MIN & MAX.

Step-4: Repeat steps-2 & 3 recursively to find final result of MIN & MAX values.

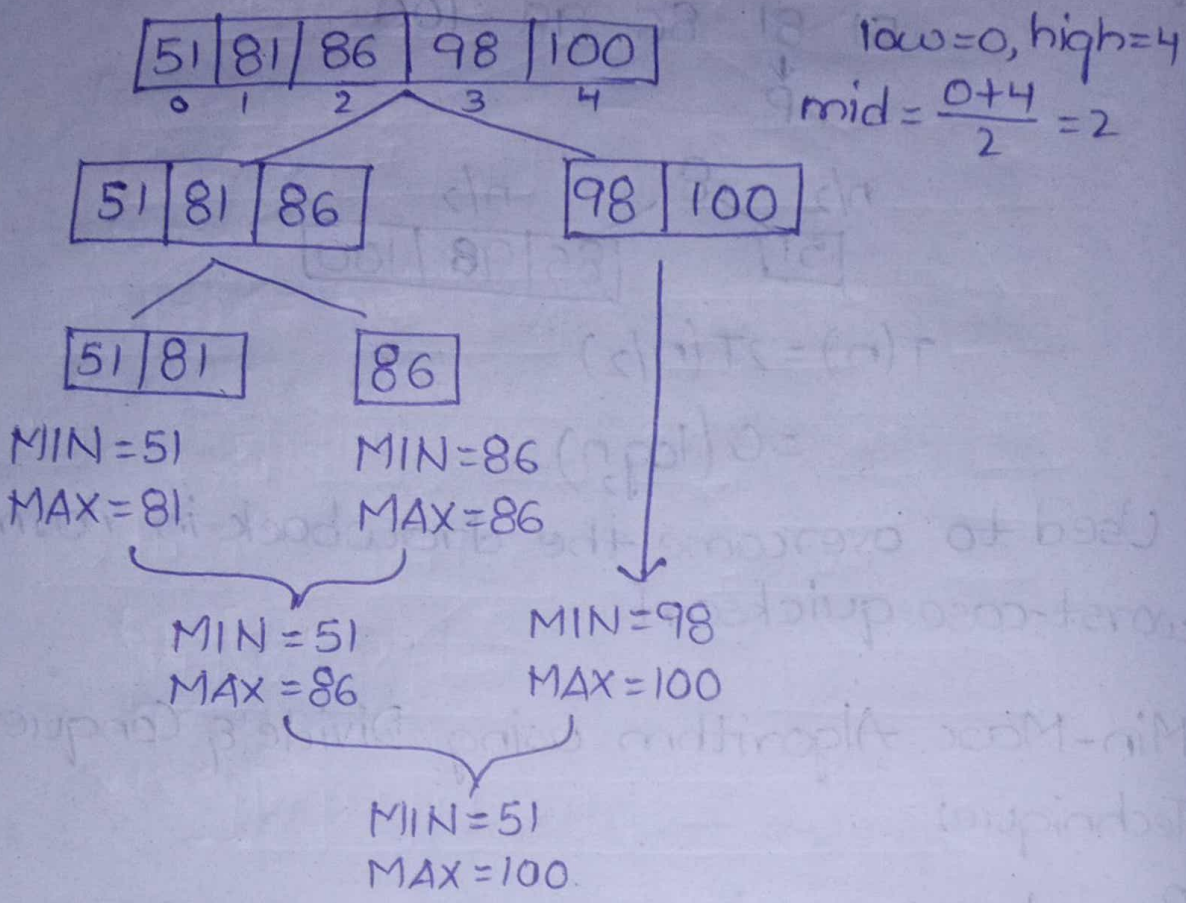
Div Policy:

D to mid } sublist-1

mid+1 to n-1 } sublist-2

Div policy is to be implemented till each sublist contains almost two eles.

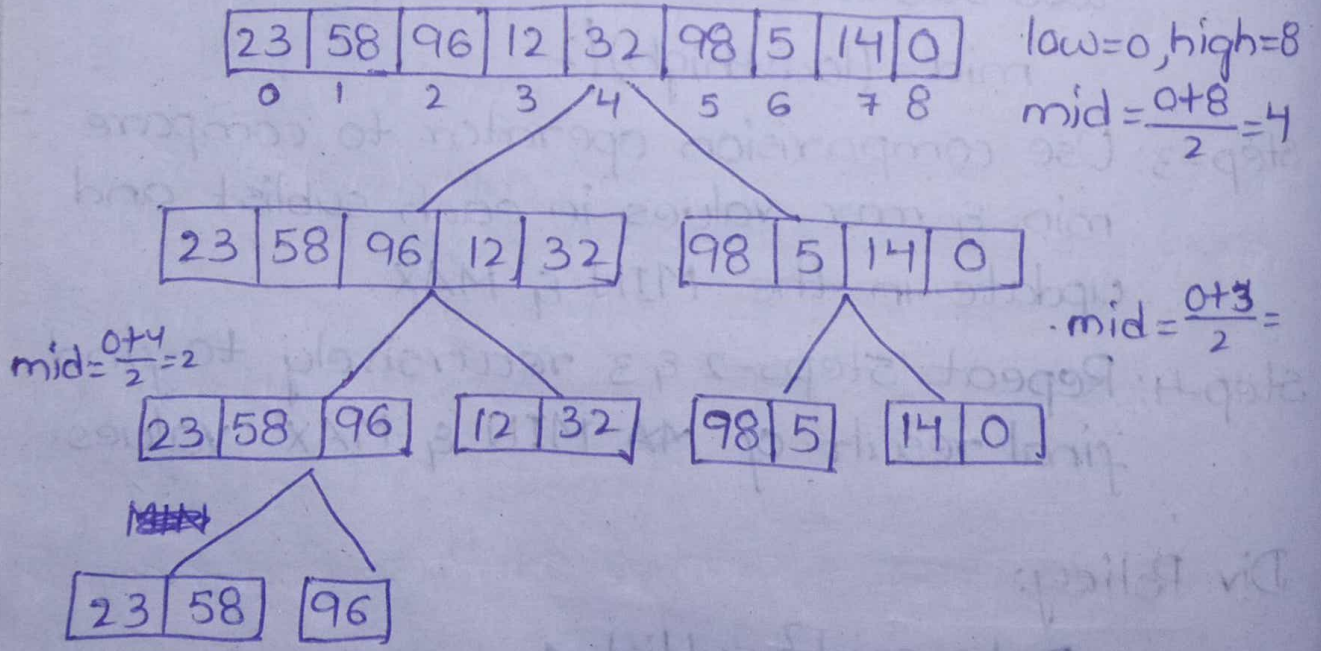
Eg:

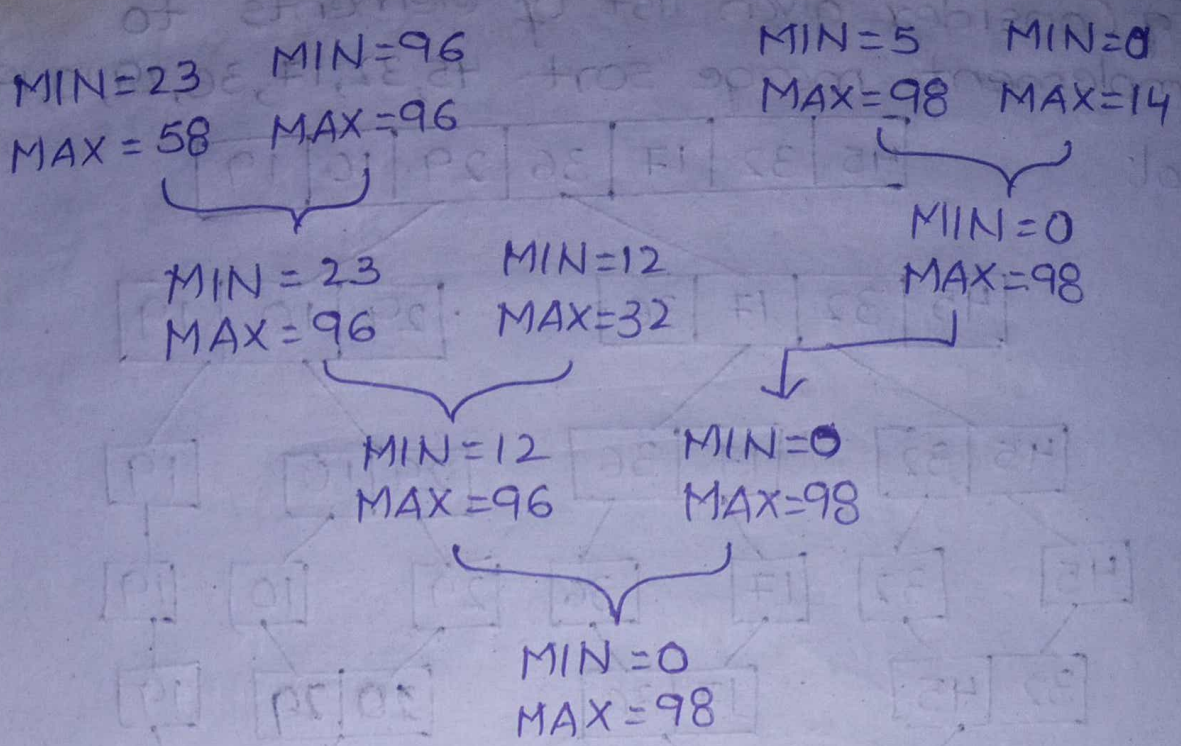


Eg: Consider the following list of eles. to find MIN & MAX values using DC Technique.

23, 58, 96, 12, 32, 98, 5, 14, 0.

Given no. of eles. = 9





Div Time → 1
Com-Time → 1
} 2

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

Merge Sort:

Step-1: Identify no. of elements in the given input & store the value in N.

Step-2: Calculate mid value based on low & high values in the given list.

Step-3: Divide the entire list into two halves (ie. index=0 to mid & index=mid+1 to n-1).

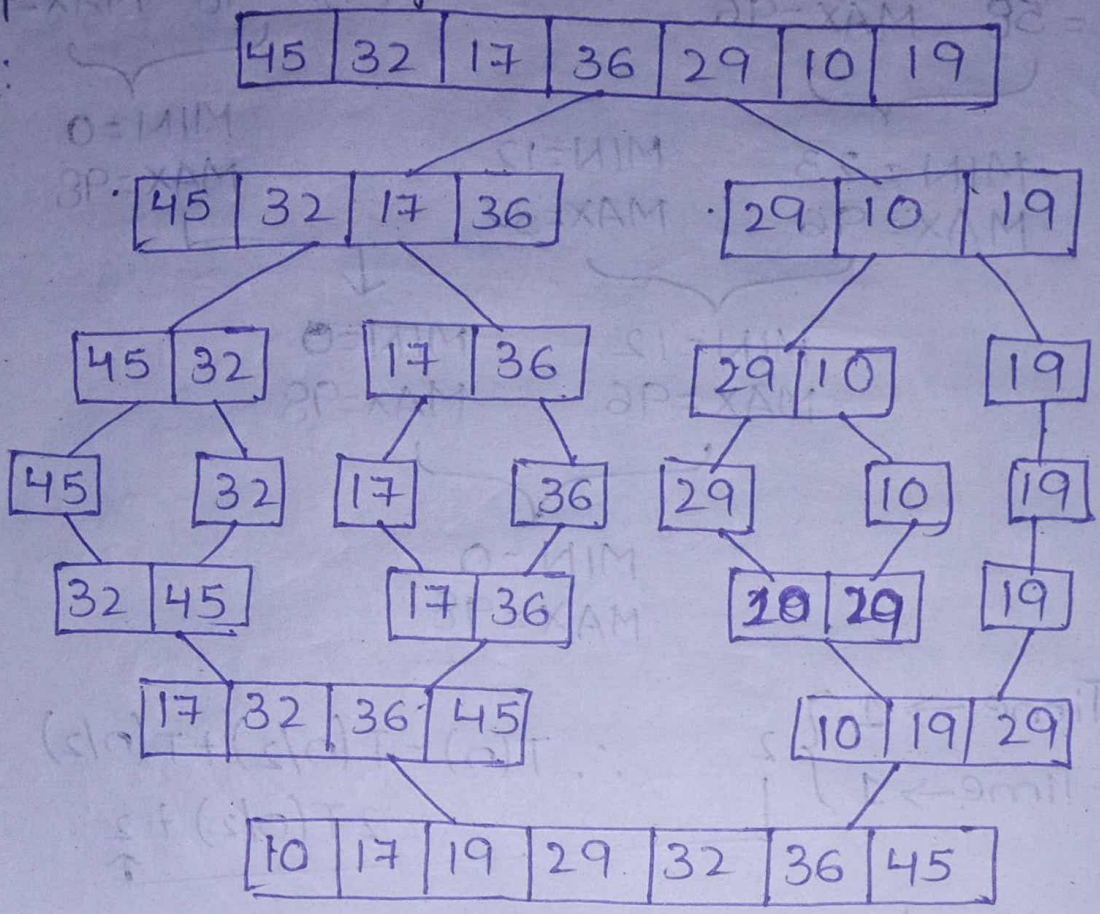
Step-4: Repeat step-3 until we reach to a single element in sublists.

Step-5: Compare each sublist element with other sublist element & perform swapping operation to arrange elements in sorted order.

Step-6: Repeat step-5 recursively from LHS to RHS sublists in order.

Eg: Consider given list of elements to implement merge sort 45, 32, 17, 36, 29, 10, 19.

Sol:



Time Complexity Analysis of Merge sort:

Time Complexity of merge sort for both sorted and unsorted list of elements can be given as -

$O(n \log_2 n)$ (best case, average case, worst case)

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

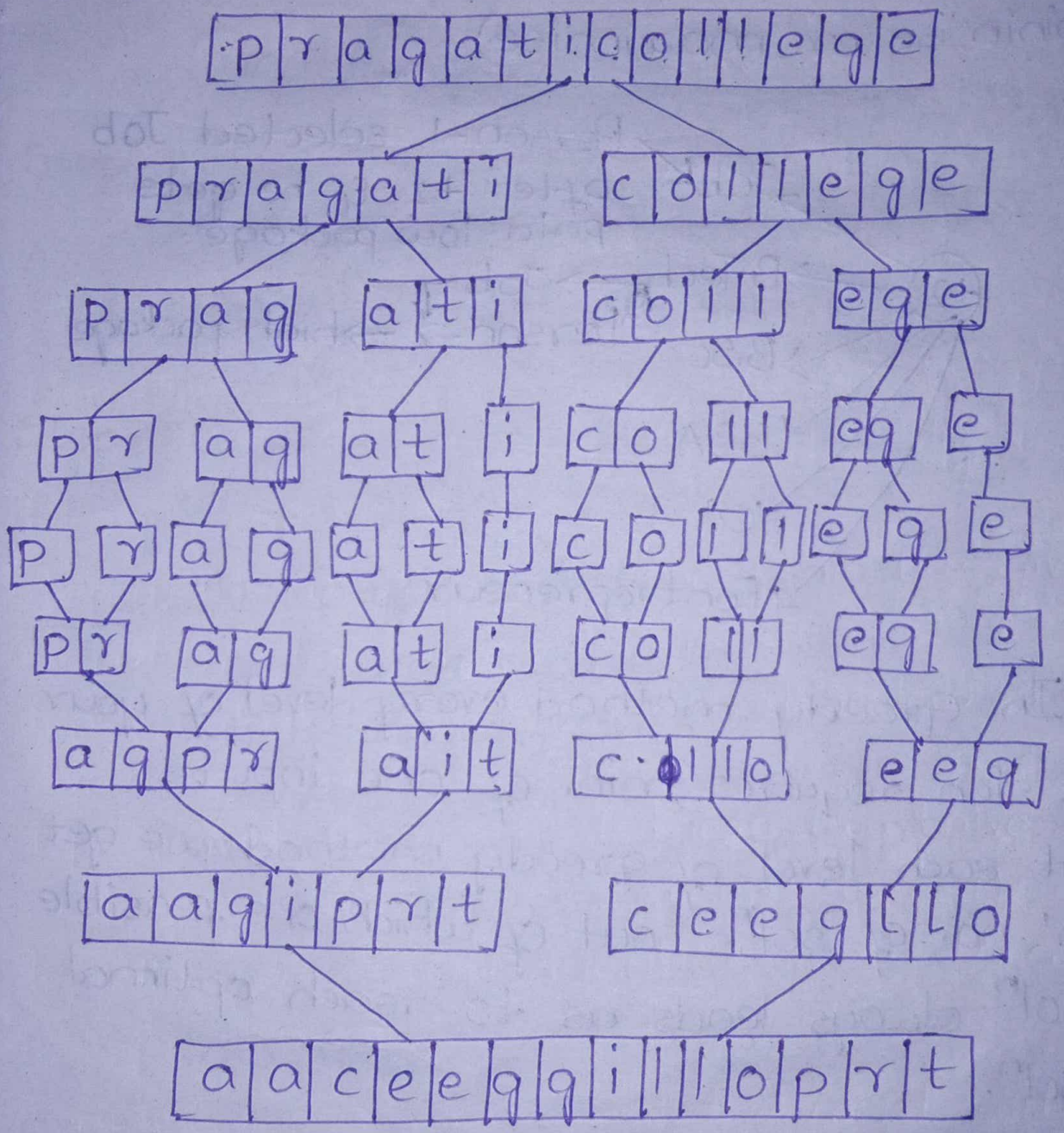
$\therefore O(n \log_2 n)$

where $T(n/2)$ indicates time complexity of first sublist & second sublist, n indicates total time required to divide the list into two halves to identify the sol^n and to merge each subsol^n to get final answer.

Recurrence Relation of merge sort can be given as -

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

1. Consider the following string to implement merge sort & identify how many total no. of sublists are available on left and right hand side sublists. String is "PragatiCollege".



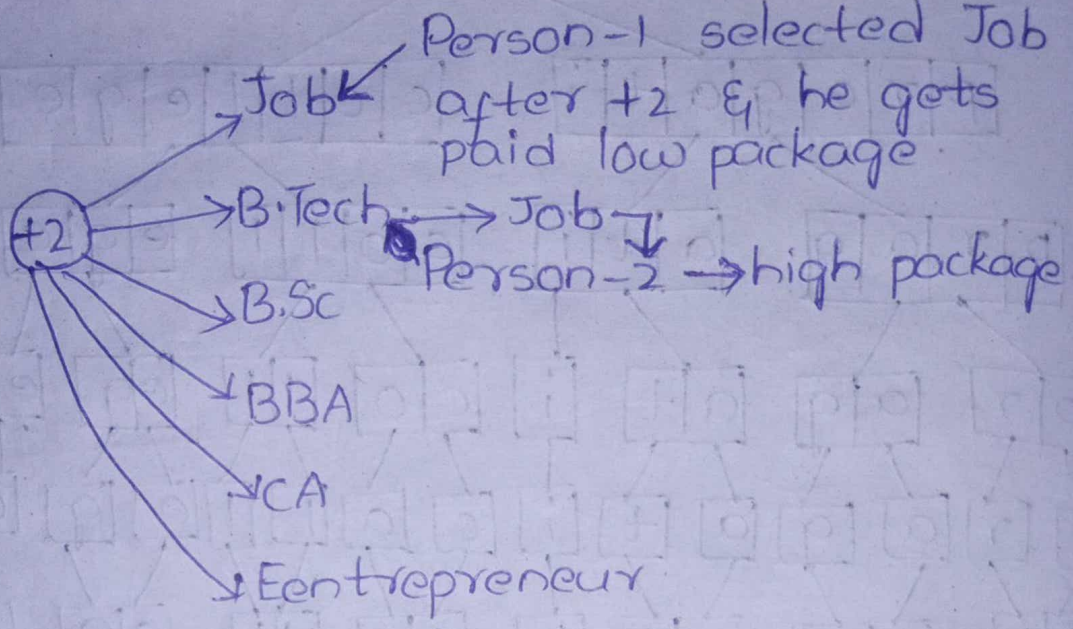
∴ After sorting the string is-

aaceeggilloprt.

Greedy method:

* Greedy method is a step-by-step procedure to take hierarchial decisions while solving any problem statement P which has optimal solution (either to minimise or maximise)

Eg:



* In Greedy method every level of your decision requires min. of one input.
* At each level of greedy method we get 'n' no. of sol^n's out of which one feasible sol^n always leads us to reach optimal sol^n.

B.Tech → Job
Feasible sol^n Optimal sol^n

Applications of Greedy method:

* Knapsack problem / Fractional knapsack:

Procedure:

Step-1: Identify the no. of elements and bag size/knapsack size.
n → no. of elements
M → bag size.

Step-2: Identify the sequence of items to be placed in knapsack by using (profit/weight) calculation.

Step-3: Place the items one after another into the knapsack either fully or partially based on the current availability of knapsack size.

Step-4: Calculate max. profit and check max weight of the knapsack is fulfilled.
Max. profit = $\sum x_i P_i$ where x_i is item placed status of item placed which ranges from 0 to 1, where 0 indicates unable to place the item and 1 indicates complete placement of the item. All fractional numbers indicates partial placement of items into knapsack.

Eg1: Consider the following information to implement -

Item -	1	2	3	4	5	6
Profit	10	19	28	63	7	23
Weight	17	69	6	25	26	36

M = 150

Step-1: Calculation of P/w ratio:

Item:	1	2	3	4	5	6
P/w :	10/17	19/69	28/6	63/25	7/26	23/36
	=0.58	=0.27	=4.6	=2.5	=0.26	=0.638

Step-2:

* Select I_3 to place in knapsack.
* Wt. of I_3 = 6.
  Latest status of ks = 150 − 6 = 144
* Select I_4 to place in ks.
  Wt. of I_4 = 25.
  Latest status of ks = 144 − 25 = 119.
* Select I_6 to place in ks.
  Wt. of I_6 = 36.
  Latest status of ks = 119 − 36 = 83
* Select I_1 to place in ks.
  Wt. of I_1 = 17
  Latest status of ks = 83 − 17 = 66.
* Select I_2 to place in ks.
  Wt. of I_2 = 69.
  But status of ks = 66.
  So, I_2 can only be placed partially.
* Select I_5 to place in ks.
  Wt. of I_5 = 26.
  But status of ks = 0.
  So, I_5 cannot be placed at all.

$x_1 = 1$ (fully placed)

$x_2 = x$ (fully placed) 66/69 (partially placed)

$x_3 = 1$ (fully placed)

$x_4 = 1$ (fully placed)

$x_5 = 0$ (not placed at all)

$x_6 = 1$ (fully placed)

$$\sum x_i P_i = (1 \times 10) + (66/69 \times 19) + (1 \times 28) + (1 \times 63) + (0 \times 7) + (1 \times 23)$$

$$= 142.17$$

$$\sum x_i W_i = (1 \times 17) + (66/69 \times 69) + (1 \times 8) + (1 \times 25) + (0 \times 26) + (1 \times 36)$$

$$= 150$$

$$\sum x_i W_i <= M \checkmark$$

Eq-2:

Item	1	2	3	4
Profit	16	20	21	28
Weight	3	8	6	9

$M = 25$

Step-1: P/w ratio:

Item:	1	2	3	4
P/w :	16/3	20/8	21/6	28/9
	=5.33	2.5	3.5	3.11

Step-2:

* Select I_1.

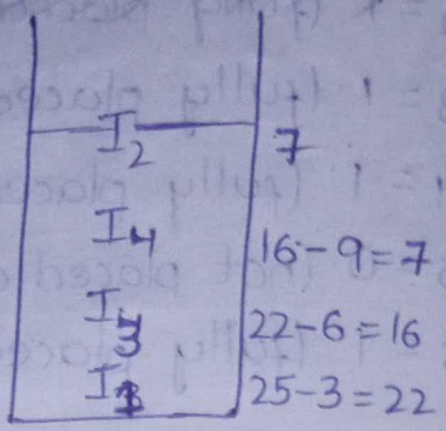
Wt. of $I_1 = 3$

Latest status of ks = 25-3 = 22

* Select I_3

I_3 wt. = 6

ks = 22-6 = 16



* Select I_4

I_4 wt. = 9

ks = 16-9 = 7

* Select I_2

I_2 wt. = 8

ks = 7

So, can be partially placed

$X_1 = 1$

$X_2 = 7/8 = 0.875$

$X_3 = 1$

$X_4 = 1$

$\Sigma X_i P_i = (1 \times 16) + (7/8 \times 20) + (1 \times 21) + (1 \times 28)$

$= 82.5$

$\Sigma X_i W_i = (1 \times 3) + (7/8 \times 8) + (1 \times 6) + (1 \times 9)$

$= 24$

$\Sigma X_i W_i <= M$

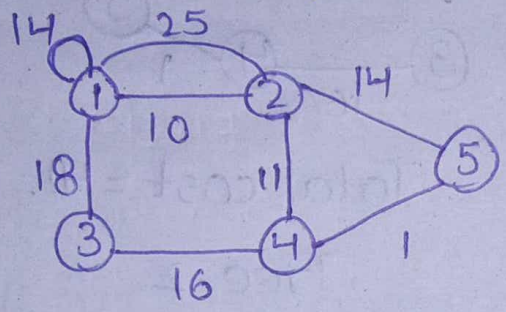
$24 <= 25$ ✓

Spanning Tree:

*A spanning tree P is a subgraph which contains all the vertices connected with min. no. of edges, Every spanning tree must not contain cycles, parallel edges and loops.

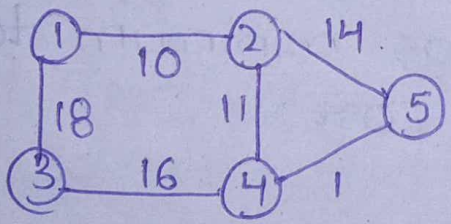
*A spanning tree has connectivity b/w one vertex to any other vertex.

Eg:

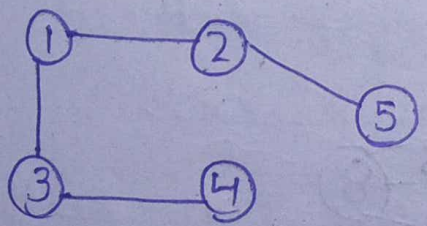


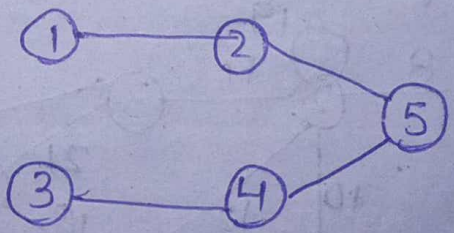
Eq.

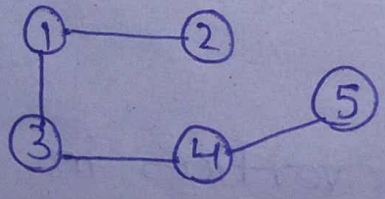
Removing the loops & parallel edges.

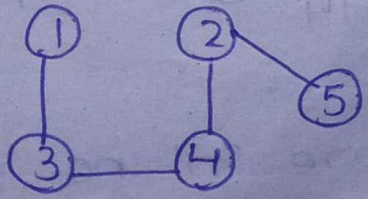


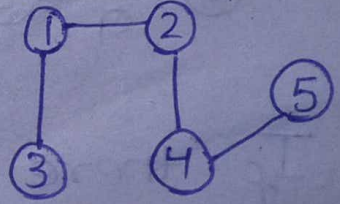
Spanning Trees:





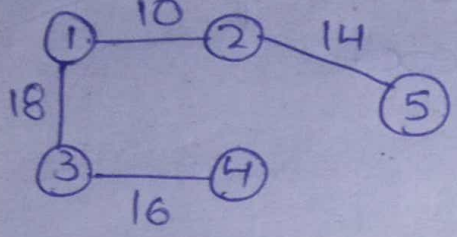






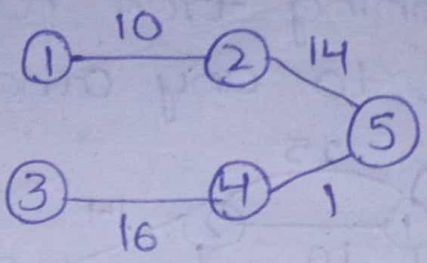
Minimum Cost Spanning Tree:

Any spanning tree can be called as minimum cost spanning tree if it has no cycles and minimum total cost ($E_1+E_2+E_3+---E_n$).



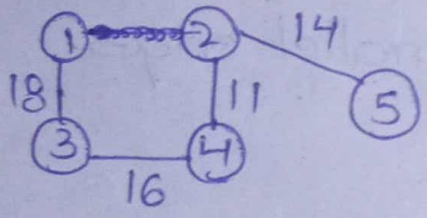
Total cost = 58

Tree-1.



Total cost = 41

Tree-2

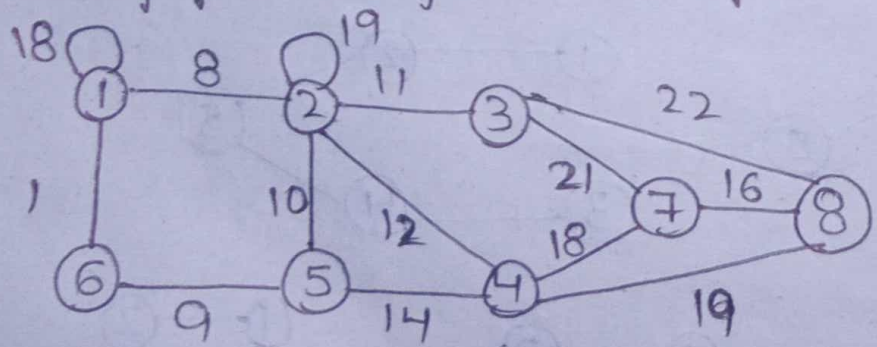


Total cost = 59

Tree-3

So, the Tree-2 is called MST since it has minimum total cost.

Identify MST for the given graph.



* If there are 'n' no. of vertices in given graph 'G' then we can extract

$[n^{(n-2)}]$ no. of spanning trees out of which only one is our required MST.

* To extract MST directly from the given graph G we can use the following algorithms -
  - Prims algorithm
  - Kruskals algorithm.

Prims algorithm:

Procedure:

Step-1: For the given graph G eliminate loops, parallel edges.

Step-2: Identify starting point or source point.

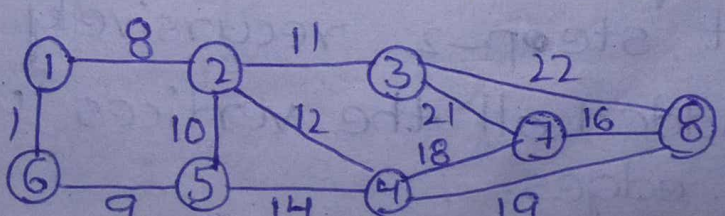
Step-3: Identify min. cost edge from the source point to move from current state to next state recursively.

Step-4: Identify total cost by adding all the edge weights in a final spanning tree structure.

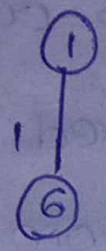
Eg: Consider the following graph.

Sol:

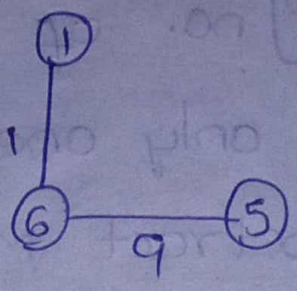
Step-1: Eliminate loops & parallel edges:



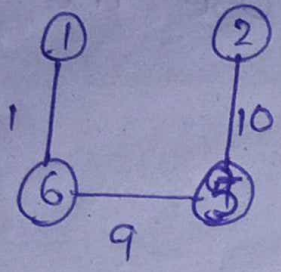
Step-2:



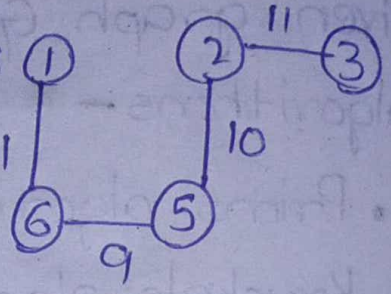
Step-3:



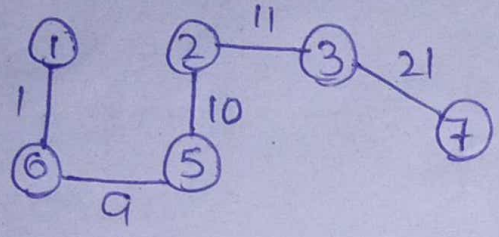
Step-4:



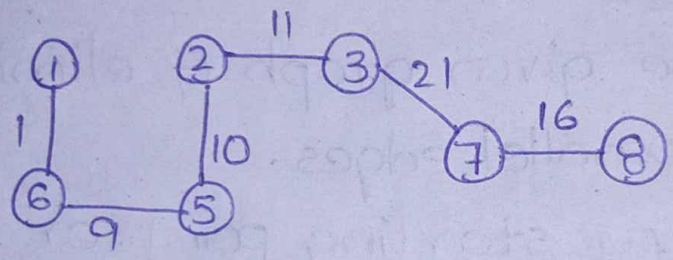
Step-5:



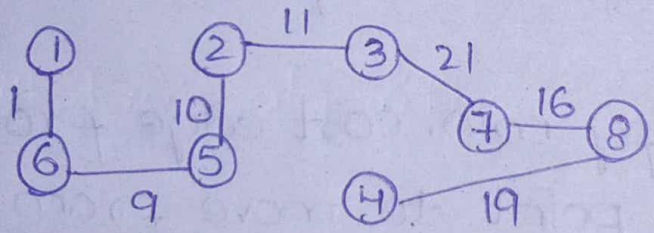
Step-6:



Step-7:



Step-8:



Kruskals algorithm:

Step-1: For the given Graph G eliminate all the loops & parallel edges.

Step-2: Identify min. cost edge in given graph G and add it to soln of spanning tree.

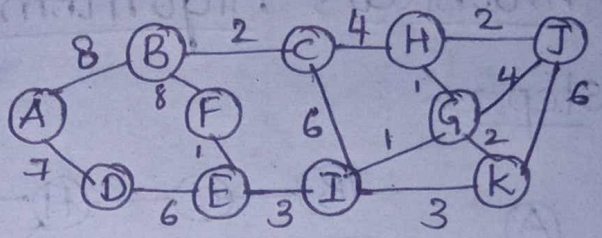
Step-3: Repeat steop-2 recursively until you added all the vertices with min. no. of edges.

Step-4: Identify total cost by adding all the edge weights in a final spanning tree structure.

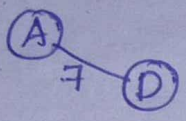
17/8/22 Wednes

Consider the given graph G to identify minimum cost spanning tree using both Prim's and Krushkal's algorithm.

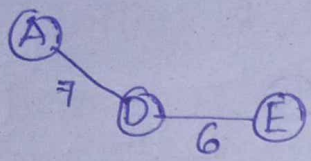
(i) Prim's Algorithm:-



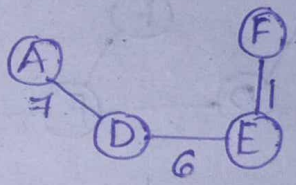
Step 2:



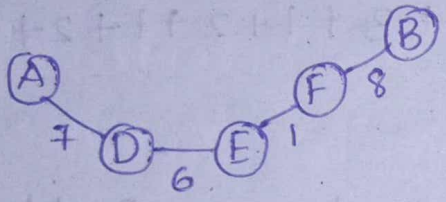
Step 3:



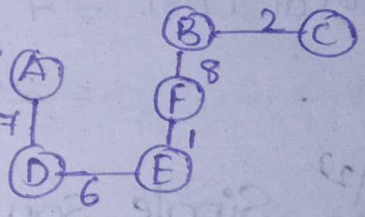
Step 4:



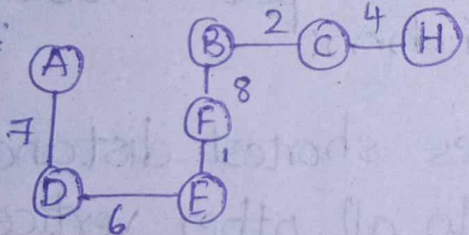
Step 5:



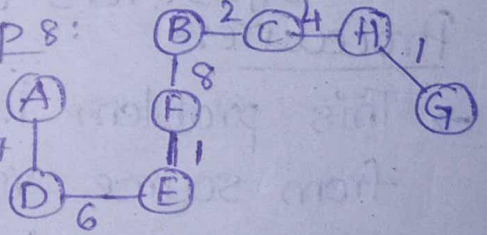
Step 6:



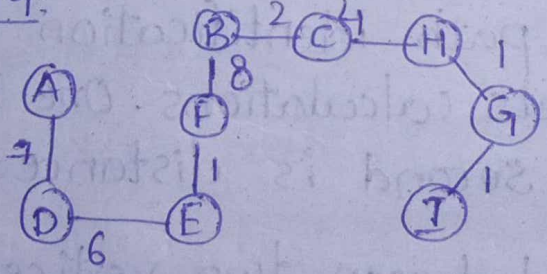
Step 7:



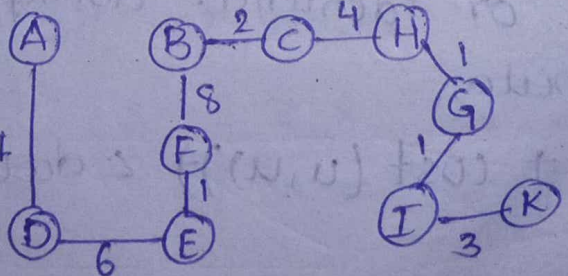
Step 8:



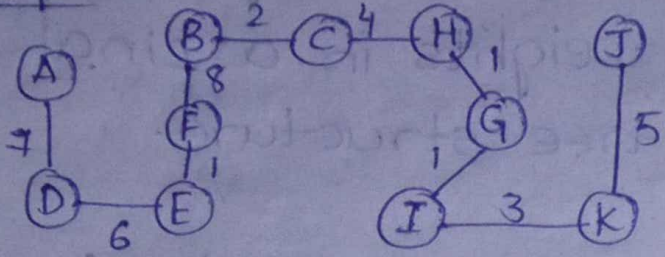
Step 9:



Step 10:



Step 11:-

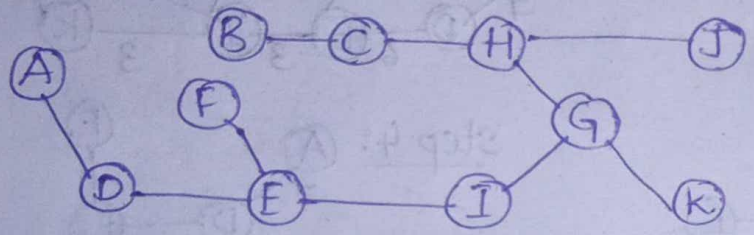


Total cost = 7 + 6 + 8 + 2 + 4 + 1 + 1 + 3 + 5

= 38

(ii) Krushkal's Algorithm:-

Step 2:-



Total cost = 7 + 6 + 1 + 3 + 1 + 2 + 1 + 2 + 4 + 2

= 29

24/8/22 wedn

Single source shortest path Problem/- Dijkstra's algorithm:

Procedure:-

→ This problem identifies shortest distance from source vertex to all other vertices in a given graph G.

→ Shortest distance path identification depends on two sets calculations. One is shortest path set, second is distance set.

→ shortest distance between two vertices(u,v) can be relaxed or identified using the following formula,

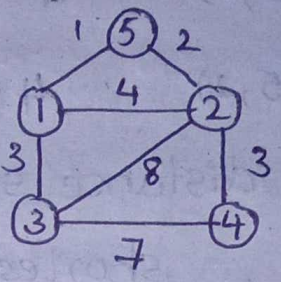
if ((dw(u) + cost(u,u)) < dw(v)

then

$$d_w(v) = d_w(u) + cost(u+v)$$

→ Identify minimum values recursively from distances set to choose upcoming vertex on source to destination path.

Ex: Consider the following graph to implement single source shortest path problem.



Sol: Step 1: Total no. of vertices = 5

Total no. of edges = 7

Let source vertex = 3

Step 2: Initial shortest path = { }

Before choosing source vertex, distance set = {∞, ∞, ∞, ∞, ∞}

After choosing, distance set = {∞, ∞, 0, ∞, ∞}

Step 3: (1,1) (1,2)

4+1 < 3

3, 2

4+2 < 8

Initial Start = 3 adjacent vertexes = 1, 2, 4

(3,1) 0+3 < ∞ → $d_w(1) = 3$

(3,2) 0+8 < ∞ → $d_w(2) = 8$

(3,4) 0+7 < ∞ → $d_w(4) = 7$

∴ Shortest path = {3} distance set = {③, 8, ⓪, 7, ∞}

least

∴ Shortest path = {3,1}

Now we are at vertex '1', adjacent vertexes = 2, 5

(1,2) 3+4 < 8 → $d_w(2) = 7$

(1,5) 3+1 < ∞ → $d_w(5) = 4$

Shortest path = {3,1}

distance set = {③, 7, ⓪, 7, ④}

∴ shortest path = {3,1,5}

Now we are at vertex '5'. The adjacent vertices = 2,1

(5,1) $4+1 \not< 3 \rightarrow d_w(1) = 3$

(5,2) $4+2 < 7 \rightarrow d_w(2) = 6$

distance set = {③, ⑥, ⓪, 7, ④}

shortest path = {3,1,5,2}

Now we are at vertex = 2, adjacent vertex = 4

(2,4) → $6+3 \not< 7 \rightarrow d_w(4) = 7$

distance set = {③, ⑥, ⓪, ⑦, ④}

shortest path = {3,1,5,2,4}

shortest distance from 3 to 1 is through direct connection cost = 3

For 3 to 2, 3 → 1 → 5 → 2

total cost = 6

Unit-2

Optimal merge patterns:

Any problem P can be solved using some methodology (D&C or greedy method or dynamic programming or backtracking) which leads to either Feasible sol^n or Optimal sol^n or Enumerated sol^n.

Feasible ⟶ {- - - - -}

↓filter(minimization/maximization)

Optimal ⟶ {0}

Enumerated ⟶ {more than 1 exact sol^n}.

Procedure:

Step-1: Consider given file length and identify total value/count.

Step-2: Sort given list of file lengths in an ascending order.

Step-3: Consider first two file lengths from the sequence & merge them to get intermediate solution node.

Step-4: Repeat step-3 until we complete to reach the single value count.

Step-5: Construct tree data structure to represent or denote optimal merge pattern sequence.

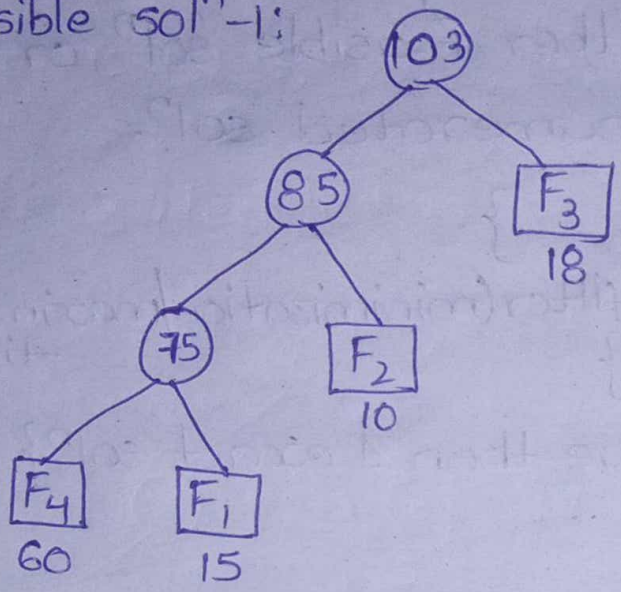
Eg: File: F_1 F_2 F_3 F_4

Record length: 15 10 18 60

[Max. two files can be choosen at a time].

Sol:

Feasible sol^n-1:

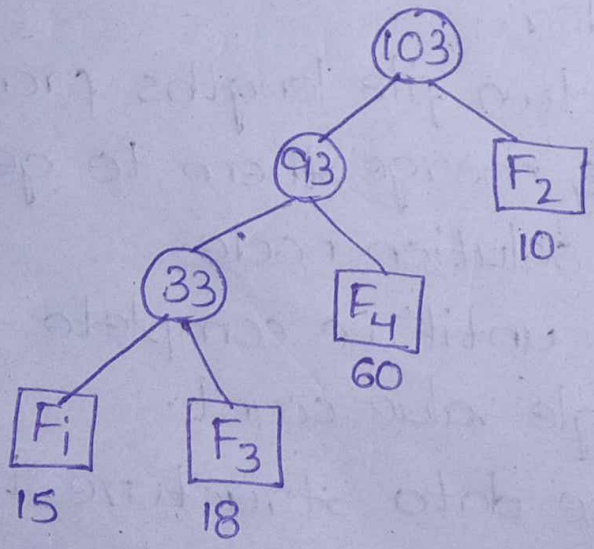


$(18 \times 1) + (10 \times 2) + (60 \times 3) + (15 \times 3) = 263$

↓

level (reachability count).

Feasible sol^n-2:



$(15 \times 3) + (18 \times 3) + (60 \times 2) + (10 \times 1) = 229.$

* As mentioned above we may get more no. of feasible sol^n's out of which only one can be selected as an optimal sol^n using greedy method.
* Optimal merge patterns is a minimization problem under greedy methodology.

Eg: Consider the following sequence of the file lengths to find out best pattern to merge using optimal merge patterns algorithm.

File lengths ⟶ 10, 5, 8, 4, 18, 2, 1, 3.

Sol: 10, 5, 8, 4, 18, 2, 1, 3

F_1 F_2 F_3 F_4 F_5 F_6 F_7 F_8

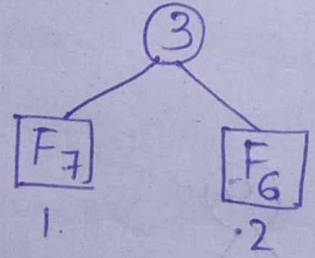
Step-1: Total files = 8

Total count = 51

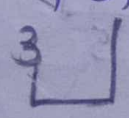
Step-2: Sort the lengths.

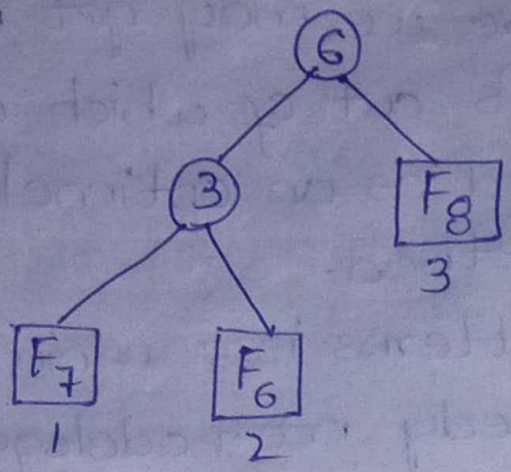
1, 2, 3, 4, 5, 8, 10, 18

Max. files = 2.



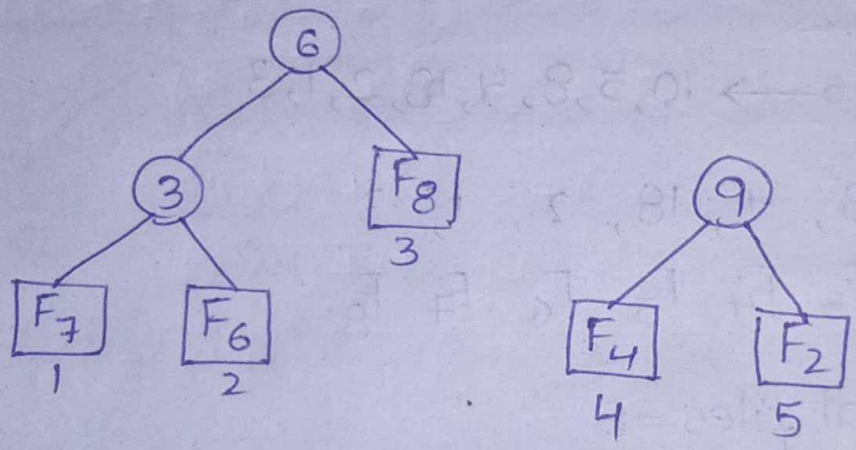
Step-3: ~~1, 2~~, 3, 4, 5, 8, 10, 18





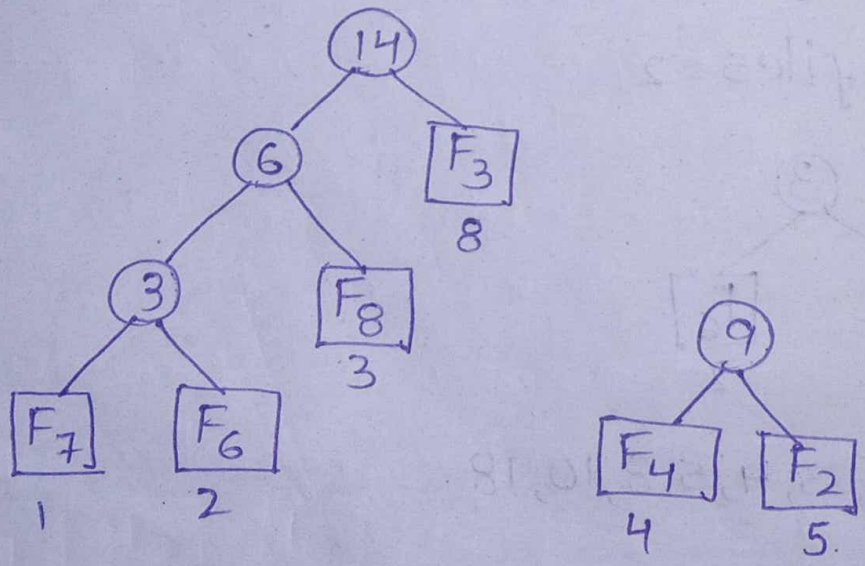
step-4: ~~1, 2, 3~~, 4, 5, 8, 10, 18
6

The sorted list is 4, 5, 6, 8, 10, 18.



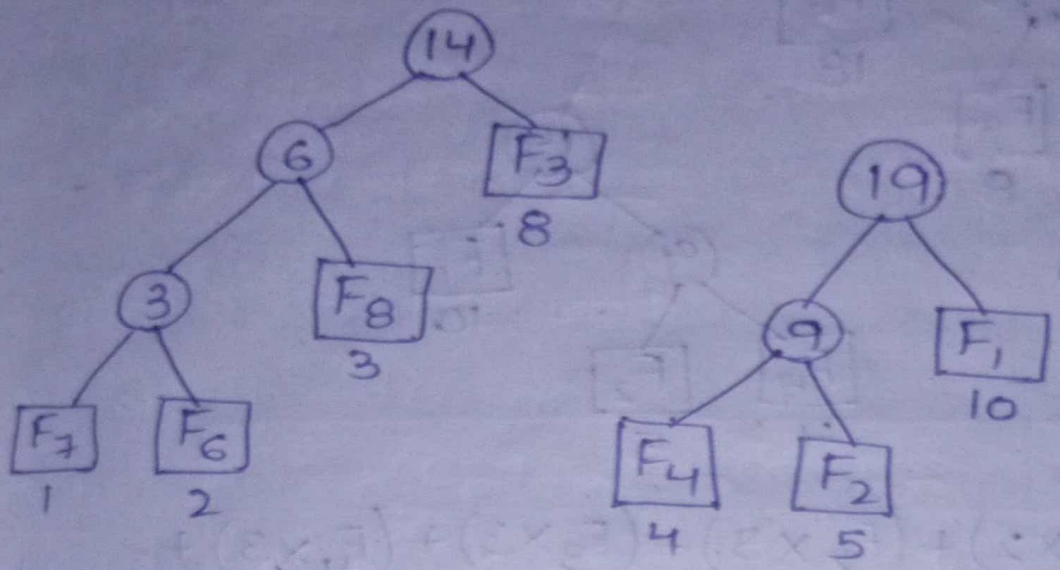
step-5: ~~4, 5~~, 6, 8, 10, 18
9

The sorted list is 6, 8, 9, 10, 18.



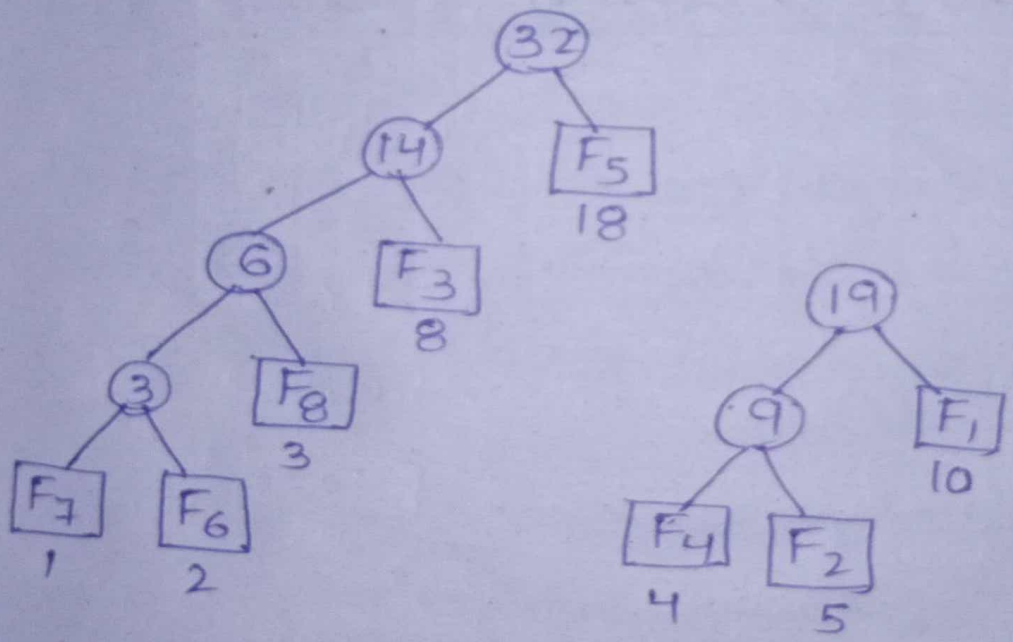
Step-6: ~~9,8~~, 9, 10, 18
14

The sorted list is 9, 10, 14, 18



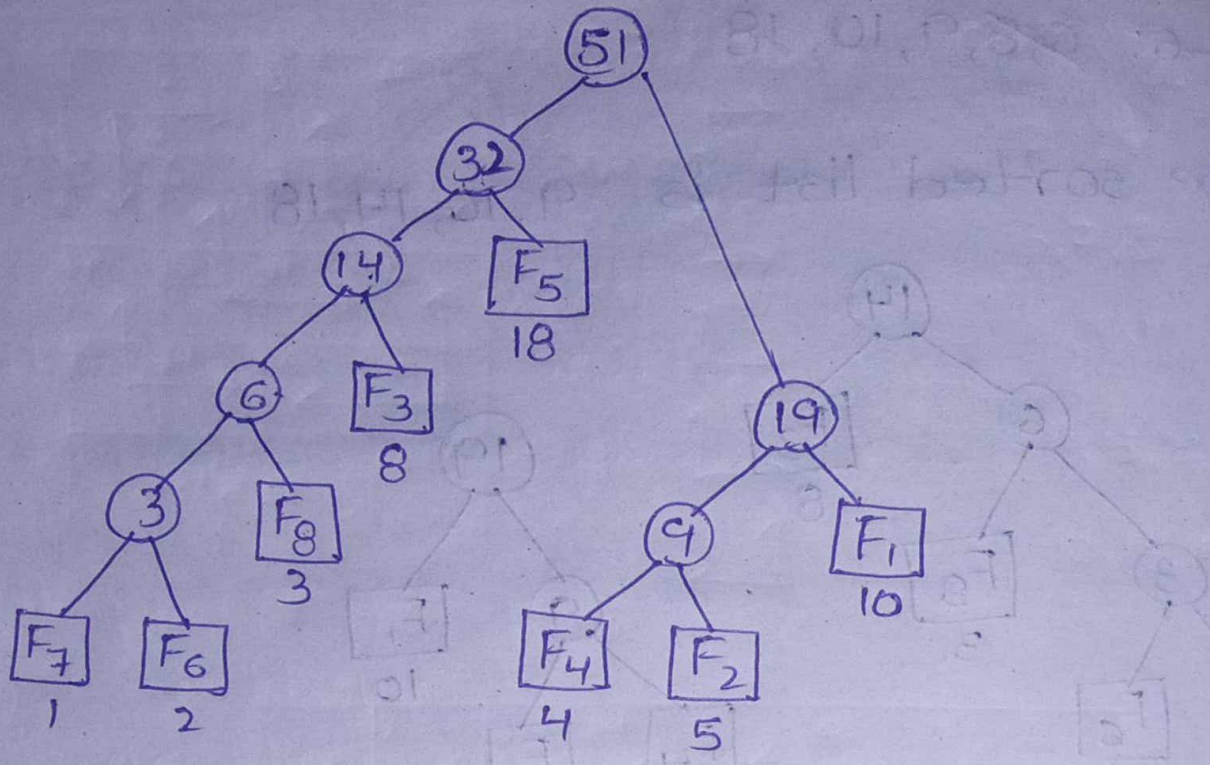
Step-7: ~~9,10~~, 14, 18
19

The sorted list is 14, 18, 19



Step-8: ~~14,18~~, 19
32

The sorted list is 19, 32



$\therefore$ cost $= (F_1 \times 2) + (F_2 \times 3) + (F_3 \times 3) + (F_4 \times 3) +$
$(F_5 \times 2) + (F_6 \times 5) + (F_7 \times 5) + (F_8 \times 4)$

$= (10 \times 2) + (5 \times 3) + (8 \times 3) + (4 \times 3) + (18 \times 2) +$
$(2 \times 5) + (1 \times 5) + (3 \times 4)$

$= 134$

Defective Chess board:

* Defective chess board is a work space of size $N \times N$, where N is equivalent to 2^k (k ranges from $0, 1, 2, \ldots n$).
* In given $n \times n$ size chess board, one of the square entry is defective or not usable, fill all other entries with the help of TROMINOES (L, ⅂, ⅃, Γ).
* In defective chess board, if there are 'n' no. of squares, then exclude one of the entry by specifying or marking.

$N \times N$ $\quad N = 2^k \quad k = \{0, 1, 2, \ldots n\}$

(1) If $k = 0$, $N = 2^0 = 1$

$\Rightarrow N \times N = 1 \times 1$

(2) If $k = 1$, $N = 2^1 = 2 \Rightarrow N \times N = 2 \times 2$

(3) If $k = 2$, $N = 2^2 = 4 \Rightarrow N \times N = 4 \times 4$

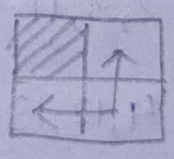
(4) If $k = 3$, $N = 2^3 = 8 \Rightarrow N \times N = 8 \times 8$

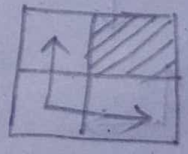
⋮

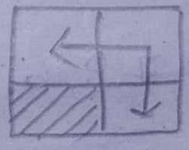
(1)

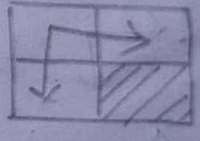
1×1

(2)



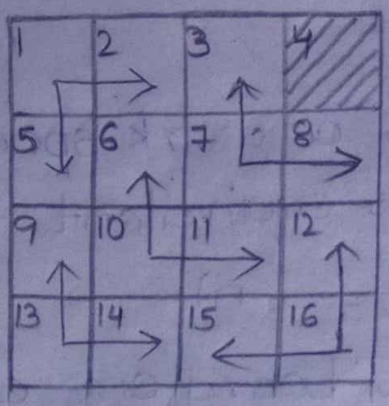




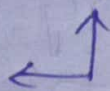


2×2

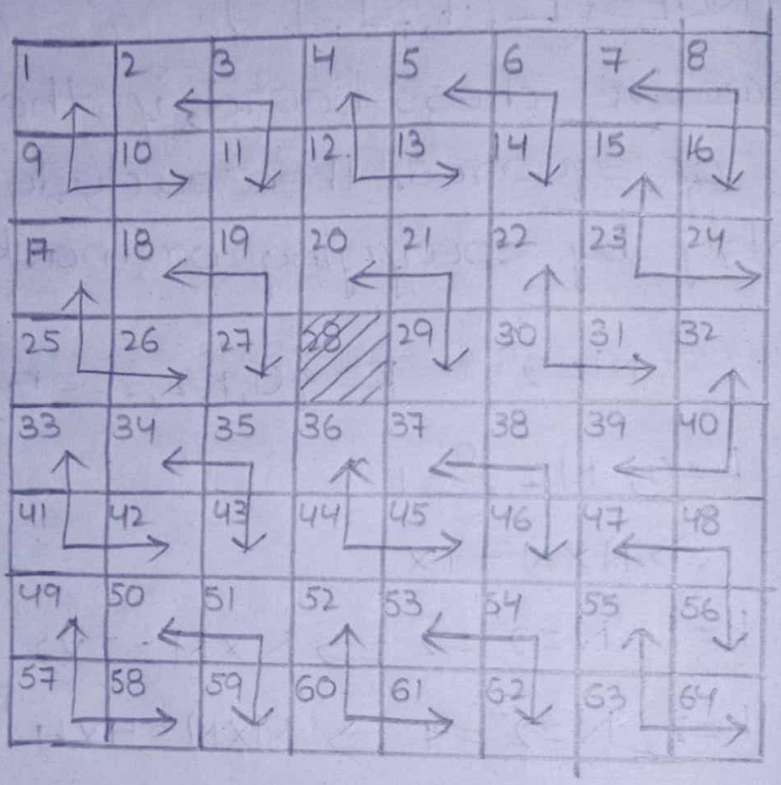
(3)

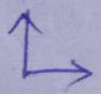
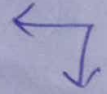
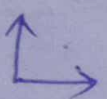


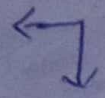
4×4

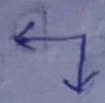
Shape	Cell No.
	2,1,5
	3,7,8
	6,10,11
	9,13,14
	12,16,15

(4)

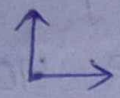


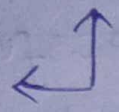
Shape	Cell No.
	1,9,10
	2,3,11
	4,12,13
	5,6,14
	7,8,16
	17,25,26

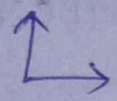

18,19,27

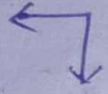

20,21,29

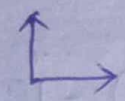

22,30,31

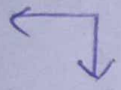

15,23,24


32,40,39


33,41,42


34,35,43

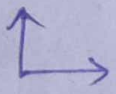

36,44,45

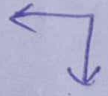

37,38,46


47,48,56


49,57,58


50,51,59


52,60,61


53,54,62


55,63,64