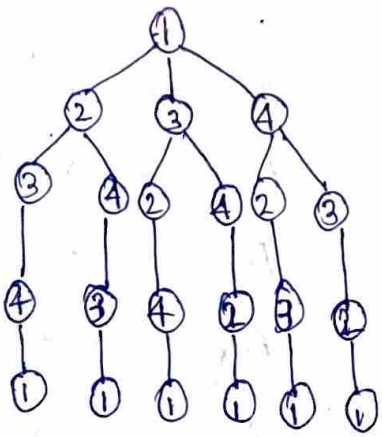
2021

CC-9

3)

b)

	1	2	3	4
1	0	10	15	20
2	5	0	9	10
3	6	13	0	12
4	8	8	9	0



$g(i,s) = \min_{j \in s} w(i,j) + g(j,\{s-j\})$

$g(1,\{2,3,4\}) = w(1,2) + g(2,\{3,4\}) = 10+25 = 35$

$= w(1,3) + g(3,\{2,4\}) = 15+25 = 40$

$= w(1,4) + g(4,\{2,3\}) = 20+23 = 43$

$g(2,\{3,4\}) = w(2,3) + g(3,\{4\}) = 9+20 = 29$

$= w(2,4) + g(4,\{3\}) = 10+15 = 25$

$g(3,\{4\}) = w(3,4) + g(4,\{\phi\}) = 12+8 = 20$

$g(4,\{3\}) = w(4,3) + g(3,\{\phi\}) = 9+6 = 15$

$g(3,\{2,4\}) = w(3,2) + g(2,\{4\}) = 13+18 = 31$

$= w(3,4) + g(4,\{2\}) = 12+13 = 25$

$g(2,\{4\}) = w(2,4) + g(4,\{\phi\}) = 10+8 = 18$

$g(4,\{2\}) = w(4,2) + g(2,\{\phi\}) = 8+5 = 13$

$g(4,\{2,3\}) = w(4,2) + g(2,\{3\}) = 8+15 = 23$

$= w(4,3) + g(3,\{2\}) = 9+18 = 27$

$g(2,\{3\}) = w(2,3) + g(3,\{\phi\}) = 9+6 = 15$

$g(3,\{2\}) = w(3,2) + g(2,\{\phi\}) = 13+5 = 18$

$\therefore 1 \rightarrow 2 \rightarrow 4 \rightarrow 3 \rightarrow 1$, cost = 35

4)

6)

$A_1 \longrightarrow 30 \times 35$

$A_2 \longrightarrow 35 \times 15$

$A_3 \longrightarrow 15 \times 5$

$A_4 \longrightarrow 5 \times 10$

$A_5 \longrightarrow 10 \times 20$

$A_6 \longrightarrow 20 \times 25$

$(A_1 \times (A_2 \times A_3)) \times A_4) \times A_5 \times A_6$

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

1	2	3	4	5	6	
(1,1) 0	(1,2) 15750	(1,3) ~~7875~~ 13125	(1,4) 9375	(1,5) 11875	(1,6) 15125	1
	(2,2) 0	(2,3) 2625	(2,4) 4375	(2,5) 7125	(2,6) 10500	2
		(3,3) 0	(3,4) 750	(3,5) 2500	(3,6) 5375	3
			(4,4) 0	(4,5) 1000	(4,6) 3500	4
				(5,5) 0	(5,6) 5000	5
					(6,6) 0	6

$$f(i,k) = \min_{i \le j < k} [f(i,j) + f(j+1,k) + r_i \times c_j \times c_k]$$

$f(i,i) = 0 \quad \forall \; i = 1 \text{ to } n$

$f(1,2) = f(1,1) + f(2,2) + 30 \times 35 \times 15 \quad [\because j = 1]$

$= 0 + 0 + 15750$

$= 15750$

$f(2,3) = f(2,2) + f(3,3) + 35 \times 15 \times 5 \quad [j = 2]$

$= 0 + 0 + 2625$

$= 2625$

$f(3,4) = f(3,3) + f(4,4) + 15 \times 5 \times 10 \quad [j = 3]$

$= 0 + 0 + 750$

$= 750$

$f(4,5) = f(4,4) + f(5,5) + 5 \times 10 \times 20 \quad [j = 4]$

$= 0 + 0 + 1000$

$= 1000$

$f(5,6) = f(5,5) + f(6,6) + 10 \times 20 \times 25 \quad [j = 5]$

$= 0 + 0 + 5000$

$= 5000$

$f(1,3) = f(1,1) + f(2,3) + 30 \times 35 \times 5$ [j=1]

$= 0 + 2625 + 5250$

$= 7875$ (min)

$f(1,3) = f(1,2) + f(3,3) + 30 \times 15 \times 5$ [j=2]

$= 15750 + 0 + 2250$

$= 18000$

$f(2,4) = f(2,2) + f(3,4) + 35 \times 15 \times 10$ [j=2]

$= 0 + 750 + 5250$

$= 6000$

$f(2,4) = f(2,3) + f(4,4) + 35 \times 5 \times 10$ [j=3]

$= 2625 + 1750$

$= 4375$ (min)

$f(3,5) = f(3,3) + f(4,5) + 15 \times 5 \times 20$ [j=3]

$= 0 + 1000 + 1500$

$= 2500$ (min)

$f(3,5) = f(3,4) + f(5,5) + 15 \times 10 \times 20$ [j=4]

$= 750 + 0 + 3000$

$= 3750$

$f(4,6) = f(4,4) + f(5,6) + 5 \times 10 \times 25$ [j=4]

$= 0 + 5000 + 1250$

$= 6250$

$f(4,6) = f(4,5) + f(6,6) + 5 \times 20 \times 25$ [j=5]

$= 1000 + 0 + 2500$

$= 3500$ (min)

$f(1,4) = f(1,1) + f(2,4) + 30 \times 35 \times 10$ [j=1]

$= 0 + 4375 \times 10500$

$= 14875$

$f(1,4) = f(1,2) + f(3,4) + 30 \times 15 \times 10$ [j=2]

$= 15750 + 750 + 4500$

$= 21000$

$f(1,4) = f(1,3) + f(4,4) + 30 \times 5 \times 10$ [j=3]

$= 7875 + 0 + 1500$

$= 9375$ (min)

$f(2,5) = f(2,2) + f(3,5) + 35 \times 15 \times 20$ [j=2]

$= 0 + 2500 + 10500$

$= 13000$

$f(2,5) = f(2,3) + f(4,5) + 35 \times 5 \times 20$ [j=3]

$= 4375$ ~~4375~~ $2625 + 1000 + 3500$

$= 7125$ (min)

$f(2,5) = f(2,4) + f(5,5) + 35 \times 10 \times 20$ [j=4]

$= 4375 + 0 + 7000$

$= 11375$

$f(3,6) = f(3,3) + f(4,6) + 15 \times 5 \times 25$ [j=3]

$= 0 + 1000 + 3500 + 1875$

$= 5375$ (min)

$f(3,6) = f(3,4) + f(5,6) + 15 \times 10 \times 25$ [j=4]

$= 750 + 5000 + 3750$

$= 9500$

$f(3,6) = f(3,5) + f(6,6) + 15 \times 20 \times 25$ [j=5]

$= 2500 + 0 + 7500$

$= 10000$

$f(1,5) = f(1,1) + f(2,5) + 30 \times 35 \times 20$ [j=1]

$= 0 + 7125 + 15000$ ~~15000~~ 21000

$= 28125$

$f(1,5) = f(1,2) + f(3,5) + 30 \times 15 \times 20$ [j=2]

$= 15750 + 2500 + 9000$

$= 27250$

$f(1,5) = f(1,3) + f(4,5) + 30 \times 5 \times 20$ [j=3]

$= 7875 + 1000 + 3000$

$= 11875$ (min)

$f(1,5) = f(1,4) + f(5,5) + 30 \times 10 \times 20$ [j=4]

$= 9375 + 0 + 6000$

$= 15375$

$f(2,6) = f(2,2) + f(3,6) + 35 \times 15 \times 25$ [j=2]

$= 0 + 5375 + 13125$

$= 18500$

$f(2,6) = f(2,3) + f(4,6) + 35 \times 5 \times 25$ [j=3]

$= 2625 + 3500 + 4375$

$= 10500$ (min)

$f(2,6) = f(2,4) + f(5,6) + 35 \times 10 \times 25$ [j=4]

$= 4375 + 5000 + 8750$

$= 18125$

$f(2,6) = f(2,5) + f(6,6) + 35 \times 20 \times 25$ [j=5]

$= 7125 + 0 + 15500$

$= 24625$

$f(1,6) = f(1,1) + f(2,6) + 30 \times 35 \times 25$ [j=1]

$= 0 + 10500 + 26250$

$= 36750$

$f(1,6) = f(1,2) + f(3,6) + 30 \times 15 \times 25$ [j=2]

$= 15750 + 5375 + 11250$

$= 32,375$

$f(1,6) = f(1,3) + f(4,6) + 30 \times 5 \times 25$ [j=3]

$= 7875 + 3500 + 3750$

$= 15125$

$f(1,6) = f(1,4) + f(5,6) + 30 \times 10 \times 25$ [j=4]

$= 5375 + 5000 + 7500$

$= 13,435$ (min)

$f(1,6) = f(1,5) + f(6,6) + 30 \times 20 \times 25$ [j=5]

$= 11875 + 0 + 15000$

$= 26875$

$\therefore \left(\left((A_1)_{30\times35} \times (A_2)_{35\times15}\right) \times (A_3)_{15\times5}\right) \times (A_4)_{5\times10}\right) \times \left((A_5)_{10\times20} \times (A_6)_{20\times25}\right)$

1)

a) The time complexity of BFS traversal of a graph depends on the data structure we are using the represent the graph i.e. adjacency matrix or adjacency list.

In case of adjacency matrix for both dense and sparse graph the complexity is $O(|V|^2)$.

In case of adjacency list the ~~co~~time complexity for sparse graph is $O(|V|+|E|)$ i.e. $O(|V|)$ & for dense graph it is $O(|V|^2)$.

b) $f(n) = 10n^2+6n+3$

$= n^2(10+\frac{6}{n}+\frac{3}{n^2})$

$\leq c.g(n)$

Here, $g(n) = n^2$, $n_0 \geq n_0$

Ω define lower bound of a function.

$\therefore f(n) = 10n^2+6n+3$ is in order of $\Omega(g(n))$ or, $\Omega(n^2)$

c) Worst case time complexity of an algorithm refers to the maxime time (on the basis of no. of operations) taken by the program to do certain task.

For example for BFS traversal in adjacency matrix $n \times n$ we need to traverse each row of the matrix where n is no. of vertices. To traverse the graph we need to see cases for all the vertices which leads to $|V|^2$ number of iteration.

Worst case scenario is represented by BigOh (O) notation, so here worst case time complexity is $O(|V|^2)$

d) P class of problem

P class problem refers to decision problems that are efficiently solvable by deterministic Turing machine in polynomial time. It includes problems for which there exists an algorithm that can determine the answer in a time bound by a polynomial function of the input size. These problems are considered tractable and practical to solve in real-world scenarios as their running time grows polynomially with the size of the input.

e)

Recursive Algorithm	Non-recursive Algorithm
① A recursive algorithm is an algorithm that solves a problem by calling itself.	A non-recursive algorithm that solves a problem using iteration loops.
② A recursive algorithm typically consists of a base case and recursive case.	Non recursive algorithm use loops or iterative constructs to repeatedly perform a set of instruction until problem is solved.
③ It breaks the problem in smaller subproblems	It does not break problem to get the desired result.
④ They can simplify the implementation of certain problems, particularly those exhibit a recursive structure	Implementation of code is tough in a non recursive algorithm.
⑤ It may suffer from stack overflow error as the called functions are stored in stack.	It does not suffer from stack overflow error as functions are not needed to be stored.

g) Advantage

One advantage of greedy algorithm is, it is often easy to understand and implement since they make locally optimal choices at each step. They do not require complex data structures or exhaustive search, making them easy to design and code.

Disadvantage

One disadvantage of a greedy algorithm ~~is its simplicity~~ is that it may not always lead to the globally optimal solution. Greedy algorithms make decisions based on the best available option at each step, without considering the overall consequences.

h) A minimum spanning tree is a concept in graph theory specifically in the ~~con~~ context of weighted undirected graphs. It refers to a subset of edges that connects all the vertices of the graph while minimizing the total sum of edge weights. In other words, it is a tree (a connected acyclic graph) that includes all the vertices of the original graph and has the minimum possible total edge weight.

Properties:

① It is a connected subgraph of the original graph.

② It contains all the vertices of the original

③ It has $|V|-1$ edges, where $|V|$ is the number of vertices in the graph.

④ It has the minimum possible sum of edge weights among all spanning trees of the graph.

2)

a) The Floyd-Warshall algorithm is used to find the shortest paths between all pairs of vertices in a weighted graph. It efficiently computes the minimum distance between any two vertices, making it valuable in various applications, such as network routing, transportation and logistics, GPS systems, social network analysis, game development, data visualization and more. With its versatility and ability to handle both positive and negative edge weights, the Floyd-Warshall algorithm is a fundamental tool in optimizing distances and paths in diverse real-world scenarios.

b) Algo Floyd-Warshall (Gr(V, E), w)

```
{
  n ← |V|
  for (i=1 to n)
  { for (j=1 to n)
    { for (k=1 to n)
      { if ((dji + dik) < djk)
        { djk = dji + dik
        }
      }
    }
  }
}
```

Time-complexity of Floyd-Warshall algorithm is $O(|V|^3)$.

here in every iteration j is the source vertex & k is the destination vertex and i is the via vertex, through which the distance between source and destination is calculated. If any via distance is shorter then the distance matrix is updated.

3)

a) The Travelling Salesman Problem (TSP) is a classic optimization problem in computer science.

Problem Statement:

Given a list of cities and the distances between each pair of cities, the travelling salesman wants to find the shortest possible route that visits each city exactly once and returns to the starting city.

Mathematically, the TSP can be described as finding the Hamiltonian cycle with the minimum total weight in a complete weighted graph.

Key Points

① It is a NP-Hard problem that means it can't be solved for large instances of the problem in polynomial time.

② There are several variants of TSP, such as asymetric TSP, metric TSP etc.

③ There are several applications of TSP i.e. logistics, transportation, circuit board drilling etc.

④ Solved using various algorithm; eg. branch and bound, dynamic programming, nearest neighbour etc.

4)

a)

Divide and Conquer	Dynamic Programming
① Follows Top-down approach	Follows bottom-up approach.
② Used to solve decision problem.	Used to solve optimi-zation problem.
③ Solution of subprob-lem is computed recursively more than once.	The solution of subproblems is compu-ted once and stored in a table for later use
④ It is used to obtain a solution to the given problem, it does not aim for the optimal solution.	It always generate optimal solution.
⑤ Less efficient and slower	More efficient but slower than greedy.
⑥ Some memory is required	More memory is required to store subproblems for later use

5)

a) Graph Colouring Problem

The graph colouring problem is a classic optimization problem in graph theory.

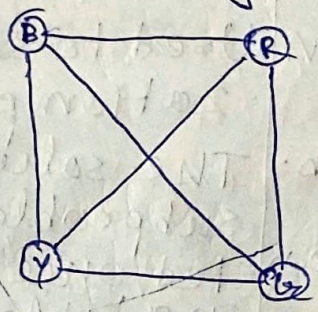
Problem Statement

Given an undirected graph (G, E) where V represents the set of vertices and E represents the set of edges, the graph colouring problem aims to assign colors to the vertices in such a way that no two adjacent vertices (connected by an edge) share the same color, while minimizing the total number of colours used.

Mathematically, the graph coloring problem can be represented as finding a

function $C: V \rightarrow \{1, 2, 3, \ldots k\}$, where k is the minimum number of colours needed, such that for every edge $(u,v) \in E, C(u) \neq C(v)$

It is a NP hard problem ie. it can't be solved in polynomial time complexity



Here 4 vertices are colored with Blue, Red, Yellow & Green according to the requirement.

Here the ~~croma~~ chromatic number is 4, that is minimum 4 colors are required.

b) Algo BFS(~~int **~~G(v,e), s,n) //G represents graph
//s represents source vertex.
// n => no. of vertex

```
{
// d => distance array
// par => parent array
// q => Queue
 for(i=0 to n)
 { d[i] = -1
   par[i] = -1
 }
 d[s] = 0
 front = -1, rear = -1;
 q[++rear] = s
 while(front != n)
 { u = q[++front]
   count++;
   for(v = 0 to n)
   {
```

```
if(Gr[u][v]==1)
{ if(d[v]<0)
  { d[v]=d[u]+1
    par[v]=u
    q[++rear]=v
  }
}
}
}
if(count=n)
{ print "The graph is connected"
}else
{ print "The graph is disconnected"
}
```

6)

a) Algo Kruskal (Gr(V, E))

① sort in ascending order of weights

② ~~At~~ n ← |V|

T ← ϕ

③ Consider n components where each component contains single vertex.

④ While (|T| ≠ n-1)

{ select e ← {u, v} of minimum weight

u_component ← findcomponent(u)

v_component ← findcomponent(v)

if(u_component ≠ v_component

{ T ← T ∪ {e}

cost = cost + w_e

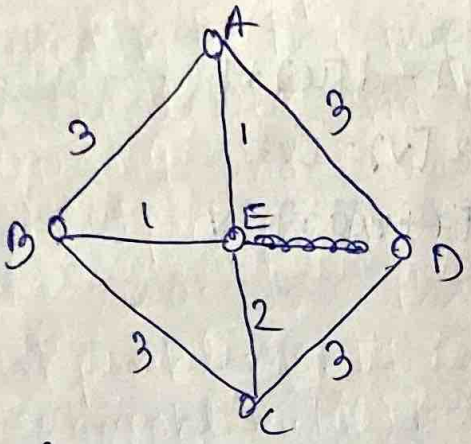
merge(u_component v_component)

}

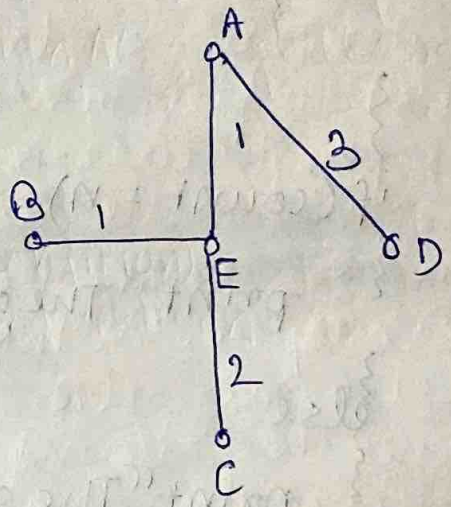
return T

Complexity: $O(|E| \log |E|)$ using heapsort
For sparse graph $O(|V| \log |V|)$
For dense graph $O(2|V|^2 \log |V|)$

b)



u, v	w_e
{A, B} →	3
{A, D} →	3
{B, C} →	3
{C, D} →	3
{C, E} →	2
{B, E} →	1
{A, E} →	1



After arranging using heapsort

u, v	w_e
{A, E} →	1
{B, E} →	1
{C, E} →	2
{A, B} →	3
{A, D} →	3
{B, C} →	3
{C, D} →	3

Step	Edge Considered	Components
1	—	{A}, {B}, {C}, {D}, {E}
2	{A, E}	{A, E}, {B}, {C}, {D}
3	{B, E}	{A, B, E}, {C}, {D}
4	{C, E}	{A, B, C, E}, {D}
5	{A, B}	Reject
6	{A, D}	{A, B, C, D, E}

∴ Cost = 1 + 1 + 2 + 3
= 7

7)

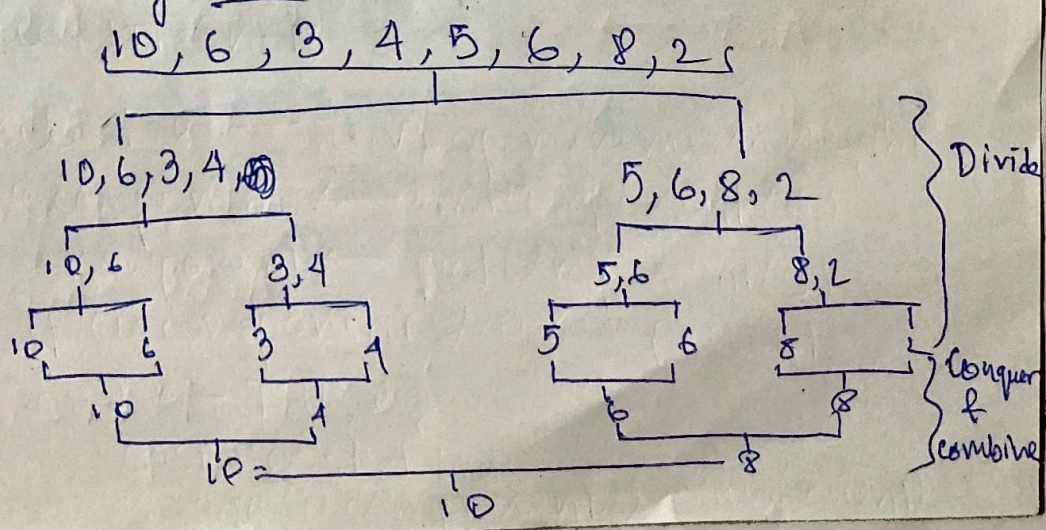
a) The divide and conquer (D&C) method is a problem-solving strategy used in computer science. It involves the following strategy to find the solution of the original problem. The strategy follows three key steps: divide, conquer and combine.

① Divide: The first step is to divide the original problem into smaller subproblems that are structurally similar to the original problem, but simpler in nature. The divison is typically performed recursively until the subproblems become small enough to solved directly or are considered base cases.

② Conquer: In this step, the D&C method recursively solves each of the subproblems generated during the divison step. If the subproblems are simple enough to be solved directly, the algorith finds their solution directly.

③ Combine: Once all the subproblems are solved, the D&C method combines their solutions to form the solution to the original problem. This compination may involve merging the subproblem solutions or performing some additional computation to derive the final answer.

Eg:- Merge Sort



6) From Fall add

For the given problem P let A() be the algorithm & n be the input size with L and U are the lower and upper indices. Also we assume I and O are respectively the input and output for A. Without losing generality we assume that process uses two subproblems. T1 and T2 are the output of the subproblem. M() is the merge operation. With these we can design A() as follows:

```
Function A(I,L,U,O)
Begin
    if (L and U meet base condition)
        then solve it and return
    M = floor((L+U)/2)
    A(I,L,M,T1)
    A(I,M+1,U,T2)
    M(L,M,T1,M+1,U,T2,O)
end
```

For time complexity analysis following recursion relation for the above solution structure can be written:

$T(n) = T(L,M) + T(M+1, U) + M(L, M, M+1, U)$

can be simplified as

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

$T(\text{Base condition}) = \text{constant}.$

8)

a) Problem statement: Given a set of items, each with a weight and a value and a knapsack with a maximum weight capacity, the goal of the knapsack problem is to determine the most valuable combination of items that can be placed in the knapsack without exceeding its weight capacity.

Formally,

w_k = the weight of each type-k item, for $k = 1, 2, \ldots, N$

r_k = the value associated with each type-k item, for $k = 1, 2, \ldots, N$

c = the weight capacity of the knapsack

Then, our problem can be formulated as

$$\text{Maximize} \sum_{k=1}^{N} r_k x_k$$

$$\text{Subject to:} \sum_{k=1}^{N} w_k x_k \leq c, \quad (0 \ldots 1)$$

Where $x_1, x_2, x_3 \ldots, x_N$ are nonnegative ~~integer~~ -valued decision variables defined by

x_k = the number or fraction of type-k items that are loaded into the knapsack.

b) ~~Ddcc~~ When we use Greedy algorithm, we select present best choice to select best choice we need to sort the value per weight ratio in descending order.

We sort this using Mergesort.

And in Merge sort the time complexity is $O(n \log n)$ and space complexity $O(n)$.

The overall complexity using Greedy's Algorithm remains same.