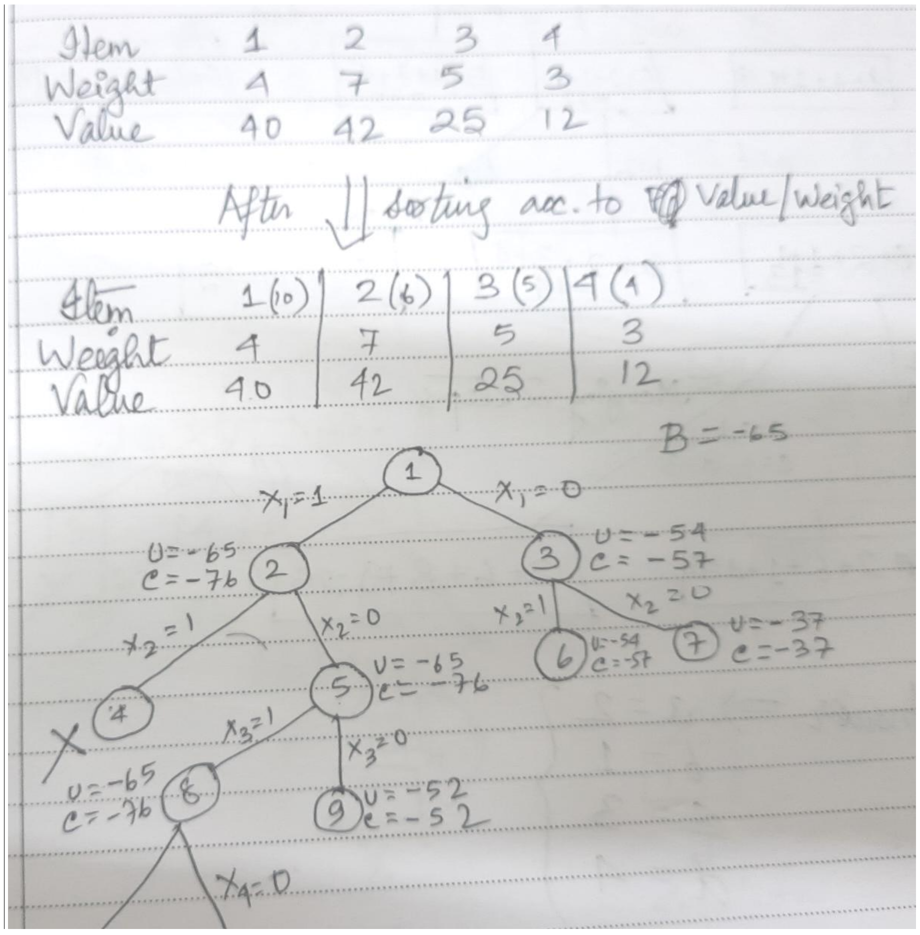
DESIGN AND ANALYSIS OF ALGORITHMS – 22MCA15 - IAT 3 SOLUTIONS

1.

Parameter	**Backtracking**	**Branch and Bound**
Approach	Backtracking is used to find all possible solutions available to a problem. When it realises that it has made a bad choice, it undoes the last choice by backing it up. It searches the state space tree until it has found a solution for the problem.	Branch-and-Bound is used to solve optimisation problems. When it realises that it already has a better optimal solution that the pre-solution leads to, it abandons that pre-solution. It completely searches the state space tree to get optimal solution.
Traversal	Backtracking traverses the state space tree by DFS(Depth First Search) manner.	Branch-and-Bound traverse the tree in any manner, DFS or BFS.
Function	Backtracking involves feasibility function.	Branch-and-Bound involves a bounding function.
Problems	Backtracking is used for solving Decision Problem.	Branch-and-Bound is used for solving Optimisation Problem.
Searching	In backtracking, the state space tree is searched until the solution is obtained.	In Branch-and-Bound as the optimum solution may be present any where in the state space tree, so the tree need to be searched completely.
Efficiency	Backtracking is more efficient.	Branch-and-Bound is less efficient.
Applications	Useful in solving N-Queen Problem, Sum of subset, Hamilton cycle problem, graph coloring problem	Useful in solving Knapsack Problem, Travelling Salesman Problem.
Solve	Backtracking can solve almost any problem. (chess, sudoku, etc).	Branch-and-Bound can not solve almost any problem.

Used for	Typically backtracking is used to solve decision problems.	Branch and bound is used to solve optimization problems.
Nodes	Nodes in stat space tree are explored in depth first tree.	Nodes in tree may be explored in depth-first or breadth-first order.
Next move	Next move from current state can lead to bad choice.	Next move is always towards better solution.
Solution	On successful search of solution in state space tree, search stops.	Entire state space tree is search in order to find optimal solution.



Item	1	2	3	4
Weight	4	7	5	3
Value	40	42	25	12

After ⇓ sorting acc. to Value/Weight

Item	1 (10)	2 (6)	3 (5)	4 (4)
Weight	4	7	5	3
Value	40	42	25	12

2. In a deterministic algorithm, for a given particular input, the computer will always produce the same output going through the same states but in the case of the **non-deterministic algorithm**, for the same

input, the compiler may produce different output in different runs. In fact, non-deterministic algorithms can't solve the problem in polynomial time and can't determine what is the next step. The non-deterministic algorithms can show different behaviors for the same input on different execution and there is a degree of randomness to it.

- A non-deterministic algorithm is one in which the outcome cannot be predicted with certainty, even if the inputs are known.
- For a particular input the computer will give different outputs on different execution.
- Can't solve the problem in polynomial time.
- Cannot determine the next step of execution due to more than one path the algorithm can take.
- Operation are not uniquely defined.
- Time complexity of non-deterministic algorithms is often described in terms of expected running time.
- Non-deterministic algorithms may produce different outputs for the same input due to random events or other factors.
- Non-deterministic algorithms are often used in applications where finding an exact solution is difficult or impractical, such as in artificial intelligence, machine learning, and optimization problems.
- Examples of non-deterministic algorithms include probabilistic algorithms like Monte Carlo methods, genetic algorithms, and simulated annealing.

Cook–Levin theorem shows that general CNF-SAT (Boolean satisfiability problem) is NP-complete. To get to Knapsack, one easy way is to:

1. reduce CNF-SAT to 3-CNF-SAT (i.e., three literals per clause) — this is easy
2. reduce 3-CNF-SAT to SubsetSum — see below for details
3. reduce SubsetSum to Knapsack — again, should be easy enough to do on your own.

For the second step, we need to find a many-one reduction from 3-SAT to SubsetSum — in other words, a poly-time-computable function that maps each 3-SAT instance to a collection of positive integers + a goal value in a way that preserves solvability. The construction below is slightly redundant for ease of explanation.

Suppose your 3-SAT instance has n variables, labeled x_0 through x_{n-1}, and m clauses, numbered 0 through m−1. The goal in our SubsetSum instance will be the following number (in base-10):
s=33….33(upto m times)11…11(upto n times)
The general idea is that the individual numbers for the instance will be constructed in such a way that:

- Getting all the 1s at the end of s will correspond to choosing either "true" or "false" (but not both) for each variable.
- Getting the 3s before them will correspond to having at least one true literal in each clause.

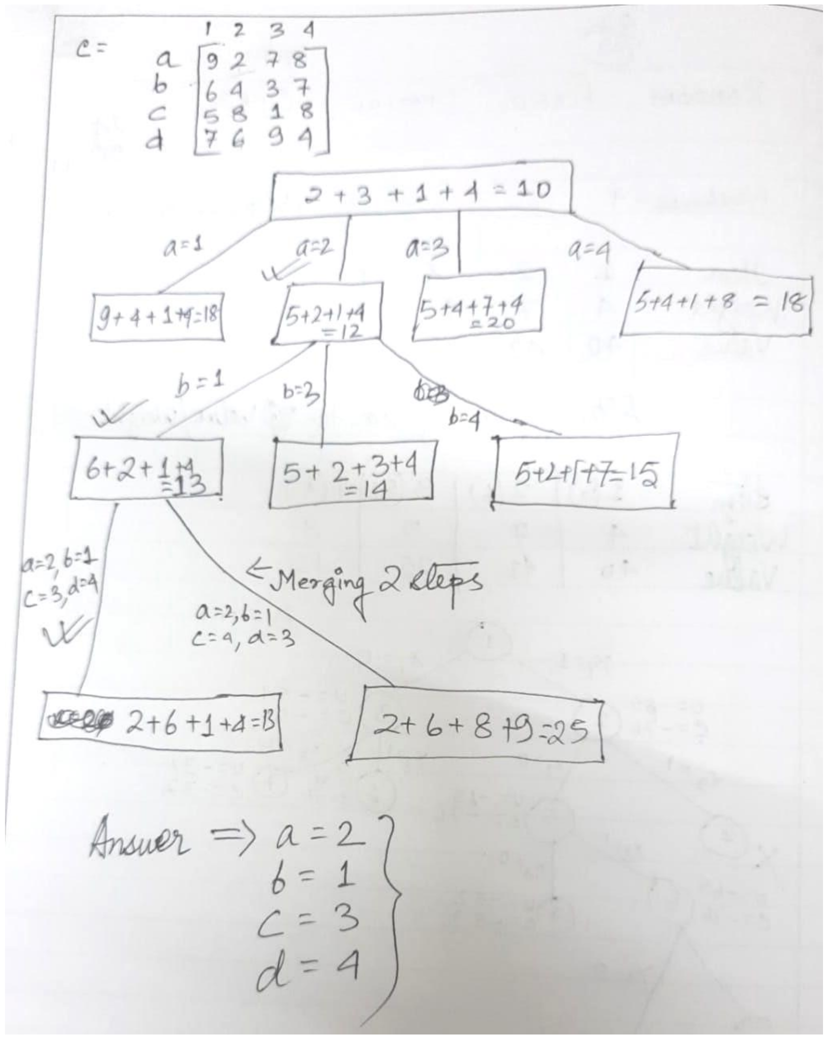
Here is the construction:

- For each positive literal x_i we will have one integer $a_i=10^i+\sum j 10^{n+j}$, where the sum goes over all clauses that contain this literal.

- For each negative literal $\neg x_i$ we will have one integer $b_i=10^i+\sum j10^{n+j}$, where the sum goes over all clauses that contain this literal.
- For each clause j, we will have two separate numbers $c_j=d_j=10^{n+j}$

  And that's the whole instance. So, we translated the 3-SAT instance "here is a list of clauses, is there a satisfying assignment of truth values?" to the SubsetSum instance "here is a collection of integers: all the ai,bi,cj,dj is there a subset of this collection with sum s?".

3.



4. A Las Vegas algorithm is an algorithm which uses randomness, but gives guarantees that the solution obtained for given problem is correct. It takes the risk with resources used. A quick-sort algorithm is a simple example of Las-Vegas algorithm. To sort the given array of n numbers quickly we

use the quick sort algorithm. For that we find out central element which is also called as pivot element and each element is compared with this pivot element. Sorting is done in less time or it requires more time is dependent on how we select the pivot element. To pick the pivot element randomly we can use Las-Vegas algorithm.

Let us consider the above example of quick sort algorithm. In this algorithm we choose the pivot element randomly. But the result of this problem is always a sorted array. A Las-Vegas algorithm is having one restriction i.e. the solution for the given problem can be found out in finite time. In this algorithm the numbers of possible solutions arc limited. The actual solution is complex in nature or complicated to calculate but it is easy to verify the correctness of candidate solution.

These algorithms always produce correct or optimum result. Time complexity of these algorithms is based on a random value and time complexity is evaluated as expected value. For example, Randomized Quick Sort always sorts an input array and expected worst case time complexity of Quick Sort is O(nLogn).

The computational algorithms which rely on repeated random sampling to compute their results such algorithm are called as **Monte-Carlo algorithms.**
The random algorithm is Monte-carlo algorithms if it can give the wrong answer sometimes.

Whenever the existing deterministic algorithm is fail or it is impossible to compute the solution for given problem then Monte-Carlo algorithms or methods are used. Monte-carlo methods are best repeated computation of the random numbers, and that’s why these algorithms are used for solving physical simulation system and mathematical system.

This Monte-carlo algorithms are specially useful for disordered materials, fluids, cellular structures. In case of mathematics these method are used to calculate the definite integrals, these integrals are provided with the complicated boundary conditions for multidimensional integrals. This method is successive one with consideration of risk analysis when compared to other methods.

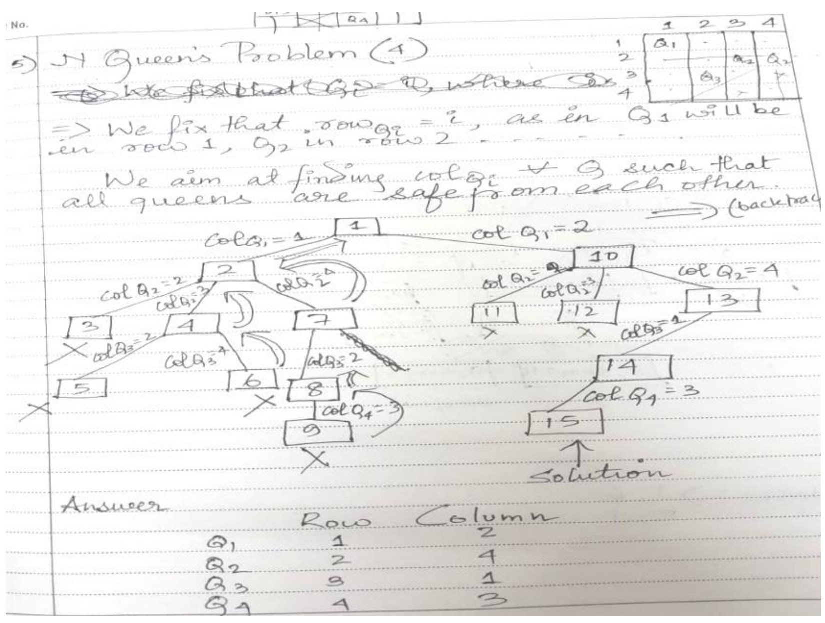
The Monte-carlo methods has wider range of applications. It uses in various areas like physical science, Design and visuals, Finance and business, Telecommunication etc. In general Monte carlo methods are used in mathematics. By generating random numbers we can solve the various problem. The problems which are complex in nature or difficult to solve are solved by using Monte-carlo algorithms. Monte carlo integration is the most common application of Monte-carlo algorithm.

The deterministic algorithm provides a correct solution but it takes long time or its runtime is large. This run-time can be improved by using the Monte carlo integration algorithms. There are various methods used for integration by using Monte-carlo methods such as,

i) Direct sampling methods which includes the stratified sampling, recursive
stratified sampling, importance sampling.

ii) Random walk Monte-carlo algorithm which is used to find out the integration for
given problem.
iii) Gibbs sampling.

5.



5) N Queen's Problem (4)

~~We fix that Q_i = i, where~~

=> We fix that $row_{Q_i} = i$, as in Q_1 will be in row 1, Q_2 in row 2 . . . .

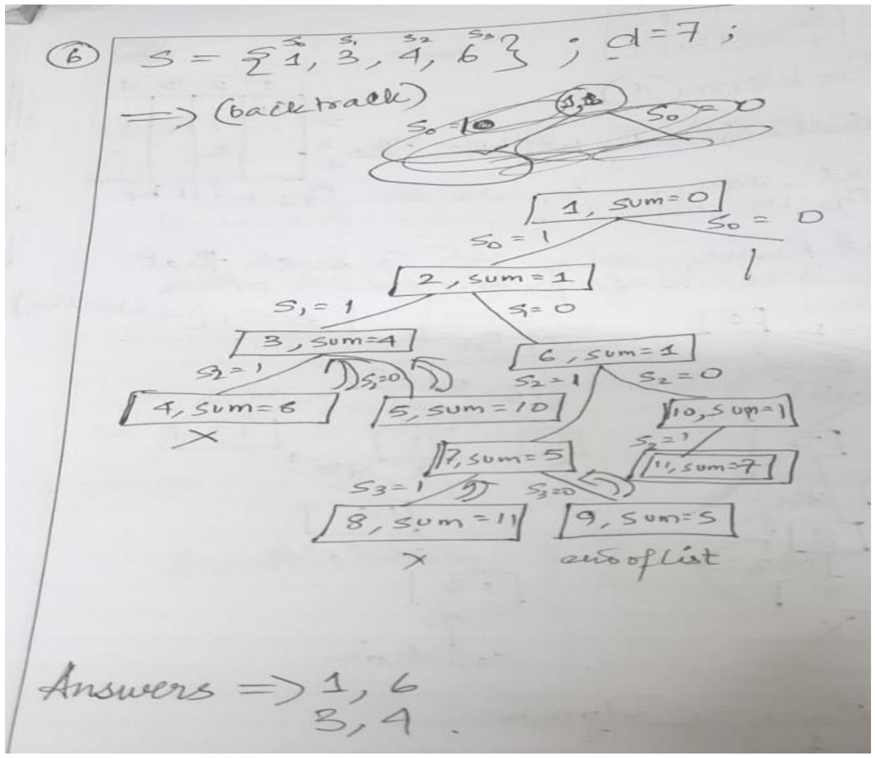
We aim at finding col_{Q_i} ∀ Q such that all queens are safe from each other.

=> (backtrac

Answer

	Row	Column
Q_1	1	2
Q_2	2	4
Q_3	3	1
Q_4	4	3

6.



⑥ $S = \{\overset{S_0}{1}, \overset{S_1}{3}, \overset{S_2}{4}, \overset{S_3}{6}\}$; d=7 ;

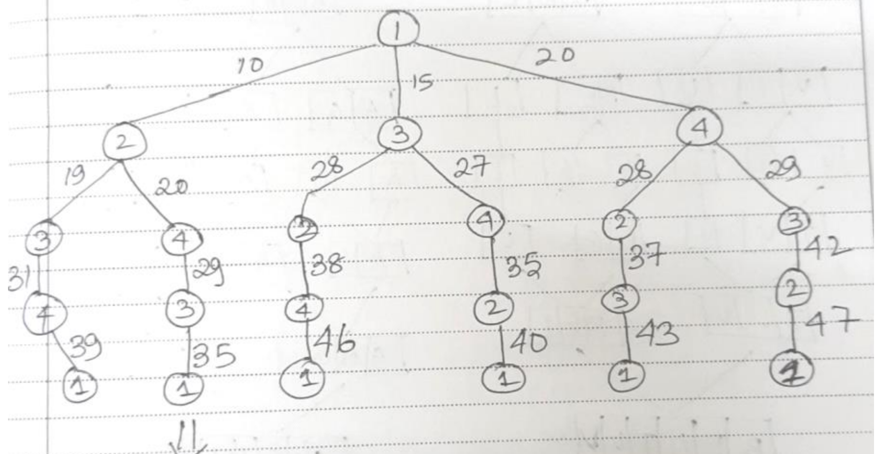
=> (backtrack)

Answers => 1, 6
3, 4 .

7.

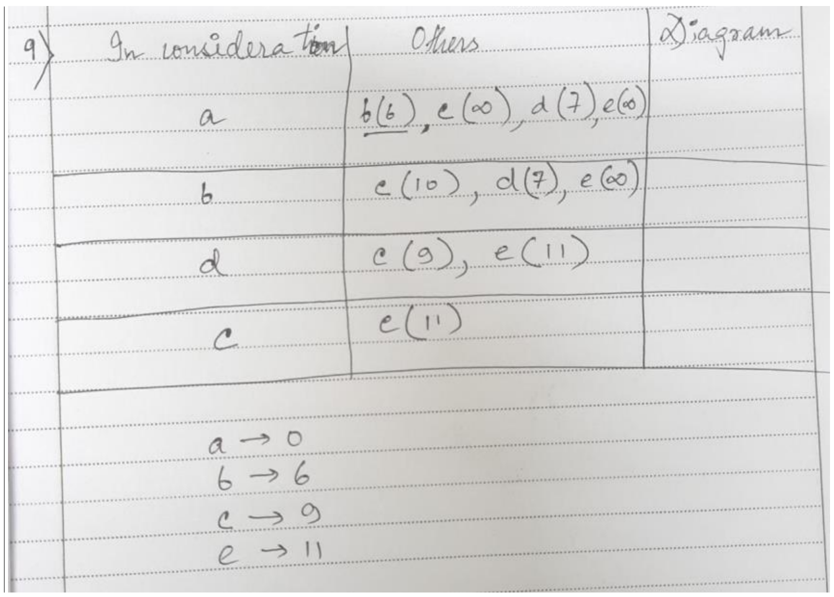
7)

$$\begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ 1 & \infty & 10 & 15 & 20 \\ 2 & 5 & \infty & 9 & 10 \\ 3 & 6 & 13 & \infty & 12 \\ 4 & 8 & 8 & 9 & \infty \end{array}$$



Optimal Tour $\Longrightarrow 1 \rightarrow 2 \rightarrow 4 \rightarrow 3 \rightarrow 1$

9.

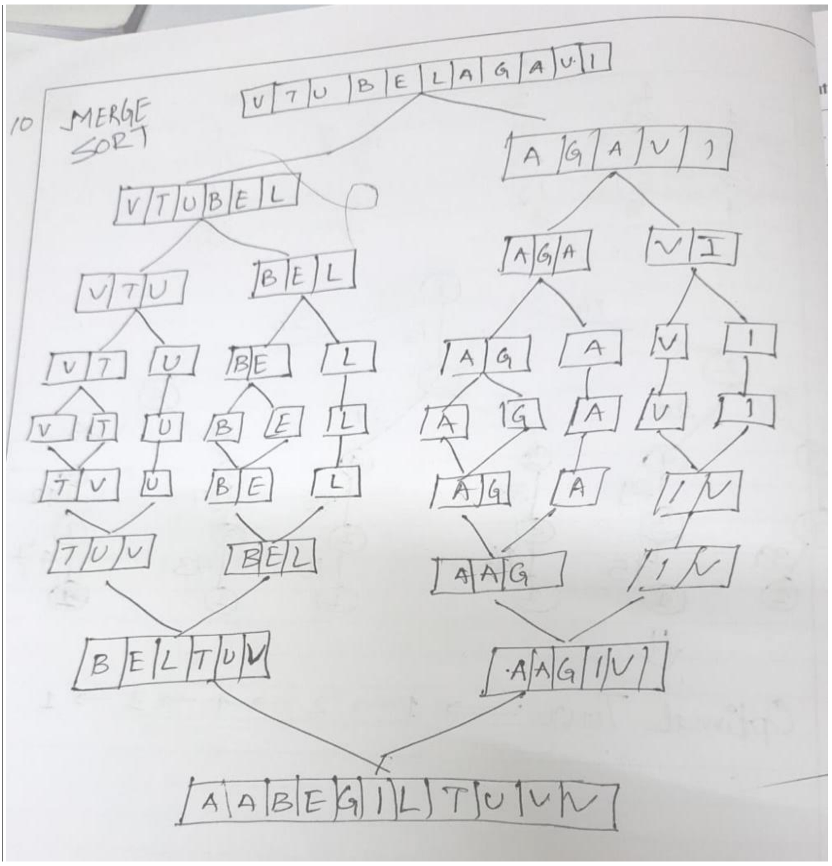


9)

In consideration	Others	Diagram
a	b(6), c(∞), d(7), e(∞)	
b	c(10), d(7), e(∞)	
d	c(9), e(11)	
c	e(11)	

$a \rightarrow 0$
$b \rightarrow 6$
$c \rightarrow 9$
$e \rightarrow 11$

10.


10
MERGE SORT
V T U B E L A G A V I
V T U B E L
A G A V I
V T U
B E L
A G A
V I
V T
U
B E
L
A G
A
V
I
V
T
U
B
E
L
A
G
A
V
I
T V
U
B E
L
A G
A
I V
T U V
B E L
A A G
I V
B E L T U V
A A G I V
A A B E G I L T U V V