2016 – Section A

1. (a) How an algorithm can be devised? Explain with example.

Solution:

In order to devise an algorithm, one needs to understand the design patterns that are generally used to develop an algorithm. Some of the design patterns are:

i. Greedy Method
ii. Divide and Conquer Method
iii. Dynamic Programming Method
iv. Branch and Bound Method
v. Backtracking method

The development of an algorithm is a key step in solving a problem. Algorithm development process consists of five major steps.

Step 1: Obtain a description of the problem.

Step 2: Analyze the problem.

Step 3: Develop a high-level algorithm.

Step 4: Refine the algorithm by adding more detail.

Step 5: Review the algorithm.

Problem Statement (Step 1)

To devise an algorithm, we have to understand the problem description properly and think about different corner cases for which the algorithm will not work perfectly

Analysis of the Problem (Step 2)

After understanding the problem description, we have to analyze the problem and try to find efficient technique to solve the problem.

High-level Algorithm (Step 3)

There are different algorithm design techniques, we have to follow those techniques for construct a very good and efficient algorithm.

Detailed Algorithm (Step 4)

The devised algorithm should be detailed to show the steps of solving the problem.

Review the Algorithm (Step 5)

Finally, we have to review the algorithm that for any test case or any input the algorithm works perfectly.

Source: http://sofia.cs.vt.edu/cs1114-ebooklet/chapter4.html

1. (b) Give -the best Big-Oh characterization for each of the following running time estimates (where n is the size of the input problem)

i. $\boldsymbol{n + (n-1) + (n-2) + \cdots + 3 + 2 + 1}$

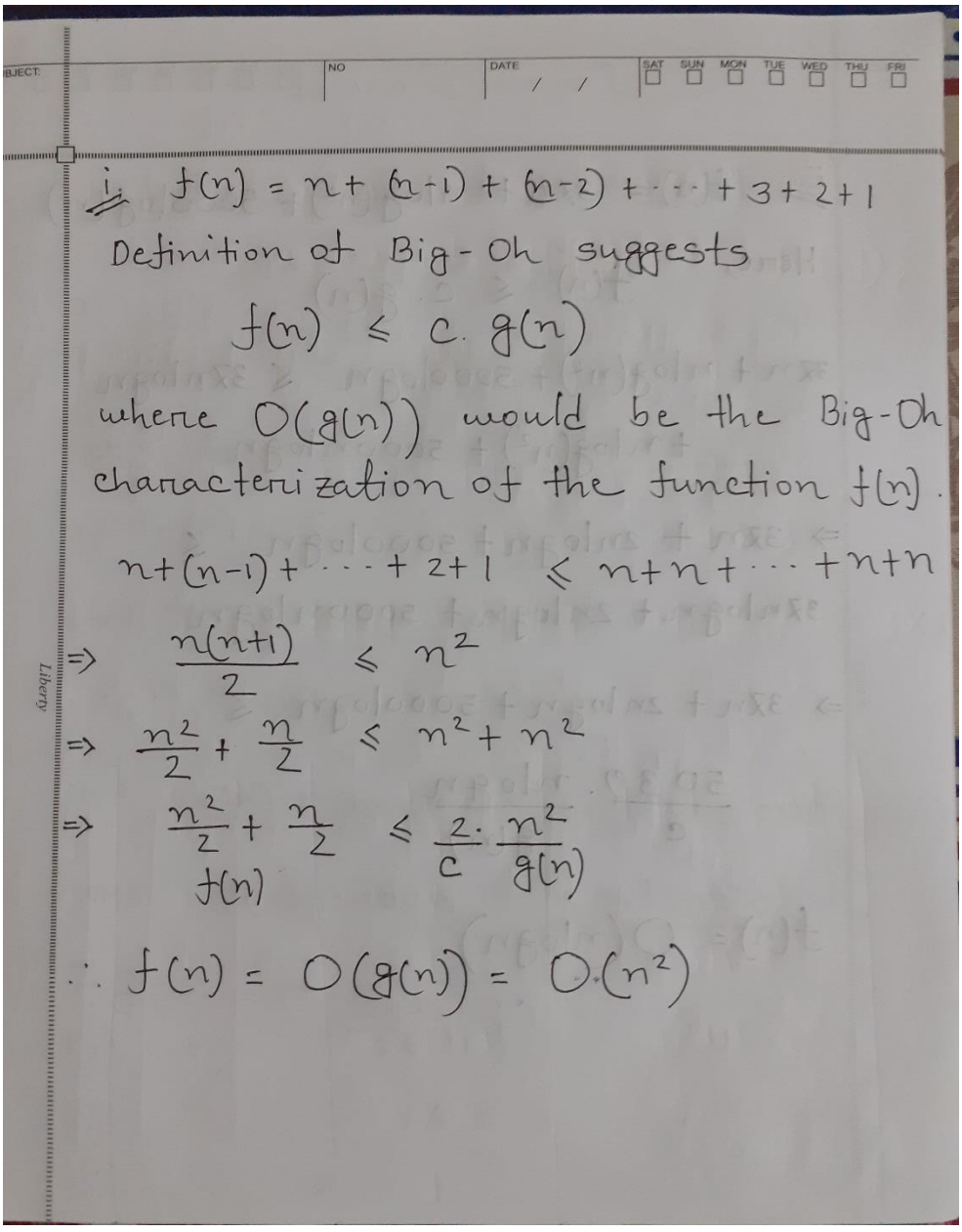
ii. $\boldsymbol{37n + nlog(n^2) + 5000\log(n)}$

iii. $\boldsymbol{1000n^2 + 16n + 2^n}$

iv. $\boldsymbol{\log(n) + 10000}$

v. $\boldsymbol{nlog(n) + 15n + 0.002n^2}$

Solution:



i. $f(n) = n + (n-1) + (n-2) + \cdots + 3 + 2 + 1$

Definition of Big-Oh suggests

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

where $O(g(n))$ would be the Big-Oh characterization of the function $f(n)$.

$$n + (n-1) + \cdots + 2 + 1 \leq n + n + \cdots + n + n$$

$$\Rightarrow \frac{n(n+1)}{2} \leq n^2$$

$$\Rightarrow \frac{n^2}{2} + \frac{n}{2} \leq n^2 + n^2$$

$$\Rightarrow \underbrace{\frac{n^2}{2} + \frac{n}{2}}_{f(n)} \leq \frac{2}{c} \cdot \underbrace{n^2}_{g(n)}$$

$$\therefore f(n) = O(g(n)) = O(n^2)$$

SUBJECT | NO | DATE / / | SAT SUN MON TUE WED THU FRI

ii. $f(n) = 3n + n\log(n^2) + 5000\log(n)$

Here, $f(n) \leq c \cdot g(n)$

$3n + n\log(n^2) + 5000\log n \leq 3n\log n + n\log(n^2) + 5000 n\log n$

$\Rightarrow 3n + 2n\log n + 5000\log n \leq 3n\log n + 2n\log n + 5000 n\log n$

$\Rightarrow 3n + 2n\log n + 5000\log n \leq \frac{5039}{c} \cdot \frac{n\log n}{g(n)}$

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

Liberty

iii. $f(n) = 1000n^2 + 16n + 2^n$

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

$1000n^2 + 16n + 2^n \leq 1000 2^n + 16 2^n + 2^n$

or, $1000n^2 + 16n + 2^n \leq \underline{1018}\ \underline{2^n}$

$\therefore f(n) = O(2^n)$

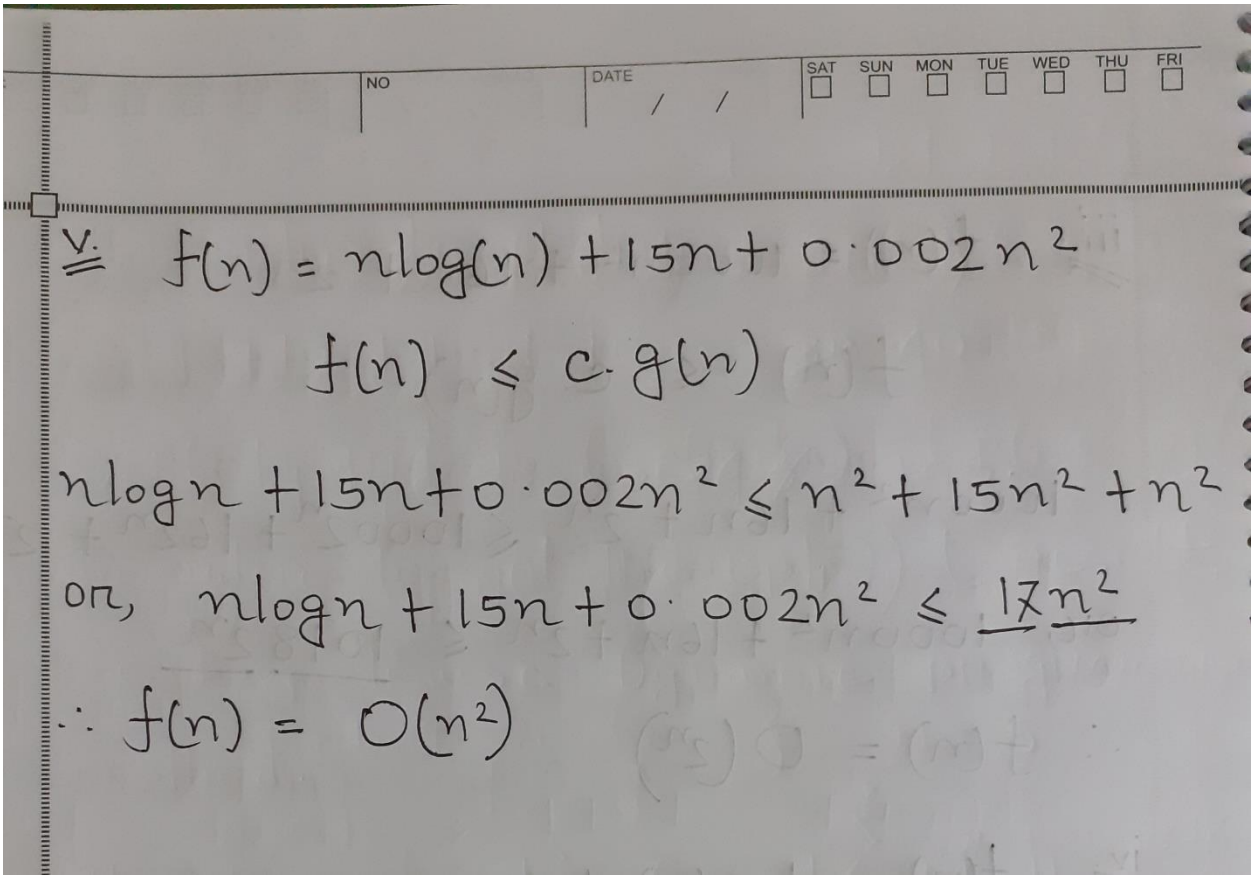
iv. $f(n) = \log(n) + 10000$

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

$\log(n) + 10000 \leq \log(n) + 10000 \log(n)$

or, $\log(n) + 10000 \leq \underline{10001}\ \underline{\log(n)}$

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



Sources:

1. https://www.youtube.com/watch?v=A03oI0znAoc&list=PLDN4rrl48XKpZkf03iYFl-O29szjTrs_O&index=12

2. https://www.youtube.com/watch?v=Nd0XDY-jVHs&list=PLDN4rrl48XKpZkf03iYFl-O29szjTrs_O&index=12

1. (c) Consider the following-coin denominations: I, 3, 5, 10, 22, 50. Can the greedy-method be used to find the smallest number of coins of these denominations with total value equal to some number B? Why?

Solution:

Coin Denominations: a = [1, 3, 5, 10, 22, 50]

Greedy Method won't provide the optimal solution for all cases.

Suppose, B = 40

Greedy Approach:

1. Sort array a in descending order. a = [50, 22, 10, 5, 3, 1]
2. If it is possible to take the current coin, take it. Otherwise, process next coin by continuing 2 until the last coin is processed.

Greedy Approach will give us the solution:

$$22 + 5 + 5 + 5 + 3 = 40$$

That is, we would need a total of 5 coins to get a total of 40. But we can do better.

The optimal solution to the problem:

$$10 + 10 + 10 + 10 = 40$$

In this case, we only had to use 4 coins to get to 40.

To get the optimal result, we have to use Dynamic Programming Method.

The reason why Greedy Method doesn't work is because, it doesn't calculate the result from all possible combinations of coins to get to value B. The algorithm picks the best option in each step and hoping that it might produce the best overall result. But that isn't always the case.

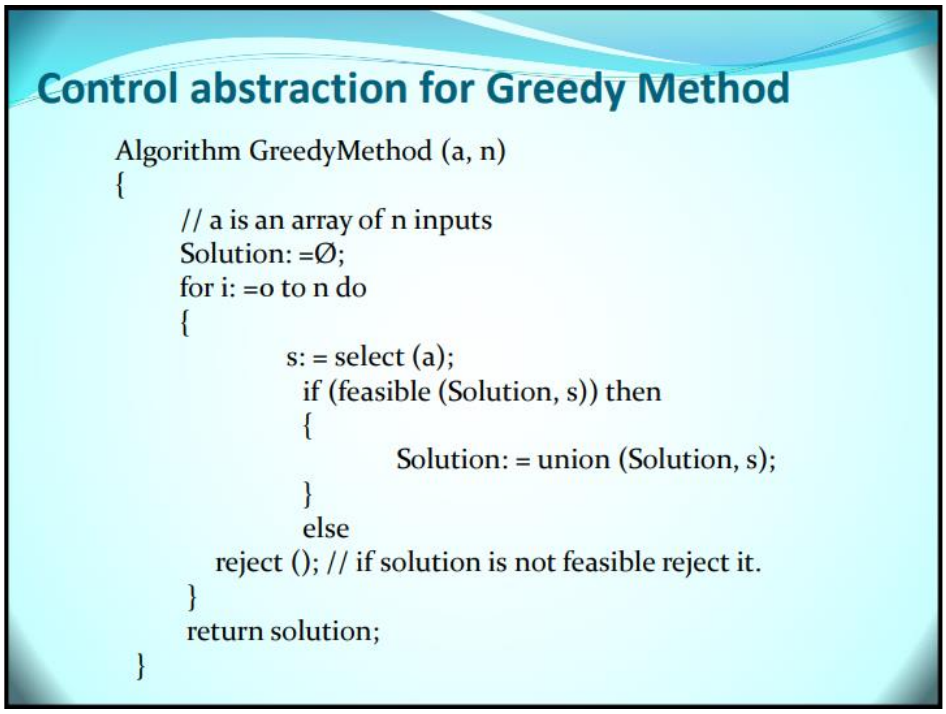
Sources:

1. https://www.log2base2.com/algorithms/greedy/greedy-algorithm.html

2. http://www.shafaetsplanet.com/?p=3638

1. **(d) Write the control abstraction for greedy method.**

Solution:



Procedure GreedyMethod describes the essential way that a greedy based algorithm will look, once a particular problem is chosen and the functions select, feasible and union are properly implemented.

The function select selects an input from 'a', removes it and assigns its value to 'x'. Feasible is a Boolean valued function, which determines if 'x' can be included into the solution vector. The function Union combines 'x' with solution and updates the objective function.

Sources:

1. https://anilkumarprathipati.files.wordpress.com/2017/03/daa-unit-iii.pdf
2. http://www.gvpcew.ac.in/LN-CSE-IT-22-32/CSE-IT/3-Year/32-DAA/DAA_UNIT-3-1.pdf

2. (a) What are Implicit and Explicit constraints? Write Implicit and Explicit constraints for n-queens and sum of subset problems.

Solution:

1. **Explicit constraints** are rules that restrict each x_i to take on values only from a given set.
   – Examples of explicit constraints
   * $x_i \geq 0$, or all nonnegative real numbers
   * $x_i = \{0, 1\}$
   * $l_i \leq x_i \leq u_i$

2. **Implicit constraints** are rules that determine which of the tuples in the solution space of I satisfy the criterion function.
   – Implicit constraints describe the way in which the x_i s must relate to each other.

• N-Queens Problem:

Let us assume, N = 8

–Explicit constraints

* Explicit constraints using 8-tuple formulation are $S_i = \{1, 2, 3, 4, 5, 6, 7, 8\}, 1 \leq i \leq 8$
* Solution space of 8^8 8-tuples.

–Implicit constraints

* No two x_i can be the same, or all the queens must be in different columns.
  · All solutions are permutations of the 8-tuple (1, 2, 3, 4, 5, 6, 7, 8).
  · Reduces the size of solution space from 8^8 to 8! tuples.
* No two queens can be on the same diagonal.

• Sum of subsets

– Explicit constraints

* $x_i \in \{j \mid j \text{ is an integer and } 1 \leq j \leq n\}$

– Implicit constraints

* No two x_i can be the same
* $\sum w_{x_i} = m$
* $x_i < x_i + 1, 1 \leq i < k$ (total order in indices)
  · Helps in avoiding the generation of multiple instances of same subset; (1, 2, 4) and (1,4, 2) are the same subset

Source: http://www.cs.umsl.edu/~sanjiv/classes/cs5130/lectures/bt.pdf

2. (b) Could Not Solve.

Sources:

1. https://www.youtube.com/watch?v=vLS-zRCHo-Y

2. https://www.youtube.com/watch?v=wAy6nDMPYAE

2. (c) Define parallel algorithms. Explain Speedup and Efficiency of parallel algorithms.

Solution:

Parallel Algorithm In computer science, a parallel algorithm or concurrent algorithm, is an algorithm which can do multiple operations at a given time on many different processing devices, and then put back together again at the end to get the correct result. Parallel algorithm is opposite of a traditional sequential (or serial) algorithm.

Speedup

• The speedup is defined as the ratio of the serial runtime of the best sequential algorithm for solving a problem to the time taken by the parallel algorithm to solve the same problem on p processors.

$$S = \frac{T_s}{T_p}$$

Efficiency

• The efficiency is defined as the ratio of speedup to the number of processors. Efficiency measures the fraction of time for which a processor is usefully utilized.

$$E = \frac{S}{p} = \frac{T_s}{pT_p}$$

Source: https://www.cs.uky.edu/~jzhang/CS621/chapter7.pdf?fbclid=IwAR3r-XAI9Esvifmn7exwsqvGfboS4dhb8QKmRnRMwxbvQ4p_rvPuzE5ZkTM

2. (d) In the Floyd-Warshall all-pairs shortest algorithm, there are three nested loops. There are six possible permutations of these three loops. Which ones provide a correct algorithm?

Solution:

There are 6 possible orderings of the loops in the Floyd-Warshall algorithm and they are,

1. IJK 2. JIK

3. IKJ 4. JKI

5. KIJ 6. KJI

Only the last two orderings **KIJ** and **KJI** will provide us with the correct result.

Floyd-Warshall algorithm is the dynamic-programming formulation to solve all-pairs shortest paths problem. The recursive formula is given by:

$$shortestPath(i, j, 0) = w(i, j)$$

$$shortesPath(i, j, k+1) = \min\left(shortestPath(i, j, k), shortestPath(i, k+1, k) + shortesPath(k+1, j, k)\right)$$

where $shortestPath(i,\ j,\ k)$: represents the shortest possible path from i to j using vertices only from the set {1, 2, ..., k} as intermediate points along the way.

Based on this recurrence, the bottom-up procedure computes the value of $shortestPath(i,\ j,\ k)$: in order of increasing value of k. That means, we can find the shortest path from i to j with intermediate vertices {1, 2, ..., k}, if we know

$shortestPath(i,\ j,\ k-1)$: the shortest path from i to j with intermediate vertices {1, 2, ..., k-1}

$shortestPath(i,\ k,\ k-1)$: the shortest path from i to k with intermediate vertices {1, 2, ..., k-1}

$shortestPath(k,\ j,\ k-1)$: the shortest path from k to j with intermediate vertices {1, 2, ..., k-1}

Since smaller sub-problems need to be solved first and the sub-problems size is controlled by k so we should always put k in the outermost for loop (As, k governs which vertices you are permitted to use internal to your path from source i to destination j). The order of i and j are irrelevant but k must go first as it controls the size of the sub-problems.

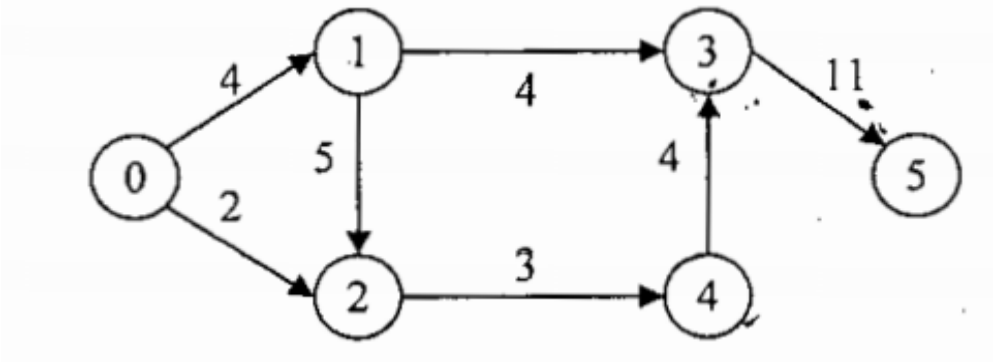
Sources:

1. https://www.quora.com/Why-is-the-order-of-the-loops-in-Floyd-Warshall-algorithm-important-to-its-correctness

2. https://discuss.codechef.com/t/floyd-warshall-algorithm-doubt/13716

3. https://stackoverflow.com/questions/27700629/why-does-order-of-3-loops-in-floyd-warshall-matters

3. (a) Consider the following travelling salesperson problem (TSP):



i. Convert it into a multistage graph.

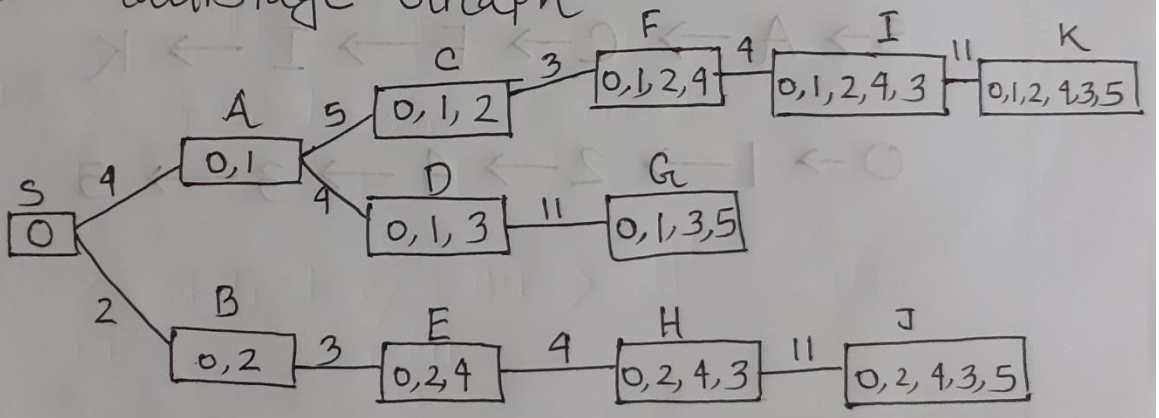
ii. Find the minimum cost path in the multistage graph (from (i). Do this using the forward reasoning approach.

Solution:

Source: https://www.youtube.com/watch?v=9iE9Mj4m8jk

	0	1	2	3	4	5
0	∞	4	2	∞	∞	∞
1	∞	∞	5	4	∞	∞
2	∞	∞	∞	∞	3	∞
3	∞	∞	∞	∞	∞	11
4	∞	∞	∞	4	∞	∞
5	∞	∞	∞	∞	∞	∞

i. Multistage Graph



ii. $d(S, A) = 4$ $d(S, B) = 2$

$d(S, C) = 9$, $d(S, D) = 8$, $d(S, E) = 5$

$d(S, F) = 12$, $d(S, G) = 19$, $d(S, H) = 9$

$d(S, I) = 16$, $d(S, J) = 20$

$d(S, K) = 27$

∴ Path will be:

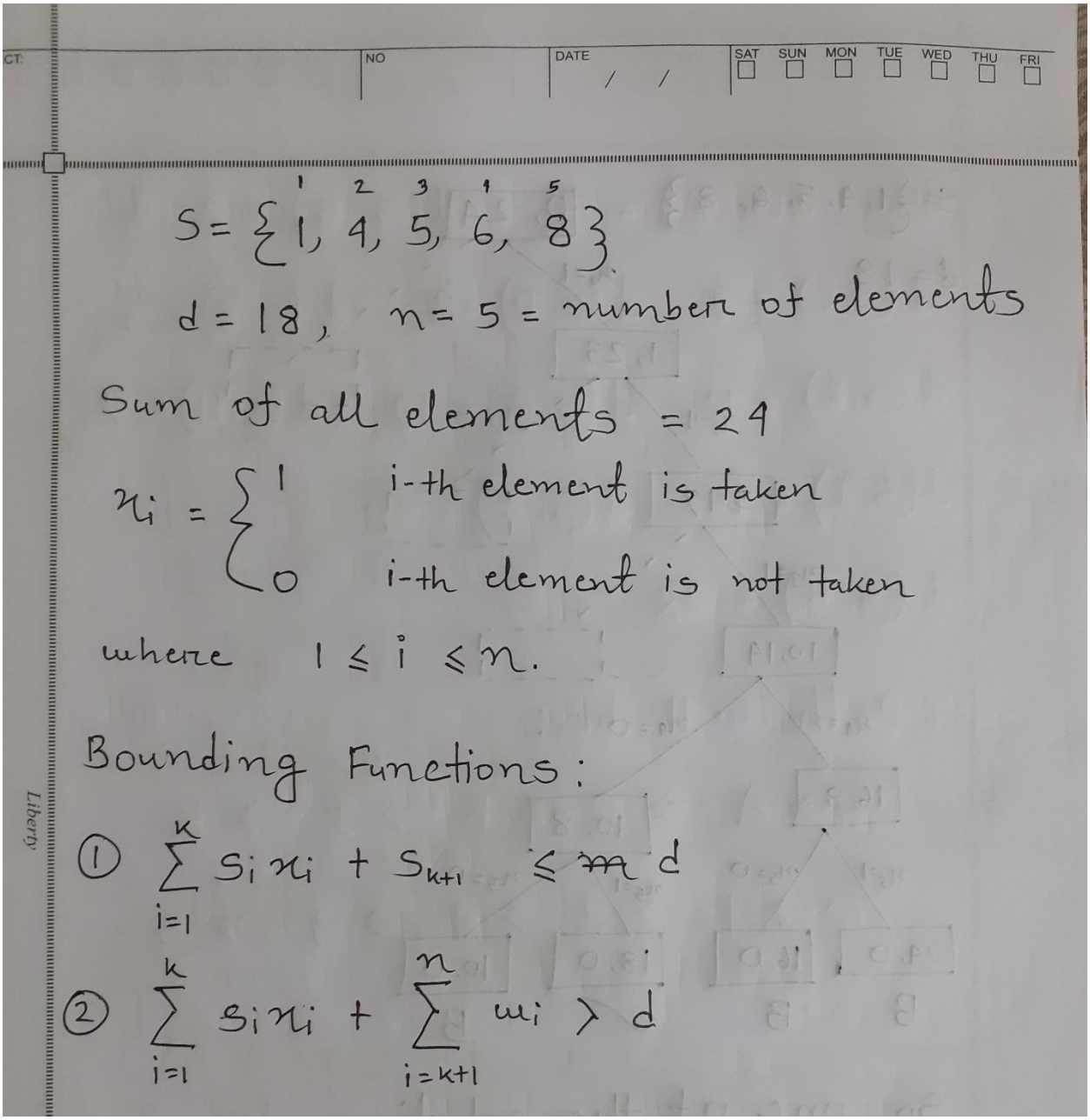
$S \rightarrow A \rightarrow C \rightarrow F \rightarrow I \rightarrow K$

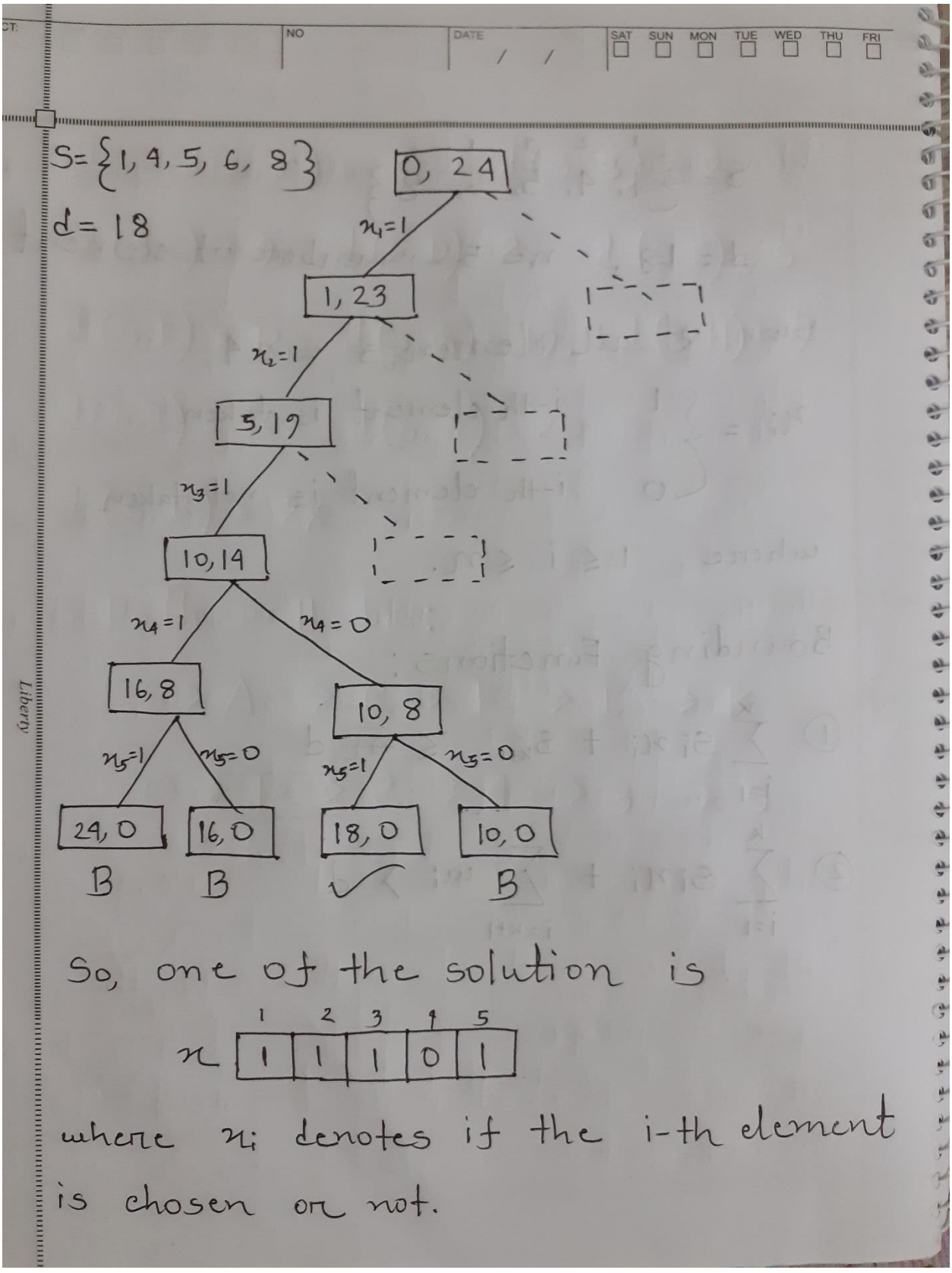
$0 \rightarrow 1 \rightarrow 2 \rightarrow 4 \rightarrow 3 \rightarrow 5$

3. (b) Apply backtracking technique to solve the following instance of subset sum problem:

S = {1, 4, 5, 6, 8} and d = 18.

Solution:





Source: https://www.youtube.com/watch?v=kyLxTdsT8ws

3. (c) What are the differences between branch-and-bound and backtracking paradigm?

Solution:

Branch and Bound	Backtracking
1. It is used to solve optimization problem.	1. It is used to find all possible solutions available to a problem.
2. It may traverse the tree in any manner, DFS or BFS.	2. It traverses the state space tree by **DFS**(Depth First Search) manner.
3. It realizes that it already has a better optimal solution that the pre-solution leads to so it abandons that pre-solution.	3. It realizes that it has made a bad choice & undoes the last choice by backing up.
4. It completely searches the state space tree to get optimal solution.	4. It searches the state space tree until it has found a solution.
5. It involves a bounding function.	5. It involves feasibility function.

Source: https://stackoverflow.com/questions/30025421/difference-between-backtracking-and-branch-and-bound

3. (d) Draw the state space tree for 4-queen problem. How is the solution reduced from 4^4 to optimal solution? Explain.

Solution:

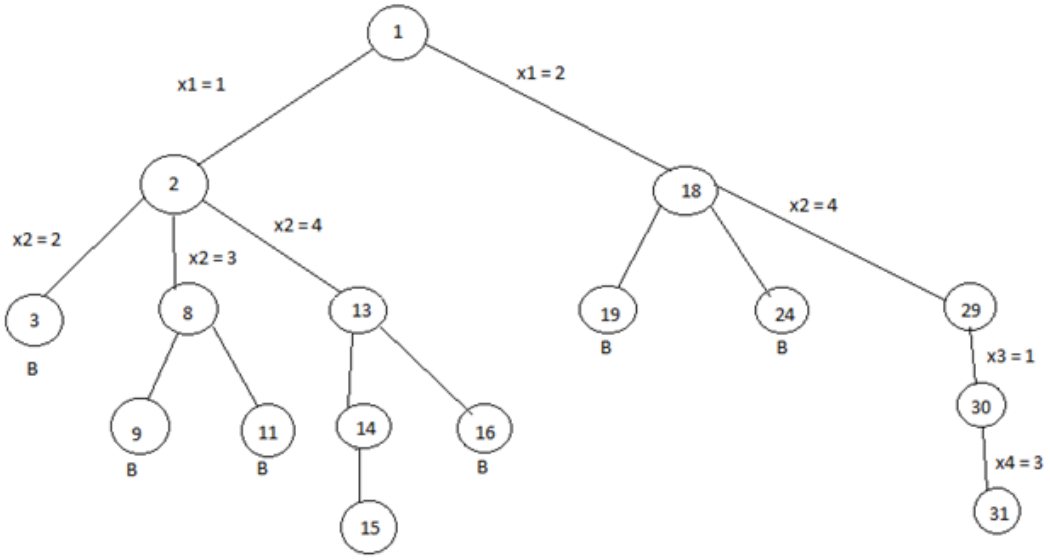


Figure: State Space Tree for 4-Queen's problem.

The reason why solution space is reduced from 4^4 to 4! is due to the constraints of the problem. The constraints are:

1. All the queens must be in different columns.

2. No two queens can be on the same diagonal.

This means, all solutions must the permutations of (1, 2, 3, 4). We won't have solutions like (1, 1, 1, 1) or (1, 1, 2, 2). [(1, 2, 3, 4) these indicate column numbers]

We will have 4 columns to put the first queen. Then, we will have to put the second queen on one of the remaining 3 columns and so on.

So, the solution space is

$$4 * 3 * 2 * 1 = 4!$$

If the constraints weren't there, our solution space would be

$$4 * 4 * 4 * 4 = 4^4$$

Sources:

1. https://www.youtube.com/watch?v=xFv_Hl4B83A&t=608s

2. https://www.includehelp.com/algorithms/4-queens-problem-and-solution-using-backtracking-algorithm.aspx

4. (a) What are the differences between performance analysis and performance measurement of an algorithm?

Solution:

Performance analysis is the estimation that is applied after developing an algorithm by checking the following questions:

1. Does it do what we want it to do?
2. Does it work correctly according to the original specification of the task?
3. Is there documentation that describes how to use it and how it works?
4. Are the procedures created in such a way that they function logically?
5. Is the code readable?

On the other hand, performance measurement is the process of executing a correct program on data sets and measuring the time and space it takes to compute the result. These timing figures are useful as they may confirm a previously done analysis and point out logical places to perform useful optimizations.

4. (b) What is the running time in θ-notation (as a function of n) of the following code? Give an explanation.

Solution:

STATEMENT	TOTAL STEPS
for x = 1 to n do	n + 1
begin	
y = x	n
while (y > 1) do	n * (logn + 1)
y = y / 2	n * logn
end	
Total	3n + 2nlogn + 1

Here, $f(n) = 3n + 2nlogn + 1$

The function $f(n) = \theta(g(n))$ if and only if there exist positive constants c_1, c_2 and n_0 such that

$c_1 g(n) \leq f(n) \leq c_2\, g(n)$ for all n, $n > n_0$

So, we can say that,

$$nlogn \leq 3n + 2nlogn + 1 \leq 3nlogn + 2nlogn + nlogn$$

or, $nlogn \leq 3n + 2nlogn + 1 \leq 6nlogn$

That is, $f(n) = \theta(nlogn)$

Assertion: The inner while loop will run at most $log_2 n$ times.

Suppose, n = 8. The while loop will compute the following values when y = x = 8

y = 8 / 2 = 4

then, y = 4 / 2 = 2

then, y = 2 / 2 = 1

When y = 1, the condition y > 1 will fail and the loop will terminate after 3 iterations.

As $\log_2 8 = 3$, we can say that, the number of iterations for the inner while loop will always be $log_2 n$ times at most.

Since the outer for loop runs for *n* times and the inner while loop runs for *logn* times the running time of the algorithm would be $\theta(nlogn)$.

Sources:

1. Fundamentals of Computer Algorithms by Ellis Horowitz, **Page-25**

2. https://www.youtube.com/watch?v=Nd0XDY-jVHs

4. (c) Describe the steps in applying dynamic programming strategy when developing an algorithm.

Solution:

Two main properties of a problem suggest that the given problem can be solved using Dynamic Programming. These properties are **overlapping sub-problems and optimal substructure**.

Overlapping Sub-Problems:

Similar to Divide-and-Conquer approach, Dynamic Programming also combines solutions to sub-problems. It is mainly used where the solution of one sub-problem is needed repeatedly. The computed solutions are stored in a table, so that these don't have to be re-computed. Hence, this technique is needed where overlapping sub-problem exists.

Optimal Sub-Structure:

A given problem has Optimal Substructure Property, if the optimal solution of the given problem can be obtained using optimal solutions of its sub-problems.

Steps of Dynamic Programming Approach:

Dynamic Programming algorithm is designed using the following four steps −

- Characterize the structure of an optimal solution.
- Recursively define the value of an optimal solution.
- Compute the value of an optimal solution, typically in a bottom-up fashion.
- Construct an optimal solution from the computed information.

Source:https://www.tutorialspoint.com/design_and_analysis_of_algorithms/design_and_analysis_of_algorithms_dynamic_programming.htm

4. (d) The knapsack problem takes 2 inputs: the capacity of the knapsack and a list of items. Each item has a value and weight. The goal is to put items into the knapsack such that the value is maximized (optimal) but we cannot go over the weight capacity of the knapsack (valid). As an example, consider a knapsack with capacity 15 and items (25, 5), (45, 11), (12, 3), (7, 2), (6, 2), (10, 7) and (5, 4) where the first number in each pair indicates the value and the second number the weight. Valid solutions are: (), (1, 3, 4), (2, 3, 4, 5) and an optimal solution is (2, 7).

i) · Describe a possible way of creating a state space tree for this problem.

ii) Design a heuristic for a branch-and-bound algorithm.

A) Will the branch-and-bound algorithm return a valid solution?

B) Will the branch-and-bound algorithm return an optimal solution?

Solution:

(d)(i) In this problem, the capacity is 15, and the item are,

item	weight(w)	value(p)	$\frac{p}{w}$
x_1	5	25	5
x_2	11	45	4.09
x_3	3	12	4
x_4	2	7	3.5
x_5	2	6	3
x_6	7	10	1.43
x_7	4	5	1.25

we considered the items as $\{x_1 \ldots x_7\}$.

If $x_i = 1$, that means we included x_i element; otherwise, $x_i = 0$.

So

So, ~~if we create~~ to create state space tree, we will use Least Cost Branch and bound.

Here, each node will contain

Cost $c(x) = -\sum_{i=1}^{n} p_i x_i$ where, $x_i = 0$ or 1

and, total weight $w \leq 15$ include

each total value will also ~~calculate~~ fractional value.

and,

upperbound $ub(x) = -\sum_{i=1}^{n} p_i x_i$ where $x_i = 0$ or 1

and total weight $w \leq 15$ and ~~total~~ $ub(x)$ will not contain fractional value.

So, the state space tree is:

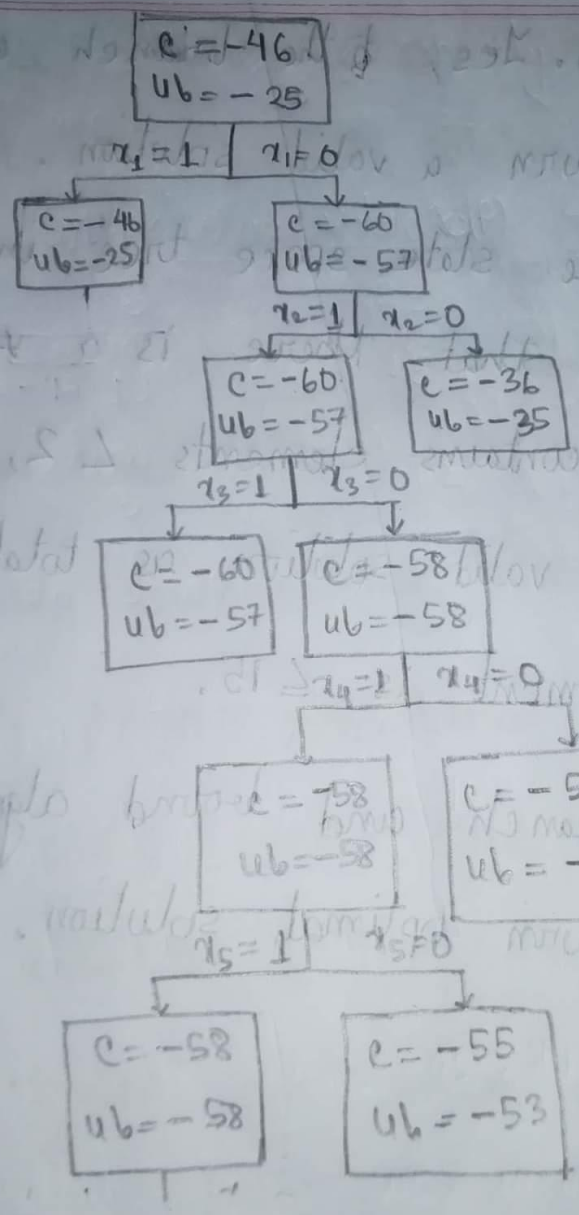


Figure: State space tree for the problem

Sources:

1. https://www.facebook.com/profile.php?id=100005486505136

2. https://www.youtube.com/watch?v=yV1d-b_NeK8