FAT Exam

About Subject

- Understand the concepts via solving problems
- No question will be asked as it is taught, your originality will be tested – (i.e.) Do not expect a question "Write LCS"
- Apply your ideas
- No memorization

About Question Paper

- Read a question fully – Do not start thinking / understand terms used in the question inbetween
- You need to put more time on understanding and thinking rather than writing
- 10 questions will be given
- All carry equal marks but may not be of same complexity
- Some questions may easy to solve and some may require more time

Good Scoring

- Preparation and Knowledge in subject – 50%
- Time management and Presentation – 50%
- Pre–requisite: Good health and sleep before attending exams
- Eat well, sleep well
- Talk less, discuss more before exam and <u>*do not discuss after exam*</u>
- Don't worry about anything which is not in your control

Basic Preparation

- Go through problems in the handout mentioned from page 3 to 7
- Have a thorough understanding of the problem
- Better work out (write on a paper) and learn
- Pen and paper will help you a lot always to have a good understanding

Better Preparation

- Learn algorithms of problems dealt in the class
- Proofs may be skipped for exams but they are very important to understand
- Take slides and go through

Day of Examination

- Enter into exam hall earlier
- Fill first page of your answer sheet carefully
- Question paper will be distributed 5 to 10 minutes ahead
- Read the question paper thoroughly
- Don't panic if it takes time
- You can read and understand the question paper till 9 : 15 am

Day of Examination

- Have a rough estimate of time required to answer each question
- Arrange the question so that first you answer whichever is easy for you
- Basically shortest job first algorithm
- Never start a question without fully understanding

Components to be Written

- Write the basic logic / technique that you are going to use
- Write the algorithm with minimal English words
- Do an illustration
- Refine algorithm if required
- Do time and space complexity analysis of the steps of the algorithm
- Express time complexity in Big–Oh notation
- Everything can be done with pen. No beautification is required

Points to remember

- If you spend more time on a question then leave some space and move on to next question
- You can come back if time permits
- Never leave a question unanswered
- Put question number with clarity
- Do not write any theory
- Do only what is asked in the question

Example Problem 1

- Given two sequences of numbers S1 and S2 of equal size, design an algorithm to find the maximum sum of a^b where a is a number in S1 and b is the number in S2. Numbers a and b can used in the computation only once
- max $(\sum a^b)$ where $a \in S1$ and $b \in S2$
- For example if S1 = {12, 4, 45} and S2 = {4, 2, 3} maximum value possible is 4102369

Example Problem 1

- This is a easy problem
- Question setter will not expect the student to answer more than 15 minutes to answer this question
- For few questions answer can be a single page also
- You will given marks as long as it is complete and correct

Basic Idea / Technique Followed

- Arranging number in descending order and taking power of corressponding elements
- Greedy

Algorithm

Step 1: Read the number of elements in sets S1 and S2, let it be n

Step 2: Read elements of S1 and S2

Step 3: Sort the elements in descending order

Sep 4: Let sum = 0

Sep 5: For i = 1 to n

sum + = S1[i] ^ S2[i]

Step 6: print sum

Illustration

S1 = {12, 4, 45} and S2 = {4, 2, 3}

sum = 0

Sort in descending order

S1 = {45, 12, 4}, S2 = {4, 3, 2}

sum += 45^4 = 4100625

sum += 12^3 = 4102353

sum += 4^ 2 = 4102369

print 4102369

Complexity Analysis

Step 3: Sort the elements in descending order

Best possible – Merge sort – O(n log n)

Sep 4: Let sum = 0 – O(1)

Sep 5: For i = 1 to n – O(n)

sum + = S1[i] ^ S2[i]

Step 6: print sum – O(1)

Time Complexity – O(n log n)

Space Complexity – No extra space other than input – O(1)

Example 2 – Partition problem

- Determine whether a given set can be partitioned into two subsets such that the sum of elements in both subsets is the same
- a = {1, 5, 11, 5} – True
- a = {1, 5, 3} – False

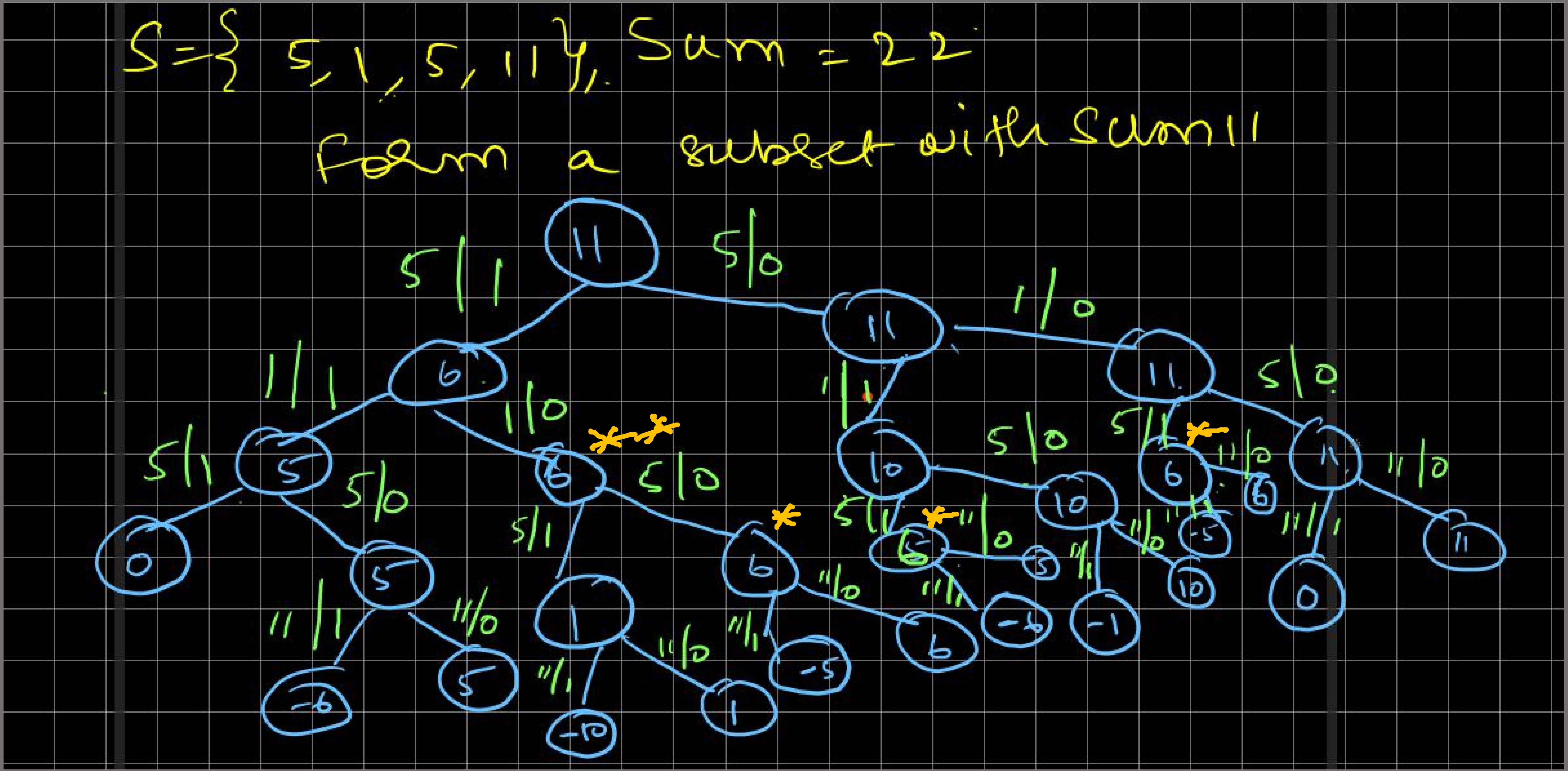
Example 2 – Partition problem

- This is a difficult question which may need more time to analyze different cases and figure out an algorithm
- Question setter will expect the student to take 20 to 30 minutes to answer this question

Basic Idea

- Include / Not include a number in a set
- Dynamic programming – Problem has optimal substructure
- Optimal solution can be constructed from optimal solutions of its subproblems
- {1, 5} – sum 6 can be formed from this
- {1, 5, 11} – still sum 6 can be formed by picking a subset with elements 1 and 5 only
- Overlapping subproblem – draw state space tree and see

Basic Idea



Recursive Solution

- In the previous image, 6 with only one star are same beacuse they have still the same subset remaining
- 6 with two stars are not same as others with one star as the subset remaining is different

Recursive Solution

1) Calculate sum of the array. If sum is odd, there can not be two subsets with equal sum, so return false.

2) If sum of array elements is even, calculate sum/2 and find a subset of array with sum equal to sum/2.

Recursive Solution

We start from the last element of the set and branch out for including the n^{th} element and not including it

subsetSum (arr, n, sum/2) = subsetSum (arr, n–1, sum/2) or

subsetSum (arr, n–1, sum/2–arr[n])

Bottom up DP

- Form a two dimenstional table – Store boolean values
- Rows representing desired sum from 1 to sum
- Columns representing subsets of given set with elements one by one included that is jth column of table contains j elements from given set

Bottom up DP

- table[i, j] = true if subset represented by column j can form sum i
- False otherwise
- Fill all cells in partition table from 1 x 1 to sum x n, where n is the number of elements in given set

Table Construction in Bottom up DP

Condition 1 – Applied always

- Copy previous column value (i.e.) table[i,j] = table[i][j–1] based on optimal substructure
- If a sum can be fromed from a set S1 then it can be formed from its superset
- that is {1,5} can form sum 6 then from its superset {1, 5, 5} we can form sum 6

Table Construction in Bottom up DP

- Condition 2 – Apply only if sum is greater than or equal to element in the given array which is getting entry added in the subset
- entry in the current cell = entry in the previous column of current row or entry in the previous column of row corressponding to sum without the element that is added into the subset
- sum>= arr[j–1] then
- table[i][j] or table[i–arr[j–1]][j–1]

Initialization of Table

	{}	{5}	{5,1}	{5,1,5}	{5,1,5,11}
0	True	True	True	True	True
1	False				
2	False				
3	False				
4	False				
5	False				
6	False				
7	False				
8	False				
9	False				
10	False				
11	False				

	{}	{5}	{5,1}	{5,1,5}	{5,1,5,1 1}
0	True	True	True	True	True
1	False	False	False		
2	False				
3	False				
4	False				
5	False				
6	False				
7	False				
8	False				
9	False				
10	False				
11	False				

- table[1][2] is false as per condition 1
- Since 1 >= 1 the new element in the subset apply condition 2
- table[1][2] = false or table[1–1][0]
- table[1][2] = false or true = true

Filling of Table

	{}	{5}	{5,1}	{5,1,5}	{5,1,5,11}
0	True	True	True	True	True
1	False	False	True	True	True
2	False	False	False	False	False
3	False	False	False	False	False
4	False	False	False	False	False
5	False	True	True	True	True
6	False	False	True	True	True
7	False	False	False	False	False
8	False	False	False	False	False
9	False	False	False	False	False
10	False	False	False	True	True
11	False	False	False	True	True

Time and Space Complexity

- Time complexity – O(Sum*n) where n is the number of elements in the set

Wish you all the best