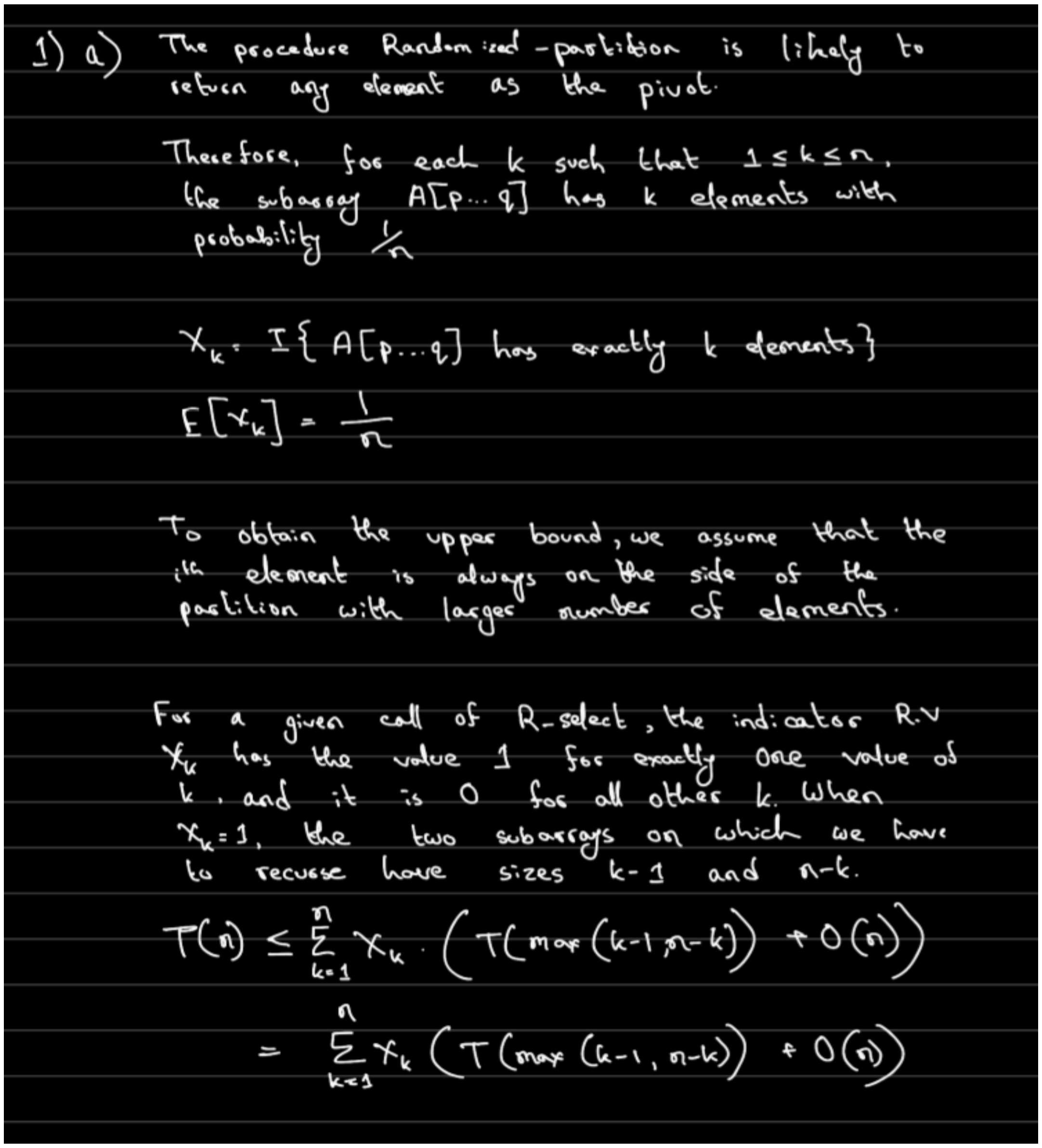
2020-2021

1. a) Prove that the expected running time of a randomized select algorithm is ***O(n)***.



1) a) The procedure Randomized-partition is likely to return any element as the pivot.

Therefore, for each k such that $1 \leq k \leq n$, the subarray $A[p \ldots q]$ has k elements with probability $1/n$

$X_k = I\{A[p \ldots q] \text{ has exactly } k \text{ elements}\}$

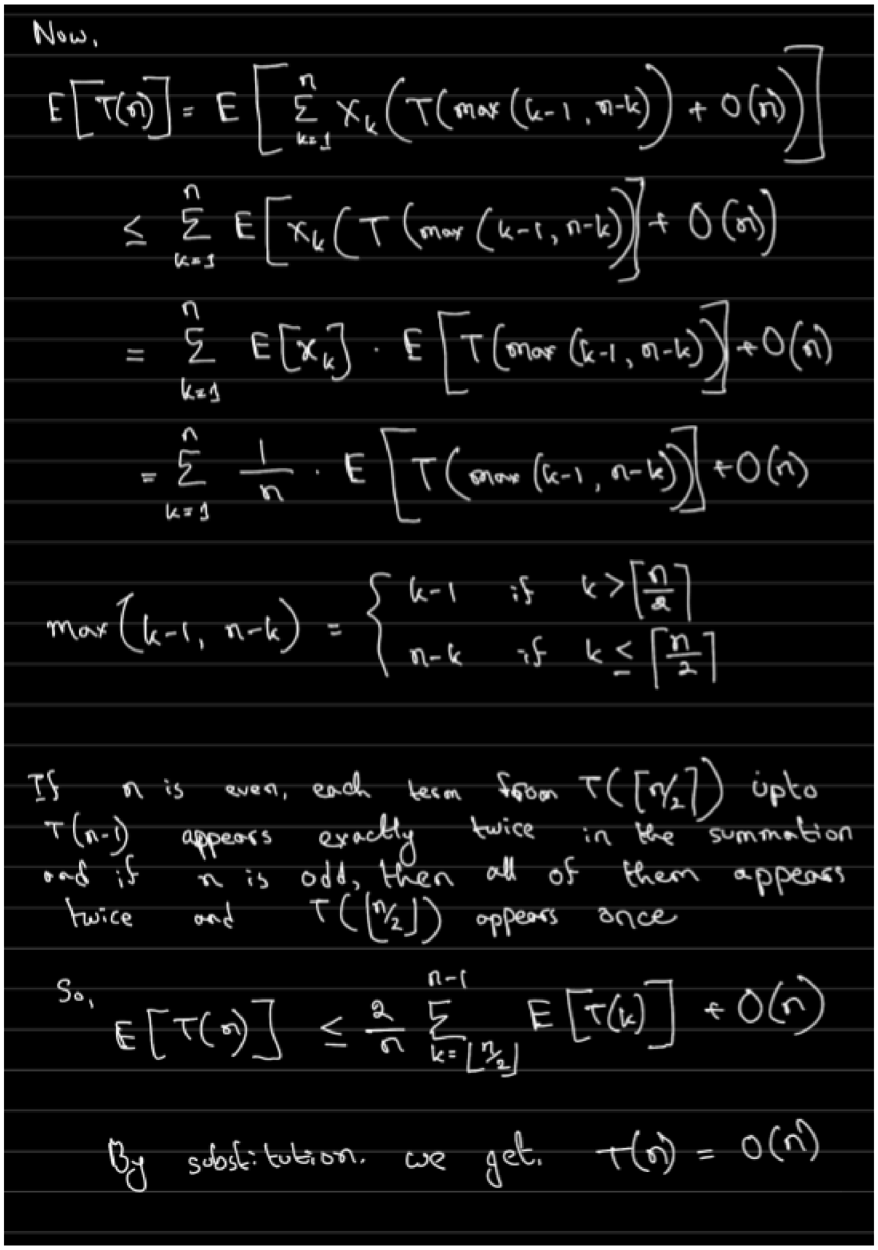
$E[X_k] = \frac{1}{n}$

To obtain the upper bound, we assume that the ith element is always on the side of the partition with larger number of elements.

For a given call of R-select, the indicator R.V X_k has the value 1 for exactly one value of k, and it is 0 for all other k. When $X_k = 1$, the two subarrays on which we have to recurse have sizes $k-1$ and $n-k$.

$$T(n) \leq \sum_{k=1}^{n} X_k \cdot \left(T(\max(k-1, n-k)) + O(n)\right)$$

$$= \sum_{k=1}^{n} X_k \left(T(\max(k-1, n-k)) + O(n)\right)$$



Now,

$$E[T(n)] = E\left[\sum_{k=1}^{n} X_k\left(T(\max(k-1, n-k)) + O(n)\right)\right]$$

$$\le \sum_{k=1}^{n} E\left[X_k\left(T(\max(k-1, n-k))\right)\right] + O(n)$$

$$= \sum_{k=1}^{n} E[X_k] \cdot E\left[T(\max(k-1, n-k))\right] + O(n)$$

$$= \sum_{k=1}^{n} \frac{1}{n} \cdot E\left[T(\max(k-1, n-k))\right] + O(n)$$

$$\max(k-1, n-k) = \begin{cases} k-1 & \text{if } k > \lceil \frac{n}{2} \rceil \\ n-k & \text{if } k \le \lceil \frac{n}{2} \rceil \end{cases}$$

If n is even, each term from $T(\lceil n/2 \rceil)$ upto $T(n-1)$ appears exactly twice in the summation and if n is odd, then all of them appears twice and $T(\lfloor n/2 \rfloor)$ appears once

So,

$$E[T(n)] \le \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} E[T(k)] + O(n)$$

By substitution, we get, $T(n) = O(n)$

b) Prove that the expected number of comparisons in a randomized quicksort is ***nlgn***.

Randomized Quick Sort

Let us think about when two items are *not* compared. Consider an input to quicksort of the numbers 1 through 10 (in any order), and suppose that the first pivot element is 7. Then the first call to PARTITION separates the numbers into two sets: $\{1, 2, 3, 4, 5, 6\}$ and $\{8, 9, 10\}$. In doing so, the pivot element 7 is compared to all other elements, but no number from the first set (e.g., 2) is or ever will be compared to any number from the second set (e.g., 9).

In general, because we assume that element values are distinct, once a pivot x is chosen with $z_i < x < z_j$, we know that z_i and z_j cannot be compared at any subsequent time. If, on the other hand, z_i is chosen as a pivot before any other item in Z_{ij}, then z_i will be compared to each item in Z_{ij}, except for itself. Similarly, if z_j is chosen as a pivot before any other item in Z_{ij}, then z_j will be compared to each item in Z_{ij}, except for itself. In our example, the values 7 and 9 are compared because 7 is the first item from $Z_{7,9}$ to be chosen as a pivot. In contrast, 2 and 9 will never be compared because the first pivot element chosen from $Z_{2,9}$ is 7. Thus, z_i and z_j are compared if and only if the first element to be chosen as a pivot from Z_{ij} is either z_i or z_j.

We now compute the probability that this event occurs. Prior to the point at which an element from Z_{ij} has been chosen as a pivot, the whole set Z_{ij} is together in the same partition. Therefore, any element of Z_{ij} is equally likely to be the first one chosen as a pivot. Because the set Z_{ij} has $j-i+1$ elements, and because pivots are chosen randomly and independently, the probability that any given element is the first one chosen as a pivot is $1/(j-i+1)$. Thus, we have

$$\begin{aligned}
\Pr\{z_i \text{ is compared to } z_j\} &= \Pr\{z_i \text{ or } z_j \text{ is first pivot chosen from } Z_{ij}\} \\
&= \Pr\{z_i \text{ is first pivot chosen from } Z_{ij}\} \\
&\quad + \Pr\{z_j \text{ is first pivot chosen from } Z_{ij}\} \\
&= \frac{1}{j-i+1} + \frac{1}{j-i+1} \\
&= \frac{2}{j-i+1} \, . \qquad (7.3)
\end{aligned}$$

The second line follows because the two events are mutually exclusive. Combining equations (7.2) and (7.3), we get that

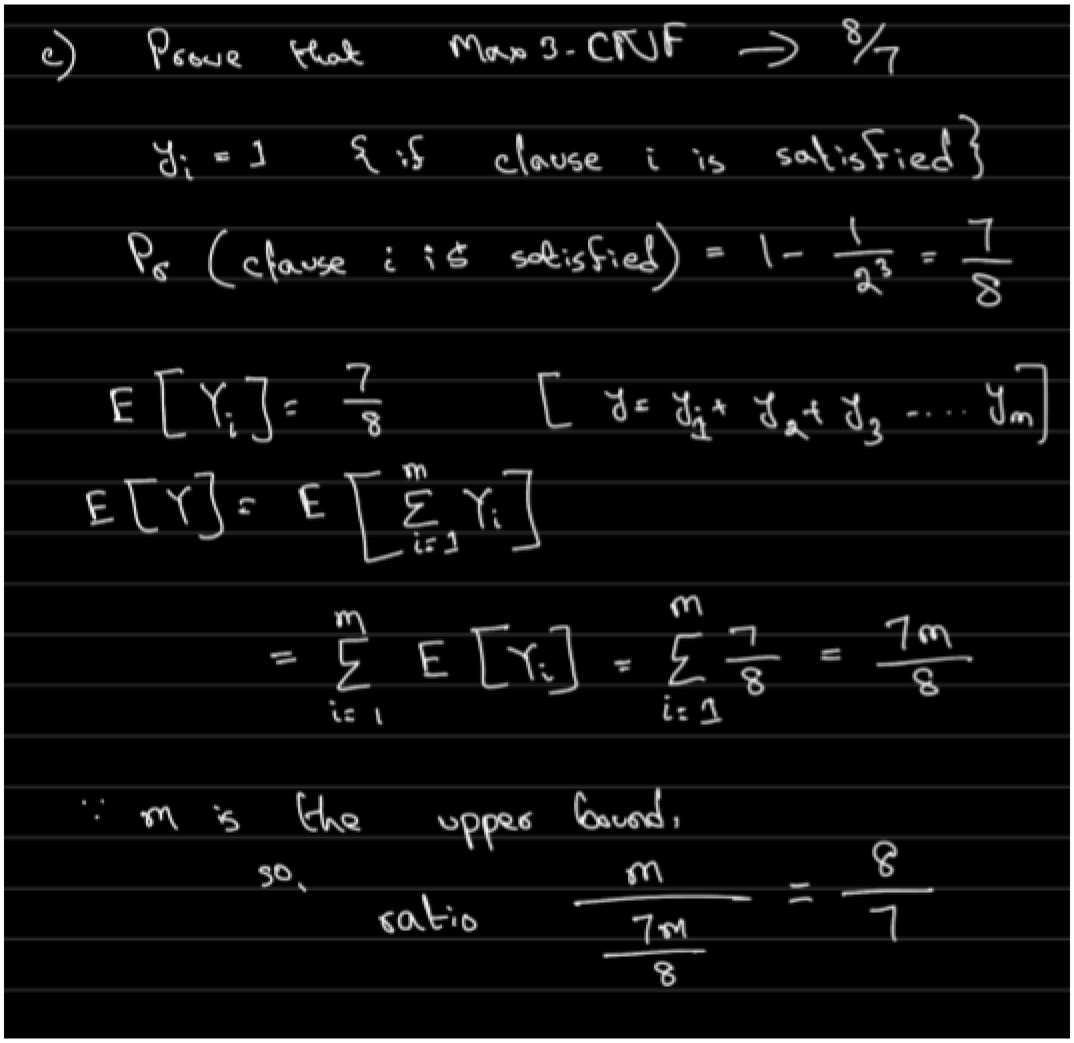
$$\mathrm{E}[X] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1} .$$

We can evaluate this sum using a change of variables ($k = j - i$) and the bound on the harmonic series in equation (A.7):

$$\begin{aligned}
\mathrm{E}[X] &= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1} \\
&= \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{2}{k+1} \\
&< \sum_{i=1}^{n-1} \sum_{k=1}^{n} \frac{2}{k} \\
&= \sum_{i=1}^{n-1} O(\lg n) \\
&= O(n \lg n) . \qquad (7.4)
\end{aligned}$$

Thus we conclude that, using RANDOMIZED-PARTITION, the expected running time of quicksort is $O(n \lg n)$ when element values are distinct.

c) Prove that randomized MAX-3-CNF algorithm is an ***8/7 -approximation*** algorithm.



c) Prove that Max 3-CNF $\rightarrow 8/7$

$y_i = 1$ $\{$ if clause i is satisfied $\}$

$$\Pr(\text{clause } i \text{ is satisfied}) = 1 - \frac{1}{2^3} = \frac{7}{8}$$

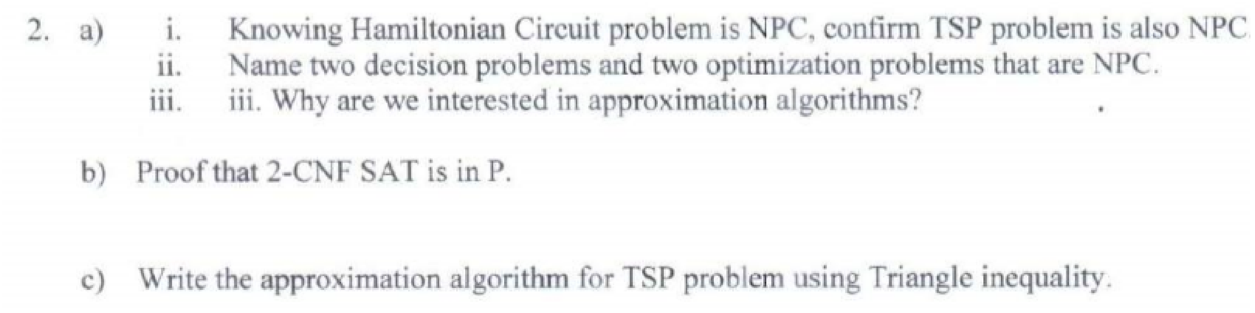
$$E[Y_i] = \frac{7}{8} \qquad [\, Y = y_1 + y_2 + y_3 \dots y_m \,]$$

$$E[Y] = E\left[\sum_{i=1}^{m} Y_i\right]$$

$$= \sum_{i=1}^{m} E[Y_i] = \sum_{i=1}^{m} \frac{7}{8} = \frac{7m}{8}$$

$\because$ m is the upper bound,

so, ratio $\dfrac{m}{\frac{7m}{8}} = \dfrac{8}{7}$



2. a) i. Knowing Hamiltonian Circuit problem is NPC, confirm TSP problem is also NPC.
   ii. Name two decision problems and two optimization problems that are NPC.
   iii. iii. Why are we interested in approximation algorithms?

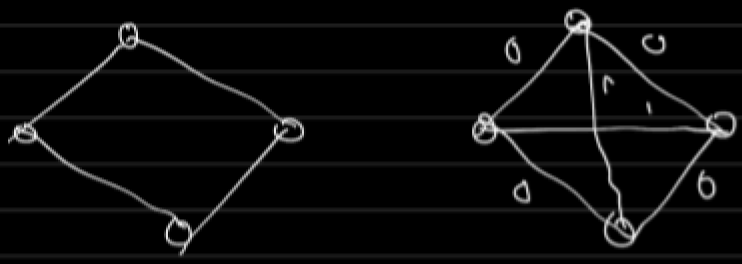
b) Proof that 2-CNF SAT is in P.

c) Write the approximation algorithm for TSP problem using Triangle inequality.

2 a)

2) a) i) TSP is NP [because verifiable]

Hamilton can be reduced to TSP



$$c(i,j) = \begin{cases} 0 & \text{if } (i,j) \in E \\ 1 & \text{if } (i,j) \notin E \end{cases}$$

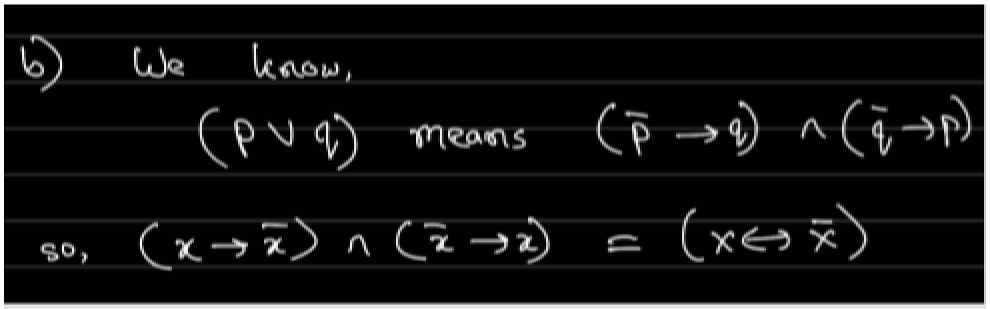
if tour-cost at most 0 ; then Hamitonian cycle exist.

ii) Decision → 3-CNF, Boolean SAT,

Optimization → Hamilton, TSP

iii) ii) Approx can approx in polynomial time.

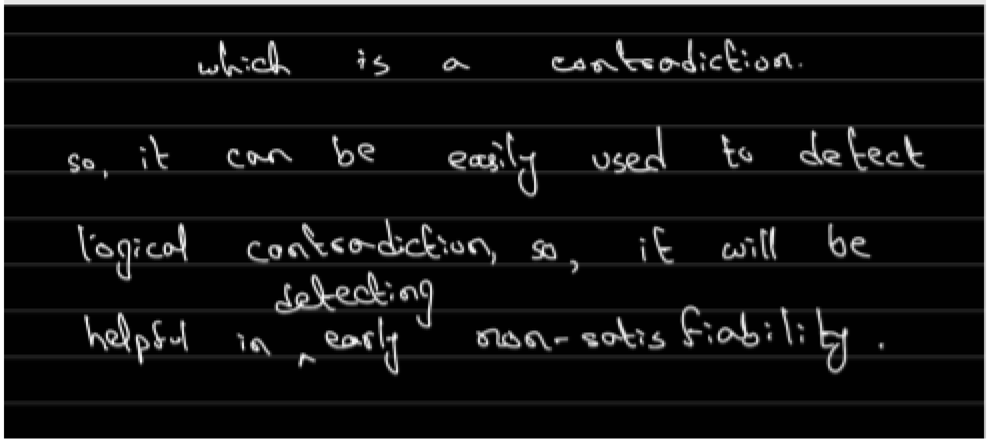
2b)



b) We know,

$(p \vee q)$ means $(\bar{p} \rightarrow q) \wedge (\bar{q} \rightarrow p)$

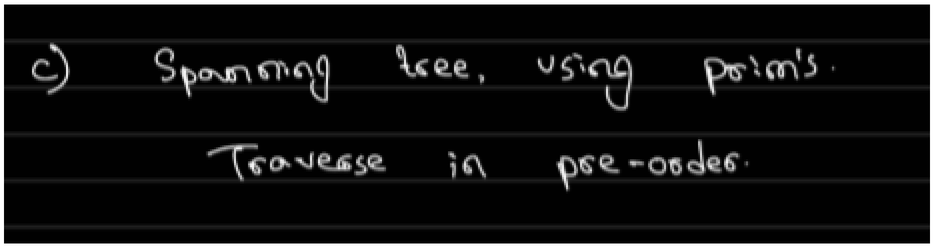
so, $(x \rightarrow \bar{x}) \wedge (\bar{x} \rightarrow x) = (x \Leftrightarrow \bar{x})$



which is a contradiction.

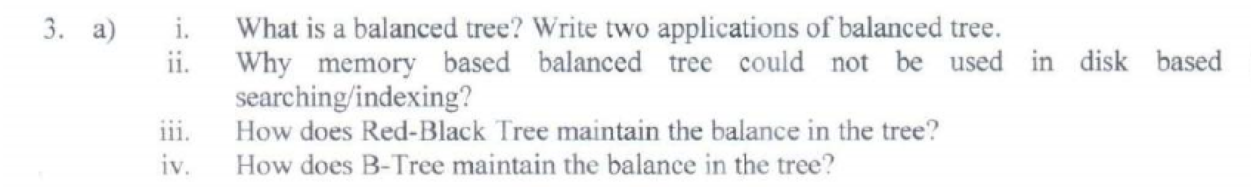
so, it can be easily used to defect logical contradiction, so, it will be helpful in detecting early non-satisfiability.

2c)



c) Spanning tree, using prim's.

Traverse in pre-order.



3. a) i. What is a balanced tree? Write two applications of balanced tree.
   ii. Why memory based balanced tree could not be used in disk based searching/indexing?
   iii. How does Red-Black Tree maintain the balance in the tree?
   iv. How does B-Tree maintain the balance in the tree?

i. What is a balanced tree?

A balanced tree is a type of binary tree structure where the height of the tree is kept to a minimum possible level by ensuring that **the depth of the two subtrees of any node never differ by more than one**. This balance helps maintain efficient operations like insertion, deletion, and search.

Databases: Balanced trees, such as B-trees and Red-Black trees, are widely used in databases and file systems for efficient data retrieval and storage.

Network Routing: Certain types of balanced trees are used in network routing algorithms to manage and store routing tables efficiently

ii. Why memory-based balanced trees could not be used in disk-based searching/indexing?

Memory-based balanced trees, such as AVL trees, typically require frequent re-balancing through rotations which are efficient in RAM. However, for disk-based storage, the cost of frequent random access and data modifications (as required by rotations) is very high. Disk-based systems benefit from structures like B-trees which minimize disk reads and writes by maximizing the use of sequential access and reducing the frequency of restructuring operations.

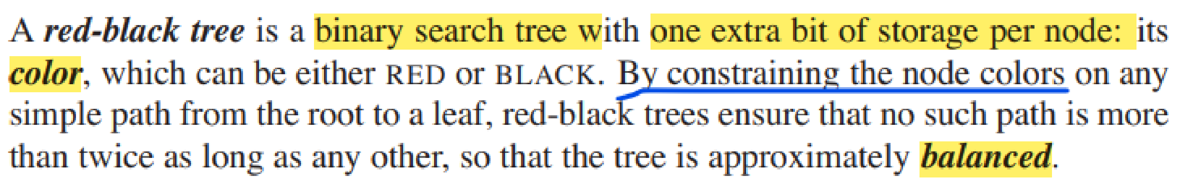
iii. How does a Red-Black Tree maintain the balance in the tree?

Red-Black Trees maintain balance by ensuring that:

DAA28: Red Black Tree | Binary Search Tree Insertion and Deletion | RB-Tree insert delete

DAA29| Red Black Tree Introduction | Red Black Tree Rules| RB Tree Properties

DAA30| Red Black Tree| A Red-Black Tree with n internal nodes has height at most 2log(n + 1)


A ***red-black tree*** is a binary search tree with one extra bit of storage per node: its ***color***, which can be either RED or BLACK. By constraining the node colors on any simple path from the root to a leaf, red-black trees ensure that no such path is more than twice as long as any other, so that the tree is approximately ***balanced***.

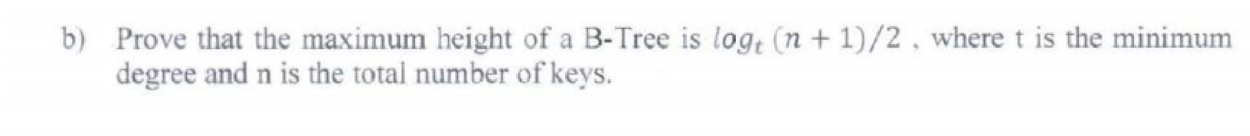
1. Every node is either red or black.
2. The root is always black.
3. Red nodes cannot have red children (no two red nodes can be adjacent).
4. Every path from a node to its descendant NULL nodes has the same number of black nodes. These properties ensure that the longest path from the root to a leaf is no more than twice the length of the shortest path, guaranteeing that the tree remains approximately balanced.

iv. How does a B-Tree maintain the balance in the tree?

B-Trees maintain balance by ensuring that all leaves are at the same depth. This is accomplished by a process of **splitting and merging** nodes:

1. Each node has a minimum and maximum number of children, typically depending on the order of the tree.
2. When a node exceeds the maximum allowable children (upon insertion), it is split into two nodes, and the middle key is pushed up into the parent node.

3. Conversely, if a node falls below the minimum number of children (after deletion), nodes are merged or keys are redistributed from neighboring nodes to maintain the minimum count. This maintains the tree's height and ensures efficient access at all levels.


b) Prove that the maximum height of a B-Tree is $log_t\,(n+1)/2$, where t is the minimum degree and n is the total number of keys.

Ans :

No of key on root = 1

In depth 1 children number= 2,

In depth 2 children number =2t,

In depth3 children number= 2t^2

……

Number of key n>= 1+ (t-1) $\sum_{i=1}^{h} 2t^{(i-1)}$

>= 1+ 2(t-1)(1+t+t^2+....+t^(h-1))

>= 1 +2(t-1)*$\frac{(t^h-1)}{t-1}$

>=1+2$(t^h - 1)$

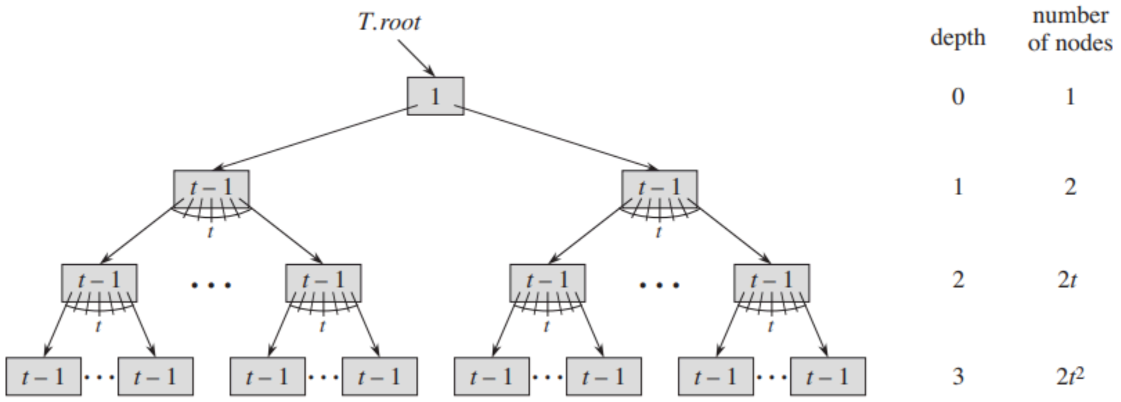
>=1+2t^h-2

>=2$t^h - 1$

So, n+1>=2t^h

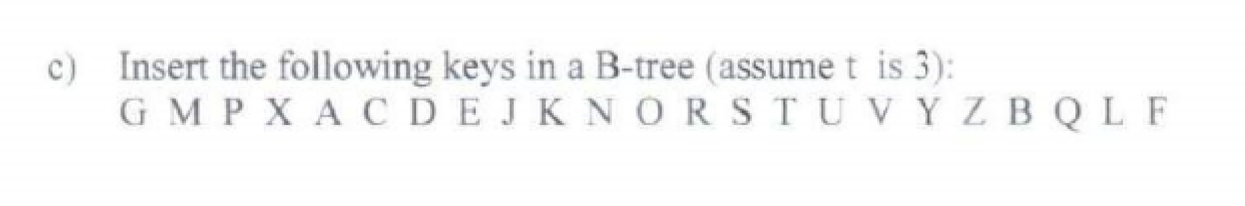
=>$\frac{n+1}{2}$>= t^h

=> h= $log_t \frac{n+1}{2}$



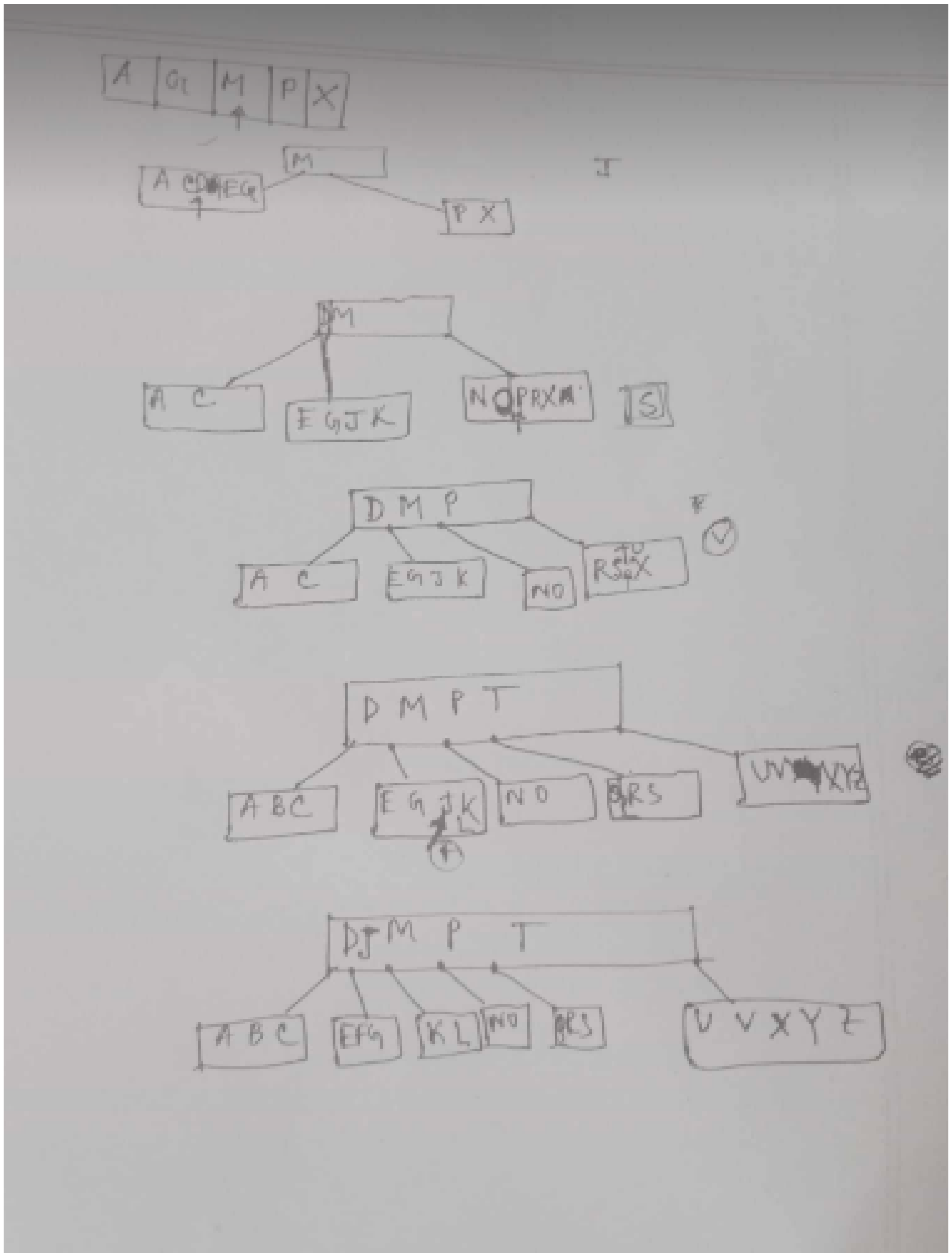
DAA38: B-Tree Definition| B Tree Properties | B Tree Explained In Hindi

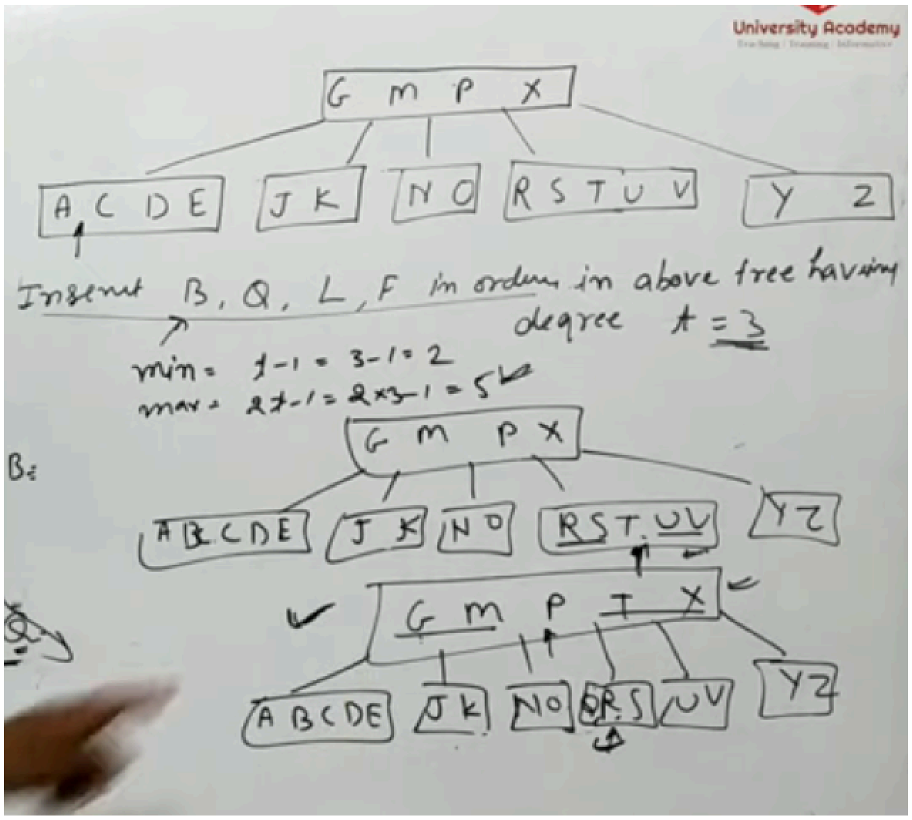
DAA39:B-Tree|If n =1,then for any n-key B-tree T of height h minimum degree t=2,h=…



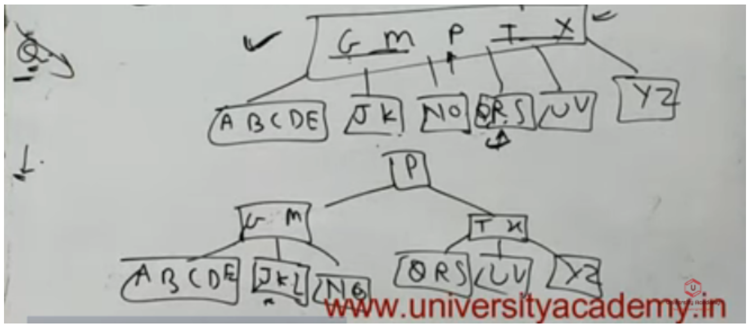
c) Insert the following keys in a B-tree (assume t is 3):
G M P X A C D E J K N O R S T U V Y Z B Q L F

Ans : **t=3 then lower bound = t-1= 3-1=2, upper bound= 2*t-1= 2*3-1=5**


A G M P X
M
A C D E G
P X
J
M
A C
E G J K
N O P R X
S
D M P
A C
E G J K
N O
R S X
D M P T
A B C
E G J K
N O
Q R S
U V X Y Z
D J M P T
A B C
E F G
K L
N O
Q R S
U V X Y Z



- Create empty node
- Insert element
- Do the split of root before when full in parent. Splitting: break on median and insert there in left and right
- If full key then send the median to its parent

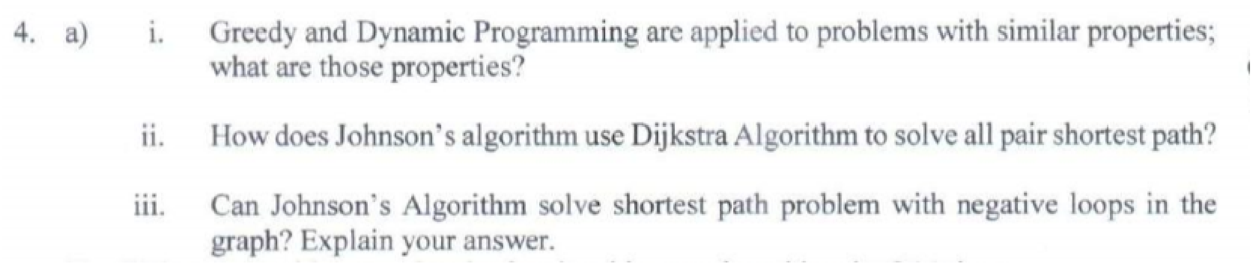


DAA41: Creating an empty B-tree| Splitting a node in a B-tree|Inserting a key into a B-tree in hindi

DAA42: B Tree insertion Solved example| B Tree Insertion Important Question

DAA43: Delete Operation in B-Tree| B Tree Deletion Algorithm | B Tree Deletion process

DAA44: Delete Operation in B-Tree| B Tree Deletion Solved Examples



4. a) i. Greedy and Dynamic Programming are applied to problems with similar properties; what are those properties?

ii. How does Johnson's algorithm use Dijkstra Algorithm to solve all pair shortest path?

iii. Can Johnson's Algorithm solve shortest path problem with negative loops in the graph? Explain your answer.

i. **Greedy and Dynamic Programming properties**: Both approaches solve problems with optimal substructure and overlapping subproblems, but greedy algorithms make locally optimal choices, while dynamic programming uses a bottom-up approach to build solutions.

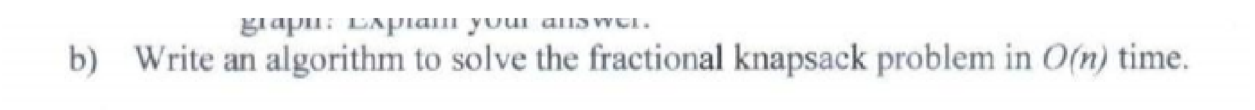
ii. **Johnson's algorithm and Dijkstra's Algorithm**:

Johnson's algorithm reweights the graph using the Bellman-Ford algorithm,after that no neg edges remains.

then uses Dijkstra's algorithm on each vertex **to find the shortest paths efficiently.** By using formula:

$\partial(u, v) = \partial(u, v) + \partial(v) - \partial(u)$

iii. Johnson's Algorithm and negative loops:
Johnson's Algorithm cannot solve the shortest path problem if there are negative weight cycles in the graph, as it relies on the assumption that reweighted edges are non-negative. It does allow edges to have negative weights, but there can be **no negative weight cycles** (because then no shortest path would exist for vertices reachable by that cycle).



graph? Explain your answer.

b) Write an algorithm to solve the fractional knapsack problem in *O(n)* time.

Sort Items:

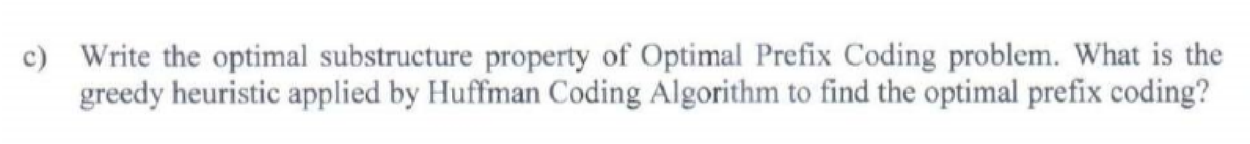
- Sort items in descending order based on their value-to-weight rati**o**
- Set `total_value` to 0.
- Set `remaining_capacity` to WWW.

Greedy Selection:

- For each item in the sorted list:
    - If the item's weight wiw_iwi is less than or equal to `remaining_capacity`:
        - Add the entire item to the knapsack:
            - Increase `total_value` by viv_ivi.
            - Decrease `remaining_capacity` by wiw_iwi.
    - Else:
        - Add a fraction of the item to fill the knapsack:
            - Increase `total_value` by v_i \times \frac{\text{remaining_capacity}}{w_i}.
            - Set `remaining_capacity` to 0.
        - Break the loop as the knapsack is full.

Output:

- Return `total_value`.



c) Write the optimal substructure property of Optimal Prefix Coding problem. What is the greedy heuristic applied by Huffman Coding Algorithm to find the optimal prefix coding?

Optimal Substructure Property of Optimal Prefix Coding

The optimal substructure property of the Optimal Prefix Coding problem states that an optimal code for a given set of characters can be constructed by combining optimal codes for subsets of those characters. Specifically, if you merge the two least frequent characters and treat them as a single character, then an optimal code for the original set can be obtained by solving the subproblem for this reduced set and then expanding the merged character back into the original two characters.

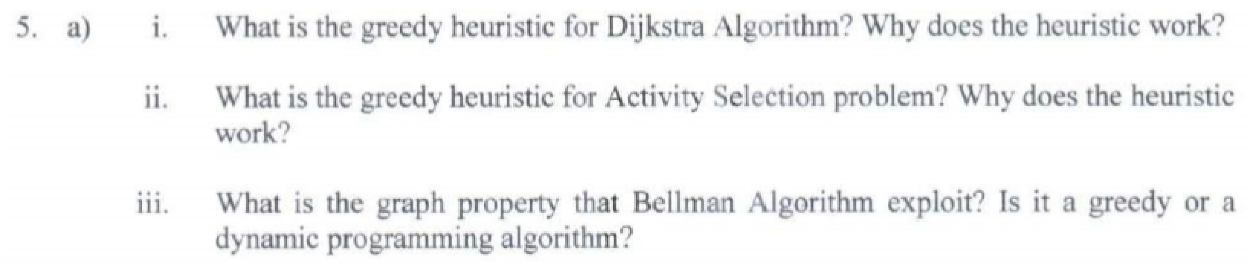
Greedy Heuristic Applied by Huffman Coding Algorithm

The greedy heuristic used by the Huffman Coding Algorithm is to repeatedly merge the two characters with the smallest frequencies. This approach ensures that the most frequently occurring characters have the shortest codes, leading to a minimal overall encoding length.

Steps:

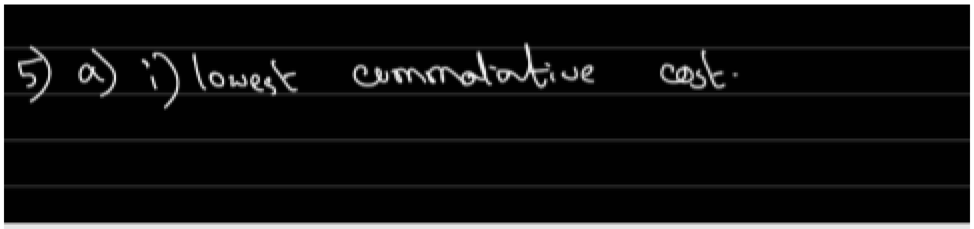
1. **Build a min-heap** of all characters based on their frequencies.
2. **Extract the two nodes** with the smallest frequencies.
3. **Create a new internal node** with these two nodes as children and a frequency equal to the sum of their frequencies.

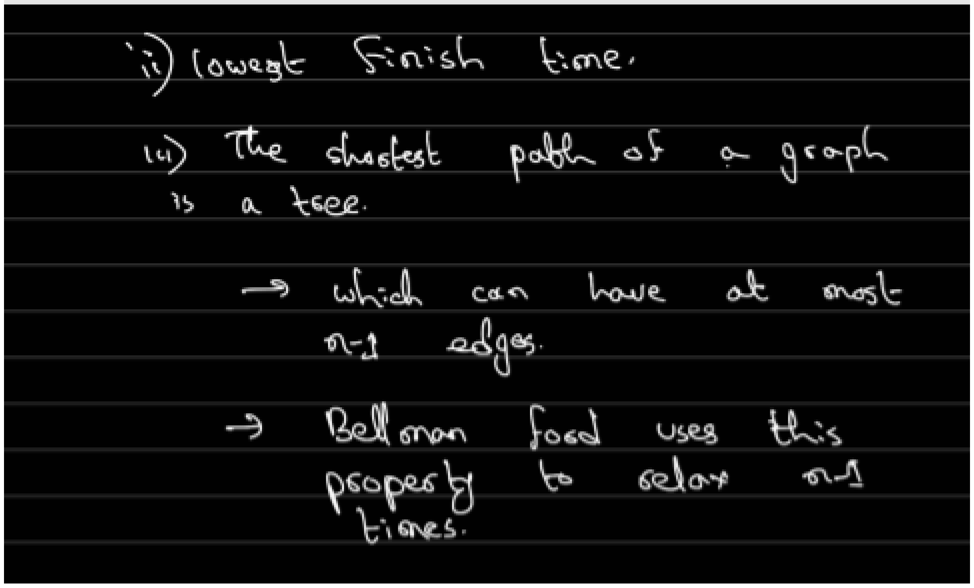
4. **Insert the new node** back into the min-heap.
5. **Repeat** until there is only one node left in the heap. This node becomes the root of the Huffman tree.


5. a) i. What is the greedy heuristic for Dijkstra Algorithm? Why does the heuristic work?

ii. What is the greedy heuristic for Activity Selection problem? Why does the heuristic work?

iii. What is the graph property that Bellman Algorithm exploit? Is it a greedy or a dynamic programming algorithm?


5) a) i) lowest commulative cost.


ii) lowest finish time.

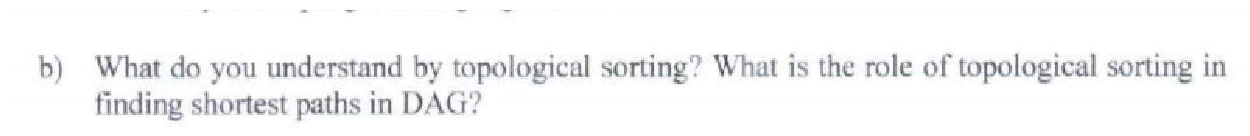
iii) The shortest path of a graph is a tree.

→ which can have at most n-1 edges.

→ Bellman ford uses this property to relax n-1 times.

The Bellman-Ford algorithm exploits the property of edge relaxation in graphs, which is the process of iteratively updating the shortest path estimates by considering edges of the graph.

It is a dynamic programming algorithm, not a greedy algorithm. This is because it systematically examines all possible ways to reach each vertex, updating the shortest paths through multiple iterations, rather than making a series of locally optimal choices.

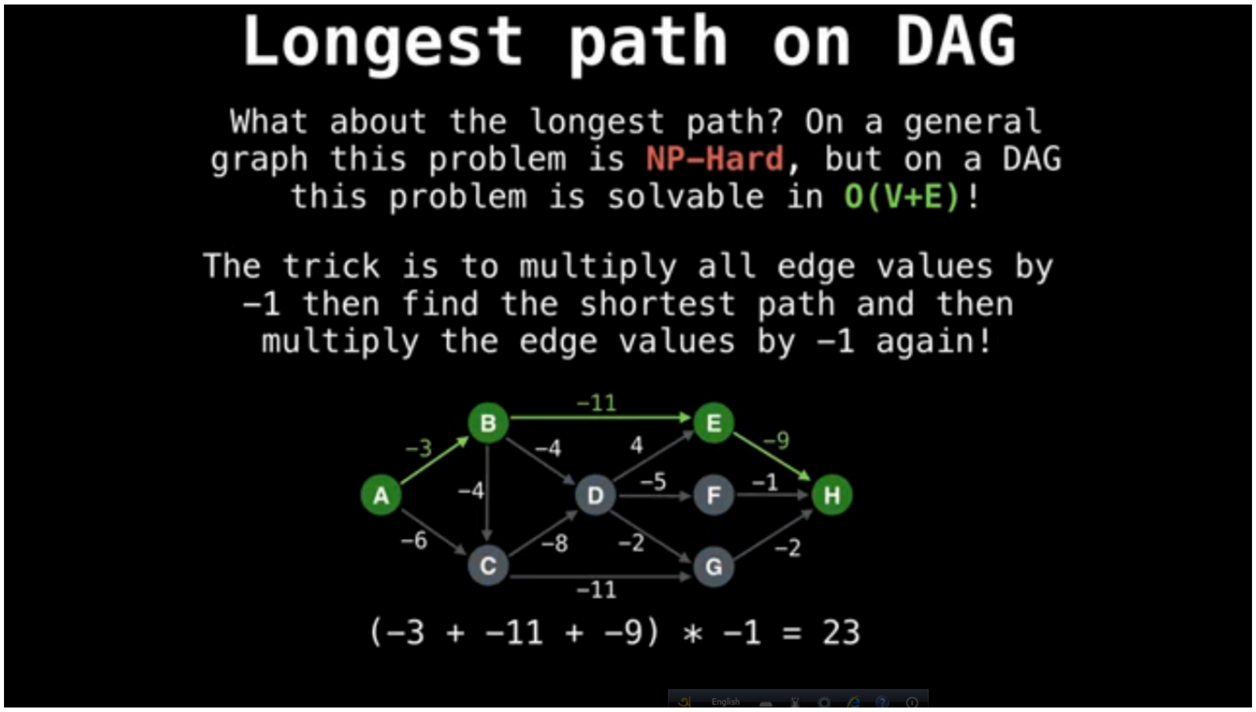


b) What do you understand by topological sorting? What is the role of topological sorting in finding shortest paths in DAG?

Topological sorting for Directed Acyclic Graph (DAG) is a linear ordering of vertices such that for every directed edge u-v, vertex u comes before v in the ordering.

Shortest Path in Directed Acyclic Graph (DAG) Explained With Solved Example in Hindi

Shortest/Longest path on a Directed Acyclic Graph (DAG) | Graph Theory



Longest path on DAG

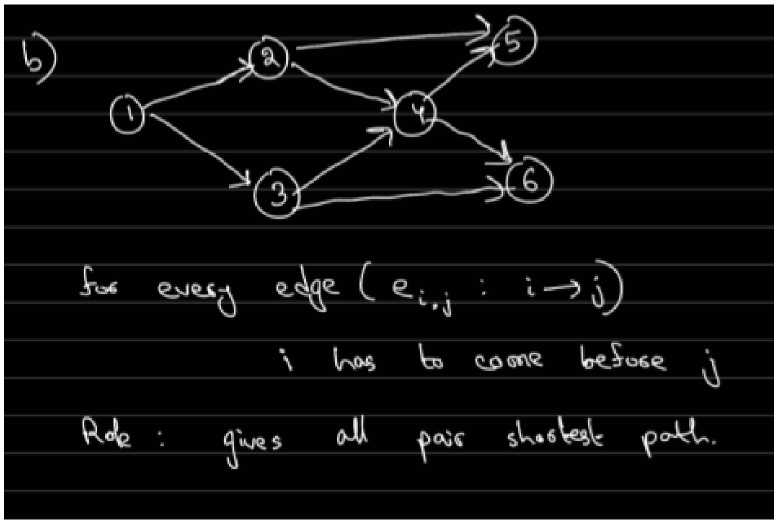
What about the longest path? On a general graph this problem is NP-Hard, but on a DAG this problem is solvable in O(V+E)!

The trick is to multiply all edge values by -1 then find the shortest path and then multiply the edge values by -1 again!

(-3 + -11 + -9) * -1 = 23

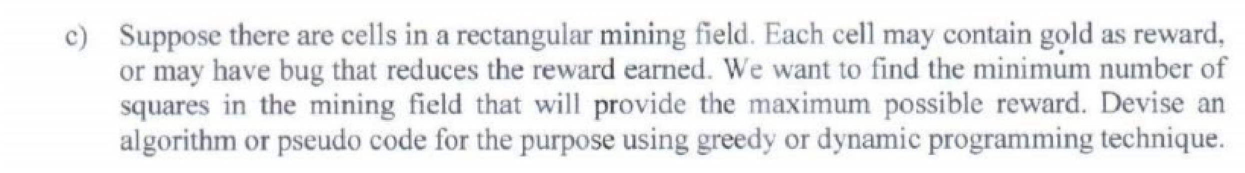
- Find indegree
- If indegree 0 found then delete that node and edes and print. Do same for all the nodes.
- Then you will get the sorted list.

- Topological Order/ Sort in DAG(Direct Acyclic Graph)



for every edge ($e_{i,j}$: $i \to j$)
i has to come before j

Role : gives all pair shortest path.



c) Suppose there are cells in a rectangular mining field. Each cell may contain gold as reward, or may have bug that reduces the reward earned. We want to find the minimum number of squares in the mining field that will provide the maximum possible reward. Devise an algorithm or pseudo code for the purpose using greedy or dynamic programming technique.

Define the Problem:

- Let `grid[i][j]` represent the reward in the cell at row `i` and column `j`.
- Define `dp[i][j]` as the maximum reward that can be collected up to cell `(i, j)`.

Initialize:

- Set `dp[0][0]` to `grid[0][0]`.

State Transition:

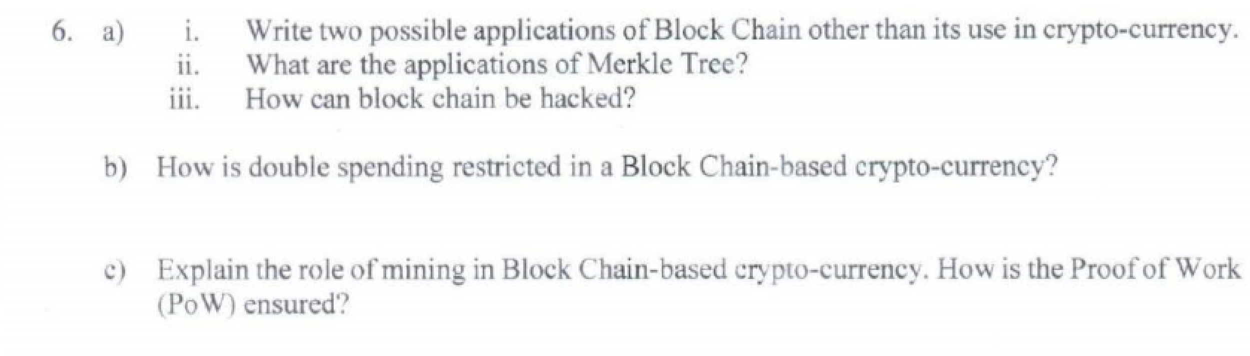
- For each cell `(i, j)`, compute the maximum reward as:
    - If coming from the left: `dp[i][j] = dp[i][j-1] + grid[i][j]`
    - If coming from above: `dp[i][j] = dp[i-1][j] + grid[i][j]`
- Thus, `dp[i][j] = max(dp[i-1][j], dp[i][j-1]) + grid[i][j]`.

Compute the DP Table:

- For each cell `(i, j)`, update `dp[i][j]` using the state transition formula.

Output:

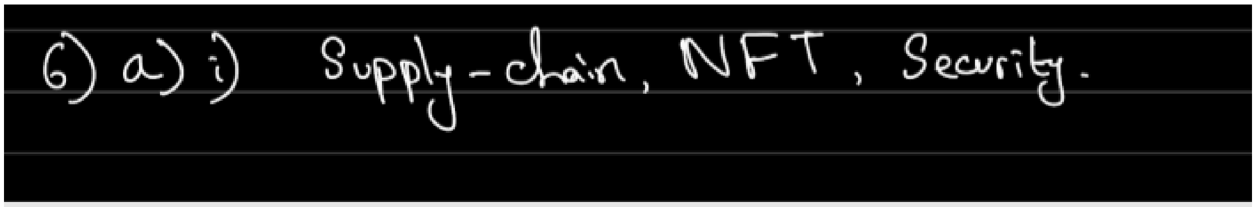
- The maximum reward is found at `dp[m-1][n-1]` for a grid of size `m x n`.



6. a) i. Write two possible applications of Block Chain other than its use in crypto-currency.
ii. What are the applications of Merkle Tree?
iii. How can block chain be hacked?

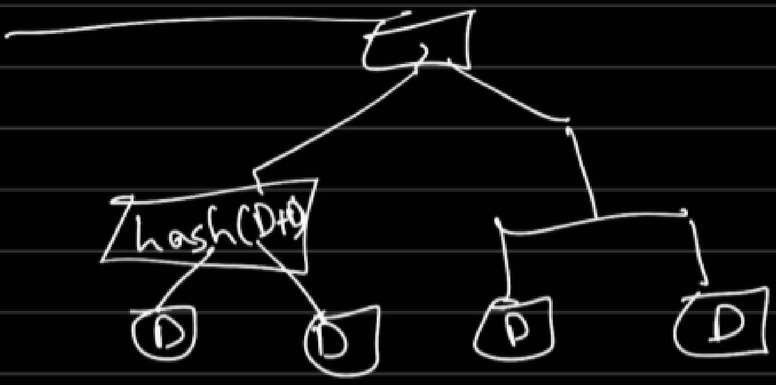
b) How is double spending restricted in a Block Chain-based crypto-currency?

c) Explain the role of mining in Block Chain-based crypto-currency. How is the Proof of Work (PoW) ensured?

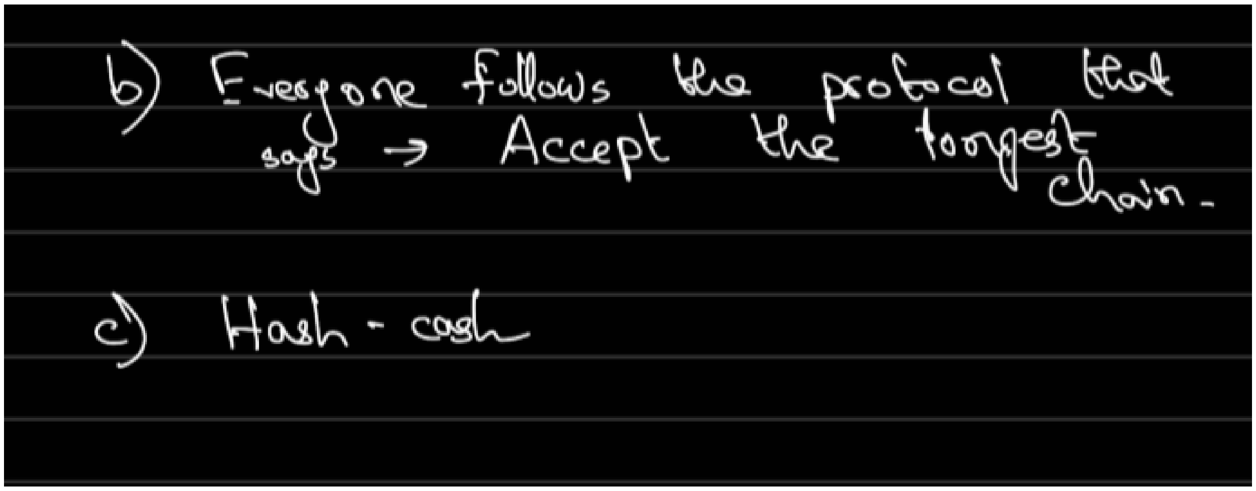


6) a) i) Supply-chain, NFT, Security.

ii) Merkel tree

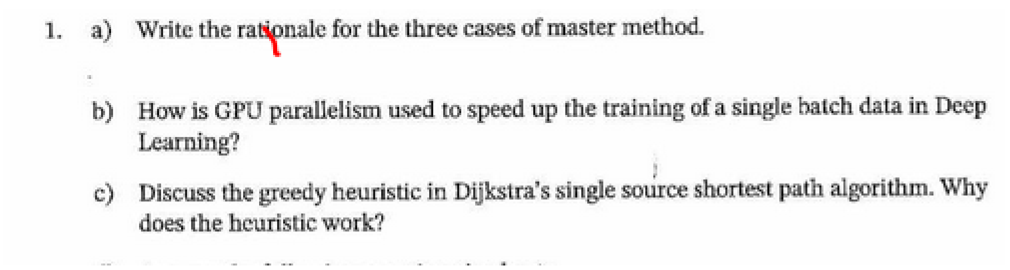


iii)Blockchain can be hacked through vulnerabilities in its implementation, such as exploiting weak smart contract code, gaining control over the majority of the network's computing power (51% attack), or compromising the cryptographic keys.


b) Everyone follows the protocol that says → Accept the longest chain.

c) Hash-cash

2021-2022


1. a) Write the rationale for the three cases of master method.

   b) How is GPU parallelism used to speed up the training of a single batch data in Deep Learning?

   c) Discuss the greedy heuristic in Dijkstra's single source shortest path algorithm. Why does the heuristic work?

1. a) mid

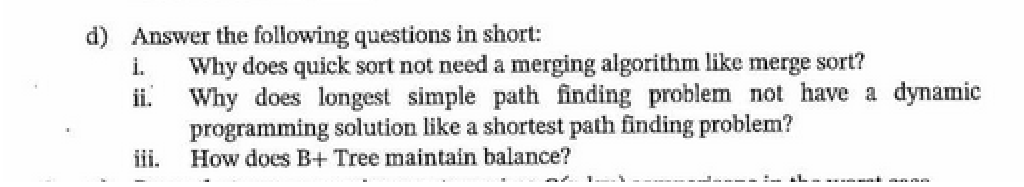
Question 1(b):

How is GPU parallelism used to speed up the training of a single batch data in Deep Learning?
GPU parallelism accelerates deep learning training by leveraging the massive number of cores in GPUs to perform matrix and vector operations simultaneously. Key points include:
Parallel Processing: GPUs can handle thousands of threads in parallel, making them ideal for the highly parallel nature of neural network computations.
Batch Operations: During training, operations like matrix multiplications, which are common in forward and backward propagation, are performed on batches of data in parallel.
High Throughput: GPUs have high memory bandwidth and throughput, which enables faster data transfer and processing, reducing the training time significantly.

Question 1(c):

Discuss the greedy heuristic in Dijkstra's single source shortest path algorithm. Why does the heuristic work?

Dijkstra's algorithm uses a greedy approach to find the shortest path from a single source node to all other nodes in a graph with non-negative edge weights. The steps include:
Initialization: Set the distance to the source node as 0 and to all other nodes as infinity.
Greedy Selection: Repeatedly select the unvisited node with the smallest known distance.
Relaxation: Update the distances to the neighboring nodes of the selected node.
The heuristic works because each step ensures that the shortest path to the current node is finalized before moving to the next node. Since edge weights are non-negative, once a node's shortest path is determined, it cannot be improved by any subsequent paths.


d) Answer the following questions in short:
i. Why does quick sort not need a merging algorithm like merge sort?
ii. Why does longest simple path finding problem not have a dynamic programming solution like a shortest path finding problem?
iii. How does B+ Tree maintain balance?

Question 1(d):

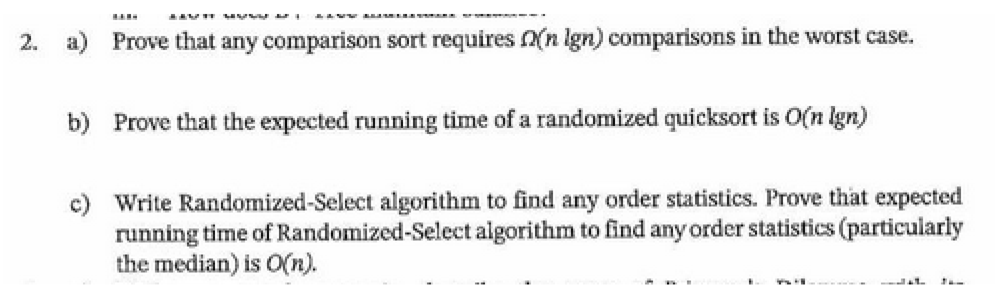
i. **Why does quick sort not need a merging algorithm like merge sort?**

- Quick sort is an in-place sorting algorithm, meaning it sorts the elements within the same array and uses partitioning to divide the array into smaller sub-arrays. Once partitioning is done, the elements are already in place, so no merging is required.

ii. Why does longest simple path finding problem not have a dynamic programming solution like a shortest path finding problem?

- The longest simple path problem is **NP-hard** because **it requires checking all possible paths** to ensure the longest one is simple (i.e., no repeated vertices). Dynamic programming is efficient for problems with optimal substructure and overlapping subproblems, which the longest simple path problem lacks due to its combinatorial explosion in possibilities.

Iii. How does B+ Tree maintain balance?
B+ Trees maintain balance through their properties: all leaf nodes are at the same level, and internal nodes are maintained to ensure that the tree remains balanced during insertions and deletions. When nodes exceed their capacity, they are split, and the middle key is promoted to the parent, ensuring logarithmic height and balanced structure.


2. a) Prove that any comparison sort requires $\Omega(n \lg n)$ comparisons in the worst case.

b) Prove that the expected running time of a randomized quicksort is $O(n \lg n)$

c) Write Randomized-Select algorithm to find any order statistics. Prove that expected running time of Randomized-Select algorithm to find any order statistics (particularly the median) is $O(n)$.

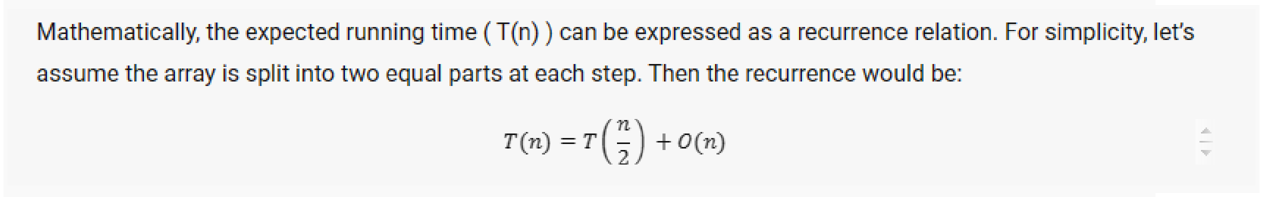
2)a) Any comparison sort algorithm requires Ω(nlogn) comparisons in worst case

2b)repeat

2c)

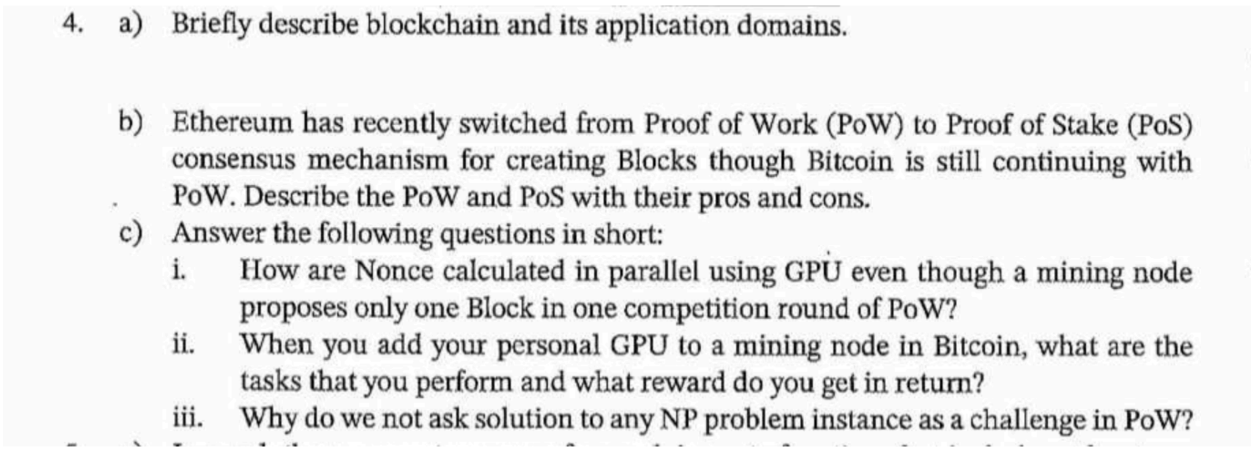
- Choose pivot from a random selection
- Partition into 2 sub arrays: elements less than the pivot and elements greater than the pivot.
- Determine the order of the pivot in sorted array
- If k=i => ith order statistics
- If (i < k), recursively apply the algorithm to the left subarray.
- If (i > k), recursively apply the algorithm to the right subarray, adjusting (i) to (i - k).

The expected running time of the Randomized Select algorithm is (O(n))



Mathematically, the expected running time (T(n)) can be expressed as a recurrence relation. For simplicity, let's assume the array is split into two equal parts at each step. Then the recurrence would be:

$$T(n) = T\left(\frac{n}{2}\right) + O(n)$$



4. a) Briefly describe blockchain and its application domains.

   b) Ethereum has recently switched from Proof of Work (PoW) to Proof of Stake (PoS) consensus mechanism for creating Blocks though Bitcoin is still continuing with PoW. Describe the PoW and PoS with their pros and cons.

   c) Answer the following questions in short:

      i. How are Nonce calculated in parallel using GPU even though a mining node proposes only one Block in one competition round of PoW?

      ii. When you add your personal GPU to a mining node in Bitcoin, what are the tasks that you perform and what reward do you get in return?

      iii. Why do we not ask solution to any NP problem instance as a challenge in PoW?

Question 4(a):

Briefly describe blockchain and its application domains.

Blockchain:
Blockchain is a distributed ledger technology that allows data to be stored across a network of computers in a way that is secure, transparent, and immutable. Each block contains a list of transactions, and these blocks are linked together in chronological order, forming a chain. Blockchain relies on consensus mechanisms to validate and record transactions, ensuring that all participants in the network agree on the state of the ledger.

Application Domains:

1. **Cryptocurrencies:**
   - The most well-known application, where blockchain is used to support digital currencies like Bitcoin and Ethereum. It provides a secure and decentralized way to conduct financial transactions.
2. **Supply Chain Management:**
   - Blockchain can be used to track the origin and journey of products through the supply chain, ensuring transparency and reducing fraud.
3. **Smart Contracts:**
   - Self-executing contracts with the terms of the agreement directly written into code. They automatically execute and enforce the terms of a contract when predefined conditions are met.
4. **Healthcare:**
   - Blockchain can secure patient records, ensuring privacy and improving the interoperability of health information across different healthcare providers.
5. **Voting Systems:**
   - Provides a transparent and tamper-proof way to conduct elections, ensuring that votes are securely recorded and counted.
6. **Identity Verification:**
   - Enables secure and verifiable digital identities, reducing fraud and improving access to services that require identity verification.

Question 4(b):

Ethereum has recently switched from Proof of Work (PoW) to Proof of Stake (PoS) consensus mechanism for creating blocks though Bitcoin is still continuing with PoW. Describe the PoW and PoS with their pros and cons.

Proof of Work (PoW):

- Description:
    - Miners compete to solve a complex mathematical problem. The first to solve the problem gets to add the next block to the blockchain and is rewarded with cryptocurrency.
- Pros:
    - High level of security due to the computational difficulty of the puzzles.
    - Proven track record of success, as seen with Bitcoin.
- Cons:
    - Extremely energy-intensive, leading to environmental concerns.
    - High barrier to entry due to the need for specialized hardware.
    - Slow transaction speeds and scalability issues.

Proof of Stake (PoS):

- Description:
    - Validators are chosen to create new blocks and validate transactions based on the number of coins they hold and are willing to "stake" as collateral.
- Pros:
    - Much more energy-efficient than PoW.
    - Lower barrier to entry as it doesn't require specialized hardware.
    - Potential for faster transaction speeds and better scalability.
- Cons:
    - May lead to centralization, as those with more coins have more influence.
    - Newer and less tested than PoW, with potential security concerns still being explored.
    - "Nothing at Stake" problem, where validators might validate multiple blockchain forks, complicating consensus.

Question 4(c):

Answer the following questions in short:

i. **How are Nonce calculated in parallel using GPU even though a mining node proposes only one Block in one competition round of PoW?**

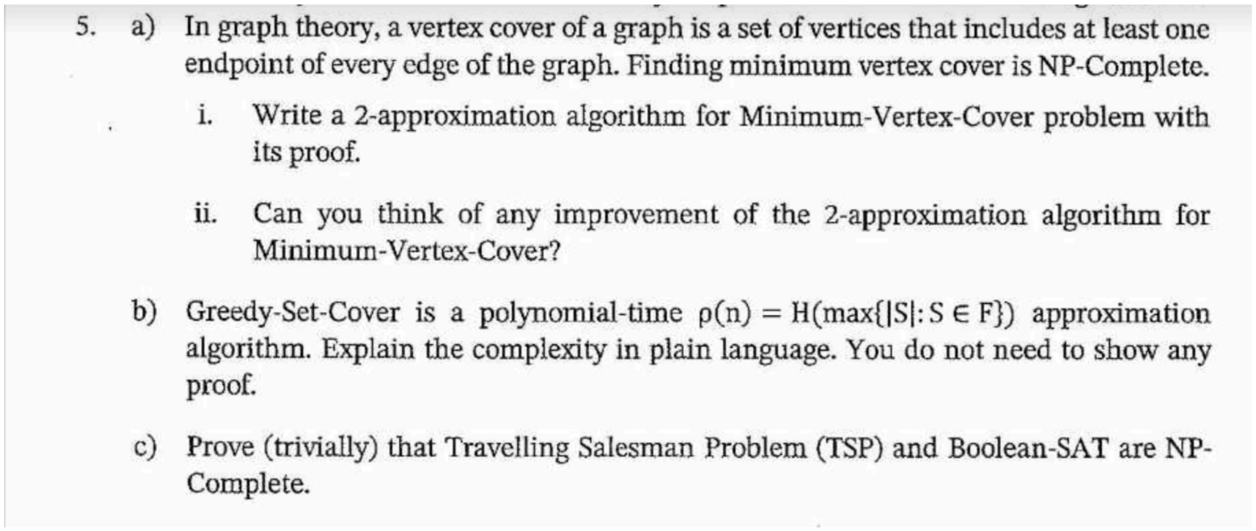
- Parallel Nonce Calculation:
    - GPUs are highly efficient at performing parallel computations. In PoW mining, each thread of the GPU can attempt to find a valid nonce by hashing different values in parallel.
    - Even though only one block is proposed in the end, the parallel approach significantly increases the chances of finding a valid nonce quickly, as many nonces are tried simultaneously.

ii. **When you add your personal GPU to a mining node in Bitcoin, what are the tasks that you perform and what reward do you get in return?**

- Tasks Performed:
    - Your GPU will perform the hashing calculations required for mining. It will repeatedly hash block header data, including a nonce, and check if the hash meets the difficulty target.
    - The goal is to find a nonce that, when hashed with the block data, produces a hash below the network's difficulty target.
- Reward:
    - If your GPU finds a valid nonce, the mining node you are connected to may propose the block to the network. If the block is accepted, you receive a reward in the form of newly minted bitcoins and transaction fees from the transactions included in the block.

iii. **Why do we not ask solution to any NP problem instance as a challenge in PoW?**

- Reason:
    - NP problems are computationally intensive and verifying the solution may also require significant resources, potentially making it impractical for real-time consensus mechanisms.
    - The PoW puzzle needs to be difficult to solve but easy to verify, ensuring that once a solution is found, it can be quickly validated by other nodes. NP problems do not typically meet this criterion as their solutions can be hard to verify efficiently.



5. a) In graph theory, a vertex cover of a graph is a set of vertices that includes at least one endpoint of every edge of the graph. Finding minimum vertex cover is NP-Complete.

   i. Write a 2-approximation algorithm for Minimum-Vertex-Cover problem with its proof.

   ii. Can you think of any improvement of the 2-approximation algorithm for Minimum-Vertex-Cover?

   b) Greedy-Set-Cover is a polynomial-time $\rho(n) = H(\max\{|S| : S \in F\})$ approximation algorithm. Explain the complexity in plain language. You do not need to show any proof.

   c) Prove (trivially) that Travelling Salesman Problem (TSP) and Boolean-SAT are NP-Complete.

5 a) i) ??????????????????????

ii. Can you think of any improvement of the 2-approximation algorithm for Minimum-Vertex-Cover?

- The 2-approximation algorithm is among the best known approximation algorithms for the Minimum-Vertex-Cover problem in terms of the approximation ratio.
- Improvements in specific instances or using different heuristics might yield better results in practice, but no polynomial-time algorithm with a better guaranteed approximation ratio for the general case is currently known.

Question 5(b):

Greedy-Set-Cover is a polynomial-time $\rho(n)=H(\max|S|:S\in F)\rho(n)=H(\max|S|:S\in F)$ approximation algorithm. Explain the complexity in plain language. You do not need to show any proof.

- Greedy-Set-Cover Algorithm:
  - This algorithm is used to cover all elements of a universe with the minimum number of sets from a given collection.
  - The greedy strategy selects the set that covers the largest number of uncovered elements in each step until all elements are covered.
- Complexity Explanation:
  - The algorithm runs in polynomial time, meaning that the number of steps required grows at most as a polynomial function of the input size.
  - The approximation ratio $\rho(n)$ indicates how close the solution is to the optimal solution. $H(n)$ is the nth Harmonic number, which is approximately $\log n$.

    $\rho(n)$

    $H(n)$

    n

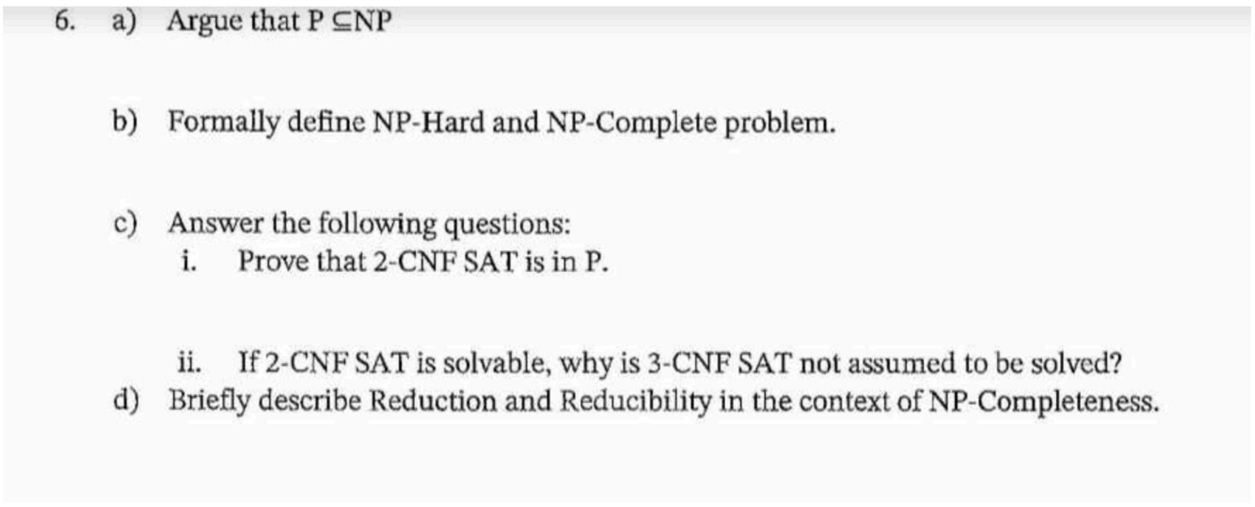
    $\log n$
  - Therefore, the algorithm guarantees a solution within a factor of $\log n$ of the optimal solution.

    $\log n$

Question 5(c):

Prove (trivially) that Travelling Salesman Problem (TSP) and Boolean-SAT are NP-Complete.

- **Travelling Salesman Problem (TSP):**
  - **Definition:** Given a set of cities and distances between each pair of cities, the problem asks if there exists a tour that visits each city exactly once and returns to the starting city with a total distance less than or equal to a given value.
  - **NP-Completeness:**
    - TSP is in NP because given a tour, we can verify in polynomial time if the tour's length is within the given limit.
    - TSP is NP-hard because any instance of the Hamiltonian Cycle problem (which is NP-complete) can be reduced to TSP.
- **Boolean-SAT (Satisfiability Problem):**
  - **Definition:** Given a Boolean formula, the problem asks if there exists an assignment of truth values to variables that makes the formula true.
  - **NP-Completeness:**
    - SAT is in NP because given an assignment, we can verify in polynomial time if the formula evaluates to true.
    - SAT is NP-complete because every problem in NP can be reduced to SAT, as shown by the Cook-Levin theorem.



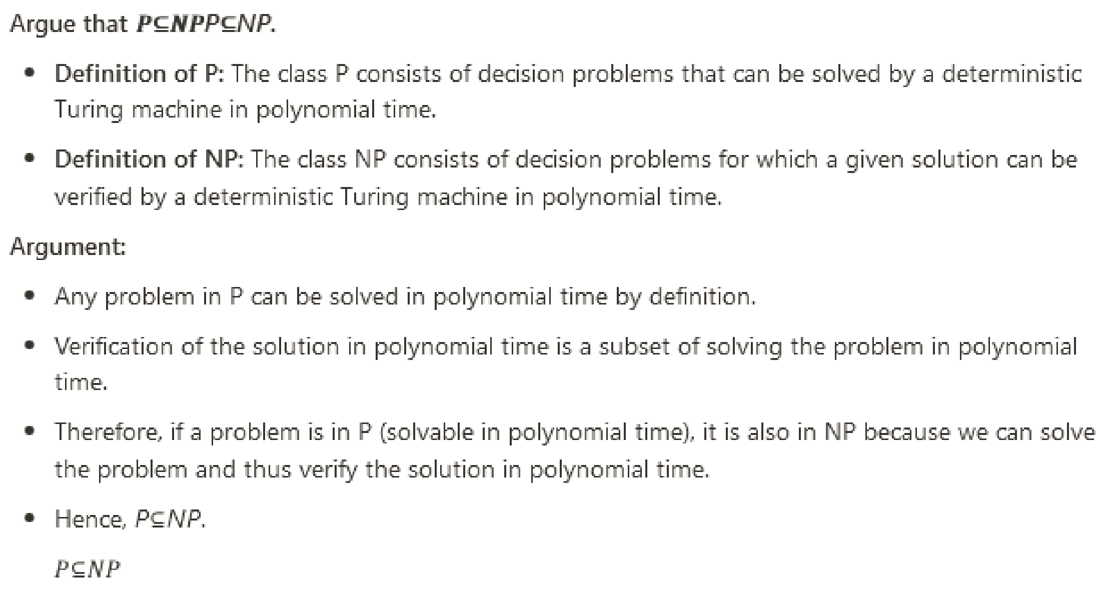
6. a) Argue that P ⊆NP

b) Formally define NP-Hard and NP-Complete problem.

c) Answer the following questions:
   i. Prove that 2-CNF SAT is in P.

   ii. If 2-CNF SAT is solvable, why is 3-CNF SAT not assumed to be solved?

d) Briefly describe Reduction and Reducibility in the context of NP-Completeness.



Argue that ***P⊆NP****P⊆NP*.

- **Definition of P:** The class P consists of decision problems that can be solved by a deterministic Turing machine in polynomial time.
- **Definition of NP:** The class NP consists of decision problems for which a given solution can be verified by a deterministic Turing machine in polynomial time.

Argument:

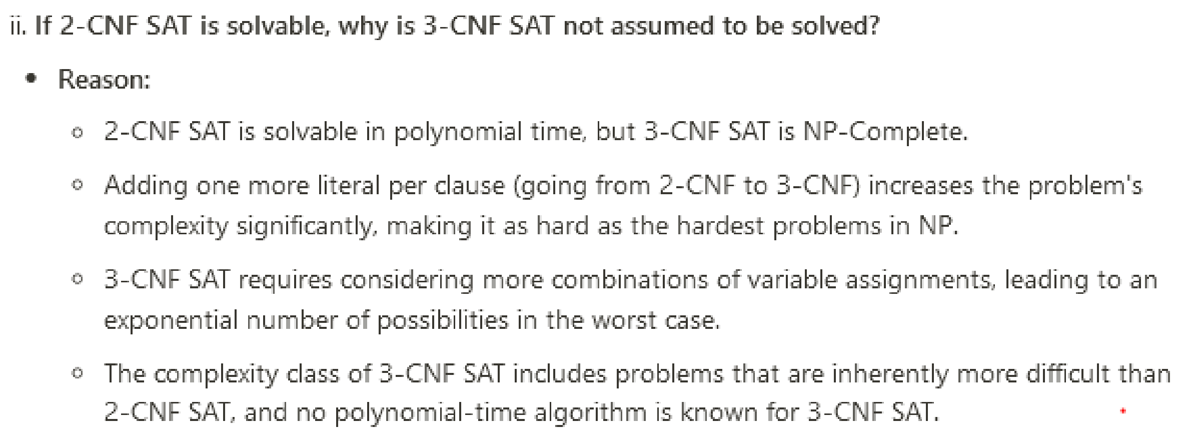
- Any problem in P can be solved in polynomial time by definition.
- Verification of the solution in polynomial time is a subset of solving the problem in polynomial time.
- Therefore, if a problem is in P (solvable in polynomial time), it is also in NP because we can solve the problem and thus verify the solution in polynomial time.
- Hence, *P⊆NP*.

  P⊆NP

6b)NP-Hard problems are at least as hard as the hardest problems in NP, but they do not necessarily have to be in NP themselves.
NP-Complete problems are the hardest problems in NP, and if any NP-Complete problem can be solved in polynomial time, then every problem in NP can be solved in polynomial time (i.e., *P*=*NP*).

6c) i) repeat

6c}ii)



ii. **If 2-CNF SAT is solvable, why is 3-CNF SAT not assumed to be solved?**

- Reason:
  - 2-CNF SAT is solvable in polynomial time, but 3-CNF SAT is NP-Complete.
  - Adding one more literal per clause (going from 2-CNF to 3-CNF) increases the problem's complexity significantly, making it as hard as the hardest problems in NP.
  - 3-CNF SAT requires considering more combinations of variable assignments, leading to an exponential number of possibilities in the worst case.
  - The complexity class of 3-CNF SAT includes problems that are inherently more difficult than 2-CNF SAT, and no polynomial-time algorithm is known for 3-CNF SAT.

6d)Reduction is a way of **converting one problem into another problem** in such a way that a solution to the second problem can be used to solve the first problem.Reduction is a way of converting one problem into another problem in such a way that a solution to the second problem can be used to solve the first problem.

Reducibility is a concept used to **compare the complexity of different problems.** If a problem *A* can be reduced to another problem *B* in polynomial time, it means that *B* is at least as hard as *A*.his concept is crucial in proving NP-Completeness. If we can show that any known NP-Complete problem can be reduced to a new problem *C* in polynomial time, and *C* is in NP, then *C* is also NP-Complete.his concept is crucial in proving NP-Completeness. If we can show that any known NP-Complete problem can be reduced to a new problem *C* in polynomial time, and *C* is in NP, then *C* is also NP-Complete.

Reductions are fundamental tools in computational complexity theory. They allow us to transfer the hardness of one problem to another, thereby showing that solving the new problem is at least as hard as solving the original problem.Polynomial-time reductions are used to demonstrate that if a polynomial-time algorithm exists for one NP-Complete problem, it exists for all NP problems, leading to the conclusion

NP completeness videos

NP, NP-Hard, NP-Complete, Circuit Satisfiability Problem - Algorithm - Online Class - JnU - (Bangla)
Travelling Salesman, Clique, 3CNF, Formula Satisfiability- Algorithm - Online Class - JnU - (Bangla)
Approximation Algorithm (Vertex Cover, TSP) - Algorithm - Online Class - JnU - (Bangla)