1. **Describe the concept of heuristic and meta heuristic .give an example of a problem that can be solved using a metaheuristic approach?**

=>

Heuristic:

A heuristic is a problem-solving strategy or a practical approach that uses intuitive judgment or uses information about the problem being solved to quickly find a solution, especially in situations where finding an optimal solution is computationally expensive or time-consuming. Heuristics do not guarantee the optimal solution but often provide good approximate solutions in a reasonable amount of time. They are efficient and easy to implement but might not always lead to the best possible solution.

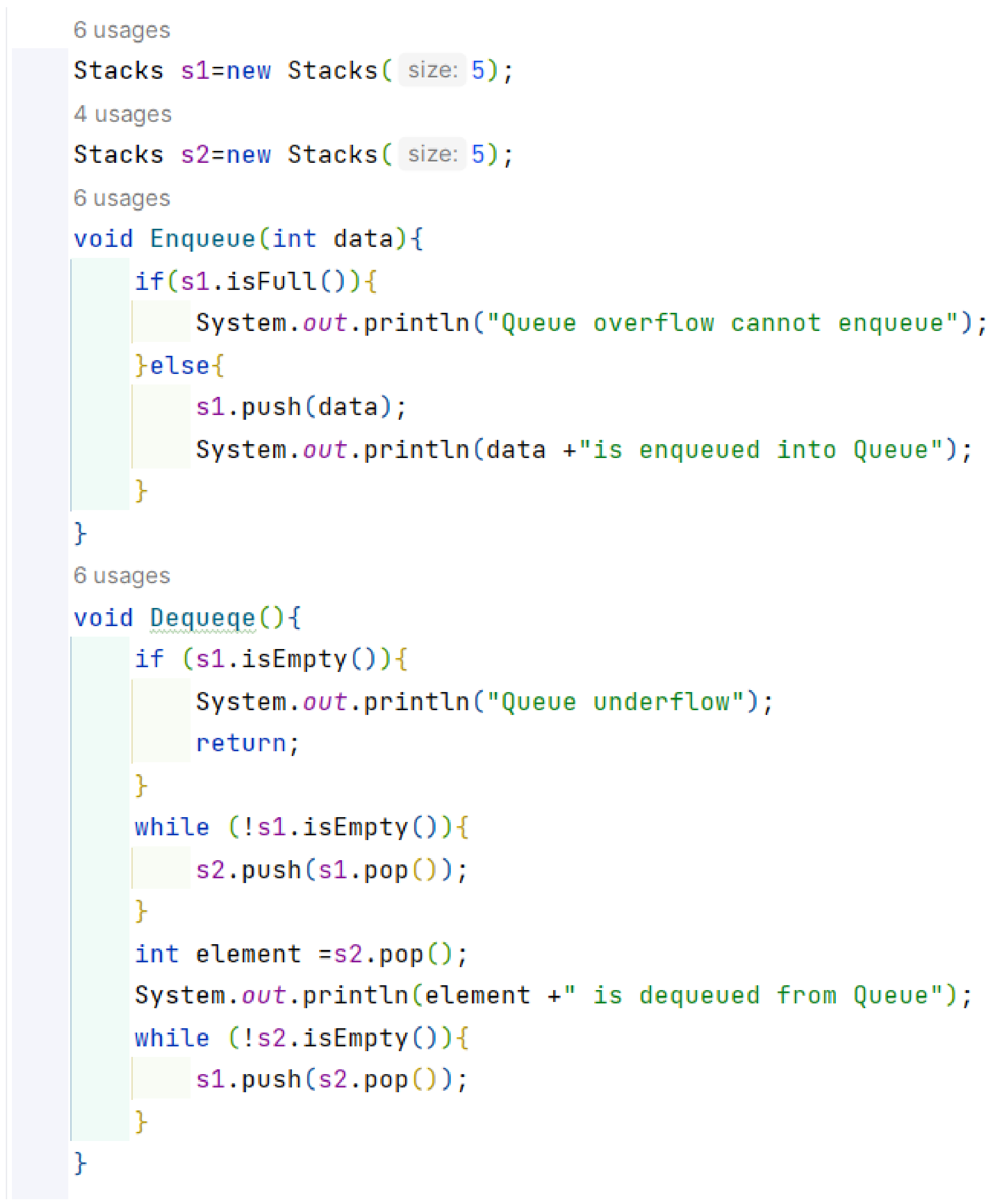
Metaheuristic:

Metaheuristics are high-level problem-solving strategies used to find good solutions for a wide range of optimization problems. They are general techniques that can be applied to different problems without requiring problem-specific adaptations. Metaheuristics are often inspired by natural phenomena or physical processes and are designed to explore the search space efficiently and effectively.

Example: Genetic Algorithm (GA) is a widely used metaheuristic inspired by the process of natural selection. It mimics the evolution process, where a population of potential solutions (individuals) evolves over generations, with fitter individuals being more likely to reproduce and pass on their traits to the next generation

For instance, in the "Knapsack Problem," where a set of items with different weights and values need to be packed into a knapsack of limited capacity to maximize the total value, a Genetic Algorithm can efficiently explore various combinations of items to find a near-optimal solution.

2. **implement queue using stack?**



```
6 usages
Stacks s1=new Stacks( size: 5);
4 usages
Stacks s2=new Stacks( size: 5);
6 usages
void Enqueue(int data){
    if(s1.isFull()){
        System.out.println("Queue overflow cannot enqueue");
    }else{
        s1.push(data);
        System.out.println(data +"is enqueued into Queue");
    }
}
6 usages
void Dequeqe(){
    if (s1.isEmpty()){
        System.out.println("Queue underflow");
        return;
    }
    while (!s1.isEmpty()){
        s2.push(s1.pop());
    }
    int element =s2.pop();
    System.out.println(element +" is dequeued from Queue");
    while (!s2.isEmpty()){
        s1.push(s2.pop());
    }
}
```

3. Explain in brief about the backtracking approach for algorithm design? how it differs with recursion ?explain the n queen problems and its algorithm using backtracking and analyze its time complexity?

=>
Backtracking Approach:

Backtracking is a systematic algorithmic technique used to find solutions to combinatorial problems, especially when the problem space is too large to be explored using a brute-force approach.
It is based on a trial-and-error method where the algorithm makes choices at each step and then backtracks to undo those choices that leads to a dead-end or an invalid solution. The backtracking

approach explores the search space by constructing a partial solution and incrementally extending it until a complete and valid solution is found.

Difference between Backtracking and Recursion:
Backtracking and recursion are related concepts, but they are not the same. Recursion is a general programming technique where a function calls itself directly or indirectly to solve smaller instances of the same problem. On the other hand, backtracking is a specific problem-solving strategy that uses recursion as its underlying mechanism. Backtracking applies recursion to explore possible solutions, and it systematically "backs up" or "undoes" choices that don't lead to a valid solution.

N-Queen Problem:
The N-Queen problem is a classic combinatorial problem in which we need to place N queens on an N×N chessboard in such a way that no two queens threaten each other (i.e., no two queens share the same row, column, or diagonal). The task is to find all possible configurations of placing the N queens on the chessboard.

Algorithm for N-Queen Problem using Backtracking:
1. Start from the first column (column 0) of the chessboard.
2. Place a queen in the first row of the current column.
3. Check if the placement of the queen is safe (i.e., it does not attack any previously placed queens).
4. If the placement is not safe, backtrack and try the next row in the current column.
5. If all rows have been explored, backtrack to the previous column and try the next row in that column.
6. Repeat steps 2 to 5 until all queens are placed on the chessboard, or all possibilities are explored.
7.If all possibilities are explored without finding a solution, return false.

Time Complexity Analysis:
The time complexity of the backtracking algorithm for the N-Queen problem is exponential. In the worst case, the algorithm explores all possible configurations of placing queens on the chessboard, and the number of possibilities grows exponentially with the size of the chessboard (N). The time complexity is usually represented as O(N!), where N is the size of the chessboard.

4. **What are the advantage and disadvantage of linked list over an array?**
=>
Advantages of Linked Lists over Arrays:

Dynamic size
Efficient insertion and deletion
Efficient memory utilization
Ease of reorganization

Disadvantage
Slower Random access
Extra memory overhead

Cache unfriendly
More complex implementation

discuss algorithm for inserting a node at front position of the linked list and deleting its last item in singly linked list?

To insert an element at the front position of a linked list and delete its last element, we can follow the following algorithm:

1. Create a new node with the desired data.
2. Set the next pointer of the new node to the current head of the linked list.
3. Set the head of the linked list to the new node, making it the new first element.
4. Traverse the linked list until the second-to-last node.
5. Set the next pointer of the second-to-last node to null,
6. effectively removing the last element.
7. Optionally, store the value of the deleted
8. element for further use or display.

5. what is heap ?

A heap is a specialized tree-based data structure that satisfies the heap property. The heap property states that for a max heap, the value of each parent node is greater than or equal to the values of its children node, and for a min heap, the value of parent node is less than or equal to the values of its children node.

explain quick sort algorithm with big -oh notation in best case, average case and worst case and trace it to sort the data :8,10,5,12,14,5,7,13.
Ans:
QuickSort is a popular divide-and-conquer sorting algorithm that efficiently sorts an array or list of elements. It works by selecting a pivot element from the array and partitioning the other elements into two sub-arrays based on whether they are less than or greater than the pivot. The sub-arrays are then recursively sorted until the entire array is sorted.

Here's an explanation of the QuickSort algorithm with Big-O notation for its best case, average case, and worst case:

Best Case Time Complexity (Big-O notation): O(n log n)

> The best-case scenario occurs when the pivot chosen at each step happens to be the median element, ideally dividing the array into two equal halves.

Average Case Time Complexity (Big-O notation): O(n log n)

> On average, the pivot selection will result in reasonably balanced partitions.
> Like the best case, the average case also has a time complexity of O(n log n).

Worst Case Time Complexity (Big-O notation): O(n^2)

The worst-case scenario occurs when the pivot selection consistently results in the smallest or largest element in the array.
In such cases, the algorithm will create highly unbalanced partitions with one partition containing n-1 elements and the other empty.

Now, let's trace QuickSort on the given data: 8, 10, 5, 12, 14, 5, 7, 13.

Step 1: Select a pivot (usually the last element): Let's choose 13 as the pivot.

Step 2: Partition the array:

Lesser elements: 8, 10, 5, 12, 5, 7
Pivot: 13
Greater elements: 14
Step 3: Recursively apply QuickSort on the lesser and greater partitions:

Applying QuickSort on the lesser elements [8, 10, 5, 12, 5, 7]:
Step 1: Select a pivot (let's choose 7 as the pivot).
Step 2: Partition the array:
- Lesser elements: 5, 5
- Pivot: 7
- Greater elements: 8, 10, 12

Step 3: Recursively apply QuickSort on the lesser and greater partitions:

Applying QuickSort on the lesser elements [5, 5]:
- Already sorted.

Applying QuickSort on the greater elements [8, 10, 12]:
- Already sorted.

The lesser and greater elements of the original partition are now sorted.

Step 4: Combine the sorted partitions and pivot:
Sorted Lesser elements: 5, 5, 7,8,10,12
Pivot: 13
Sorted Greater elements: 14

The entire array is now sorted: [5, 5, 7, 8, 10, 12,13, 14].

6. **Design multithread merge sort algorithm?** **Done**

7. **how does an asynchronous framework handle resource sharing and synchronization in a multithreaded 5/5 environment?**

=> In a multithreaded environment, asynchronous frameworks handle resource sharing by using a single thread to execute multiple tasks concurrently, rather than creating a separate thread for each task.
Here are some common techniques used by asynchronous frameworks:

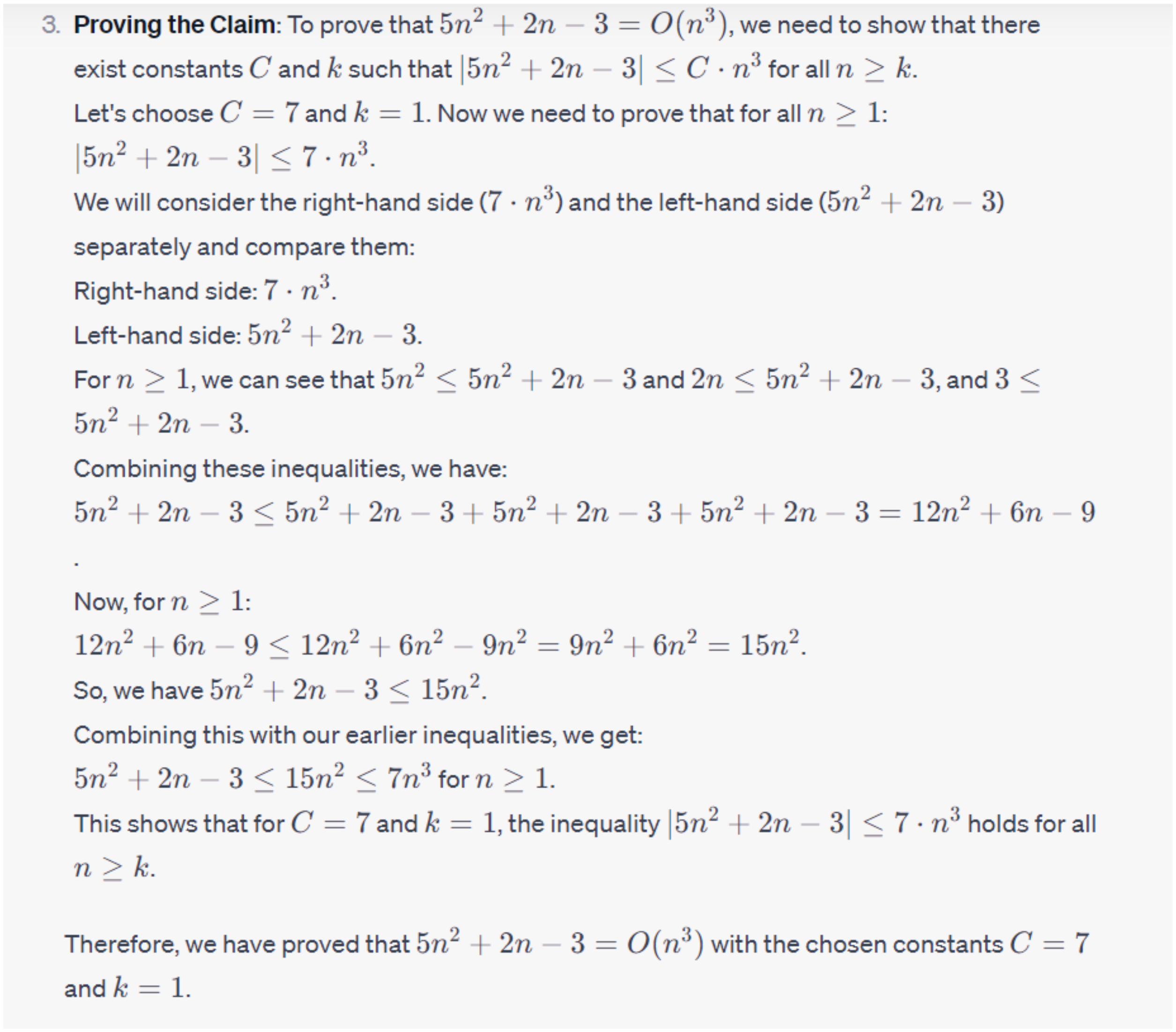
Event loop:
An event loop manages tasks and callbacks, enabling concurrent execution without excessive threading overhead.

Non-blocking I/O:
Non-blocking I/O operations allow threads to perform other tasks while waiting for I/O to complete.

Lock-Free Data Structures:
Lock-free data structures and atomic operations facilitate efficient data sharing and synchronization between threads.

Thread Pool:
A thread pool manages worker threads to avoid frequent thread creation and destruction.

Callback Mechanism:
Callbacks are extensively used, replacing blocking operations to enhance concurrency and responsiveness.



3. **Proving the Claim**: To prove that $5n^2 + 2n - 3 = O(n^3)$, we need to show that there exist constants C and k such that $|5n^2 + 2n - 3| \leq C \cdot n^3$ for all $n \geq k$.
   Let's choose $C = 7$ and $k = 1$. Now we need to prove that for all $n \geq 1$:
   $|5n^2 + 2n - 3| \leq 7 \cdot n^3$.
   We will consider the right-hand side ($7 \cdot n^3$) and the left-hand side ($5n^2 + 2n - 3$) separately and compare them:
   Right-hand side: $7 \cdot n^3$.
   Left-hand side: $5n^2 + 2n - 3$.
   For $n \geq 1$, we can see that $5n^2 \leq 5n^2 + 2n - 3$ and $2n \leq 5n^2 + 2n - 3$, and $3 \leq 5n^2 + 2n - 3$.
   Combining these inequalities, we have:
   $$5n^2 + 2n - 3 \leq 5n^2 + 2n - 3 + 5n^2 + 2n - 3 + 5n^2 + 2n - 3 = 12n^2 + 6n - 9$$
   .
   Now, for $n \geq 1$:
   $12n^2 + 6n - 9 \leq 12n^2 + 6n^2 - 9n^2 = 9n^2 + 6n^2 = 15n^2$.
   So, we have $5n^2 + 2n - 3 \leq 15n^2$.
   Combining this with our earlier inequalities, we get:
   $5n^2 + 2n - 3 \leq 15n^2 \leq 7n^3$ for $n \geq 1$.
   This shows that for $C = 7$ and $k = 1$, the inequality $|5n^2 + 2n - 3| \leq 7 \cdot n^3$ holds for all $n \geq k$.

Therefore, we have proved that $5n^2 + 2n - 3 = O(n^3)$ with the chosen constants $C = 7$ and $k = 1$.

1. **Explain how AVL tree balances of BST?**

Ans-> An AVL tree (Adelson-Velsky and Landis tree) is a self-balancing binary search tree (BST) that maintains a specific property known as the AVL balance condition . The AVL balance condition ensures that the difference of heights of the left and right subtrees of any node comes within {-1,0,1} this domain.

When performing insertion or deletion operations on an AVL tree, the tree may become unbalanced, violating the AVL balance condition.
To maintain AVL balance condition AVL tree uses a technique called rotation. A rotation is a local operation that changes the structure of the tree without affecting the BST property.

There are four possible cases when inserting or deleting a node that can result in an imbalance in the AVL tree:

Left-Left (LL) Case: This case occurs when a node is inserted or deleted in the left subtree of the left child of a node. It causes a left-leaning imbalance. To restore balance, a right rotation is performed.

Left-Right (LR) Case: This case occurs when a node is inserted or deleted in the right subtree of the left child of a node. It causes a double left-leaning imbalance. To restore balance, a left rotation is performed on the left child, followed by a right rotation on the node.

Right-Right (RR) Case: This case occurs when a node is inserted or deleted in the right subtree of the right child of a node. It causes a right-leaning imbalance. To restore balance, a left rotation is performed.

Right-Left (RL) Case: This case occurs when a node is inserted or deleted in the left subtree of the right child of a node. It causes a double right-leaning imbalance. To restore balance, a right rotation is performed on the right child, followed by a left rotation on the node.

When a node is inserted or deleted from an AVL tree, the balance of the tree may be affected. If the balance of a node is violated, the tree will perform one or more rotations to restore the balance.

The following diagram shows an example of how an AVL tree balances itself after a node is inserted.

Before:

```
   A
  / \
 B   C
/ \
D  E
```

After:

```
   A
  / \
 B   C
 / \  \
D  E  F
```
In this example, the node F is inserted into the tree. This causes the balance of the node C to be violated. The tree then performs a right rotation on the node C to restore the balance.
In this way avl tree balances of BST.

The AVL balance condition ensures that AVL trees can be searched, inserted, and deleted efficiently.

4. **Explain Dijkstra Algorithm**

Ans-> Dijkstra's algorithm is a popular algorithms for solving many single-source shortest path problems having non-negative edge weight in the graphs i.e., it is to find the shortest distance between two vertices on a graph.

The algorithm maintains a set of visited vertices and a set of unvisited vertices. It starts at the source vertex and iteratively selects the unvisited vertex with the smallest tentative distance from the source. It then visits the neighbors of this vertex and updates their tentative distances if a shorter path is found. This process continues until the destination vertex is reached, or all reachable vertices have been visited.

A. **discuss an algorithm to insert element at front position of a linkedlist and deleting its last element**

Ans->
To insert an element at the front position of a linked list and delete its last element, we can follow the following algorithm:
1. Create a new node with the desired data.
2. Set the next pointer of the new node to the current head of the linked list.
3. Set the head of the linked list to the new node, making it the new first element.
4. Traverse the linked list until the second-to-last node.
5. Set the next pointer of the second-to-last node to null, e
6. ffectively removing the last element.
7. Optionally, store the value of the deleted element for further use or display.

For implementation heads toward intelijIdea

5. **Explain np hard and np complete .and explain why it is difficult to solve np complete problem.and also provide eg of np hard and np complete problem?**

Ans=> NP-hard and NP-complete are two classes of problems that are related to the complexity of solving them. NP-hard problems are those that are at least as hard as any problem in NP, meaning that any problem in NP can be transformed into an NP-hard problem in polynomial time12. NP-complete problems are those that are both in NP and NP-hard, meaning that they are the hardest problems in NP and that solving any of them in polynomial time would solve all problems in NP12.

NP-hard:
NP-hard stands for Non-deterministic Polynomial-time hard. A problem is classified as NP-hard if it is at least as difficult as the hardest problems in the class NP (Nondeterministic Polynomial-time). NP-hard problems may or may not be in NP themselves, but they have the property that if any NP-hard problem can be solved in polynomial time, then all problems in

NP can be solved in polynomial time as well.

Example of NP-hard problem: The traveling salesman problem (TSP) is a classic example of an NP-hard problem. It asks for the shortest possible route that visits a set of cities and returns to the starting city, without visiting any city twice.

NP-complete:
NP-complete stands for Non-deterministic Polynomial-time complete. A problem is classified as NP-complete if it is both in NP and Np-hard
and every problem in NP can be reduced to it in polynomial time. In other words, an NP-complete problem is one for which a polynomial-time solution would allow all other problems in NP to be solved in polynomial time.

Example of NP-complete problem: The boolean satisfiability problem (SAT) is a classic example of an NP-complete problem.

Why NP-complete problems are difficult to solve:
NP-complete problems are difficult to solve due to their inherent complexity and the lack of known polynomial-time algorithms for them. The difficulty arises from the fact that the number of possible solutions grows exponentially with the size of the problem. Therefore, exhaustive search algorithms that check every possible solution become infeasible for large problem instances.

Aru baaki xa....

6. **Define Heurestic and also define meta heurestic?**

Ans=> A heuristic is a strategy that uses information about the problem being solved to find promising solutions.
A heuristic is a problem-solving technique that relies on practical experience, intuition, or rules of thumb to find a reasonably good solution, especially when an optimal solution is hard to compute or not required. It provides an approximate solution quickly, though not necessarily the best one.

Meta-heuristic: A meta-heuristic is like a super smart problem-solving strategy that can work for lots of different problems. It's good at exploring lots of possibilities to find a pretty good answer, even in situations where finding the perfect answer is really hard. Examples include methods like Genetic Algorithms, Simulated Annealing, and others

Metaheuristic: A higher-level problem-solving methodology that guides the search for solutions using a set of heuristics, offering a more general and effective approach for solving complex optimization problems.

7. **Is hashing better than Binary Search Algorithm?Define any two collision resolution technique any two eg?**

Ans=> Hashing and binary search algorithms are two different algorithms used for different purposes. Hashing is used for searching and inserting elements in a hash table. Binary search is used for searching elements in a sorted array.

searching and inserting elements in a hash table. Binary search is used for searching elements in a sorted array.

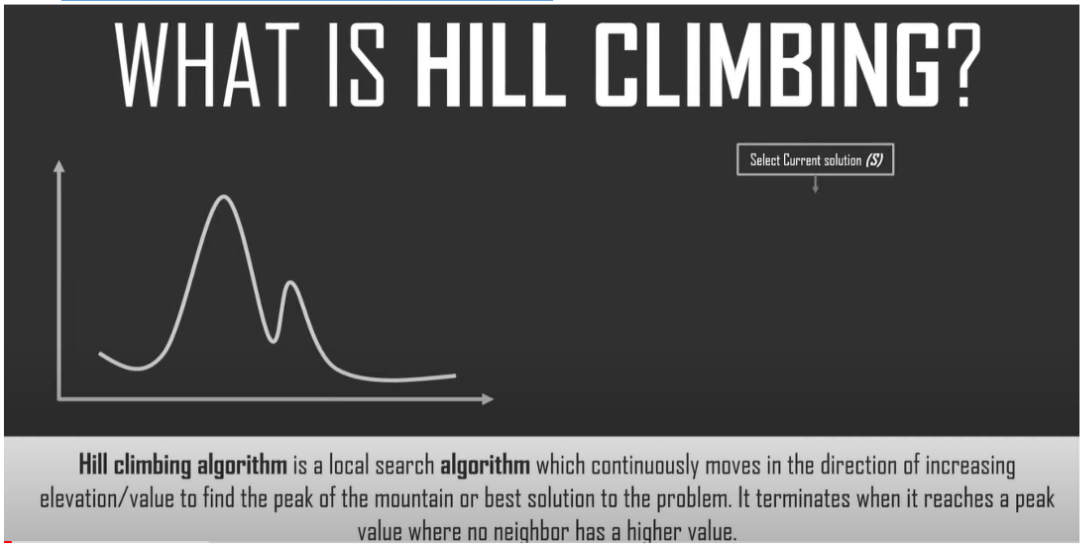
Collision Resolution Technique:[Collision Resolution Techniques - GeeksforGeeks]
Collision resolution techniques are used when two or more keys are mapped to the same index in the hash table. There are several collision resolution techniques such as chaining, linear probing, quadratic probing, double hashing, etc.
Chaining is one of the simplest techniques where each slot of the hash table contains a linked list of elements that hash to that slot. Linear probing is another technique where if a collision occurs at slot i, then the next slot i+1 is checked and so on until an empty slot is found.

8. **Compare Heurestic and meta heurestic .provide example of a problem solve by metaheurestic algorithm?**

Ans=>

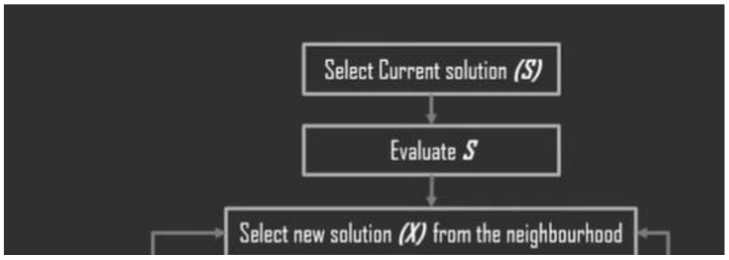
9. **Explain hill climbing algorithm**

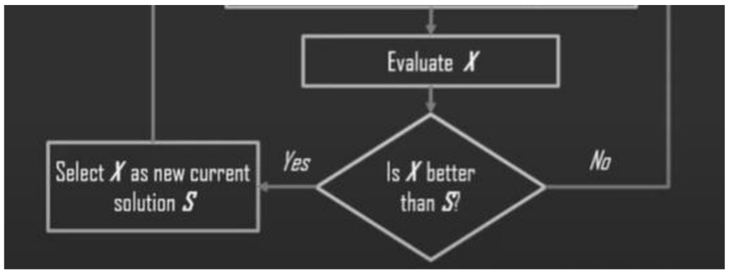
Ans=>hill climbing algorithm_1688569619664.pdf (schoolworkspro.com)



WHAT IS **HILL CLIMBING?**

Hill climbing algorithm is a local search **algorithm** which continuously moves in the direction of increasing elevation/value to find the peak of the mountain or best solution to the problem. It terminates when it reaches a peak value where no neighbor has a higher value.


Select Current solution (S)
Evaluate S
Select new solution (X) from the neighbourhood



1. **Explain ant colony optimization and how it works. What are disadvantage and advantage of it and provide an example of a problem that can be solved by ant colony algorithm?**

Ans=> Ant Colony Optimization (ACO) is a metaheuristic algorithm inspired by the foraging behavior of ants. It is commonly used to solve optimization problems such as the Traveling Salesman Problem (TSP) and the Vehicle Routing Problem (VRP). ACO mimics the behavior of ants, which communicate with each other through pheromone trails to find the shortest path between their nest and food sources.

The algorithm works as follows:

- Initialization: A set of artificial ants is created and placed randomly on the problem space.
- Ants' movement: Each ant chooses the next destination to visit based on a probabilistic decision influenced by the pheromone trails and heuristic information (such as the distance between destinations).
- Pheromone update: After each ant has completed its tour, the pheromone trail intensities are updated. The trails of the ants that found shorter routes are reinforced, while the trails of others gradually evaporate.
- Iteration: Steps 2 and 3 are repeated for a certain number of iterations or until a stopping criteria is met.
- Solution construction: The best solution found by the ants is considered the final result.

Advantages of Ant Colony Optimization:

1. Adaptability: ACO can handle complex optimization problems that may involve large solution spaces, multiple constraints, and changing environments.
2. Robustness: It is less prone to getting stuck in local optima due to the use of pheromone trails and probabilistic decision-making.
3. Distributed computation: The algorithm allows for parallel processing as each ant explores the problem space independently, which can speed up the solution process.

Disadvantages of Ant Colony Optimization:

1. Convergence speed: ACO may require a large number of iterations to converge to an optimal solution, making it slower compared to some other optimization algorithms.

2. Parameter tuning: The performance of ACO can be sensitive to the selection of parameters, such as the evaporation rate and the importance given to pheromone trails versus heuristic information.
3. Memory requirements: Maintaining the pheromone trail information for each path can be memory-intensive, especially for large-scale problems.

One example of a problem that can be solved using the Ant Colony Optimization (ACO) algorithm is the Traveling Salesman Problem (TSP).

The TSP involves finding the shortest possible route that a salesman can take to visit a set of cities and return to the starting city, visiting each city only once. The goal is to minimize the total distance traveled.

1. Initialization: Randomly distribute a set of artificial ants across the cities.
2. Ants' movement: Each ant chooses the next city to visit based on a probabilistic decision influenced by the pheromone trails and heuristic information (e.g., the distance between cities).
3. Pheromone update: After each ant completes its tour, the pheromone trail intensities on the chosen paths are updated. The trails of the ants that found shorter routes are reinforced, while the trails on other paths gradually evaporate.
4. Iteration: Repeat steps 2 and 3 for a certain number of iterations or until a stopping criterion is met.
5. Solution construction: The best solution found (shortest route) by any of the ants is considered the final solution.

By iteratively constructing solutions and updating the pheromone trails, ACO converges towards an optimal or near-optimal solution to the TSP.

11. **explain backtracking startegy how it differn with recursion and explain n queen problem and its algorithm using backtracking and analyze it's time complexity?**

Ans=>

What is Recursion?

The process in which a function calls itself directly or indirectly is called recursion and the corresponding function is called a recursive function.

Properties of Recursion:

- Performing the same operations multiple times with different inputs.
- In every step, we try smaller inputs to make the problem smaller.
- A base condition is needed to stop the recursion otherwise infinite loop will occur.

What is Backtracking?

Backtracking is an algorithmic technique for solving problems recursively by trying to build a solution incrementally, one piece at a time, removing those solutions that fail to satisfy the constraints of the problem at any point in time (by time, here, is referred to the time elapsed till reaching any level of the search tree).

What is the difference between Backtracking and Recursion?

What is the difference between Backtracking and Recursion?

Sl. No.	Recursion	Backtracking
1	Recursion does not always need backtracking	Backtracking always uses recursion to solve problems
2	A recursive function solves a particular problem by calling a copy of itself and solving smaller subproblems of the original problems.	Backtracking at every step eliminates those choices that cannot give us the solution and proceeds to those choices that have the potential of taking us to the solution.
3	Recursion is a part of backtracking itself and it is simpler to write.	Backtracking is comparatively complex to implement.
4	Applications of recursion are Tree and Graph Traversal, Towers of Hanoi, Divide and Conquer Algorithms, Merge Sort, Quick Sort, and Binary Search.	Application of Backtracking is N Queen problem, Rat in a Maze problem, Knight's Tour Problem, Sudoku solver, and Graph coloring problems.

N-Queen Problem:

The N Queen is the problem of placing **N** chess queens on an **N×N** chessboard so that no two queens attack each other.

Algorithm:
Make a recursive function that takes the state of the board and the current row number as its parameter.
Start in the topmost row.
If all queens are placed return true
Try all columns in the current row. Do the following for every tried column.
If the queen can be placed safely in this column
Then mark this [row, column] as part of the solution and recursively check if placing queen here leads to a solution.
If placing the queen in [row, column] leads to a solution then return true.
If placing queen doesn't lead to a solution then unmark this [row, column] and track back and try other columns.
If all columns have been tried and nothing worked return false to trigger backtracking.)

Time complexity O(n!)

12. Define heap . Explain quicksort algo with its big(O) big omega big theta ?(best ,average and worst case?
Ans=>

Heap:

A heap is a specialized tree-based data structure that satisfies the heap property. The heap property states that for a max heap, the value of each node is greater than or equal to the values of its children, and for a min heap, the value of each node is less than or equal to the values of its children. In other words, the root node of a max heap contains the largest element, and the root node of a min heap contains the smallest element. Heaps are typically implemented using arrays and are commonly used to efficiently maintain and retrieve the minimum or maximum element in constant time, which is the top/root of the heap.

Quicksort Algorithm:

Quicksort is a popular sorting algorithm based on the divide-and-conquer strategy. It works by selecting a "pivot" element from the array and partitioning the other elements into two sub-arrays according to whether they are less than or greater than the pivot. The process is then recursively applied to the two sub-arrays until the base case is reached (sub-arrays of size 0 or 1). Quicksort has an average-case time complexity of O(n log n) and performs very well in practice. However, in the worst-case scenario, it can degrade to O(n^2), but this can be mitigated with good pivot selection techniques.

Best Case Time Complexity (Big Omega):
The best-case time complexity occurs when the pivot selected in each partitioning step happens to be the median element of the array. In this scenario, the array is evenly divided into two sub-arrays at each step, resulting in balanced recursion. The best-case time complexity of quicksort is Ω(n log n).

Average Case Time Complexity (Big Theta):
The average-case time complexity occurs when the pivot selection is random or selects a median-of-three (or similar) strategy. In this case, the array is partitioned reasonably well in each step, leading to balanced recursion on average. The average-case time complexity of quicksort is Θ(n log n).

Worst Case Time Complexity (Big O):
The worst-case time complexity occurs when the pivot selected in each partitioning step results in highly unbalanced sub-arrays. For example, if the pivot is always the smallest or largest element of the array, one of the sub-arrays will contain n - 1 elements, and the other sub-array will have 0 elements. This will lead to the worst-case time complexity. In the worst-case scenario, quicksort has a time complexity of O(n^2).

13. how does asynchronous framework handle resource sharing and synchronization in Multithreaded Environment?
Ans=>

In a multithreaded environment, asynchronous frameworks handle resource sharing and synchronization using various techniques to ensure thread safety and efficient resource utilization. Asynchronous frameworks are designed to allow multiple tasks to run concurrently, often using non-blocking I/O operations, to avoid unnecessary waiting and maximize CPU utilization. Some common techniques employed by asynchronous frameworks for resource sharing and synchronization are:
Non-blocking I/O: Asynchronous frameworks use non-blocking I/O operations, avoiding waiting and enabling tasks to continue executing other tasks until resources are ready.

Callbacks and Event-Driven: Asynchronous frameworks rely on callbacks and event-driven programming, allowing tasks to register callbacks that execute when specific events occur.

Task Scheduling: Asynchronous frameworks manage task execution with schedulers, ensuring tasks run when required resources are available.

Thread Pooling: Asynchronous frameworks use pre-allocated threads in a pool, reducing thread creation overhead and improving performance.

Lock-Free Data Structures: Asynchronous frameworks use lock-free data structures for shared data access, avoiding explicit locking and potential issues like deadlocks.

Asynchronous Communication: Asynchronous frameworks facilitate communication between threads without explicit blocking, using message passing or event-based mechanisms.

Atomic Operations: Asynchronous frameworks use atomic operations for thread-safe modifications of shared data, allowing multiple threads to access data concurrently.

Q) What are the advantages and disadvantage of linkedlist over array?

89. list out the advantages of using a linked list?

It is not necessary to specify the number of elements in a linked list during its declaration.
linked list can grow and shrink in size depending upon the insertion and deletion that occurs in the list.
Insertions and deletions at any place in a list can be handled easily and efficiently.
a linked list does not waste any memory space.

Implementation gareko xa Code hernus:

Dynamic programming is a problem-solving strategy that involves breaking down a complex problem into smaller overlapping subproblems and solving them in a top-down or bottom-up manner. It is typically used for optimization problems where the solution can be expressed as the optimal solution to a series of overlapping subproblems.

The main idea behind dynamic programming is to solve each subproblem only once and store its solution for future reference, rather than recomputing it every time it is encountered. This approach reduces the overall time complexity and improves the efficiency of solving the problem.

Strategy: explain garne if possible:

- **Examples:**

Q44. What Are The Advantages Of Linked List Over Array (static Data Structure)?

The disadvantages of array are:

i) unlike linked list it is expensive to insert and delete elements in the array.
ii) One can't double or triple the size of array as it occupies block of memory space.

In linked list

i) each element in list contains a field, called a link or pointer which contains the address of the next element.
ii) Successive element's need not occupy adjacent space in memory.

14. **Explain deadlock in multithreading and also explain how can you avoid deadlock in multithreading?**
=>
Deadlock in multithreading refers to a situation where two or more threads are blocked indefinitely, waiting for each other to release resources they hold. It occurs when threads acquire resources in such a way that a circular dependency is formed, preventing any thread from progressing.

A typical scenario leading to a deadlock involves the following conditions, known as the "four Coffman conditions":

Mutual Exclusion: At least one resource is held in a mutually exclusive mode, meaning only one thread can use it at a time.
Hold and Wait: A thread holds at least one resource and waits for additional resources that are currently held by other threads.
No Preemption: Resources cannot be forcibly taken away from a thread. A thread can only release resources voluntarily.
Circular Wait: There is a circular chain of two or more threads, where each thread is waiting for a resource held by the next thread in the chain.

To avoid deadlocks in multithreading, several techniques and best practices can be employed:

Avoid Circular Dependencies:
Design resource acquisition order in a way that avoids circular dependencies. By establishing a consistent order for acquiring resources, we can prevent the occurrence of circular waits.

Use Resource Ordering:
If you need to acquire multiple resources, ensure that threads always acquire resources in the same order. This prevents potential conflicts and eliminates the possibility of circular waits.

Implement Timeout Mechanisms:
Set timeouts for acquiring resources. If a thread cannot acquire a resource within a specified time, it releases the already acquired resources and retries later. This helps break potential deadlocks by avoiding indefinite waiting.

Resource Preemption:
Consider introducing mechanisms to forcibly preempt resources from threads when necessary. This approach requires careful consideration and should be used sparingly to avoid undesirable side effects.

Deadlock Detection and Recovery:
Implement deadlock detection mechanisms that periodically analyze the state of the system to identify potential deadlocks. If a deadlock is detected, appropriate recovery actions can be taken, such as terminating one or more

threads or rolling back operations to a safe state.

Proper Resource Management:
Ensure proper resource management practices, such as releasing resources as soon as they are no longer needed. Avoid holding onto resources longer than necessary to minimize the chances of deadlocks.

Use Synchronization Primitives Wisely:
When using synchronization primitives like locks or semaphores, be mindful of their scope and duration of use. Acquire locks only when necessary and release them as soon as they are no longer needed. Avoid acquiring multiple locks simultaneously whenever possible.

By following these techniques and best practices, the risk of deadlocks in multithreading can be minimized. It is crucial to carefully analyze the resource dependencies and design the threading model to ensure efficient and deadlock-free execution.

15. **Binary Search Algorithm:**
=>Binary Search is a searching algorithm used in a sorted array by repeatedly dividing the search interval in half and the correct interval to find is decided based on the searched value and the mid value of the interval.
Or
Binary search is one of the searching techniques applied when the input is sorted as here we are focusing on finding the middle element that acts as a reference frame whether to go left or right to it as the elements are already sorted. This searching helps in optimizing the search technique with every iteration is referred to as binary search and readers do stress over it as it is indirectly applied in solving questions. Now you must be thinking what if the input is not sorted then the results are undefined.

Best Case Time Complexity of Binary Search Algorithm: O(1)

Worst Case Time Complexity of Binary Search Algorithm: O(logn)

16. **What is travelling a salesman problem and how it can be solved using dynamic programming provide step by step flow with examples.? LINKS:** Traveling Salesman Problem (TSP): Everything You Need to Know in 2023 (upperinc.com)

=>

The traveling Salesman Problem (TSP) is a combinatorial problem that deals with finding the shortest and most efficient route to follow for reaching a list of specific destinations.

It is a common algorithmic problem in the field of delivery operations that might hamper the multiple delivery process and result in financial loss. TSP turns out when we have multiple routes available but choosing a minimum cost path is really hard for us or a traveling person.

3. Dynamic Programming Approach for Solving TSP

Let's first see the pseudocode of the dynamic approach of TSP, then we'll discuss the steps in detail:

```
Algorithm 1: Dynamic Approach for TSP
  Data: $s$: starting point; $N$: a subset of input cities; $dist()$:
        distance among the cities
  Result: $Cost$ : TSP result
  $Visited[N] = 0$;
  $Cost = 0$;
  Procedure TSP($N$, $s$)
      $Visited[s] = 1$;
      if $|N| = 2$ and $k \neq s$ then
          $Cost(N, k) = dist(s, k)$;
          Return Cost;
      else
          for $j \in N$ do
              for $i \in N$ and $visited[i] = 0$ do
                  if $j \neq i$ and $j \neq s$ then
                      $Cost(N, j) = \min ( TSP(N - \{i\}, j) + dist(j, i))$
                      $Visited[j] = 1$;
                  end
              end
          end
      end
      Return $Cost$;
  end
```

In this algorithm, we take a subset N of the required cities needs to be visited, distance among the cities $dist$ and starting city s as inputs. Each city is identified by unique city id like $\{1, 2, 3, \cdots, n\}$.

17. Explain dynamic programming strategy?

=>

The two dynamic programming properties which can tell whether it can solve the given problem or not are:

- Optimal substructure: an optimal solution to a problem contains optimal solutions to sub problems.

- Overlapping sub problems: a recursive solution contains a small number of distinct sub problems repeated many times.

Strategy:

- Breaks down the complex problem into simpler steps
- Find optimal solution to these subproblems
- Store the result of subproblems (memoization)
- Reuse them so same subproblem is not calculated more than once
- Finally calculate the result of complex problem.

Dynamic programming is a problem-solving technique that involves breaking down
a complex problem into smaller overlapping subproblems and solving each subproblem only once.
It is typically used for optimization problems where the solution can be expressed as the optimal
solution to a series of overlapping subproblems.

The key to solving a dynamic programming problem is finding the optimal substructure and then
recursively finding the optimal solution to each subproblem. The optimal solution to the overall
Dynamic programming is often used for problems
such as the knapsack problem, longest common subsequence, longest increasing subsequence,
edit distance, and many others.

v