***1.Bubble Sort:***

Bubble Sort is a simple comparison-based sorting algorithm.

It repeatedly steps through the list, compares adjacent elements, and swaps them if they are in the wrong order.

It continues iterating through the list until no swaps are needed, indicating that the list is sorted.

- Best Case Time Complexity: O(n)

- Average Case Time Complexity: O(n^2)

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

- Space Complexity: O(1) - Bubble sort is an in-place sorting algorithm, meaning it doesn't require additional space other than the array being sorted.

***2. Selection Sort:***

Selection Sort sorts an array by repeatedly finding the minimum element from the unsorted part and moving it to the beginning.

It maintains two subarrays: sorted and unsorted.

It repeatedly selects the smallest element from the unsorted part and swaps it with the first unsorted element.

- Best Case Time Complexity: O(n^2)

- Average Case Time Complexity: O(n^2)

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

- Space Complexity: O(1) - Similar to bubble sort, selection sort operates in-place, so it doesn't require extra space.

***3. Insertion Sort:***

Insertion Sort builds the final sorted array one element at a time.

It iterates through the list, removing one element at a time and finding its correct position in the sorted part of the array.

It shifts larger elements to the right to make space for the inserted element.

- Best Case Time Complexity: O(n)

- Average Case Time Complexity: O(n^2)

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

- Space Complexity: O(1) - Insertion sort, like the other two, works in-place.

***4. Quicksort:***

Quicksort is a divide-and-conquer sorting algorithm.

It selects a pivot element and partitions the array into two sub-arrays: elements less than the pivot and elements greater than the pivot.

It recursively sorts the sub-arrays.

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

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

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

- Space Complexity: O(log n) - Quicksort is a recursive algorithm and requires O(log n) space for the call stack in the best and average cases. However, in the worst case, it may require O(n) space due to a large number of recursive calls.

These complexities give an idea of how these algorithms perform under different scenarios. Quicksort generally outperforms the other three algorithms in most cases, especially for larger datasets, due to its efficient average and best-case time complexities. However, it's important to consider the worst-case scenario and space requirements when choosing an algorithm for a particular problem.

In terms of speed, the fastest algorithm among the four you mentioned is typically Quicksort. Quicksort has an average-case time complexity of O(n log n), which means it generally performs better than the others, especially for larger datasets.

Following Quicksort, Insertion Sort tends to be faster than Bubble Sort and Selection Sort, particularly for small datasets or nearly sorted arrays, because of its better best-case time complexity of O(n).

However, it's important to note that the performance of these algorithms can vary based on factors like the size of the dataset, the distribution of elements, and the specific implementation details. Additionally, Quicksort's worst-case time complexity of O(n^2) can sometimes be a concern, although randomized or carefully implemented variants can mitigate this issue in practice.

In summary, the sequence of increasing speed among the four algorithms is typically:

1. Quicksort

2. Insertion Sort

3. Selection Sort

4. Bubble Sort

1. Which of the following sorting algorithms has the best average-case time complexity?

a) Bubble Sort

b) Selection Sort

c) Insertion Sort

d) Quicksort

2. Which sorting algorithm exhibits the worst-case time complexity of O(n^2)?

a) Quicksort

b) Bubble Sort

c) Insertion Sort

d) Selection Sort

e) AOT

3. Which sorting algorithm is most suitable for sorting small datasets or nearly sorted arrays efficiently?

a) Bubble Sort

b) Selection Sort

c) Insertion Sort

d) Quicksort

4. Among the given sorting algorithms, which one typically requires the least additional space?

a) Bubble Sort

b) Selection Sort

c) Insertion Sort

d) Quicksort

5. In which sorting algorithm is the sorting process performed in-place without requiring additional space?

a) Quicksort

b) Bubble Sort

c) Insertion Sort

d) Selection Sort

***1. Linear Search :***

- Linear search is a simple searching algorithm that sequentially checks each element in the list until the desired element is found or the end of the list is reached.

- It is applicable to both sorted and unsorted lists.

- Time Complexity:

- Best Case: O(1) - when the target element is the first element in the list.

- Worst Case: O(n) - when the target element is at the end of the list or not present.

- Average Case: O(n/2) - on average, the target element is expected to be found in the middle of the list.

- Space Complexity: O(1) - Linear search does not require additional space.

- Linear search is straightforward to implement and is suitable for small lists or when the list is not sorted.

***2. Binary Search :***

- Binary search is a more efficient searching algorithm that works on sorted lists.

- It repeatedly divides the search interval in half until the target element is found or the search interval becomes empty.

- It requires the list to be sorted beforehand.

- Time Complexity:

- Best Case: O(1) - when the target element is the middle element of the list.

- Worst Case: O(log n) - when the target element is not present in the list or is at one of the ends.

- Average Case: O(log n) - similar to the worst case, as binary search consistently divides the search interval in half.

- Space Complexity: O(1) - Binary search is an iterative algorithm and does not require additional space.

- Binary search is highly efficient for large sorted lists but requires the list to be sorted initially.

In summary, while linear search is simple and applicable to both sorted and unsorted lists, it is less efficient compared to binary search, especially for large sorted lists. Binary search, on the other hand, offers a significant improvement in performance but requires the list to be sorted beforehand.

1. What is a linked list?

a) A linear data structure

b) A non-linear data structure

c) A hierarchical data structure

d) A graph data structure

2. Which of the following statements about a doubly linked list is true?

a) Each node contains only one reference or link.

b) It allows traversal of elements only in one direction.

c) Each node contains references to both the next and previous nodes.

d) It cannot contain cycles.

3. Which of the following operations in a singly linked list has a time complexity of O(1)?

a) Insertion at the end

b) Deletion at the beginning

c) Deletion at the end

d) Searching for an element

4. In a singly linked list, which operation requires traversal of the entire list?

a) Insertion at the beginning

b) Insertion at the end

c) Deletion at the beginning

d) Deletion at the end

5. Which of the following is a disadvantage of linked lists compared to arrays?

a) Linked lists have better cache locality.

b) Linked lists have constant-time access to elements.

c) Linked lists have a fixed size.

d) Linked lists require more memory overhead.

6. What is the time complexity of searching for an element in a singly linked list?

a) O(1)

b) O(log n)

c) O(n)

d) O(n^2)

7. Which of the following operations can be performed more efficiently in a doubly linked list compared to a singly linked list?

a) Insertion at the beginning

b) Deletion at the end

c) Traversal from the end to the beginning

d) Searching for an element

8. Which of the following statements about circular linked lists is true?

a) They have no beginning or end.

b) They contain nodes arranged in a linear order.

c) They allow traversal in both forward and backward directions. \*\*

d) They cannot contain cycles.

1. Which of the following is not an operation typically associated with stacks?

a) Push

b) Pop

c) Enqueue

d) Peek

2. What is the data structure principle that governs the behavior of a stack?

a) First-In-First-Out (FIFO)

b) Last-In-First-Out (LIFO)

c) Last-In-Last-Out (LILO)

d) First-In-Last-Out (FILO)

3. Which operation adds an element to the top of the stack?

a) Insert

b) Enqueue

c) Push

d) Add

4. What happens when attempting to pop from an empty stack?

a) The operation fails

b) The stack is resized

c) An exception is thrown

d) The top element is removed

5. Which data structure is commonly used to implement a stack?

a) Array

b) Linked List

c) Queue

d) Hash Table

6. In a stack, which operation retrieves the top element without removing it?

a) Peek

b) Top

c) Pop

d) Access

7. What is the time complexity of the push and pop operations in a stack implemented using an array?

a) O(1)

b) O(log n)

c) O(n)

d) O(n log n)

8. In a stack, which element is the last one to be removed?

a) Bottom

b) Base

c) Top \*\*

d) Peak

1. Which of the following data structure principles governs the behavior of a queue?

a) First-In-First-Out (FIFO)

b) Last-In-First-Out (LIFO)

c) Last-In-Last-Out (LILO)

d) First-In-Last-Out (FILO)

2. Which operation adds an element to the rear of the queue?

a) Enqueue

b) Push

c) Insert

d) Add

3. What happens when attempting to dequeue from an empty queue?

a) The operation fails

b) The queue is resized

c) An exception is thrown

d) The front element is removed

4. Which data structure is commonly used to implement a queue?

a) Array

b) Linked List

c) Stack

d) Hash Table

5. In a queue, which operation retrieves the front element without removing it?

a) Peek

b) Front

c) Dequeue

d) Access

6. What is the time complexity of the enqueue and dequeue operations in a queue implemented using a linked list?

a) O(1)

b) O(log n)

c) O(n)

d) O(n log n)

7. Which of the following is not a valid application of a queue data structure?

a) CPU scheduling

b) Breadth-first search (BFS)

c) Undo functionality in text editors

d) Depth-first search (DFS)

8. In a priority queue, which element is removed first?

a) Highest priority

b) Lowest priority

c) Random priority

d) First-in priority

Certainly! Here are some multiple-choice questions (MCQs) that teachers commonly ask about recursion:

1. What is recursion?

a) A loop that repeats a set of instructions

b) A function that calls itself

c) An algorithm that uses stacks

d) A sorting technique

2. What is the base case in a recursive function?

a) The case where the function returns a value

b) The case where the function calls itself

c) The case where the function terminates without further recursion

d) The case where the function has no parameters

3. What is the process of breaking a problem into smaller subproblems in recursion called?

a) Dividing

b) Conquering

c) Partitioning

d) Decomposition

4. Which of the following statements about recursion is true?

a) Recursion always leads to better performance than iteration

b) Recursion can be used only with mathematical problems

c) Recursion can lead to stack overflow if not implemented properly

d) Recursion is never used in computer programming

5. What is tail recursion?

a) Recursion that occurs at the end of a function

b) Recursion with a base case

c) Recursion that involves a stack

d) Recursion with multiple base cases

6. Which of the following is NOT a characteristic of recursion?

a) It requires more memory compared to iteration

b) It can lead to elegant and concise solutions

c) It can simplify certain problems

d) It can lead to infinite loops if not implemented properly

7. In recursion, what is the function that calls itself called?

a) Initial function

b) Main function

c) Recursive function

d) Base function

8. What is the process of a recursive function calling itself with different arguments?

a) Tail recursion

b) Binary recursion

c) Mutual recursion

d) Indirect recursion

One data structure we could use to implement this graph is called an *adjacency list*.

The primary advantage of hash tables is their ability to perform constant-time (O(1)) average case access to elements, making them highly efficient for large datasets.

Hash tables are a data structure that provides efficient data storage and retrieval operations. They are commonly used to implement associative arrays, dictionaries, or maps, where data is stored in key-value pairs. The primary advantage of hash tables is their ability to perform constant-time (O(1)) average case access to elements, making them highly efficient for large datasets.



A hash table consists of an array where data elements are stored, and each element is associated with a unique key. To determine the location in the array where an element should be stored, a hash function is used. The hash function takes the key as input and computes an index (hash code) within the array range. This index is where the element will be placed.

To handle collisions, various collision resolution methods are employed:

**Chaining:** In this method, each element in the array points to a linked list or other data structure. When a collision occurs, new elements are simply appended to the existing data structure at that location.

**Open Addressing:** In this method, when a collision occurs, the algorithm searches for the next available empty slot in the array and places the element there. There are different approaches within open addressing, such as linear probing, quadratic probing, and double hashing.

Hashing is a technique that is used to uniquely identify a specific object from a group of similar objects. Suppose an object is to be assigned a key to it to make searching easy. To store the key/value pair, one can use a simple array like a data structure where keys (integers) can be used directly as an index to store values. However, in cases where the keys are large and cannot be used directly as an index, one should use hashing. In hashing, large keys are converted into small keys by using hash functions. The values are then stored in a data structure called hash table. Linear Probing, It may happen that the hashing technique is used to create an already used index of the array. In such a case, we can search for the next empty location in the array by looking into the next cell until we find an empty cell. This technique is called linear probing.

There are three basic operations linked with linear probing which are as follows:

Search

Insert

Delete

Implementation: Hash tables with linear probing by making a helper class and testing this in the main class.

**Linear probing is a technique used in hash tables to resolve collisions. When a collision occurs (i.e., two keys hash to the same index), linear probing involves probing sequentially through the hash table until an empty slot is found. The probing sequence is usually of the form hash(key) + i, where i is incremented until an empty slot is found.**