Other Topics in SE

Recursion:

Questions:

1. What is potential consequence of not including a base case in a recursive function?

A. Not including a base case in a recursive function can lead to infinite loop,, causing the program to run out of stack space and resulting in a crash or stack overflow error.

2. What does memoization allow you to avoid?

A. Memoization allows you to avoid redundant calculations by caching and reusing previously computed results(it means that it doesn’t run for the same inputs more than once by keeping the records). This can significantly improve the performance of recursive functions. It especially useful for recursion or dynamic programming problems, where the same subproblems are solved repeatedly.

3. What are the steps to writing a recursive function?

A. Steps to writing a recursive function:

- Identify the base case(s) to terminate the recursion.
- Define the problem in terms of smaller instances of itself.
- Ensure progress towards the base case with each recursive call
- Implement the recursive calls in the function.

Here is an example:

```
function factorial(x)
{

if (x === 0)
{
return 1;
}
return x * factorial(x-1);

}
console.log(factorial(5));
```

A recursive function must have at least one Condition where it will stop calling itself, or the function Will call itself indefinitely until js throws an error.The condition that stops a recursive function from calling itself is known as the base case.

Linked Lists:

Questions:

1. Why would we use a Linked List instead of an array?

A. Linked lists are dynamic data structures, allowing for efficient insertion and deletion of elements at any position, unlike arrays where resizing can be costly.

   Memory for linked lists is allocated as needed, reducing the potential for wasted space compared to fixed-size arrays.

   Linked lists are particularly useful when the size of the data structure is unknown or subject to change.

   The properties of Linked List is, It starts with aHEAD which is the starting point or the Memory location of the first node. Linked list ends with the last node pointing to NULL value. Unlike array elements are not stored in contiguous memory locations but are stored in random locations.There are 3 types of linked List

   a.Singly LinkedList

   b.Doubly Linked List

   C. Circular Linked List.

2. When we want to add a node to a linked list, do we have to scoot over the subsequent nodes (like we do for arrays)?

A. No, in a linked list, you do not need to shift subsequent nodes when adding a new node. You simply update pointers to insert or remove elements. This makes insertions and deletions more efficient compared to arrays, where shifting elements may be necessary.

   The Way to INSERT a NODE in Linked List is:

   1.Allocate memory for the new node.

   2. Put our data into the new node.

   3.set Next of the new node to point to the PRevious Head.

4. Reset Head to point to the new node.

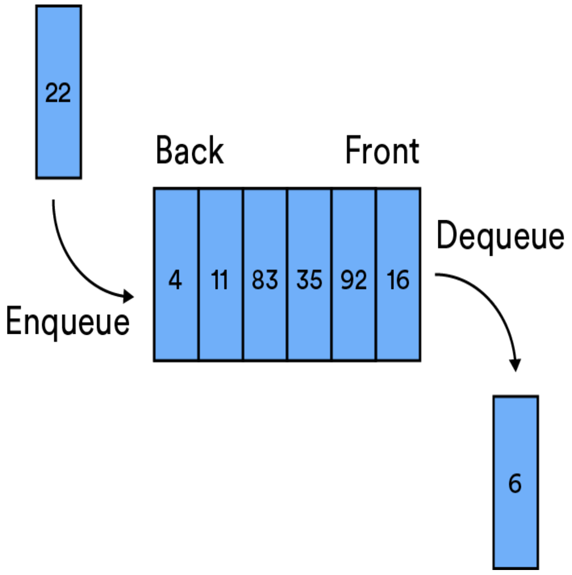
3. Can we do index access like we can with arrays with linked lists?

A. Unlike arrays, linked lists don't support direct index access. To access an element, you typically start from the head (or tail) and traverse the list by following pointers until you reach the desired position. This results in O(n) time complexity for access, as opposed to O(1) for arrays. Linked lists are better suited for scenarios where insertion and deletion operations are more frequent than random access.

STACKS AND QUEUES:

What is Stack?

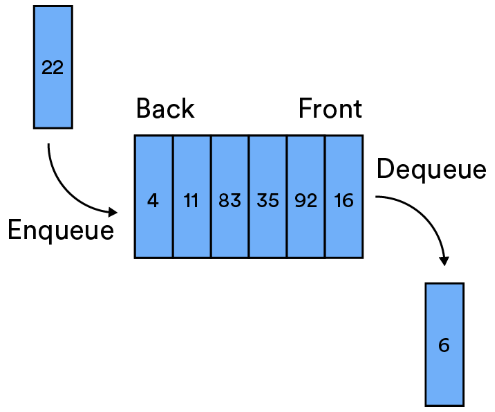
Stack is a linear data structure that follows a particular order in which the operations are performed. The order may be LIFO(Last In First Out) or FILO(First In Last Out). LIFO implies that the element that is inserted last, comes out first and FILO implies that the element that is inserted first, comes out last.



The function call stack is a common example of stacks in programming. When you call a function to execute, it’s pushed to the top of the stack and runs until we add another function to the stack, which then runs until it returns (or another function is pushed to the top), on a last in, first out basis. You can keep adding functions until you’ve run out of space in the stack, in which case you’ve reached stack overflow.

What is Queue?

A Queue is defined as a linear data structure that is open at both ends and the operations are performed in First In First Out (FIFO) order.We define a queue to be a list in which all additions to the list are made at one end, and all deletions from the list are made at the other end. The element which is first pushed into the order, the operation is first performed on that.



What data structure would be represented by the back button in the browser and sending the documents to the printer is it stack or Queue?

Back button in the browser, represents LIFO(last IN First OUT) which means it is a Stack.Sending documents to the Printer represents FIFO(First IN First OUT) which means it is a queue.

Implementing Stacks & Queues:

The other thing that stacks and queues have in common is that they can both be implemented as an array or as a linked list.

The major difference between a linked list and an array is how they store data in a computer's memory?
In terms of memory allocation, arrays require a contiguous block of memory, which can be a limitation if large arrays are needed and memory is fragmented. Linked lists, however, do not require contiguous memory, as each node can be stored anywhere in memory.

Which of the following statements is true about how linked lists and arrays store data?
Ans:Linked lists can store data anywhere in a computer's memory using pointers.

Queue Variations: 1.Priority Queues; 2.Double-Ended Queues
These have three rules in addition to regular queues:

1. Every item has a priority associated with it.
2. An element with high priority is dequeued before an element with low priority.
3. If two elements have the same priority, they are served according to their order in the queue.

Examples: Airport priority boarding, CPU scheduling.
Double-Ended Queues
A double-ended queue, or "deque", performs insertions and deletions at both ends of the queue.
They're usually implemented with doubly-linked lists or dynamic arrays.
Have you ever been last in a long line at the grocery store only to see a cashier one lane over open up their register? Typically, that cashier will beckon to you to jump into their lane so you can be checked out right away. That's basically what happens in a deque.
Example: spreading tasks between different servers as equally as possible

Q. You're working on building a message board and want to display only the 10 most recent messages a user posted, in the order they were posted. Which data structure should you use?
Ans: We use arrays and Stacks data structures.

Q:You and your partner are going out for your anniversary dinner. When you arrive, you ask the host for a table for two and your name goes on the waiting list. While you're waiting, a group of seven people walks right in and gets seated. What gives?!
Ans:Double-Ended Queues:Have you ever been last in a long line at the grocery store only to see a cashier one lane over to open up their register? Typically, that cashier will beckon to you to jump into their lane so you can be checked out right away. That's basically what happens in a dequeue.
Testing:

1.Why do we create tests?
As programmers, we use code to solve problems. Most libraries and frameworks have test libraries available that let us write code to evaluate the robustness, completeness and flex of our applications. In a production-level application, providing a high level of test coverage an application is almost always required in order to guarantee that code is bug-free and functions as intended.
There are many types of tests that we can create for our applications:

- Unit tests: the smallest, most microscopic level of testing. Evaluates individual methods and functions within a codebase. The kind we'll be writing today! Integration tests: ensure that different services and modules work together.
- End-to-end tests: verify that application responds as expected to user interactions, such as evaluating how user input edge cases are handled.
- Performance tests: also known as load testing, and evaluate application's response to heavy traffic (number of requests, large amounts of data).
- Acceptance tests: ensure that the application meets its given business requirements.

TDD: Test-Driven Development:

JavaScript Testing with Mocha & Chai:
Mocha will be our testing framework, but we're mostly just using it as a test runner. "Mocha is a feature-rich JavaScript test framework running on Node.js and the browser, making asynchronous testing simple and fun. Mocha tests run serially, allowing for flexible and accurate reporting, while mapping (associating) uncaught exceptions to the correct test cases." For assertions, we will use Chai:
"Chai is a BDD / TDD assertion library for Node and the browser that can be delightfully paired with any JavaScript testing framework."

1. What the heck is an assertion?
Ans: It's a way of writing a unit test that tests whether or not a test case is passing or failing by comparing the expected result with the actual result of a test.

2. What is an API? What is API Testing?
APIs are mechanisms that enable two software components to communicate with each other using a set of definitions and protocols. For example, the weather bureau's software system contains daily weather data. The weather app on your phone "talks" to this system via AP/s and shows you daily weather updates on your phone.
Websocket APIs
Websocket API is another modern web API development that uses JSON objects to pass data.
A WebSocket API supports two-way communication between client apps and the server. The
server can send callback messages to connected clients, making it more efficient than REST API.

What are REST APIs?
REST stands for Representational State Transfer. REST defines a set of functions like GET, PUT, DELETE, etc. that clients can use to access server data. Clients and servers exchange data using HTTP.

What is API Testing?

Let's distinguish between three kinds of test flows which comprise our test plan:

1. Testing requests in isolation - Executing a single API request and checking the response accordingly. Such basic tests are the minimal building blocks we should start with, and there's no reason to continue testing if these tests fail.
2. Multi-step workflow with several requests - Testing a series of requests which are common user actions, since some requests can rely on other ones. For example, we execute a POST request that creates a resource and returns an auto-generated identifier in its response. We then use this identifier to check if this resource is present in the list of elements received by a GET request. Then we use a PATCH endpoint to update new data, and we again invoke a GET request to validate the new data. Finally, we DELETE that resource and use GET again to verify it no longer exists.
3. Combined API and web UI tests - This is mostly relevant to manual testing, where we want to ensure data integrity and consistency between the UI and API.

RESTful HTTP Methods:

RESTful HTTP Methods:

Method	Crud functionality	DB Action
GET	read	retrieve data
POST	create	add data
PUT	update	modify existing data
PATCH	update	modify existing data
DELETE	delete	delete existing data