1.3 Array Operations | Deletion from Array | Explanation with Code | Data Structure

I will discuss the deletion operation using the example I used in a previous video. Understanding how to delete data from a specific position will also make it easier to delete data from the beginning or end of the array. I declared an array of size 50 in the previous code, and the memory manager allocated 200 bytes of memory for this array. One variable, size, is used to determine the maximum size of the array. If the user wants to insert only 10 or 5 elements, for example, I will ask them how much size they want for the array, and this will allocate an additional 4 bytes of memory. The user will then enter the elements of the array, which will be initialized at runtime. To delete data from the array, I will ask the user from which position they want to delete the data. For example, if they want to delete data from position 2, I will shift the values from position 3 to 4 and reduce the size of the array by 1.I will start a loop from the position to be deleted to the end of the array and shift the values to the left. To print the updated array after deletion, I will use a for loop to print each element of the array. It's important to check the validity of the position entered by the user before deleting data from the array. If the position is invalid, such as -1 or greater than the size of the array, I will print "invalid position". If there is no data in the array, it's also an underflow condition and the data cannot be deleted. When deleting data from the end of the array, I will simply decrement the size of the array. When deleting data from the beginning of the array,I will shift all the elements to the left and decrement the size of the array. The time complexity of the deletion operation depends on the position from which the data is to be deleted. If the data is to be deleted from a specific position, all the elements to the right of that position must be shifted to the left, which takes O(n) time. However, if the array is unsorted, a quicker algorithm is to pick the last element of the array and put it in the position to be deleted. The best algorithm for this case takes constant time, which is O(1).