**1️ Algorithm for create()**

**Purpose:** Create an array with user-defined elements.  
**Steps:**

1. **Input**: size of the array.
2. Check if size > 0 and size <= 20. If invalid, print an error.
3. **Loop**: From i = 0 to size - 1 to input elements.
4. Store each element in the array a[i].
5. **End**

**Time Complexity:** O(n)

**2️ Algorithm for display()**

**Purpose:** Display the array elements.  
**Steps:**

1. Check if the array is empty (size == 0). If yes, print a message and stop.
2. **Loop**: From i = 0 to size - 1.
3. Print each element a[i].
4. **End**

**Time Complexity:** O(n)

**3️ Algorithm for insert()**

**Purpose:** Insert an element at the beginning, end, or a specific position.  
**Steps:**

1. Input a choice:
   * 'a' for the beginning
   * 'b' for a specific position
   * 'c' for the end
2. Call the corresponding insertion function based on the input.
3. **End**

**Time Complexity:** Depends on the chosen method:

* Insert at beginning: O(n)
* Insert at a specific position: O(n)
* Insert at the end: O(1)

**4️ Algorithm for insertatbeginning()**

**Purpose:** Insert an element at the start of the array.  
**Steps:**

1. Check if the array is full. If full, print an error and stop.
2. Input the num to insert.
3. **Shift elements**: From i = size - 1 down to 0, move a[i] to a[i+1].
4. Assign a[0] = num.
5. Increment size by 1.
6. **End**

**Time Complexity:** O(n)

**5️ Algorithm for insertatspecipos()**

**Purpose:** Insert an element at a specified position.  
**Steps:**

1. Check if the array is full. If yes, print an error.
2. Input pos (position) and num (element).
3. Validate if pos is within the valid range (1 <= pos <= size+1).
4. **Shift elements**: From i = size - 1 down to pos - 1, move a[i] to a[i+1].
5. Insert num at a[pos - 1].
6. Increment size by 1.
7. **End**

**Time Complexity:** O(n)

**6️ Algorithm for insertatend()**

**Purpose:** Insert an element at the end.  
**Steps:**

1. Check if the array is full.
2. Input num to insert.
3. Assign a[size] = num.
4. Increment size by 1.
5. **End**

**Time Complexity:** O(1)

**7️ Algorithm for delete()**

**Purpose:** Delete an element from the beginning, end, or a specific position.  
**Steps:**

1. Input a choice:
   * 'a' for the beginning
   * 'b' for a specific position
   * 'c' for the end
2. Call the corresponding deletion function based on the input.
3. **End**

**Time Complexity:** Depends on the chosen method:

* Delete from beginning: O(n)
* Delete from a specific position: O(n)
* Delete from the end: O(1)

**8️ Algorithm for deleteatbeginning()**

**Purpose:** Delete an element from the start.  
**Steps:**

1. Check if the array is empty. If yes, print an error.
2. **Shift elements**: From i = 0 to size - 2, move a[i+1] to a[i].
3. Decrement size by 1.
4. **End**

**Time Complexity:** O(n)

**9️ Algorithm for deleteatspecipos()**

**Purpose:** Delete an element from a specified position.  
**Steps:**

1. Check if the array is empty. If yes, print an error.
2. Input the pos to delete.
3. Validate the position (1 <= pos <= size).
4. **Shift elements**: From i = pos - 1 to size - 2, move a[i+1] to a[i].
5. Decrement size by 1.
6. **End**

**Time Complexity:** O(n)

**🔟 Algorithm for deleteatend()**

**Purpose:** Delete the last element of the array.  
**Steps:**

1. Check if the array is empty. If yes, print an error.
2. Decrement size by 1.
3. **End**

**Time Complexity:** O(1)