explain Linear and Binary Search algorithm and bubble sorting algorithm for array and linked list

Linear search, binary search, and bubble sort are fundamental algorithms with distinct implementations for arrays and linked lists. Here's a breakdown of their mechanisms and characteristics:

Linear Search

For Arrays

Sequentially checks each element until finding the target.

- **Time Complexity**: $O(n)$ [1]
- **Implementation**:

```
def linearSearch(array, x):
    for i in range(len(array)):
        if array[i] == x:
            return i
    return -1
```

  Works for both sorted and unsorted arrays[1] [2].

For Linked Lists

Traverses nodes sequentially from head to tail:

- **Time Complexity**: $O(n)$ [3] [4]
- **Method**:

```
Node* linearSearch(Node* head, int target) {
    Node* current = head;
    while (current != NULL) {
        if (current->data == target) return current;
        current = current->next;
    }
    return NULL;
}
```

Binary Search

For Arrays
Requires a sorted array and divides the search range by half repeatedly:

- **Time Complexity**: $O(\log n)$ [5]
- **Implementation**:

```
int binarySearch(int[] arr, int target) {
    int left = 0, right = arr.length - 1;
    while (left <= right) {
        int mid = left + (right - left)/2;
        if (arr[mid] == target) return mid;
        if (arr[mid] < target) left = mid + 1;
        else right = mid - 1;
    }
    return -1;
}
```

For Linked Lists
Possible only if the list is sorted, but inefficient due to lack of random access:

- **Approach**:
  1. Find the middle node (requires $O(n)$ time per step)
  2. Compare and adjust search range[6]
- **Time Complexity**: $O(n \log n)$ [6]
- **Limitations**: Not practical for most use cases; linear search is often preferred[3] [6].

Bubble Sort

For Arrays
Repeatedly swaps adjacent elements to move larger values to the end:

- **Time Complexity**:
  - Best: $O(n)$ (already sorted)
  - Average/Worst: $O(n^2)$ [7]
- **Optimized Implementation**:

```
void bubbleSort(int[] arr) {
    boolean swapped;
    for (int i = 0; i < arr.length-1; i++) {
        swapped = false;
        for (int j = 0; j < arr.length-i-1; j++) {
            if (arr[j] > arr[j+1]) {
                swap(arr, j, j+1);
                swapped = true;
            }
        }
        if (!swapped) break;
```

```
    }
}
```

For Linked Lists
Swaps adjacent nodes instead of elements:

- **Time Complexity**: $O(n^2)$ [8]
- **Key Steps**:
  1. Traverse the list, comparing adjacent nodes
  2. Swap nodes by adjusting pointers (not just data)[8]
- **Example**: Sorting `6 → 3 → 1 → 9` becomes `1 → 3 → 6 → 9` after multiple passes[8].

Key Comparisons

Algorithm	Array Complexity	Linked List Complexity	Key Constraints
Linear Search	$O(n)$ [1]	$O(n)$ [4]	Works on unsorted data
Binary Search	$O(\log n)$ [5]	$O(n \log n)$ [6]	Requires sorted data
Bubble Sort	$O(n^2)$ [7]	$O(n^2)$ [8]	Inefficient for large datasets

Linear search is versatile but slow for large datasets. Binary search excels in speed for sorted arrays but struggles with linked lists. Bubble sort, while simple, is rarely used in practice due to quadratic time complexity, though it serves as an educational tool for understanding sorting mechanics.

⁂

1. https://www.programiz.com/dsa/linear-search
2. https://www.digitalocean.com/community/tutorials/linear-search-algorithm-c
3. https://www.w3schools.com/dsa/dsa_algo_linkedlists_operations.php
4. https://dotnettutorials.net/lesson/linear-search-in-linked-list/
5. https://www.w3schools.com/dsa/dsa_algo_binarysearch.php
6. https://www.prepbytes.com/blog/linked-list/binary-search-on-linked-list/
7. https://dev.to/chinonsoike/understanding-bubble-sort-algorithm-with-examples-in-java-3gae
8. https://www.prepbytes.com/blog/linked-list/c-program-for-performing-bubble-sort-on-linked-list/