2. Linear Search

2.1 Concept

Linear Search (also called Sequential Search) checks every element in the array sequentially until the target is found or the end is reached.

How it works:

1. Start from the first element
2. Compare each element with the target
3. If match found, return the position
4. If end reached without match, return -1

Visual Representation:

```
Array: [10, 25, 30, 15, 40, 35]
Target: 15

Step 1: Check 10 ≠ 15
Step 2: Check 25 ≠ 15
Step 3: Check 30 ≠ 15
Step 4: Check 15 = 15 ✓ Found at index 3!
```

2.2 Implementation

Basic Linear Search

```
#include <stdio.h>

int linearSearch(int arr[], int n, int target) {
    for (int i = 0; i < n; i++) {
        if (arr[i] == target) {
            return i;  // Return index if found
        }
    }
    return -1;  // Return -1 if not found
```

```
}

int main() {
    int arr[] = {10, 25, 30, 15, 40, 35};
    int n = sizeof(arr) / sizeof(arr[0]);
    int target = 15;

    int result = linearSearch(arr, n, target);

    if (result != -1) {
        printf("Element %d found at index %d\n", target, result);
    } else {
        printf("Element %d not found\n", target);
    }

    return 0;
}
```

Linear Search with Count

```
int linearSearchCount(int arr[], int n, int target, int *comparisons) {
    *comparisons = 0;

    for (int i = 0; i < n; i++) {
        (*comparisons)++;
        if (arr[i] == target) {
            return i;
        }
    }
    return -1;
}

// Usage
int main() {
    int arr[] = {10, 25, 30, 15, 40, 35};
    int n = sizeof(arr) / sizeof(arr[0]);
    int target = 15;
    int comparisons = 0;

    int result = linearSearchCount(arr, n, target, &comparisons);
```

```
    printf("Result: %d\n", result);
    printf("Comparisons made: %d\n", comparisons);

    return 0;
}
```

Find All Occurrences

```
void linearSearchAll(int arr[], int n, int target) {
    int found = 0;

    printf("Element %d found at indices: ", target);

    for (int i = 0; i < n; i++) {
        if (arr[i] == target) {
            printf("%d ", i);
            found = 1;
        }
    }

    if (!found) {
        printf("Not found");
    }
    printf("\n");
}

int main() {
    int arr[] = {10, 25, 30, 25, 40, 25};
    int n = sizeof(arr) / sizeof(arr[0]);

    linearSearchAll(arr, n, 25);
    // Output: Element 25 found at indices: 1 3 5

    return 0;
}
```

2.3 Complexity Analysis

Case	Time Complexity	Description

Best Case	O(1)	Element found at first position
Average Case	O(n)	Element found in middle
Worst Case	O(n)	Element at last position or not found
Space Complexity	O(1)	No extra space needed

Visualization:

```
Best Case (1 comparison):
[15, 25, 30, ...] → Found immediately!

Average Case (n/2 comparisons):
[10, 25, 30, 15, ...] → Found in middle

Worst Case (n comparisons):
[10, 25, 30, 40, 35, 15] → Found at end
or
[10, 25, 30, 40, 35, 20] → Not found (checked all)
```

2.4 Advantages and Disadvantages

Advantages:

- Simple to implement
- Works on unsorted arrays
- No preprocessing required
- Works well for small datasets
- No extra memory needed

Disadvantages:

- Slow for large datasets
- Inefficient compared to other algorithms
- Time complexity increases linearly

When to Use:

- Small datasets (n < 100)
- Unsorted data
- When simplicity is priority
- When data changes frequently

Revision #2
Created 2025-11-03 02:01:35 UTC by DS
Updated 2025-11-03 02:02:21 UTC by DS