Notes: Searching and Sorting Algorithms

Searching in a List

Today, I explored searching and sorting techniques used on lists. These are fundamental operations used in data structures and algorithms.

1. Linear Search

Linear search checks each element one by one until it finds the target or reaches the end.

Code:

```
def linear_search(arr, target):
    for i in range(len(arr)):
        if arr[i] == target:
            return i
    return -1
```

Time: O(n), Space: O(1)

2. Binary Search

Binary search works on sorted lists and halves the search space each time.

Code:

```
def binary_search(arr, target):
    low, high = 0, len(arr) - 1
    while low <= high:
        mid = (low + high) // 2
        if arr[mid] == target:
            return mid
        elif arr[mid] < target:
            low = mid + 1
        else:
```

```
            high = mid - 1
    return -1
```

Time: O(log n), Space: O(1)

3. Selection Sort

Selection sort repeatedly finds the minimum and places it at the start.

Code:

```
def selection_sort(arr):
    for i in range(len(arr)):
        min_idx = i
        for j in range(i+1, len(arr)):
            if arr[j] < arr[min_idx]:
                min_idx = j
        arr[i], arr[min_idx] = arr[min_idx], arr[i]
```

Time: $O(n^2)$, Space: O(1)

4. Insertion Sort

Insertion sort builds the sorted list one element at a time.

Code:

```
def insertion_sort(arr):
    for i in range(1, len(arr)):
        key = arr[i]
        j = i - 1
        while j >= 0 and key < arr[j]:
            arr[j + 1] = arr[j]
            j -= 1
```

```
        arr[j + 1] = key
```

Time: O(n²), Space: O(1)

5. Merge Sort

Merge sort is a divide-and-conquer algorithm that splits and merges arrays recursively.

Code:

```
def merge_sort(arr):
    if len(arr) > 1:
        mid = len(arr) // 2
        L = arr[:mid]
        R = arr[mid:]
        merge_sort(L)
        merge_sort(R)
        i = j = k = 0
        while i < len(L) and j < len(R):
            if L[i] < R[j]:
                arr[k] = L[i]
                i += 1
            else:
                arr[k] = R[j]
                j += 1
            k += 1
        while i < len(L):
            arr[k] = L[i]
            i += 1
            k += 1
        while j < len(R):
            arr[k] = R[j]
```

```
        j += 1
        k += 1
```

Time: O(n log n), Space: O(n)