Computer Science By SAMPAT LILER

These complete notes have been made for class 12th board computer science exam.

5. Sorting

Sorting is a fundamental concept in computer science that involves arranging data in a specific order, such as ascending or descending. Sorting is used in many real-life applications like ranking students, managing inventory, and organizing search results.

1. Bubble Sort

Bubble Sort is a simple sorting algorithm that works by repeatedly swapping adjacent elements if they are in the wrong order. It is called "Bubble Sort" because the largest element bubbles up to the last position after each pass.

Working of Bubble Sort

1. Compare the first two elements.
2. If the first element is greater than the second, swap them.
3. Move to the next pair and repeat the process until the largest element reaches the last position.
4. Repeat the same process for the remaining elements until the entire list is sorted.

Example

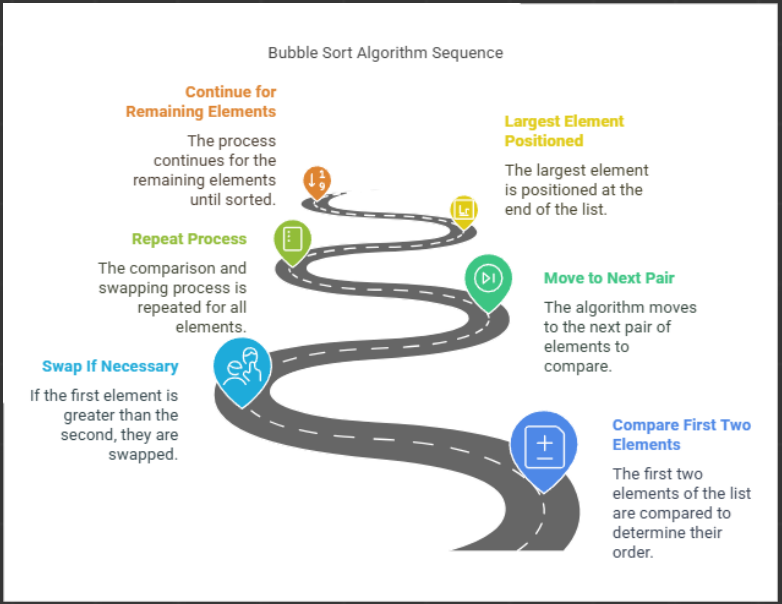
Unsorted List: [8, 7, 13, 1, -9, 4]

Pass 1:

- Compare 8 and 7 ⟶ Swap ⟶ [7, 8, 13, 1, -9, 4]
- Compare 8 and 13 ⟶ No swap
- Compare 13 and 1 ⟶ Swap ⟶ [7, 8, 1, 13, -9, 4]
- Compare 13 and -9 ⟶ Swap ⟶ [7, 8, 1, -9, 13, 4]
- Compare 13 and 4 ⟶ Swap ⟶ [7, 8, 1, -9, 4, 13]

Pass 2:

- Compare 7 and 8 ⟶ No swap
- Compare 8 and 1 ⟶ Swap ⟶ [7, 1, 8, -9, 4, 13]
- Compare 8 and -9 ⟶ Swap ⟶ [7, 1, -9, 8, 4, 13]
- Compare 8 and 4 ⟶ Swap ⟶ [7, 1, -9, 4, 8, 13]
- Now, 13 is sorted



Python Code for Bubble Sort

```
def bubble_sort(arr):
    n = len(arr)
    for i in range(n-1):
        for j in range(n-i-1):
            if arr[j] > arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[j]
    return arr

arr = [8, 7, 13, 1, -9, 4]
sorted_arr = bubble_sort(arr)
print("Sorted List:", sorted_arr)
```

Time Complexity of Bubble Sort

- Best Case: O(n) (If already sorted)
- Average Case: $O(n^2)$
- Worst Case: $O(n^2)$ (If sorted in reverse order)

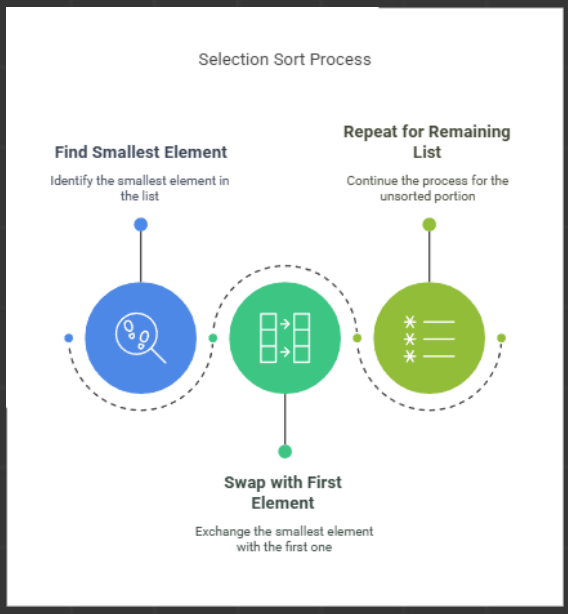
---

2. Selection Sort

Selection Sort sorts a list by selecting the smallest element and swapping it with the first element. It repeats this process until the entire list is sorted.

Working of Selection Sort

1. Find the smallest element in the list.
2. Swap it with the first element.
3. Repeat the process for the remaining unsorted portion of the list.
4.



Example

Unsorted List: [8, 7, 13, 1, -9, 4]

Pass 1:

- Find the smallest element (-9) and swap it with 8
- [-9, 7, 13, 1, 8, 4]

Pass 2:

- Find the smallest element (1) and swap it with 7
- [-9, 1, 13, 7, 8, 4]

Pass 3:

- Find the smallest element (4) and swap it with 13
- [-9, 1, 4, 7, 8, 13]

Python Code for Selection Sort

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

arr = [8, 7, 13, 1, -9, 4]
sorted_arr = selection_sort(arr)
print("Sorted List:", sorted_arr)
```

Time Complexity of Selection Sort

- Best Case: $O(n^2)$
- Average Case: $O(n^2)$
- Worst Case: $O(n^2)$

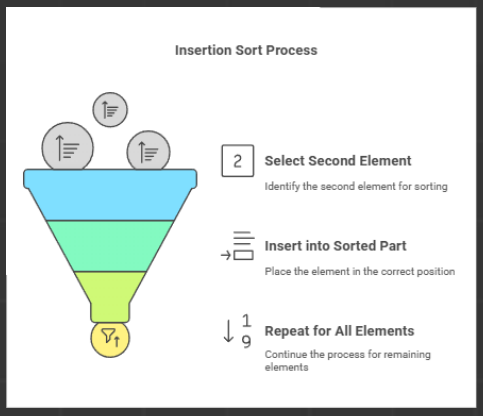
---

3. Insertion Sort

Insertion Sort builds a sorted list one element at a time by inserting each element into its correct position.

Working of Insertion Sort

1. Take the second element and insert it into the correct position in the sorted part.
2. Repeat for all elements.



Example

Unsorted List: [8, 7, 13, 1, -9, 4]

Pass 1: Insert 7 into [8] ⟶ [7, 8, 13, 1, -9, 4]

Pass 2: Insert 13 ⟶ [7, 8, 13, 1, -9, 4]

Pass 3: Insert 1 ⟶ [1, 7, 8, 13, -9, 4]

Pass 4: Insert -9 ⟶ [-9, 1, 7, 8, 13, 4]

Pass 5: Insert 4 ⟶ [-9, 1, 4, 7, 8, 13]

Python Code for Insertion Sort

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

arr = [8, 7, 13, 1, -9, 4]
sorted_arr = insertion_sort(arr)
print("Sorted List:", sorted_arr)
```

Time Complexity of Insertion Sort

- Best Case: O(n) (If already sorted)
- Average Case: $O(n^2)$
- Worst Case: $O(n^2)$ (If sorted in reverse order)

---

4. Time Complexity of Sorting Algorithms

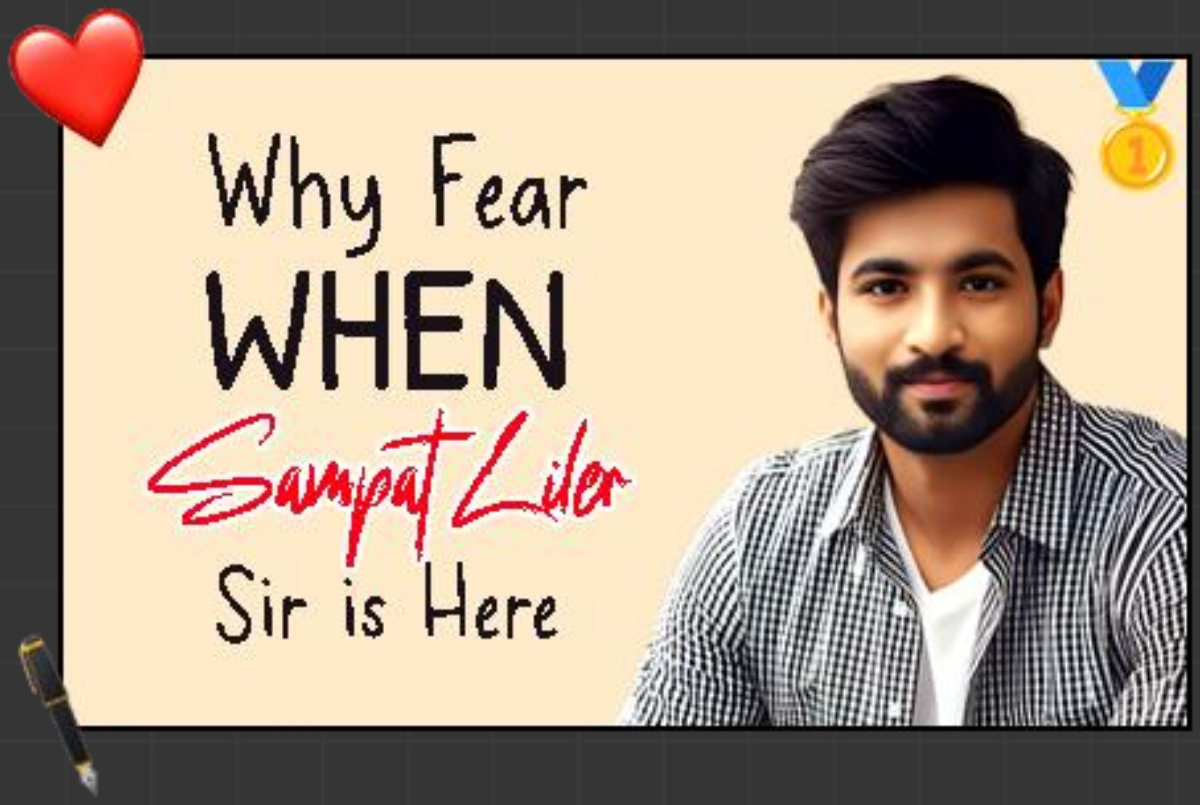
Summary of Time Complexity:

Algorithm	Best Case	Average Case	Worst Case
Bubble Sort	O(n)	$O(n^2)$	$O(n^2)$
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Insertion Sort	O(n)	$O(n^2)$	$O(n^2)$

Comparison of Sorting Algorithms

- Bubble Sort: Simple but inefficient.
- Selection Sort: More efficient than Bubble Sort but still slow.
- Insertion Sort: Works better for small or nearly sorted data.

Summary
Sorting is essential in computer science and has different methods based on efficiency. Bubble Sort, Selection Sort, and Insertion Sort are basic sorting techniques, with Insertion Sort being the most efficient for small datasets.



6. Searching

Introduction to Searching

Searching is the process of finding a specific element in a collection of elements. It helps determine whether an element exists in a dataset and, if present, its location.

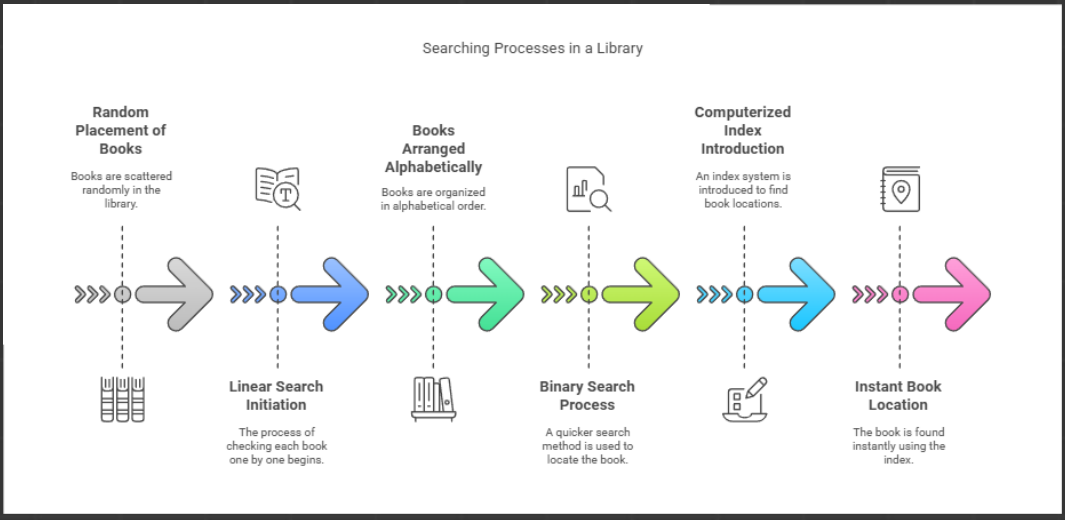
Real-Life Example

Imagine searching for a book in a library:

- If books are randomly placed, you check each book one by one (similar to **Linear Search**).
- If books are arranged alphabetically, you can quickly locate your book (similar to **Binary Search**).
- If there is a computerized index that directly tells the book's location, you get it instantly (similar to **Hashing**).

Types of Searching Techniques

1. **Linear Search (Sequential Search)**
2. **Binary Search**
3. **Search by Hashing**



---

1. Linear Search

Definition

Linear Search is a simple search method where we compare each element of the list one by one with the key (the value to be searched). It works well for small and unsorted datasets.

Algorithm

1. Start from the first element.
2. Compare each element with the key.
3. If a match is found, return the position.
4. If the end of the list is reached and no match is found, return "Not Found".

Example

Consider an array:

[8, -4, 7, 17, 0, 2, 19]

Key to search: 17

Step	Index	Element	Key Match?
1	0	8	No
2	1	-4	No
3	2	7	No
4	3	17	Yes

Result: Element found at position 4.

Python Code

```
def linear_search(arr, key):
    for index in range(len(arr)):
        if arr[index] == key:
            return index + 1  # Position in the list
    return "Element not found"

arr = [8, -4, 7, 17, 0, 2, 19]
key = 17
print("Element found at position:", linear_search(arr, key))
```

Time Complexity

- **Best Case:** O(1) (Key is found at the first position)
- **Worst Case:** O(n) (Key is at the last position or not in the list)
- **Average Case:** O(n)

Advantages

- Simple and easy to implement.
- Works on both sorted and unsorted lists.

Disadvantages

- Inefficient for large datasets.
- Requires n comparisons in the worst case.

---

2. Binary Search

Definition

Binary Search is an efficient search algorithm that works on sorted lists. It repeatedly divides the list into two halves and searches only in the relevant half.

How It Works

1. Find the **middle element** of the sorted list.
2. Compare it with the key.
    - If they match, return the position.
    - If the middle element is greater, search in the left half.
    - If the middle element is smaller, search in the right half.
3. Repeat the process until the key is found or the list is exhausted.

Example

Consider a sorted array:

[2, 3, 5, 7, 10, 12, 17, 19, 23]

Key to search: 17

Step	Left	Right	Middle	Middle Value	Action
1	0	8	4	10	Search Right
2	5	8	6	17	**Found**

Result: Element found at position 7.

Python Code

```
def binary_search(arr, key):
    left, right = 0, len(arr) - 1
    while left <= right:
        mid = (left + right) // 2
```

```
        if arr[mid] == key:
            return mid + 1
        elif arr[mid] < key:
            left = mid + 1
        else:
            right = mid - 1
    return "Element not found"

arr = [2, 3, 5, 7, 10, 12, 17, 19, 23]
key = 17
print("Element found at position:", binary_search(arr, key))
```

Time Complexity

- **Best Case:** O(1) (Key is the middle element)
- **Worst Case:** O(log n) (Repeated halving)
- **Average Case:** O(log n)

Advantages

- Faster than Linear Search (O(log n) vs O(n)).
- Suitable for large datasets.

Disadvantages

- Requires sorted data.
- Extra effort needed for sorting if the list is unsorted.

---

3. Search by Hashing

Definition

Hashing is a technique that allows searching in constant time (O(1)). It uses a **hash function** to compute an index where the element is stored.

How It Works

1. A hash function maps keys to an index in a table.
2. The element is stored at that index.
3. To search, compute the hash value of the key and check the corresponding index.

Example

Consider an array: [34, 16, 2, 93, 80, 77, 51]

Hash Table of size 10 using hash(key) = key % 10:

Element	Hash Value (key % 10)	Stored at Index
34	4	4
16	6	6
2	2	2
93	3	3
80	0	0
77	7	7
51	1	1

To search 16, compute 16 % 10 = 6.
Check index 6 → **Found at position 7.**

Python Code

```
def hash_search(key, hash_table):
    index = key % 10
    return index if hash_table[index] == key else "Element not found"

hash_table = [None] * 10
elements = [34, 16, 2, 93, 80, 77, 51]
for num in elements:
    hash_table[num % 10] = num

key = 16
print("Element found at position:", hash_search(key, hash_table))
```

Time Complexity

- **Best Case:** O(1) (Direct access)
- **Worst Case:** O(n) (If collisions occur)
- **Average Case:** O(1)

Advantages

- Extremely fast (O(1)) if there are no collisions.
- Efficient for large datasets.

Disadvantages

- Requires extra memory for the hash table.
- Collisions occur when multiple elements map to the same index.

Comparison Table

Search Type	Time Complexity (Worst)	Sorted List Required?	Best For
Linear Search	O(n)	No	Small or unsorted datasets
Binary Search	O(log n)	Yes	Large sorted datasets
Hashing	O(1) - O(n)	No	Very large datasets

Summary

- **Linear Search** is best for **small, unsorted lists.**
- **Binary Search** is best for **large, sorted lists.**
- **Hashing** is best when **speed is crucial** and extra memory can be used.

👉 Subscribe Youtube Channel - Anvira Education - YouTube

👉 Join Course - Https://Anviraeducation.Com/

👉 Follow Us On Facebook - Https://Www.Facebook.Com/Anviraedu

👉 Follow Us On Instagram - https://www.instagram.com/anvira_edu/

👉 Sampat Sir Instagram - https://www.instagram.com/writersampat/

👉 Join Our Telegram Channel - https://t.me/Anviraeducation20