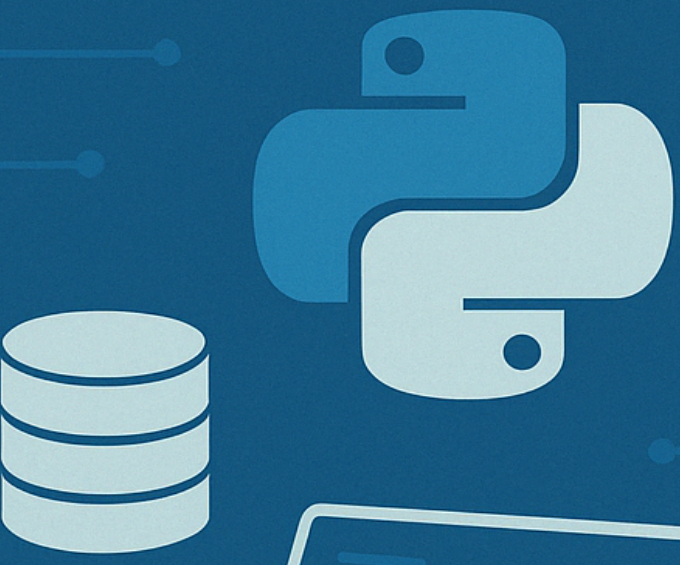
KARNATAKA SECOND PUC

COMPUTER SCIENCE

STUDY MATERIAL

MOHAMMED MATHEEN L R



PYTHON & MYSQL

Contents

LICENSE

Figure 1: Licence

CHAPTER 5: SORTING

CHAPTER NOTES

5.1 Introduction

- **Sorting** is the process of arranging elements in a particular order (ascending, descending, alphabetical, etc.).
- It is essential for easy data retrieval and efficient searching.
- Common applications:
  - Dictionaries (alphabetical order)
  - Exam seating plans (roll number order)
  - Sorting by height, weight, etc.

5.2 Bubble Sort

Concept

- Compares **adjacent elements** and **swaps** them if they are in the wrong order.
- After each **pass**, the largest unsorted element "bubbles up" to its correct position.
- Requires **n – 1 passes** for a list of size **n**.

Optimization Tip If no swaps occur in a pass, the list is already sorted — the algorithm can be terminated early.

Diagram: Bubble Sort Passes We will sort:
`[8, 7, 13, 1, -9, 4]`
Each pass compares adjacent elements and swaps if needed.

Pass 1

```
[8, 7, 13, 1, -9, 4]
 → Swap 8 and 7 → [7, 8, 13, 1, -9, 4]
 → No Swap (8,13)
 → Swap 13 and 1 → [7, 8, 1, 13, -9, 4]
 → Swap 13 and -9 → [7, 8, 1, -9, 13, 4]
 → Swap 13 and 4 → [7, 8, 1, -9, 4, 13]
```

Pass 2

```
[7, 8, 1, -9, 4, 13]
 → No Swap (7,8)
 → Swap 8 and 1 → [7, 1, 8, -9, 4, 13]
 → Swap 8 and -9 → [7, 1, -9, 8, 4, 13]
 → Swap 8 and 4 → [7, 1, -9, 4, 8, 13]
```

Pass 3

```
[7, 1, -9, 4, 8, 13]
 → Swap 7 and 1 → [1, 7, -9, 4, 8, 13]
 → Swap 7 and -9 → [1, -9, 7, 4, 8, 13]
 → Swap 7 and 4 → [1, -9, 4, 7, 8, 13]
```

Pass 4

```
[1, -9, 4, 7, 8, 13]
 → Swap 1 and -9 → [-9, 1, 4, 7, 8, 13]
```

Pass 5

```
No swaps → Sorting complete.
Final list: [-9, 1, 4, 7, 8, 13]
```

Algorithm 5.1: Bubble Sort

```
BUBBLESORT(numList, n)
1. Set i = 0
2. While i < n repeat:
3.   Set j = 0
4.   While j < n - i - 1 repeat:
5.     If numList[j] > numList[j+1], then
6.       Swap numList[j] and numList[j+1]
7.     j = j + 1
8. i = i + 1
```

Python Program

```
def bubble_Sort(list1):
    n = len(list1)
    for i in range(n):
```

```
        for j in range(0, n-i-1):
            if list1[j] > list1[j+1]:
                list1[j], list1[j+1] = list1[j+1], list1[j]

numList = [8, 7, 13, 1, -9, 4]
bubble_Sort(numList)
print("The sorted list is:")
for i in range(len(numList)):
    print(numList[i], end=" ")
```

5.3 Selection Sort

Concept

- The list is divided into **sorted and unsorted** parts.
- The **smallest** element from the unsorted list is selected and **swapped** with the first unsorted element.
- Requires **n – 1 passes** for n elements.

Diagram: Selection Sort Passes Starting List: `[8, 7, 13, 1, -9, 4]`

Pass 1 - Smallest is `-9` → Swap with `8`
`[-9, 7, 13, 1, 8, 4]`

Pass 2 - Smallest in [7,13,1,8,4] is `1` → Swap with `7`
`[-9, 1, 13, 7, 8, 4]`

Pass 3 - Smallest in [13,7,8,4] is `4` → Swap with `13`
`[-9, 1, 4, 7, 8, 13]`

Pass 4 - Smallest in [7,8,13] is `7` (already in place)
`[-9, 1, 4, 7, 8, 13]`

Pass 5 - Already sorted
`[-9, 1, 4, 7, 8, 13]`

Algorithm 5.2: Selection Sort

```
SELECTIONSORT(numList, n)
1. Set i = 0
2. While i < n repeat:
3.    Set min = i, flag = 0
4.    Set j = i + 1
5.    While j < n repeat:
```

```
6.      If numList[j] < numList[min]:
7.          min = j
8.          flag = 1
9.    If flag == 1:
10.     Swap numList[i], numList[min]
11.   i = i + 1
```

Python Program

```
def selection_Sort(list2):
    n = len(list2)
    for i in range(n):
        min = i
        for j in range(i + 1, n):
            if list2[j] < list2[min]:
                min = j
        list2[i], list2[min] = list2[min], list2[i]

numList = [8, 7, 13, 1, -9, 4]
selection_Sort(numList)
print("The sorted list is:")
for i in range(len(numList)):
    print(numList[i], end=" ")
```

5.4 Insertion Sort

Concept

- Elements from the unsorted part are picked and **inserted** into the correct position of the sorted part.
- Like arranging cards in hand.

Diagram: Insertion Sort Passes Initial List: `[8, 7, 13, 1, -9, 4]`

Pass 1 Insert 7 into sorted [8]
`[7, 8, 13, 1, -9, 4]`

Pass 2 Insert 13 into sorted [7,8]
`[7, 8, 13, 1, -9, 4]`

Pass 3 Insert 1 into sorted [7,8,13]
`[1, 7, 8, 13, -9, 4]`

Pass 4 Insert -9 into sorted [1,7,8,13]
`[-9, 1, 7, 8, 13, 4]`

Pass 5 Insert 4 into sorted [-9,1,7,8,13]
`[-9, 1, 4, 7, 8, 13]`

Algorithm 5.3: Insertion Sort

```
INSERTIONSORT(numList, n)
1. Set i = 1
2. While i < n repeat:
3.   temp = numList[i]
4.   Set j = i - 1
5.   While j >= 0 and numList[j] > temp:
6.       numList[j+1] = numList[j]
7.       j = j - 1
8.   numList[j+1] = temp
9.   i = i + 1
```

Python Program

```
def insertion_Sort(list3):
    n = len(list3)
    for i in range(n):
        temp = list3[i]
        j = i-1
        while j >= 0 and temp < list3[j]:
            list3[j+1] = list3[j]
            j -= 1
        list3[j+1] = temp

numList = [8, 7, 13, 1, -9, 4]
insertion_Sort(numList)
print("The sorted list is:")
for i in range(len(numList)):
    print(numList[i], end=" ")
```

5.5 Time Complexity of Algorithms

Basic Complexity Concepts

- **Constant Time (O(1))**: No loops.
- **Linear Time (O(n))**: Single loop.
- **Quadratic Time ($O(n^2)$)**: Nested loops.

All three sorting algorithms in this chapter (bubble, selection, insertion) have **O(n²)** complexity due to nested loops.

Summary Points

- **Sorting**: Rearranging elements for efficient access.
- **Bubble Sort**: Repeated adjacent swaps, n-1 passes.
- **Selection Sort**: Select and place smallest, reduce unsorted list.
- **Insertion Sort**: Insert each element into sorted part at the correct position.
- **Time Complexity**: Helps choose suitable algorithms for large datasets.

MULTIPLE CHOICE QUESTIONS (MCQs)

1. **Which of the following sorting techniques is based on repeatedly comparing adjacent elements and swapping them if they are in the wrong order?**

A) Selection Sort

B) Bubble Sort

C) Insertion Sort

D) Merge Sort

Answer: B) Bubble Sort

2. **What is the time complexity of Bubble Sort in the worst-case scenario?**

A) O(n)

B) O(log n)

C) O(n²)

D) O(n log n)

Answer: C) O(n²)

3. **Which of the following best describes the logic of the Bubble Sort algorithm?**

A) Select the smallest element and place it in the beginning.

B) Insert the current element into its correct position in a sorted list.

C) Swap adjacent elements until the largest reaches the end.

D) Divide the list and merge sorted parts.

Answer: C) Swap adjacent elements until the largest reaches the end.

4. **How many passes are required to sort a list of 6 elements using Bubble Sort?**

A) 5

B) 6

C) 4

D) 3

Answer: A) 5

5. **Which sorting technique finds the smallest element in the list and places it in its correct position?**

A) Insertion Sort

B) Bubble Sort

C) Selection Sort

D) Quick Sort

Answer: C) Selection Sort

6. **In Selection Sort, after the first pass, the first element of the list is:**

A) The largest element

B) The average of the list

C) The smallest element

D) Unchanged

Answer: C) The smallest element

7. **What is the time complexity of Selection Sort in the best case?**

A) O(n log n)

B) $O(n^2)$

C) O(n)

D) O(log n)

Answer: B) $O(n^2)$

8. **Which sorting technique is best described as "a sorted list is built one element at a time"?**

A) Merge Sort

B) Insertion Sort

C) Bubble Sort

D) Selection Sort

Answer: B) Insertion Sort

9. **In Insertion Sort, which of the following is TRUE about the sorted portion of the list?**

A) It decreases in size

B) It is built from left to right

C) It always remains constant

D) It is sorted in reverse order

Answer: B) It is built from left to right

10. **What is the best-case time complexity of Insertion Sort?**

A) O(n log n)

B) O(n)

C) $O(n^2)$

D) O(1)

Answer: B) O(n)

11. **Which sorting algorithm is generally better for nearly sorted data?**

A) Selection Sort

B) Bubble Sort

C) Insertion Sort

D) Heap Sort

Answer: C) Insertion Sort

12. **In Bubble Sort, which element reaches its correct position after the first pass?**

A) The smallest element

B) The middle element

C) The largest element

D) The second largest element

Answer: C) The largest element

13. **Which sorting algorithm swaps elements only when necessary and avoids unnecessary movements?**

A) Bubble Sort

B) Insertion Sort

C) Selection Sort

D) None of the above

Answer: C) Selection Sort

14. **Which of the following algorithms uses a temporary variable during swapping?**

A) Bubble Sort

B) Selection Sort

C) Insertion Sort

D) All of the above

Answer: D) All of the above

15. **Which sort algorithm is** not** efficient for large datasets?**

A) Quick Sort

B) Insertion Sort

C) Merge Sort

D) Heap Sort

Answer: B) Insertion Sort

16. **In the Insertion Sort algorithm, when is shifting required?**

A) When the current element is greater than the previous one

B) When the current element is smaller than elements in the sorted list

C) When elements are equal

D) Always

Answer: B) When the current element is smaller than elements in the sorted list

17. **The number of comparisons in the worst case for Bubble Sort is:**

A) n

B) n-1

C) n(n-1)/2

D) log n

Answer: C) n(n-1)/2

18. **Which sorting algorithm is described using the following pseudocode?**

```
For i = 0 to n-2:
    For j = 0 to n-2-i:
        If arr[j] > arr[j+1]:
            Swap arr[j] and arr[j+1]
```

A) Insertion Sort

B) Bubble Sort

C) Selection Sort

D) Quick Sort

Answer: B) Bubble Sort

19. **Which sorting method has the same time complexity in best, average, and worst cases?**

A) Selection Sort

B) Insertion Sort

C) Merge Sort

D) Bubble Sort

Answer: A) Selection Sort

20. **In Selection Sort, if the minimum element is already at its correct position, what happens?**

A) The list is reversed

B) The algorithm stops

C) No swap is made

D) Swap still happens

Answer: C) No swap is made

21.
Assertion (A): Bubble Sort repeatedly swaps adjacent elements if they are in the wrong order.
Reason (R): Bubble Sort pushes the smallest element to the end in each pass.

A. Both A and R are true, and R is the correct explanation of A.
B. Both A and R are true, but R is not the correct explanation of A.
C. A is true, but R is false.
D. A is false, but R is true.

Answer: C. A is true, but R is false.
Explanation: Bubble Sort pushes the **largest** element to the end of the list in each pass, not the smallest.

22.
Assertion (A): In Selection Sort, the smallest element is selected in each pass and placed in its correct position.
Reason (R): Selection Sort uses repeated shifting of elements to sort the list.

A. Both A and R are true, and R is the correct explanation of A.
B. Both A and R are true, but R is not the correct explanation of A.
C. A is true, but R is false.
D. A is false, but R is true.

Answer: C. A is true, but R is false.
Explanation: Selection Sort works by swapping, not shifting elements.

23.
Assertion (A): Insertion Sort is efficient for nearly sorted lists.
Reason (R): In the best case, the time complexity of Insertion Sort is O(n).

A. Both A and R are true, and R is the correct explanation of A.
B. Both A and R are true, but R is not the correct explanation of A.
C. A is true, but R is false.
D. A is false, but R is true.

Answer: A. Both A and R are true, and R is the correct explanation of A.

24.
Assertion (A): Selection Sort is a stable sorting algorithm.

Reason (R): Selection Sort preserves the order of identical elements.

A. Both A and R are true, and R is the correct explanation of A.
B. Both A and R are true, but R is not the correct explanation of A.
C. A is false, but R is true.
D. Both A and R are false.

Answer: D. Both A and R are false.
Explanation: Selection Sort is **not** stable by default and may change the relative order of identical elements.

25.
Assertion (A): Insertion Sort builds the final sorted list one item at a time.
Reason (R): It inserts each new element at the beginning of the list.

A. Both A and R are true, and R is the correct explanation of A.
B. Both A and R are true, but R is not the correct explanation of A.
C. A is true, but R is false.
D. A is false, but R is true.

Answer: C. A is true, but R is false.
Explanation: Insertion Sort inserts elements into the correct position **within** the sorted portion, not necessarily at the beginning.

26.
Assertion (A): Bubble Sort can be optimized to stop early if no swaps are made during a pass.
Reason (R): This optimization improves its best-case time complexity to O(n).

A. Both A and R are true, and R is the correct explanation of A.
B. Both A and R are true, but R is not the correct explanation of A.
C. A is true, but R is false.
D. A is false, but R is true.

Answer: A. Both A and R are true, and R is the correct explanation of A.

27.
Assertion (A): In Bubble Sort, every pair of adjacent elements is compared in each pass.
Reason (R): The number of comparisons increases with each pass.

A. Both A and R are true, and R is the correct explanation of A.
B. Both A and R are true, but R is not the correct explanation of A.
C. A is true, but R is false.
D. A is false, but R is true.

Answer: C. A is true, but R is false.
Explanation: The number of comparisons **decreases** with each pass in Bubble Sort.

28.
Assertion (A): In Selection Sort, the number of comparisons is always the same regardless of the input order.
Reason (R): Selection Sort has a time complexity of $O(n^2)$ for all cases.

A. Both A and R are true, and R is the correct explanation of A.
B. Both A and R are true, but R is not the correct explanation of A.
C. A is true, but R is false.
D. A is false, but R is true.

Answer: A. Both A and R are true, and R is the correct explanation of A.

29.
Assertion (A): In Insertion Sort, shifting elements is necessary to maintain the sorted portion of the list.
Reason (R): Elements are shifted right to make space for the current element.

A. Both A and R are true, and R is the correct explanation of A.
B. Both A and R are true, but R is not the correct explanation of A.
C. A is true, but R is false.
D. A is false, but R is true.

Answer: A. Both A and R are true, and R is the correct explanation of A.

30.
Assertion (A): All three sorting techniques (Bubble, Selection, Insertion) have a worst-case time complexity of $O(n^2)$.
Reason (R): They all use nested loops with two iterations each.

A. Both A and R are true, and R is the correct explanation of A.
B. Both A and R are true, but R is not the correct explanation of A.
C. A is true, but R is false.
D. A is false, but R is true.

Answer: A. Both A and R are true, and R is the correct explanation of A.

FILL IN THE BLANKS

1. Sorting is the process of arranging elements in a particular _____.
Answer: order

2. In Bubble Sort, after each pass, the _____ element gets placed at its correct position.
Answer: largest

3. In Bubble Sort, the process of comparing and swapping adjacent elements is repeated for _____ - 1 passes.
Answer: n

4. If no _____ are made during a pass in Bubble Sort, the list is already sorted.
Answer: swaps

5. In Selection Sort, the list is conceptually divided into a _____ list and an unsorted list.
Answer: sorted

6. In each pass of Selection Sort, the _____ element from the unsorted portion is selected and placed in the sorted list.
Answer: smallest

7. Selection Sort requires a total of _____ - 1 passes to sort a list of n elements.
Answer: n

8. In Insertion Sort, each element from the unsorted list is _____ into its correct position in the sorted part.
Answer: inserted

9. In Insertion Sort, shifting of elements occurs from _____ to find the correct position of insertion.
Answer: right to left

10. In the best-case scenario, Insertion Sort has a time complexity of _____.
Answer: O(n)

11. The time complexity of Bubble, Selection, and Insertion Sort in the worst case is _____.
Answer: $O(n^2)$

12. A _____ loop is responsible for increasing the time complexity of sorting algorithms to quadratic.
Answer: nested

13. Sorting helps to improve the efficiency of _____ operations on large datasets.
Answer: searching

14. In Python, a pair of adjacent elements in Bubble Sort can be swapped using the syntax:
`list[j], list[j+1] = __________`
Answer: list[j+1], list[j]

15. The algorithm that builds the sorted list one element at a time by comparing and shifting is _____ Sort.
Answer: Insertion

2 MARKS QUESTIONS

1. What is sorting? Give one real-life example.

Answer:
Sorting is the process of arranging or ordering a given collection of elements in a particular order, such as ascending or descending.
Example: Words in a dictionary are sorted in alphabetical order to make searching easier.

2. Define pass and swap in the context of Bubble Sort.

Answer:
A **pass** refers to one complete iteration through the list during sorting.
A **swap** is the process of exchanging the positions of two adjacent elements if they are not in the desired order.

3. Write the syntax of swapping two elements in Python with an example.

Answer:
Syntax:

```
a, b = b, a
```

Example:

```
list[j], list[j+1] = list[j+1], list[j]
```

This swaps the elements at index `j` and `j+1` in a list.

4. Differentiate between Bubble Sort and Selection Sort (any two points).

Answer:

Bubble Sort	Selection Sort
Repeatedly compares and swaps adjacent elements.	Selects the smallest element and swaps it with the leftmost unsorted element.
Pushes the largest element to the end in each pass.	Places the smallest element at the beginning in each pass.

5. What is the purpose of using a `flag` in Selection Sort algorithm?

Answer:
The `flag` is used to check whether a smaller element than the current minimum was found in the current pass. If `flag = 1`, a swap is needed; otherwise, no swap is performed.

6. Give one example where Insertion Sort is more efficient than other sorts. Explain why.

Answer:
Insertion Sort is more efficient for **nearly sorted lists** because it requires fewer comparisons and shifts, giving it a best-case time complexity of **O(n)**.

7. Write the initial list and final sorted list after applying Insertion Sort on: `[8, 7, 13, 1, -9, 4]`.

Answer:
Initial List: [8, 7, 13, 1, -9, 4]
Final Sorted List: [-9, 1, 4, 7, 8, 13]

8. Distinguish between shifting and swapping in the context of sorting.

Answer:
Shifting means moving elements one position to the right to make space (used in Insertion Sort).
Swapping means exchanging two elements' positions directly (used in Bubble and Selection Sort).

9. Explain time complexity with an example from the chapter.

Answer:
Time complexity is the amount of time an algorithm takes to run, depending on input size.
Example: Bubble Sort has **$O(n^2)$** time complexity because it uses a nested loop to perform $n \times (n - 1)/2$ comparisons.

10. Write the output of the following Bubble Sort implementation on the list `[8, 7, 13, 1, -9, 4]`.
Also, mention how many passes are needed.

Answer:
Output: `[-9, 1, 4, 7, 8, 13]`
Passes Needed: 5 (as the list contains 6 elements)

3 MARKS QUESTIONS

1. Explain the concept of Bubble Sort with an example.

Answer:
Bubble Sort compares adjacent elements and swaps them if they are in the wrong order. After each pass, the largest element is moved to its correct position at the end. This process is repeated for $n-1$ passes for a list of n elements.

Example:
For `numList = [8, 7, 13, 1, -9, 4]`, after Bubble Sort, the list becomes `[-9, 1, 4, 7, 8, 13]`.

2. Differentiate between Bubble Sort and Insertion Sort (any three points).

Answer:

Bubble Sort	Insertion Sort
Compares and swaps adjacent elements.	Inserts each element at the correct position in sorted part.
Time complexity: $O(n^2)$ in all cases.	Best case time complexity: O(n).
Poor for nearly sorted data.	Efficient for nearly sorted data.

3. Write the Python implementation of Selection Sort and explain it briefly.

Answer:

```
def selection_Sort(list2):
    n = len(list2)
    for i in range(n):
        min = i
        for j in range(i + 1, n):
            if list2[j] < list2[min]:
                min = j
        list2[i], list2[min] = list2[min], list2[i]
```

Explanation:
The algorithm selects the smallest element from the unsorted list and swaps it with the element at the current position. This process continues until the list is sorted.

4. Define the term 'pass' in sorting. How many passes are required to sort a list of 6 elements using Selection Sort?

Answer:
A **pass** is one complete traversal over the list during the sorting process.
In Selection Sort, for a list of **n = 6 elements**, the total number of passes required is **n – 1 = 5 passes**.

5. How does Insertion Sort work? Explain with an example.

Answer:
Insertion Sort builds a sorted list one element at a time. Each new element is compared with elements in the sorted list and inserted into its correct position by shifting larger elements to the right.

Example:
`[8, 7, 13, 1, -9, 4]` becomes `[-9, 1, 4, 7, 8, 13]` after applying Insertion Sort.

6. Write the algorithm for Bubble Sort and explain its steps.

Answer:
Algorithm:

```
1. Set i = 0
2. While i < n, repeat steps 3-8
3. Set j = 0
4. While j < n - i - 1, repeat steps 5-7
5. If list[j] > list[j+1]
6. Swap list[j] and list[j+1]
7. Increment j
8. Increment i
```

Explanation:
The algorithm performs nested loops to compare and swap adjacent elements so that the largest unsorted element moves to its correct position after each pass.

7. What is time complexity? Explain with respect to Bubble, Selection, and Insertion Sort.

Answer:
Time complexity is the amount of time an algorithm takes to complete based on input size.
In Bubble, Selection, and Insertion Sort, all three have **nested loops**, hence their worst-case time complexity is **O(n²)**.
Insertion Sort, however, can have a best-case complexity of **O(n)** if the list is already nearly sorted.

8. Differentiate between swapping and shifting with suitable examples from the sorting algorithms.

Answer:

Swapping	Shifting
Used in Bubble and Selection Sort.	Used in Insertion Sort.
Exchanges the positions of two elements.	Moves elements one position to the right.
Example: `list[j], list[j+1] = list[j+1], list[j]`	Example: `list[j+1] = list[j]`

5 MARKS QUESTIONS

1. Explain the working of Bubble Sort with the help of an example. Show all passes and swaps.

Answer:

Bubble Sort works by comparing adjacent elements and swapping them if they are in the wrong order. The largest element "bubbles up" to the end after each pass.

Example: numList = [8, 7, 13, 1, -9, 4]

Pass 1:

[8, 7, 13, 1, -9, 4] → [7, 8, 13, 1, -9, 4] → [7, 8, 1, 13, -9, 4] → [7, 8, 1, -9, 13, 4] → [7, 8, 1, -9, 4, 13]

Pass 2:

[7, 8, 1, -9, 4, 13] → [7, 1, 8, -9, 4, 13] → [7, 1, -9, 8, 4, 13] → [7, 1, -9, 4, 8, 13]

Pass 3:

[7, 1, -9, 4, 8, 13] → [1, 7, -9, 4, 8, 13] → [1, -9, 7, 4, 8, 13] → [1, -9, 4, 7, 8, 13]

Pass 4:

[1, -9, 4, 7, 8, 13] → [-9, 1, 4, 7, 8, 13]

Pass 5: No swap → Already sorted

Final list: [-9, 1, 4, 7, 8, 13]

2. Write the algorithm and Python implementation of Bubble Sort.

Answer:

Algorithm 5.1: Bubble Sort

```
1. Set i = 0
2. WHILE i < n repeat:
   - Set j = 0
   - WHILE j < n - i - 1:
     - IF numList[j] > numList[j+1]:
       - Swap numList[j], numList[j+1]
     - j = j + 1
   - i = i + 1
```

Python Program:

```
def bubble_Sort(list1):
    n = len(list1)
    for i in range(n):
        for j in range(0, n - i - 1):
            if list1[j] > list1[j+1]:
                list1[j], list1[j+1] = list1[j+1], list1[j]
```

3. Explain the working of Selection Sort using an example. Show all passes.

Answer:

Selection Sort selects the smallest element from the unsorted portion and swaps it with the first element in the unsorted list.

Example: numList = [8, 7, 13, 1, -9, 4]

Pass 1:

Minimum is -9 → Swap with 8 → [-9, 7, 13, 1, 8, 4]

Pass 2:

Minimum is 1 → Swap with 7 → [-9, 1, 13, 7, 8, 4]

Pass 3:

Minimum is 4 → Swap with 13 → [-9, 1, 4, 7, 8, 13]

Pass 4:

Already sorted → [-9, 1, 4, 7, 8, 13]

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

4. Write the algorithm and Python program for Selection Sort.

Answer:

Algorithm 5.2: Selection Sort

```
1. Set i = 0
2. WHILE i < n repeat:
   - Set min = i, flag = 0
   - Set j = i + 1
   - WHILE j < n repeat:
     - IF numList[j] < numList[min]:
       - min = j, flag = 1
     - j = j + 1
   - IF flag = 1 → Swap numList[i], numList[min]
   - i = i + 1
```

Python Program:

```
def selection_Sort(list2):
    n = len(list2)
```

```
    for i in range(n):
        min = i
        for j in range(i+1, n):
            if list2[j] < list2[min]:
                min = j
        list2[i], list2[min] = list2[min], list2[i]
```

5. What is Insertion Sort? Explain its working with an example and diagrammatically.

Answer:

Insertion Sort builds a sorted list one element at a time. Each new element is inserted into its correct position in the sorted list by shifting larger elements to the right.

Example: numList = [8, 7, 13, 1, -9, 4]

Pass 1: [7, 8, 13, 1, -9, 4]
Pass 2: [7, 8, 13, 1, -9, 4]
Pass 3: [1, 7, 8, 13, -9, 4]
Pass 4: [-9, 1, 7, 8, 13, 4]
Pass 5: [-9, 1, 4, 7, 8, 13]

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

6. Write the algorithm and Python implementation of Insertion Sort.

Answer:

Algorithm 5.3: Insertion Sort

```
1. Set i = 1
2. WHILE i < n repeat:
   - temp = numList[i]
   - j = i - 1
   - WHILE j  0 and numList[j] > temp repeat:
     - numList[j+1] = numList[j]
     - j = j - 1
   - numList[j+1] = temp
   - i = i + 1
```

Python Program:

```
def insertion_Sort(list3):
    n = len(list3)
    for i in range(n):
```

```
temp = list3[i]
j = i - 1
while j >= 0 and temp < list3[j]:
    list3[j+1] = list3[j]
    j -= 1
list3[j+1] = temp
```

7. Compare Bubble Sort, Selection Sort, and Insertion Sort based on working, time complexity, and best use cases.

Answer:

Criteria	Bubble Sort	Selection Sort	Insertion Sort
Basic Operation	Swaps adjacent elements	Selects minimum and swaps	Inserts in sorted portion
Time Complexity	$O(n^2)$	$O(n^2)$	$O(n^2)$, best case $O(n)$
Best Use Case	Small lists	Small or fixed-size lists	Nearly sorted lists
Stable?	Yes	No	Yes

8. What is time complexity? Describe the complexity of the three sorting algorithms discussed in the chapter.

Answer:
Time complexity is a measure of the amount of time an algorithm takes based on input size.

- **Bubble Sort:** $O(n^2)$ – Nested loop for comparisons and swaps
- **Selection Sort:** $O(n^2)$ – Regardless of input, comparisons are same
- **Insertion Sort:** $O(n^2)$ in worst case, $O(n)$ in best case (nearly sorted data)

All three algorithms are **quadratic time algorithms** due to nested loops.

9. Give the syntax and explanation of swapping and shifting with examples.

Answer:

Swapping Syntax in Python:

```
list[j], list[j+1] = list[j+1], list[j]
```

Example: In Bubble/Selection Sort → swaps adjacent or selected elements.

Shifting Syntax:

```
list[j+1] = list[j]
```

Example: In Insertion Sort → elements are moved one place right to insert a smaller element in the correct position.

10. Explain with an example how you can use Bubble Sort to find the median of a list.

Answer:

Step 1: Sort the list using Bubble Sort.

Step 2: If number of terms is odd, median is the middle term.

Step 3: If number is even, median = (term[n//2 – 1] + term[n//2]) / 2

Example:

List = [4, 1, 5, 3] → After Bubble Sort: [1, 3, 4, 5]

Median = (3 + 4)/2 = **3.5**

11. What is the significance of pass number in all sorting algorithms? Explain with examples.

Answer:

A **pass** is a complete iteration over the list.

- In **Bubble Sort**, each pass places the largest unsorted element at its correct position.
- In **Selection Sort**, each pass selects and places the next smallest element.
- In **Insertion Sort**, each pass inserts the next element into the correct position in the sorted list.

Example for Bubble Sort (n = 6): 5 passes are needed.

12. Explain how sorting can be useful in real-world applications with examples.

Answer:

Sorting simplifies and speeds up data retrieval and comparison.

Examples:

- Dictionary words sorted alphabetically
- Roll numbers arranged in order in an exam hall
- Student data sorted by height, weight, or scores
- Percentile calculation in aptitude tests using sorting

Sorting is essential before binary search or when evaluating ranks/statistics.

CHAPTER END EXERCISES

1. Consider a list of 10 elements:

```
numList = [7, 11, 3, 10, 17, 23, 1, 4, 21, 5]
```

Display the partially sorted list after three complete passes of Bubble sort.

Answer:
After applying 3 passes of **Bubble Sort**, we get:

- **Pass 1:** [7, 3, 10, 11, 17, 1, 4, 21, 5, 23]
- **Pass 2:** [3, 7, 10, 11, 1, 4, 17, 5, 21, 23]
- **Pass 3:** [3, 7, 10, 1, 4, 11, 5, 17, 21, 23]

Partially Sorted List: [3, 7, 10, 1, 4, 11, 5, 17, 21, 23]

2. Identify the number of swaps required for sorting the following list using selection sort and bubble sort and identify which is the better sorting technique with respect to the number of comparisons.
`List 1: [63, 42, 21, 9]`

Answer:

- **Bubble Sort:**
  - **Swaps:**
    - Pass 1: 63↔42, 63↔21, 63↔9 → 3 swaps
    - Pass 2: 42↔21, 42↔9 → 2 swaps
    - Pass 3: 21↔9 → 1 swap
    - **Total Swaps:** 6
- **Selection Sort:**
  - Pass 1: Swap 9↔63
  - Pass 2: Swap 21↔42
  - Pass 3: Already sorted
  - **Total Swaps:** 2

Conclusion: Selection Sort performs fewer swaps and is better in this case.

3. Consider the following lists:
`List 1: [2, 3, 5, 7, 11]`

`List 2: [11, 7, 5, 3, 2]`

If the lists are sorted using Insertion Sort then which of the lists (List 1 or List 2) will make the minimum number of comparisons? Justify using diagrammatic representation.

Answer:

- **List 1** is already sorted → Best-case scenario
- Comparisons per pass: 0 (or 1 minimal comparison per element)
- Time complexity: O(n)

- **List 2** is reverse sorted → Worst-case scenario
  - Maximum comparisons and shifts per pass
  - Time complexity: $O(n^2)$

Conclusion: List 1 will make the **minimum number of comparisons** as it represents the best-case input for Insertion Sort.

4. Write a program using user-defined functions that accepts a list of numbers as an argument and finds its median.

(Hint: Use bubble sort to sort the accepted list. If there are an odd number of terms, the median is the center term. If even, add the two middle terms and divide by 2 to get the median.)

Answer:

```
def bubble_sort(lst):
    n = len(lst)
    for i in range(n):
        for j in range(0, n-i-1):
            if lst[j] > lst[j+1]:
                lst[j], lst[j+1] = lst[j+1], lst[j]

def find_median(lst):
    bubble_sort(lst)
    n = len(lst)
    if n % 2 == 1:
        return lst[n//2]
    else:
        return (lst[n//2 - 1] + lst[n//2]) / 2

# Example
nums = [4, 2, 7, 3]
print("Median is:", find_median(nums))
```

Output: `Median is: 3.5`

5. All the branches of XYZ school conducted an aptitude test for all the students in the age group 14 - 16. There were a total of n students. The marks of n students are stored in a list. Write a program using a user-defined function that accepts a list of marks as an argument and calculates the 'xth' percentile (where x is any number between 0 and 100).

Steps to calculate the xth percentile:
I. Order all values using Selection Sort
II. Calculate index as: `x% of n`
III. Use `math.ceil()` or `math.floor()` if necessary
IV. Return the value at that index

Answer:

```
import math

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

def calculate_percentile(marks, x):
    selection_sort(marks)
    index = math.ceil((x / 100) * len(marks)) - 1
    return marks[index]

# Example
marks_list = [56, 78, 45, 90, 82, 67]
x = 90
print("90th Percentile is:", calculate_percentile(marks_list, x))
```

Output: `90th Percentile is: 90`

6. During admission in a course, the names of the students are inserted in ascending order. Thus, performing the sorting operation at the time of inserting elements in a list. Identify the type of sorting technique being used and write a program using a user-defined function that is invoked every time a name is input and stores the name in ascending order of names in the list.

Answer:
Type of Sorting Technique: Insertion Sort

Python Program:

```
def insert_in_order(name_list, new_name):
    name_list.append(new_name)
    i = len(name_list) - 1
    while i > 0 and name_list[i] < name_list[i-1]:
        name_list[i], name_list[i-1] = name_list[i-1], name_list[i]
        i -= 1
    return name_list

# Example
names = []
names = insert_in_order(names, "Zara")
names = insert_in_order(names, "Amit")
names = insert_in_order(names, "Meera")
print("Sorted Names:", names)
```

Output: `Sorted Names: ['Amit', 'Meera', 'Zara']`

For more information Visit:

https://matheenhere.blogspot.com