FREE EDITION · NOTES + 3 SAMPLE MCQS

CUET · COMPUTER SCIENCE · CLASS XII · CODE 308

Sorting

CUET unit: Sorting

By **UniDrill** · NCERT-grounded study material

WWW.UNIDRILL.IN

📌 Snapshot

- Sorting is the process of arranging a collection of elements in a particular order; it is fundamental to efficient searching and data management.
- Three sorting algorithms come with step-by-step traces and Python implementations: Bubble Sort (§5.2), Selection Sort (§5.3), and Insertion Sort (§5.4).
- Time Complexity (§5.5) covers constant, linear, and quadratic complexity, explained through the three sorting programs.
- Algorithm design, tracing passes, and complexity classification are core Class XII topics appearing in both direct and application-based MCQ formats, so CUET tests them.
- All three algorithms have the same time complexity of n^2 (quadratic), a fact that NTA frequently uses as a distractor or direct question.

📖 Detailed Notes

2.1 Core concepts

- **What is sorting?** Sorting is the process of ordering or arranging a given collection of elements in some particular order — ascending (increasing), descending (decreasing), or alphabetical. Real-life examples include words in a dictionary (alphabetical), seats in an exam hall (roll number), students by height or weight. (NCERT §5.1, p. 67)
- **Why sort?** Sorting a large collection has an upfront overhead, but that overhead is worth it because finding an element in a sorted list is far faster than searching an unsorted one. Sorting is an important area of study in computer science and many algorithms have been developed and analysed from a performance perspective. (NCERT §5.1, p. 68)
- **Bubble Sort — principle:** Sorts by repeatedly comparing adjacent elements and swapping them if they are unordered. Each full traversal through the list is called a pass. For a list of n elements, bubble sort makes n−1 passes. After each pass the largest element "bubbles up" to its correct position at the end and is excluded from subsequent passes. (NCERT §5.2, p. 68)
- **Bubble Sort — passes on numList = [8, 7, 13, 1, −9, 4]:** Pass 1 places 13 at the end; Pass 2 places 8; Pass 3 places 7; Pass 4 places 4; Pass 5 places –9 and 1. The

list is fully sorted after at most n−1 = 5 passes. Figure 5.1 illustrates all comparisons. (NCERT §5.2, p. 69)

- **Bubble Sort — Algorithm 5.1:** BUBBLESORT(numList, n) uses two nested loops — outer loop index i from 0 to n−1, inner loop index j from 0 to n−i−2 — comparing numList[j] with numList[j+1] and swapping if numList[j] > numList[j+1]. The inner range shrinks by one each pass because the last i elements are already sorted. (NCERT §5.2, p. 70)
- **Bubble Sort — Python (Program 5-1):** Implemented using `bubble_Sort(list1)` with nested for-loops. The inner loop runs `range(0, n-i-1)`. Swap is done with simultaneous assignment: `list1[j], list1[j+1] = list1[j+1], list1[j]`. Output for numList = [8,7,13,1,–9,4] is –9 1 4 7 8 13. (NCERT §5.2, p. 70)
- **Selection Sort — principle:** Makes n−1 passes. The list is conceptually divided into a sorted left part and an unsorted right part. In each pass, the smallest element is found from the unsorted part by traversing it fully, then swapped with the leftmost element of the unsorted part; that element joins the sorted part. The unsorted part shrinks by one each pass. (NCERT §5.3, p. 71)
- **Selection Sort — Algorithm 5.2:** SELECTIONSORT(numList, n) sets `min = i` and `flag = 0` at the start of each outer pass (i from 0 to n−1). An inner loop (j from i+1 to n−1) finds the index of the minimum element (updates min and sets flag=1). If flag=1, swaps numList[i] with numList[min]. (NCERT §5.3, p. 73)
- **Selection Sort — Python (Program 5-2):** Implemented using `selection_Sort(list2)`. A `flag` variable controls whether a swap is needed. Swap: `list2[min], list2[i] = list2[i], list2[min]`. Output for same numList is –9 1 4 7 8 13. (NCERT §5.3, p. 74)
- **Insertion Sort — principle:** Divides the list into a sorted part (initially one element) and an unsorted part. In each pass, the first element of the unsorted part (element e) is compared with elements of the sorted part from right to left; sorted elements larger than e are shifted right to make space, and e is inserted at its correct position. (NCERT §5.4, p. 74)
- **Insertion Sort — pass details:** Pass 1: sorted list has 1 element, unsorted has n−1. Pass 2 reduces unsorted to n−2, and so on. Figure 5.3 shows all comparisons for numList = [8,7,13,1,–9,4]. (NCERT §5.4, p. 75)
- **Insertion Sort — Algorithm 5.3:** INSERTIONSORT(numList, n) starts i=1. For each i, sets `temp = numList[i]` and `j = i-1`. While j⩾0 and numList[j]>temp, shifts numList[j+1] = numList[j] and decrements j. Finally sets numList[j+1] = temp (inserts temp at correct position). (NCERT §5.4, p. 76)
- **Insertion Sort — Python (Program 5-3):** Implemented using `insertion_Sort(list3)`. The while-loop condition is `j >= 0 and temp < list3[j]`. Output: –9 1 4 7 8 13. (NCERT §5.4, p. 77)
- **Time Complexity — introduction:** The amount of time an algorithm takes to process given data is its time complexity. For small datasets the differences between

algorithms are negligible, but for huge real-world datasets time utilisation becomes significant and must be analysed before choosing an algorithm. (NCERT §5.5, p. 77)

- **Time Complexity — classification rules:** (a) No loop → complexity = 1 (Constant time). (b) One loop (1 to n) → complexity = n (Linear time). (c) Loop within a loop (nested loop) → complexity = n^2 (Quadratic time). (d) If a nested loop and a single loop coexist, complexity is estimated based on the nested loop. (NCERT §5.5, p. 78)
- **Time Complexity of the three sorts:** All three Python programs (Bubble, Selection, Insertion) contain a nested loop (one loop inside another), so all three have time complexity of n^2. (NCERT §5.5, p. 78)

2.2 Definitions to memorise

Term	Definition	Page
Sorting	The process of placing or rearranging a collection of elements into a particular order (ascending, descending, or alphabetical).	67
Pass	One full iteration/traversal through the list (or relevant portion of it) during a sorting algorithm.	68
Swap	Changing the positions of two elements with each other.	68
Bubble Sort	A sorting algorithm that repeatedly compares and swaps adjacent unordered elements; the largest element "bubbles" to the end in each pass.	68
Selection Sort	A sorting algorithm that in each pass selects the smallest element from the unsorted part and swaps it with the leftmost element of the unsorted part.	71
Insertion Sort	A sorting algorithm that inserts each element of the unsorted part into its correct position in the sorted part by shifting larger elements right.	74
Time Complexity	The amount of time an algorithm takes to process a given data set; a measure of algorithm efficiency as input size grows.	77
Constant Time Algorithm	An algorithm with no loops; executes a fixed number of instructions regardless of data size; complexity = 1.	78
Linear Time Algorithm	An algorithm with a single loop (1 to n); complexity = n.	78
Quadratic Time Algorithm	An algorithm with a nested loop (loop within a loop); complexity = n^2.	78
Ascending order	Arranging from smallest to largest	67
Descending order	Arranging from largest to smallest	67
Flag (Selection Sort)	Boolean used to indicate whether a swap is necessary in the current pass	73

Term	Definition	Page
Outer loop	The slower-iterating loop in a nested-loop algorithm; controls passes	70
Inner loop	The faster-iterating loop in a nested-loop algorithm; performs comparisons	70
Sorted part / Unsorted part	Conceptual division of the list used in Selection and Insertion sorts	71
Big-O notation	Standard notation for asymptotic complexity (n, n², log n, ⋯); NCERT introduces the concept implicitly via "complexity"	77-78
Adjacent comparison	Comparison between immediate neighbours — characteristic of Bubble Sort	68
Minimum search	Per-pass operation in Selection Sort that scans the unsorted part for the smallest value	71
Shift operation	Movement of larger elements to the right in Insertion Sort to make space for `temp`	74
n−1 passes	Number of outer iterations needed by all three sorts on an n-element list	68, 71, 75

2.3 Diagrams / processes to remember

- **Figure 5.1 — Bubble Sort passes (p. 69):** Shows 5 passes on numList = [8,7,13,1,−9,4]. Blue cells = elements being compared/swapped; green cells = already sorted. After Pass 1: 13 is at index 5. After Pass 5: list is fully sorted as [−9,1,4,7,8,13]. Key observation: Pass 5 has only one comparison, confirming the inner loop range is n−i−1.
- **Figure 5.2 — Selection Sort passes (p. 72):** Shows 5 passes on the same numList. Arrows indicate elements being compared; blue = smaller element found; green = sorted portion. After Pass 1: −9 moves to index 0. Only one swap happens per pass (at most).
- **Figure 5.3 — Insertion Sort passes (p. 75):** Shows how each new element is inserted into the growing sorted sublist. All elements in the sorted part that are larger than the current element shift one position to the right. The sorted portion grows by one element per pass.
- **Algorithm 5.1 (Bubble Sort, p. 70), Algorithm 5.2 (Selection Sort, p. 73), Algorithm 5.3 (Insertion Sort, p. 76):** Pseudocode is given for all three; each uses two nested loops. Inner loop bounds differ: Bubble uses j < n−i−1; Selection uses j starting at i+1; Insertion uses a while-loop traversing backwards.

2.4 Common confusions / NTA trap points

- **Number of passes vs. number of comparisons:** Bubble sort makes n−1 passes but the total comparisons across all passes is n(n−1)/2. NTA may ask about passes (n−1) or comparisons; do not confuse them.
- **Bubble vs. Insertion — both use swapping, but differently:** In Bubble Sort, adjacent elements are swapped immediately if out of order. In Insertion Sort, elements are shifted (not swapped) to make room, and then the key element is placed once. NTA often tests this distinction.
- **Selection Sort — at most ONE swap per pass:** Selection Sort finds the minimum index first and only then swaps once (if flag=1). Bubble Sort can make multiple swaps in a single pass. This is a frequent distractor.
- **All three algorithms have the same time complexity n^2:** Students often think Insertion or Selection is "faster" in terms of Big-O. For CUET purposes all three are n^2 (quadratic). NTA is known to use "which has better time complexity?" as a trap — correct answer: all are the same (n^2).
- **Inner loop upper bound in Bubble Sort (NCERT Algorithm 5.1, p. 70).** The inner loop runs from j = 0 to j < n−i−1, not j < n−1. Forgetting to subtract i means you re-compare already-sorted elements.
- **Sorted output is independent of algorithm (NCERT §5.2-5.4, pp. 70-77).** All three produce `[-9, 1, 4, 7, 8, 13]` on the example numList — the only difference is the path taken.
- **n^2 for all three is for worst case as much as average (NCERT §5.5, p. 78).** NCERT keeps the discussion simple — students should not over-claim "Insertion sort is O(n) when already sorted" for CUET unless explicitly asked.
- **Selection sort: only one swap per pass (NCERT §5.3, p. 71).** Bubble may swap multiple times per pass.
- **Sort algorithms can sort numbers, strings, or any orderable type.** NCERT examples use numbers, but the same logic applies to alphabetical order.
- **Time complexity is independent of language (NCERT §5.5, p. 77).** It depends on the algorithm structure (loops), not on Python vs. C++.
- **Empty or 1-element list is already sorted (NCERT implicit).** All three algorithms make 0 passes on such inputs.

🎯 Practice MCQs

Q1. Which of the following best describes the process performed by Bubble Sort?

A. In each pass, the smallest element is selected and placed at its correct position.

B. In each pass, adjacent elements are compared and swapped if they are unordered, causing the largest element to move to the end.

C. In each pass, each unsorted element is inserted at its correct position in the sorted sublist.

D. In each pass, the list is divided into halves and each half is sorted separately.

Q2. For a list of n = 6 elements, how many passes does Bubble Sort make in total?

A. 6

B. 4

C. 5

D. 3

Q3. Consider the following statements about Selection Sort: **Statement 1:** In each pass, the smallest element from the unsorted part is found and swapped with the leftmost element of the unsorted part. **Statement 2:** Selection Sort can make more than one swap in a single pass. Which of the following is correct?

A. Both Statement 1 and Statement 2 are true.

B. Statement 1 is true, Statement 2 is false.

C. Statement 1 is false, Statement 2 is true.

D. Both Statement 1 and Statement 2 are false.

PYQ Alignment

Sorting algorithms and time complexity appear regularly in CUET Computer Science papers, with questions typically targeting algorithm tracing (state of list after k passes), identification of the correct algorithm from a description or pseudocode step, and classification of time complexity. Match-the-following and assertion–reason formats are especially common for this topic. See PYQ archive for Computer Science.