

UNIT NOTES

Sorting and Searching Algorithms

Data Structures — exam notes

SUBJECT

Data Structures

CLASS / COURSE

B.Tech / BCA • Semester 3

PREPARED BY

Subject Teacher

How to Use These Notes

These notes cover searching, elementary and efficient sorting algorithms, and their complexity. Each chapter starts with a short overview, then lists the key definitions, explains the core concepts with examples, and ends with practice questions. Answers to the multiple-choice questions are at the end.

Study plan

- Read the overview and definitions first
- Work through the solved examples without looking at the answers
- Attempt the practice questions in timed conditions
- Revise using the summary sheet the night before the exam

Contents

Unit 1: Searching.....	4
Key Definitions.....	4
Binary Search.....	4
Solved Examples.....	4
Practice Questions.....	4
Unit 2: Elementary Sorting.....	6
Key Definitions.....	6
How They Work.....	6
Solved Examples.....	6
Practice Questions.....	6
Unit 3: Efficient Sorting.....	8
Key Definitions.....	8
Merge Sort.....	8
Quick Sort.....	8
Solved Examples.....	8
Practice Questions.....	8
Quick Revision Sheet.....	10
Answer Key.....	11

Unit 1: Searching

Searching locates an element in a collection. Linear search works on any list; binary search is much faster but needs the list to be sorted.

Key Definitions

Term	Meaning
Linear search	Checks each element in turn until a match is found
Binary search	Repeatedly halves a sorted list around the middle element
Time complexity	How running time grows with input size n
Big-O	Upper bound on growth, e.g. $O(\log n)$

Binary Search

Compare the key with the middle element. If equal, stop; if smaller, search the left half; otherwise search the right half. Each step halves the search space, giving $O(\log n)$ time.

- Requires sorted data
- Iterative or recursive
- Worst case: $\lfloor \log_2 n \rfloor + 1$ comparisons

Solved Examples

Q1. How many comparisons does binary search need in the worst case for 1,000 elements?

$\lfloor \log_2 1000 \rfloor + 1 = 10$ comparisons.

Q2. When is linear search preferred?

For small or unsorted lists, or when the list changes often and sorting would cost more than searching.

Exam tip

Trace binary search in a table with low, high and mid columns — examiners look for the trace, not just the answer.

Practice Questions

A. Multiple choice

1. Binary search requires

(a) Unsorted data (b) Sorted data (c) Linked list (d) Hash table

2. Worst-case time of linear search

(a) $O(1)$ (b) $O(\log n)$ (c) $O(n)$ (d) $O(n^2)$

3. Worst-case time of binary search

- (a) $O(n)$ (b) $O(\log n)$ (c) $O(n \log n)$ (d) $O(1)$

4. Binary search is an example of

- (a) Greedy (b) Divide and conquer (c) Dynamic programming (d) Backtracking

B. Short answer

1. Write the algorithm for linear search.
2. Why must data be sorted for binary search?
3. Define time complexity.

C. Long answer

4. Write and trace binary search for the key 47 in [5, 12, 19, 23, 31, 47, 56, 72].
5. Compare linear and binary search on time complexity, requirements and use cases.

Unit 2: Elementary Sorting

Bubble, selection and insertion sort are simple to write and understand, and they take $O(n^2)$ time in the worst case. Insertion sort is efficient for nearly sorted data.

Key Definitions

Term	Meaning
Stable sort	Equal elements keep their original relative order
In-place sort	Uses only a constant amount of extra memory
Pass	One complete traversal of the list
Adaptive	Runs faster on partially sorted input

How They Work

Bubble sort repeatedly swaps adjacent elements that are out of order. Selection sort selects the minimum of the unsorted part and places it at the front. Insertion sort takes the next element and inserts it into its correct place in the sorted part.

Algorithm	Best	Average	Worst	Stable	In-place
Bubble	$O(n)$	$O(n^2)$	$O(n^2)$	Yes	Yes
Selection	$O(n^2)$	$O(n^2)$	$O(n^2)$	No	Yes
Insertion	$O(n)$	$O(n^2)$	$O(n^2)$	Yes	Yes

Table: Elementary sorts compared

Solved Examples

Q1. Show the passes of bubble sort on [5, 1, 4, 2].

Pass 1: [1, 4, 2, 5]; pass 2: [1, 2, 4, 5]; pass 3: no swaps — sorted.

Q2. Why is insertion sort good for nearly sorted data?

Each element moves only a few positions, so the inner loop runs very few times — close to $O(n)$.

Exam tip

Mention stability and in-place properties in comparison questions; they are easy marks often missed.

Practice Questions

A. Multiple choice

1. Worst-case time of bubble sort

(a) $O(n)$ (b) $O(n \log n)$ (c) $O(n^2)$ (d) $O(\log n)$

2. Which is not stable by default?

(a) Bubble (b) Insertion (c) Selection (d) Merge

3. Best case of insertion sort

(a) $O(n)$ (b) $O(n^2)$ (c) $O(\log n)$ (d) $O(1)$

4. Number of passes in bubble sort for n elements (worst)

(a) n (b) $n - 1$ (c) $n/2$ (d) $\log n$

B. Short answer

6. What is a stable sort?

7. Write the selection sort algorithm.

8. Why is bubble sort rarely used in practice?

C. Long answer

9. Explain insertion sort with an example and analyse its time complexity.

10. Compare bubble, selection and insertion sort.

Unit 3: Efficient Sorting

Merge sort and quick sort use divide and conquer to sort in $O(n \log n)$ time on average. Merge sort guarantees this bound; quick sort is usually faster in practice.

Key Definitions

Term	Meaning
Divide and conquer	Split the problem, solve the parts, combine the results
Pivot	Element used by quick sort to partition the list
Partition	Rearranging elements around the pivot
Recurrence	Equation describing running time in terms of smaller inputs

Merge Sort

Split the list into halves until each part has one element, then merge pairs of sorted lists. The recurrence $T(n) = 2T(n/2) + n$ gives $O(n \log n)$. It needs $O(n)$ extra memory.

Quick Sort

Choose a pivot, partition so that smaller elements go left and larger go right, then sort both parts recursively. Average time is $O(n \log n)$; a poor pivot on sorted input can give $O(n^2)$.

- Random or median-of-three pivot avoids the worst case
- In-place, but not stable

Solved Examples

Q1. Why is quick sort $O(n^2)$ on already sorted data with the first element as pivot?

Each partition puts all remaining elements on one side, so recursion depth is n and total work is $n + (n-1) + \dots = O(n^2)$.

Q2. Solve $T(n) = 2T(n/2) + n$.

By the master theorem (case 2), $T(n) = O(n \log n)$.

Exam tip

For merge sort, draw the recursion tree; for quick sort, show one full partition step with the pivot boxed.

Practice Questions

A. Multiple choice

1. Merge sort worst case

(a) $O(n^2)$ (b) $O(n \log n)$ (c) $O(n)$ (d) $O(\log n)$

2. Quick sort worst case

(a) $O(n \log n)$ (b) $O(n^2)$ (c) $O(n)$ (d) $O(1)$

3. Extra space of merge sort

(a) $O(1)$ (b) $O(\log n)$ (c) $O(n)$ (d) $O(n^2)$

4. Quick sort is

(a) Stable (b) Not stable (c) Always $O(n \log n)$ (d) Not in-place

B. Short answer

11. What is a pivot?

12. Write the recurrence for merge sort.

13. Why is merge sort preferred for linked lists?

C. Long answer

14. Explain merge sort with an example and derive its time complexity.

15. Explain the partition procedure of quick sort and discuss its best and worst cases.

Quick Revision Sheet

Chapter	Term	Remember
Searching	Linear search	Checks each element in turn until a match is found
Searching	Binary search	Repeatedly halves a sorted list around the middle element
Searching	Time complexity	How running time grows with input size n
Elementary Sorting	Stable sort	Equal elements keep their original relative order
Elementary Sorting	In-place sort	Uses only a constant amount of extra memory
Elementary Sorting	Pass	One complete traversal of the list
Efficient Sorting	Divide and conquer	Split the problem, solve the parts, combine the results
Efficient Sorting	Pivot	Element used by quick sort to partition the list
Efficient Sorting	Partition	Rearranging elements around the pivot

Answer Key

Chapter	Question	Answer
1	1	b
1	2	c
1	3	b
1	4	b
2	1	c
2	2	c
2	3	a
2	4	b
3	1	b
3	2	b
3	3	c
3	4	b