



# Sorting and Searching in Java

**With Rohit Mazumder**

Let's crack Competitive Programming together!

# Lecture - 1

## Introduction to Sorting

---

1. Which of the following sorting algorithms in its typical implementation gives best performance when applied on an array which is almost sorted (1 or 2 elements are misplaced) ?


- A. Selection Sort
- B. Bubble Sort
- C. Insertion Sort
- D. None of the above

- A. Selection Sort
- B. Bubble Sort
-  C. Insertion Sort
- D. None of the above

Solution : Since insertion sort only does extra work for the elements out of place. It will be most efficient in this case.

2. What is the best case time complexity we can achieve in bubble sort?

- A.  $O(n^2)$
- B.  $O(n \log n)$
- C.  $O(n)$
- D.  $O(n^3)$

- A.  $O(n^2)$
- B.  $O(n \log n)$
-  C.  $O(n)$
- D.  $O(n^3)$

Solution : By using a boolean swap variable to break when the array is already sorted we can achieve  $O(n)$   
Otherwise it is  $O(n^2)$

3. Selection Sort never makes more than  $n$  swaps?

A. True

B. False


- A. True
- B. False

Solution : Because we only swap after finding the index of minimum element and at worst all the elements could be out of their place.



4. What is the best auxiliary space complexity a sorting algorithm can have?


- A.  $O(n)$
- B.  $O(1)$
- C.  $O(n \log n)$
- D.  $O(n^2)$

- A.  $O(n)$
-  B.  $O(1)$
- C.  $O(n \log n)$
- D.  $O(n^2)$

Solution : Bubble, insertion and selection sort have  $O(1)$  space complexity.

5. Consider the array  $\{4,3,5,2\}$  how many swaps will be needed to sort the array using selection sort?

- A. 1
- B. 2
- C. 3
- D. 4

- A. 1
-  B. 2
- C. 3
- D. 4

Solution : First we swap 2 and 4, then we swap 4 and 5. So we require 2 swaps.

# Lecture - 2

## Merge Sort

---


1. What is the maximum number of inversions possible in an array of length  $n$ ?

A.  $n^2$

B.  $n*(n-1)/2$

C.  $n*(n+1)/2$

D.  $n+1$

- A.  $n^2$
-  B.  $n*(n-1)/2$
- C.  $n*(n+1)/2$
- D.  $n+1$

Solution : Worst case the array is sorted in decreasing order.

2. Which of the following mentioned algorithms works on the divide and conquer paradigm?

- A. Merge Sort
- B. Selection Sort
- C. Bubble Sort
- D. Insertion Sort



- ✓ A. Merge Sort
- B. Selection Sort
- C. Bubble Sort
- D. Insertion Sort

Solution : Merge Sort works on divide and conquer as discussed in class.

3. You have to sort 1GB of data with only 100MB of main memory available. Which sorting technique will be most appropriate?


- A. Bubble Sort
- B. Merge Sort
- C. Selection Sort
- D. Insertion Sort

- A. Bubble Sort
-  B. Merge Sort
- C. Selection Sort
- D. Insertion Sort

Solution : Merge sort works for external sorting

4. What is the total number of levels in the recursion tree while using merge sort on an array of 8 elements?

- A. 1
- B. 2
- C. 3
- D. 4

- A. 1
- B. 2
- C. 3
-  D. 4

Solution : Level -1  $\Rightarrow$  8


Level-2  $\Rightarrow$  4+4

Level-3  $\Rightarrow$  2+2+2+2

Level-4  $\Rightarrow$  1+1+1+1+1+1+1+1

5. What is the worst case time complexity of merge sort?

- A.  $O(n)$
- B.  $O(n \log n)$
- C.  $O(n^2)$
- D.  $O(\sqrt{n})$

- A.  $O(n)$
-  B.  $O(n \log n)$
- C.  $O(n^2)$
- D.  $O(\sqrt{n})$

Solution : Derived from recurrence relation

$$T(n) = 2 * T(n/2) + O(n)$$

6. Which of the following algorithms works best for sorting arrays?

- A. Bubble Sort
- B. Selection Sort
- C. Merge Sort
- D. Insertion Sort



- A. Bubble Sort
- B. Selection Sort
-  C. Merge Sort
- D. Insertion Sort

Solution : Worst case for merge sort is  $O(n \log n)$  while for others it is  $O(n^2)$


# Lecture - 3

## Doubt Clearing Session

---

1. What is the recurrence relation for worst case of merge sort and the time complexity in worst case?

- A.  $T(n) = T(n-2) + O(n)$  ,  $O(n^2)$
- B.  $T(n) = 2 * T(n/2) + O(n)$  ,  $O(n \log n)$
- C.  $T(n) = 2 * T(n/2) + O(1)$  ,  $O(n^2)$
- D.  $T(n) = 2 * T(n/2) + O(n)$  ,  $O(n^2)$

- A.  $T(n) = T(n-2) + O(n)$  ,  $O(n^2)$
-  B.  $T(n) = 2 * T(n/2) + O(n)$  ,  $O(n \log n)$
- C.  $T(n) = 2 * T(n/2) + O(1)$  ,  $O(n^2)$
- D.  $T(n) = 2 * T(n/2) + O(n)$  ,  $O(n^2)$

Solution : We divide the array in 2 halves and do  $O(n)$  work at the merge step.

2. Which of the following is not a stable algorithm in its typical implementation?

- A. Insertion Sort
- B. Merge Sort
- C. Selection Sort
- D. Bubble Sort

- A. Insertion Sort
- B. Merge Sort
-  C. Selection Sort
- D. Bubble Sort

Solution : We just discussed the implementation for stable selection sort algo.

3. Which sorting algo will take least time when all elements are identical? Consider only typical implementation.

- A. Insertion Sort
- B. Merge Sort
- C. Selection Sort
- D. Bubble Sort


- ✓ A. Insertion Sort
- B. Merge Sort
- C. Selection Sort
- D. Bubble Sort

Solution : Since the array is sorted, insertion sort will work in  $O(n)$



4. A list of  $n$  strings each of length  $n$ , is sorted into lexicographic order using the merge sort algorithm. The worst case running time is?


- A.  $O(n \log n)$
- B.  $O(n \log^2 n)$
- C.  $O(n^2 \log n)$
- D.  $O(n^2 \log^2 n)$

- A.  $O(n \log n)$
- B.  $O(n \log^2 n)$
-  C.  $O(n^2 \log n)$
- D.  $O(n^2 \log^2 n)$

Solution : Everything is same as merge sort, except while comparing two elements in merge step we need  $O(n)$  time for strings.

5. Which of the following statements are false about merge sort?

- A. It is stable by nature
- B. It is an in-place algorithm
- C. It outperforms insertion sort in best case
- D. Both B and C

- A. It is stable by nature
- B. It is an in-place algorithm
- C. It outperforms insertion sort in best case
-  D. Both B and C

Solution : Merge sort is not in-place. And insertion sort takes  $O(n)$  in best case while merge sort takes  $O(n \log n)$


6. Given an array = {4,3,2,1}, how many minimum number of operations will be required to sort the array if you are only allowed to swap the adjacent elements in one operation.?

A. 2

B. 1

C. 4

D. 6

- A. 2
- B. 1
- C. 4
-  D. 6

Solution : This is equivalent to finding the number of inversions in an array which in this case is equal to  $n*(n-1)/2$