# DATA STRUCTURE

## Chapter 8

## Searching & Sorting

---

***Solved Problems – Exam Practice Set 4***

---

*For Engineering (B.Tech / Diploma) Students*

### Contents



*Note: This set uses a fresh collection of arrays and questions, different from Practice Sets 1, 2 and 3, for additional exam practice.*

## Section A – Sorting Algorithms

**Problem 1:** Sort the array A = [18, 6, 12, 3, 15] in ascending order using Bubble Sort. Show the array after every pass and find the total number of comparisons and swaps.

**Solution:**

```
Initial Array: 18, 6, 12, 3, 15

Pass 1 (4 comparisons):
  18,6  -> swap -> 6,18,12,3,15
  18,12 -> swap -> 6,12,18,3,15
  18,3  -> swap -> 6,12,3,18,15
  18,15 -> swap -> 6,12,3,15,18
  After Pass 1 : 6, 12, 3, 15, 18   (swaps = 4)

Pass 2 (3 comparisons):
  6,12 -> no swap
  12,3 -> swap -> 6,3,12,15,18
  12,15 -> no swap
  After Pass 2 : 6, 3, 12, 15, 18   (swaps = 1)

Pass 3 (2 comparisons):
  6,3 -> swap -> 3,6,12,15,18
  6,12 -> no swap
  After Pass 3 : 3, 6, 12, 15, 18   (swaps = 1)

Pass 4 (1 comparison):
  3,6 -> no swap
  No swap occurred -- array already sorted, algorithm terminates.
```

Total comparisons = 4 + 3 + 2 + 1 = 10

Total swaps = 4 + 1 + 1 + 0 = 6

**Final Answer: Sorted Array = 3, 6, 12, 15, 18 | Comparisons = 10, Swaps = 6**

**Problem 2:** Sort the array A = [81, 34, 2, 56, 17, 90] in ascending order using Selection Sort. Show the array after each pass.

**Solution:**

```
Initial Array: 81, 34, 2, 56, 17, 90

Pass 1: minimum = 2 (index 2)  -> swap A[0],A[2] -> 2,34,81,56,17,90
Pass 2: minimum = 17 (index 4) -> swap A[1],A[4] -> 2,17,81,56,34,90
Pass 3: minimum = 34 (index 4) -> swap A[2],A[4] -> 2,17,34,56,81,90
Pass 4: minimum = 56 (already in place) -> no swap -> 2,17,34,56,81,90
Pass 5: minimum = 81 (already in place) -> no swap -> 2,17,34,56,81,90
```

**Final Answer: Sorted Array = 2, 17, 34, 56, 81, 90**

**Problem 3:** Sort the array A = [70, 40, 90, 10, 60, 20] in ascending order using Insertion Sort. Show the array after each element is inserted.

**Solution:**

```
Initial Array: 70, 40, 90, 10, 60, 20

i=1, key=40: 70>40 shift; insert 40 at 0                -> 40,70,90,10,60,20
i=2, key=90: 70<90, no shift                             -> 40,70,90,10,60,20
i=3, key=10: 90>10,70>10,40>10 shift all; insert at 0    -> 10,40,70,90,60,20
i=4, key=60: 90>60,70>60 shift; 40<60 stop; insert@2     -> 10,40,60,70,90,20
i=5, key=20: 90>20,70>20,60>20,40>20 shift; 10<20 stop   -> 10,20,40,60,70,90
```

**Final Answer: Sorted Array = 10, 20, 40, 60, 70, 90**

**Problem 4:** Sort the array A = [56, 17, 89, 24, 61, 10, 42] in ascending order using Quick Sort. Take the last element as pivot and show the first partitioning step in detail, then give the final sorted array.

**Solution:**

**Initial Array: 56, 17, 89, 24, 61, 10, 42 (pivot = 42, the last element)**

```
i = -1
j=0: 56<=42? No
j=1: 17<=42  -> i=0, swap A[0],A[1]  -> 17,56,89,24,61,10,42
j=2: 89<=42? No
j=3: 24<=42  -> i=1, swap A[1],A[3]  -> 17,24,89,56,61,10,42
j=4: 61<=42? No
j=5: 10<=42  -> i=2, swap A[2],A[5]  -> 17,24,10,56,61,89,42

Place pivot: swap A[i+1] (A[3]) with A[high] (A[6])
                                      -> 17,24,10,42,61,89,56

Pivot 42 is now fixed at index 3.
Left sub-array  : [17,24,10] -> Quick Sort gives [10,17,24]
Right sub-array : [61,89,56] -> Quick Sort gives [56,61,89]
```

**Final Answer: Sorted Array = 10, 17, 24, 42, 56, 61, 89**

**Problem 5:** Sort the array A = [20, 18, 12, 9, 15, 6, 3, 24] in ascending order using Merge Sort. Show the divide and merge steps.

**Solution:**

```
DIVIDE:
[20,18,12,9,15,6,3,24]
   /                \
[20,18,12,9]        [15,6,3,24]
  /    \              /    \
[20,18][12,9]        [15,6][3,24]
 /  \   /  \          /  \   /  \
[20][18][12][9]      [15][6][3][24]

MERGE (combine back in sorted order):
[20][18] -> [18,20]      [12][9] -> [9,12]
[18,20][9,12] -> [9,12,18,20]
[15][6] -> [6,15]        [3][24] -> [3,24]
[6,15][3,24] -> [3,6,15,24]
[9,12,18,20][3,6,15,24] -> [3,6,9,12,15,18,20,24]
```

**Final Answer: Sorted Array = 3, 6, 9, 12, 15, 18, 20, 24**

**Problem 6:** Sort the array A = [29, 17, 5, 41, 23, 11] in ascending order using Shell Sort with gap sequence 3, then 1.

**Solution:**

```
Initial Array: 29, 17, 5, 41, 23, 11   (n = 6)

GAP = 3:
  i=3 (key=41): A[0]=29<=41, no shift
  i=4 (key=23): A[1]=17<=23, no shift
  i=5 (key=11): A[2]=5<=11, no shift
  After gap=3 (unchanged) : 29, 17, 5, 41, 23, 11

GAP = 1 (standard Insertion Sort):
  i=1 (key=17): 29>17 shift; insert@0              -> 17,29,5,41,23,11
  i=2 (key=5) : 29>5,17>5 shift; insert@0           -> 5,17,29,41,23,11
  i=3 (key=41): 29<41, no shift
  i=4 (key=23): 41>23,29>23 shift; 17<23 stop; ins@2 -> 5,17,23,29,41,11
  i=5 (key=11): 41>11,29>11,23>11,17>11 shift; 5<11  -> 5,11,17,23,29,41
```

**Final Answer: Sorted Array = 5, 11, 17, 23, 29, 41**

**Problem 7:** Sort the array A = [2, 8, 5, 3, 9, 1] in ascending order using Heap Sort. Show the Max Heap constructed and the extraction steps.

**Solution:**

```
Initial Array: 2, 8, 5, 3, 9, 1   (n = 6)

BUILD MAX HEAP (i from n/2-1 = 2 down to 0):
  i=2 (node=5, child=1): 5 already largest, no change
  i=1 (node=8, children 3,9): largest=9, swap A[1],A[4]
                              -> 2,9,5,3,8,1
  i=0 (node=2, children 9,5): largest=9, swap A[0],A[1]
                              -> 9,2,5,3,8,1
     heapify index1 (node=2, children 3,8): largest=8, swap
                              -> 9,8,5,3,2,1
  Max Heap = 9, 8, 5, 3, 2, 1

EXTRACTION:
  swap A[0],A[5] -> 1,8,5,3,2,9 ; heapify(size5) -> 8,3,5,1,2,9
  swap A[0],A[4] -> 2,3,5,1,8,9 ; heapify(size4) -> 5,3,2,1,8,9
  swap A[0],A[3] -> 1,3,2,5,8,9 ; heapify(size3) -> 3,1,2,5,8,9
  swap A[0],A[2] -> 2,1,3,5,8,9 ; heapify(size2) -> 2,1,3,5,8,9
  swap A[0],A[1] -> 1,2,3,5,8,9 ; heap size1, done
```

**Final Answer: Sorted Array = 1, 2, 3, 5, 8, 9**

**Problem 8:** Sort the array A = [123, 45, 6, 789, 234, 901, 12] in ascending order using Radix Sort (treat missing higher digits as 0).

**Solution:**

```
Initial Array: 123, 45, 6, 789, 234, 901, 12
Maximum number = 901 (3 digits), so 3 passes are required.

PASS 1 (Units digit):
  Buckets: 1:[901] 2:[12] 3:[123] 4:[234] 5:[45] 6:[6] 9:[789]
  Collected -> 901, 12, 123, 234, 45, 6, 789

PASS 2 (Tens digit):
  Buckets: 0:[901,6] 1:[12] 2:[123] 3:[234] 4:[45] 8:[789]
  Collected -> 901, 6, 12, 123, 234, 45, 789

PASS 3 (Hundreds digit):
  Buckets: 0:[6,12,45] 1:[123] 2:[234] 7:[789] 9:[901]
  Collected -> 6, 12, 45, 123, 234, 789, 901
```

**Final Answer: Sorted Array = 6, 12, 45, 123, 234, 789, 901**

**Problem 9:** Sort the array A = [55, 28, 68, 12, 40, 60, 80] in ascending order using Binary Tree Sort. Show the Binary Search Tree formed and the traversal used.

**Solution:**

```
Insert elements in the given order 55,28,68,12,40,60,80 into a BST:

                55
               /  \
             28     68
            /  \   /  \
          12   40 60   80

Perform IN-ORDER traversal (Left - Root - Right):
  Visit 12 -> 28 -> 40 -> 55 -> 60 -> 68 -> 80
```

**Final Answer: Sorted Array = 12, 28, 40, 55, 60, 68, 80**

## Section B – Searching (Binary Search)

**Problem 10:** Using Binary Search, find the key = 48 in the sorted array A = [6, 11, 17, 23, 29, 35, 42, 48, 53, 61, 68, 75, 82]. Show low, high and mid at every step and the number of comparisons made.

**Solution:**

```
Array (index 0-12): 6,11,17,23,29,35,42,48,53,61,68,75,82

Step 1: low=0, high=12, mid=6 -> A[6]=42
        42 < 48  => low = 7

Step 2: low=7, high=12, mid=9 -> A[9]=61
        61 > 48  => high = 8

Step 3: low=7, high=8, mid=7 -> A[7]=48
        48 == 48 => KEY FOUND at index 7
```

**Final Answer: Key 48 found at index 7 | Number of comparisons = 3**

**Problem 11:** Using Binary Search on the same array A = [6, 11, 17, 23, 29, 35, 42, 48, 53, 61, 68, 75, 82], search for key = 25, which is not present. Show all the steps and the result.

**Solution:**

```
Step 1: low=0, high=12, mid=6 -> A[6]=42
        42 > 25  => high = 5

Step 2: low=0, high=5, mid=2 -> A[2]=17
        17 < 25  => low = 3

Step 3: low=3, high=5, mid=4 -> A[4]=29
        29 > 25  => high = 3

Step 4: low=3, high=3, mid=3 -> A[3]=23
        23 < 25  => low = 4

Now low(4) > high(3)  => SEARCH UNSUCCESSFUL
```

**Final Answer: Key 25 is not present in the array (search returns -1); comparisons made = 4**

## Section C – Time Complexity & Analytical Problems

**Problem 12:** A sorted array contains n = 4096 elements. Find the maximum number of comparisons required to search for an element using Binary Search in the worst case.

**Solution:**

Worst-case comparisons for Binary Search = $\lfloor \log_2 n \rfloor + 1$

Since $2^{12} = 4096$, $\log_2(4096) = 12$

Maximum comparisons = 12

**Final Answer: Maximum comparisons required = 12**

**Problem 13:** Find the total number of comparisons made by Selection Sort for an array of n = 9 elements.

**Solution:**

Selection Sort always scans the entire unsorted part in every pass, regardless of the arrangement of elements.

Total comparisons = $(n-1) + (n-2) + \ldots + 1 = n(n-1) / 2$

For n = 9: Total comparisons = $9 \times 8 / 2 = 72 / 2 = 36$

**Final Answer: Total comparisons = 36 (same in best, average and worst case)**

**Problem 14:** An array of n = 2000 elements is already sorted in ascending order. Estimate the number of comparisons Insertion Sort will make while processing this array.

**Solution:**

Best-case time complexity of Insertion Sort is O(n), which occurs precisely when the array is already sorted.

In this case, each of the (n-1) new keys is compared exactly once with its immediate predecessor before it is found to be already in the correct place.

Number of comparisons = n - 1 = 2000 - 1 = 1999

**Final Answer: Approximate comparisons ≈ 1999 (best-case, O(n) behaviour)**

**Problem 15:** Merge Sort takes approximately 5 seconds to sort n = 1,000,000 ($10^6$) elements. Assuming the running time follows $T(n) = c \cdot n \log_2 n$, estimate how long it will take to sort n = 4,000,000 ($4 \times 10^6$) elements.

**Solution:**

$\log_2(10^6) \approx 19.93$ and $\log_2(4 \times 10^6) = \log_2 4 + \log_2(10^6) = 2 + 19.93 = 21.93$

Ratio of running times = $[n_2 \log_2 n_2] / [n_1 \log_2 n_1] = (4 \times 10^6 \times 21.93) / (10^6 \times 19.93) = (4 \times 21.93) / 19.93 \approx 4.4$

Estimated time for $n_2 = 5 \times 4.4 \approx 22$ seconds

**Final Answer: Estimated time ≈ 22 seconds**

**Problem 16:** Prove that Selection Sort never performs more than (n – 1) swaps while sorting an array of n elements, regardless of the initial arrangement. Explain why this property makes Selection Sort useful in certain practical situations.

**Solution:**

Selection Sort works in (n – 1) passes. In each pass, it scans the unsorted part of the array to find the minimum element and performs AT MOST ONE swap – moving that minimum into its correct position at the start of the unsorted part.

If the minimum element found is already at the correct position (i.e. min_index equals the current pass index), no swap is required at all for that pass.

Since there are exactly (n-1) passes and each contributes at most one swap, the total number of swaps can never exceed (n-1), regardless of whether the input array is sorted, reverse sorted or in random order.

This is in sharp contrast to the number of comparisons, which always remains $O(n^2)$ irrespective of arrangement.

---

**Final Answer: Maximum swaps = (n-1). This makes Selection Sort useful when swapping/moving elements is expensive (e.g. large records or data on slow storage), since it minimises the number of write/move operations even though it does not minimise comparisons.**