DATA STRUCTURE

Chapter 8

Searching & Sorting

Solved Problems – Exam Practice Set

For Engineering (B.Tech / Diploma) Students

Contents

Section A – Sorting Algorithms

Problem 1: Sort the array A = [35, 10, 55, 15, 25] in ascending order using Bubble Sort. Show the array after every pass and find the total number of comparisons and swaps made.

Solution:

Initial Array: 35, 10, 55, 15, 25

```
Pass 1 (4 comparisons):
  35,10 -> swap -> 10,35,55,15,25
  35,55 -> no swap
  55,15 -> swap -> 10,35,15,55,25
  55,25 -> swap -> 10,35,15,25,55
  After Pass 1 : 10, 35, 15, 25, 55   (swaps = 3)

Pass 2 (3 comparisons):
  10,35 -> no swap
  35,15 -> swap -> 10,15,35,25,55
  35,25 -> swap -> 10,15,25,35,55
  After Pass 2 : 10, 15, 25, 35, 55   (swaps = 2)

Pass 3 (2 comparisons):
  10,15 -> no swap
  15,25 -> no swap
  After Pass 3 : 10, 15, 25, 35, 55   (swaps = 0)

No swap occurred in Pass 3, so the (optimised) algorithm
terminates early -- the array is already sorted.
```

Total comparisons = 4 + 3 + 2 = 9

Total swaps = 3 + 2 + 0 = 5

Final Answer: Sorted Array = 10, 15, 25, 35, 55 | Comparisons = 9, Swaps = 5

Problem 2: Sort the array A = [29, 10, 14, 37, 13] in ascending order using Selection Sort. Show the array after each pass.

Solution:

```
Initial Array: 29, 10, 14, 37, 13

Pass 1: Unsorted part [29,10,14,37,13] -> minimum = 10 (index 1)
        swap A[0] and A[1] -> 10, 29, 14, 37, 13

Pass 2: Unsorted part [29,14,37,13] -> minimum = 13 (index 4)
        swap A[1] and A[4] -> 10, 13, 14, 37, 29

Pass 3: Unsorted part [14,37,29] -> minimum = 14 (already in place)
        no swap needed      -> 10, 13, 14, 37, 29

Pass 4: Unsorted part [37,29] -> minimum = 29 (index 4)
        swap A[3] and A[4] -> 10, 13, 14, 29, 37
```

Final Answer: Sorted Array = 10, 13, 14, 29, 37

Problem 3: Sort the array A = [12, 11, 13, 5, 6] in ascending order using Insertion Sort. Show the array after each element is inserted into its correct position.

Solution:

```
Initial Array: 12, 11, 13, 5, 6

i=1, key=11 : 12>11, shift 12 right; insert 11 at index 0
              -> 11, 12, 13, 5, 6
i=2, key=13 : 12<13, no shift; 13 stays in place
              -> 11, 12, 13, 5, 6
i=3, key=5  : 13>5,12>5,11>5 shift all three; insert 5 at index 0
              -> 5, 11, 12, 13, 6
i=4, key=6  : 13>6,12>6,11>6 shift; 5<6 stop; insert 6 at index 1
              -> 5, 6, 11, 12, 13
```

Final Answer: Sorted Array = 5, 6, 11, 12, 13

Problem 4: Sort the array A = [10, 80, 30, 90, 40, 50, 70] in ascending order using Quick Sort. Take the last element as the pivot and show the first partitioning step in detail, then give the final sorted array.

Solution:

Initial Array: 10, 80, 30, 90, 40, 50, 70 (pivot = 70, the last element)

```
i = -1
j=0: 10<=70  -> i=0, swap A[0],A[0]      -> 10,80,30,90,40,50,70
j=1: 80<=70? No
j=2: 30<=70  -> i=1, swap A[1],A[2]      -> 10,30,80,90,40,50,70
j=3: 90<=70? No
j=4: 40<=70  -> i=2, swap A[2],A[4]      -> 10,30,40,90,80,50,70
j=5: 50<=70  -> i=3, swap A[3],A[5]      -> 10,30,40,50,80,90,70

Place pivot: swap A[i+1] (A[4]) with A[high] (A[6])
                                          -> 10,30,40,50,70,90,80

Pivot 70 is now fixed at index 4.
Left sub-array  : [10,30,40,50]  (already sorted)
Right sub-array : [90,80] -> Quick Sort gives [80,90]
```

Final Answer: Sorted Array = 10, 30, 40, 50, 70, 80, 90

Problem 5: Sort the array A = [38, 27, 43, 3, 9, 82, 10] in ascending order using Merge Sort. Show the divide and merge steps.

Solution:

```
DIVIDE:
[38,27,43,3,9,82,10]
   /                \
[38,27,43,3]        [9,82,10]
  /      \            /     \
[38,27] [43,3]      [9,82]  [10]
 /  \    /  \        /  \
[38][27][43][3]    [9][82]
```

```
MERGE (combine back in sorted order):
[38][27]      -> [27,38]
[43][3]       -> [3,43]
[27,38][3,43] -> [3,27,38,43]
[9][82]       -> [9,82]
[9,82][10]    -> [9,10,82]
[3,27,38,43][9,10,82] -> [3,9,10,27,38,43,82]
```

Final Answer: Sorted Array = 3, 9, 10, 27, 38, 43, 82

Problem 6: Sort the array A = [12, 34, 54, 2, 3] in ascending order using Shell Sort with initial gap = n/2, halving the gap after each pass.

Solution:

```
Initial Array: 12, 34, 54, 2, 3   (n = 5)

GAP = 2:
  i=2 (key=54): compare A[0]=12 -> 12<=54, no shift
  i=3 (key=2) : compare A[1]=34>2 shift -> A[3]=34
                j=-1 stop -> insert 2 at A[1]
                -> 12, 2, 54, 34, 3
  i=4 (key=3) : compare A[2]=54>3 shift -> A[4]=54
                compare A[0]=12>3 shift -> A[2]=12
                j=-2 stop -> insert 3 at A[0]
                -> 3, 2, 12, 34, 54
  After gap=2 : 3, 2, 12, 34, 54

GAP = 1 (standard Insertion Sort):
  i=1 (key=2) : A[0]=3>2 shift; insert 2 at A[0]  -> 2, 3, 12, 34, 54
  i=2 (key=12): A[1]=3<12, no shift
  i=3 (key=34): A[2]=12<34, no shift
  i=4 (key=54): A[3]=34<54, no shift
```

Final Answer: Sorted Array = 2, 3, 12, 34, 54

Problem 7: Sort the array A = [4, 10, 3, 5, 1] in ascending order using Heap Sort. Show the Max Heap constructed and the extraction steps.

Solution:

```
Initial Array: 4, 10, 3, 5, 1   (n = 5)

BUILD MAX HEAP (i from n/2-1 = 1 down to 0):
  i=1 (node=10, children 5,1): 10 already largest, no change
  i=0 (node=4, children 10,3): largest=10, swap A[0],A[1]
                               -> 10,4,3,5,1
     heapify index1 (node=4, children 5,1): largest=5, swap
                               -> 10,5,3,4,1
  Max Heap = 10, 5, 3, 4, 1

EXTRACTION:
  swap A[0],A[4] -> 1,5,3,4,10 ; heapify(size4) -> 5,4,3,1,10
  swap A[0],A[3] -> 1,4,3,5,10 ; heapify(size3) -> 4,1,3,5,10
  swap A[0],A[2] -> 3,1,4,5,10 ; heapify(size2) -> 3,1,4,5,10
  swap A[0],A[1] -> 1,3,4,5,10 ; heap size1, done
```

Final Answer: Sorted Array = 1, 3, 4, 5, 10

Problem 8: Sort the array A = [170, 45, 75, 90, 802, 24, 2, 66] in ascending order using Radix Sort.

Solution:

```
Initial Array: 170, 45, 75, 90, 802, 24, 2, 66
Maximum number = 802 (3 digits), so 3 passes are required.

PASS 1 (Units digit):
  Buckets: 0:[170,90] 2:[802,2] 4:[24] 5:[45,75] 6:[66]
  Collected -> 170, 90, 802, 2, 24, 45, 75, 66

PASS 2 (Tens digit):
  Buckets: 0:[802,2] 2:[24] 4:[45] 6:[66] 7:[170,75] 9:[90]
  Collected -> 802, 2, 24, 45, 66, 170, 75, 90

PASS 3 (Hundreds digit):
  Buckets: 0:[2,24,45,66,75,90] 1:[170] 8:[802]
  Collected -> 2, 24, 45, 66, 75, 90, 170, 802
```

Final Answer: Sorted Array = 2, 24, 45, 66, 75, 90, 170, 802

Problem 9: Sort the array A = [50, 30, 70, 20, 40, 90, 10] in ascending order using Binary Tree Sort. Show the Binary Search Tree formed and the traversal used to extract the sorted sequence.

Solution:

```
Insert elements in the given order 50,30,70,20,40,90,10 into a BST:

                50
               /  \
             30     70
            /  \      \
          20   40      90
          /
        10

Perform IN-ORDER traversal (Left - Root - Right):
  Visit 10 -> 20 -> 30 -> 40 -> 50 -> 70 -> 90
```

Final Answer: Sorted Array = 10, 20, 30, 40, 50, 70, 90

Section B – Searching (Binary Search)

Problem 10: Using Binary Search, find the key = 47 in the sorted array A = [3, 8, 15, 22, 34, 47, 56, 63, 72, 81]. Show low, high and mid at every step and give the number of comparisons made.

Solution:

```
Array (index 0-9): 3,8,15,22,34,47,56,63,72,81

Step 1: low=0, high=9, mid=(0+9)/2=4 -> A[4]=34
        34 < 47  => search right half; low = 5

Step 2: low=5, high=9, mid=(5+9)/2=7 -> A[7]=63
        63 > 47  => search left half; high = 6

Step 3: low=5, high=6, mid=(5+6)/2=5 -> A[5]=47
        47 == 47 => KEY FOUND at index 5
```

Final Answer: Key 47 found at index 5 | Number of comparisons = 3

Problem 11: Using Binary Search on the same array A = [3, 8, 15, 22, 34, 47, 56, 63, 72, 81], search for key = 50, which is not present in the array. Show all the steps and the result.

Solution:

```
Step 1: low=0, high=9, mid=4 -> A[4]=34
        34 < 50  => low = 5

Step 2: low=5, high=9, mid=7 -> A[7]=63
        63 > 50  => high = 6

Step 3: low=5, high=6, mid=5 -> A[5]=47
        47 < 50  => low = 6

Step 4: low=6, high=6, mid=6 -> A[6]=56
        56 > 50  => high = 5

Now low(6) > high(5)  => SEARCH UNSUCCESSFUL
```

Final Answer: Key 50 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 = 1024 elements. Find the maximum number of comparisons required to search for an element using (a) Binary Search and (b) Linear Search, in the worst case.

Solution:

(a) Binary Search: worst-case comparisons = $\lfloor \log_2 n \rfloor + 1 = \log_2(1024) + 1$

Since $2^{10} = 1024$, $\log_2(1024) = 10$ => maximum comparisons = 10

(b) Linear Search: worst-case comparisons = n = 1024 (the key may be the last element or absent)

Final Answer: Binary Search = 10 comparisons (worst case); Linear Search = 1024 comparisons (worst case)

Problem 13: Find the total number of comparisons made by Bubble Sort in the worst case for an array of n = 6 elements.

Solution:

In the worst case, Bubble Sort compares every pair in every pass without early termination.

Total comparisons = (n-1) + (n-2) + ... + 1 = n(n-1) / 2

For n = 6: Total comparisons = 6 × 5 / 2 = 30 / 2 = 15

Final Answer: Total worst-case comparisons = 15

Problem 14: An array of n = 5 elements is already sorted in ascending order. If Quick Sort always chooses the first element as the pivot, find the total number of comparisons made in the worst case.

Solution:

Since the array is already sorted and the pivot is always the first (smallest remaining) element, every partition step produces one empty sub-array and one sub-array of size (current size – 1).

This is the worst case for Quick Sort, and the number of comparisons follows the same pattern as Bubble Sort's worst case:

Total comparisons = (n-1) + (n-2) + (n-3) + (n-4) = 4 + 3 + 2 + 1 = 10

Final Answer: Total worst-case comparisons = 10, giving overall worst-case time complexity $O(n^2)$

Problem 15: Find the total number of comparisons made by Selection Sort for an array of n = 7 elements. Does this number depend on the initial arrangement of the array?

Solution:

Selection Sort always scans the entire unsorted part of the array in every pass to find the minimum element, regardless of how the elements are arranged.

Total comparisons = (n-1) + (n-2) + ... + 1 = n(n-1) / 2

For n = 7: Total comparisons = 7 × 6 / 2 = 42 / 2 = 21

Final Answer: Total comparisons = 21 in every case (best, average and worst) – it does NOT depend on the initial arrangement

Problem 16: Compare Merge Sort and Quick Sort for sorting a very large data set of n = 100000 elements that is already almost sorted, in terms of expected time complexity, and justify which one would be preferred.

Solution:

Quick Sort: if the pivot is chosen as the first or last element, an almost-sorted array drives Quick Sort close to its worst case, $O(n^2)$, because partitions become highly unbalanced (one side nearly empty).

Merge Sort: its performance is O(n log n) in the best, average and worst cases regardless of the initial arrangement of the data, since it always divides the array into two equal halves.

For n = 100000, $O(n^2)$ for Quick Sort would require about 10,000,000,000 operations, whereas Merge Sort would require only about $100000 \times \log_2(100000) \approx 1,700,000$ operations.

Final Answer: Merge Sort is preferred here, since its guaranteed O(n log n) performance is far more reliable than Quick Sort's worst-case $O(n^2)$ on nearly sorted data (unless a randomised or median pivot is used).