DATA STRUCTURE

Chapter 8

Searching & Sorting

Solved Problems – Exam Practice Set 3

For Engineering (B.Tech / Diploma) Students

Contents

- Section A – Sorting Algorithms: Solved Numerical Problems (Bubble, Selection, Insertion, Quick, Merge, Shell, Heap, Radix, Binary Tree Sort)
- Section B – Searching: Solved Numerical Problems on Binary Search
- Section C – Time Complexity & Analytical Problems

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

Section A – Sorting Algorithms

Problem 1: Sort the array A = [6, 3, 9, 5, 1] in ascending order using Bubble Sort. Show the array after every pass and find the total number of comparisons and swaps.

Solution:

```
Initial Array: 6, 3, 9, 5, 1

Pass 1 (4 comparisons):
  6,3 -> swap -> 3,6,9,5,1
  6,9 -> no swap
  9,5 -> swap -> 3,6,5,9,1
  9,1 -> swap -> 3,6,5,1,9
  After Pass 1 : 3, 6, 5, 1, 9   (swaps = 3)

Pass 2 (3 comparisons):
  3,6 -> no swap
  6,5 -> swap -> 3,5,6,1,9
  6,1 -> swap -> 3,5,1,6,9
  After Pass 2 : 3, 5, 1, 6, 9   (swaps = 2)

Pass 3 (2 comparisons):
  3,5 -> no swap
  5,1 -> swap -> 3,1,5,6,9
  After Pass 3 : 3, 1, 5, 6, 9   (swaps = 1)

Pass 4 (1 comparison):
  3,1 -> swap -> 1,3,5,6,9
  After Pass 4 : 1, 3, 5, 6, 9   (swaps = 1)
```

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

Total swaps = 3 + 2 + 1 + 1 = 7

Final Answer: Sorted Array = 1, 3, 5, 6, 9 | Comparisons = 10, Swaps = 7

Problem 2: Sort the array A = [45, 23, 8, 71, 19, 56] in ascending order using Selection Sort. Show the array after each pass.

Solution:

```
Initial Array: 45, 23, 8, 71, 19, 56

Pass 1: minimum = 8 (index 2)  -> swap A[0],A[2] -> 8,23,45,71,19,56
Pass 2: minimum = 19 (index 4) -> swap A[1],A[4] -> 8,19,45,71,23,56
Pass 3: minimum = 23 (index 4) -> swap A[2],A[4] -> 8,19,23,71,45,56
Pass 4: minimum = 45 (index 4) -> swap A[3],A[4] -> 8,19,23,45,71,56
Pass 5: minimum = 56 (index 5) -> swap A[4],A[5] -> 8,19,23,45,56,71
```

Final Answer: Sorted Array = 8, 19, 23, 45, 56, 71

Problem 3: Sort the array A = [40, 10, 60, 20, 50, 30] in ascending order using Insertion Sort. Show the array after each element is inserted.

Solution:

```
Initial Array: 40, 10, 60, 20, 50, 30

i=1, key=10: 40>10 shift; insert 10 at 0          -> 10,40,60,20,50,30
i=2, key=60: 40<60, no shift                       -> 10,40,60,20,50,30
i=3, key=20: 60>20,40>20 shift; 10<20 stop; ins@1  -> 10,20,40,60,50,30
i=4, key=50: 60>50 shift; 40<50 stop; insert@3     -> 10,20,40,50,60,30
i=5, key=30: 60>30,50>30,40>30 shift; 20<30 stop   -> 10,20,30,40,50,60
```

Final Answer: Sorted Array = 10, 20, 30, 40, 50, 60

Problem 4: Sort the array A = [48, 25, 79, 60, 11, 92, 36] 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: 48, 25, 79, 60, 11, 92, 36 (pivot = 36, the last element)

```
i = -1
j=0: 48<=36? No
j=1: 25<=36  -> i=0, swap A[0],A[1]  -> 25,48,79,60,11,92,36
j=2: 79<=36? No
j=3: 60<=36? No
j=4: 11<=36  -> i=1, swap A[1],A[4]  -> 25,11,79,60,48,92,36
j=5: 92<=36? No

Place pivot: swap A[i+1] (A[2]) with A[high] (A[6])
                                      -> 25,11,36,60,48,92,79

Pivot 36 is now fixed at index 2.
Left sub-array  : [25,11] -> Quick Sort gives [11,25]
Right sub-array : [60,48,92,79] -> Quick Sort gives [48,60,79,92]
```

Final Answer: Sorted Array = 11, 25, 36, 48, 60, 79, 92

Problem 5: Sort the array A = [8, 3, 5, 4, 7, 6, 1, 2] in ascending order using Merge Sort. Show the divide and merge steps.

Solution:

```
DIVIDE:
[8,3,5,4,7,6,1,2]
   /            \
[8,3,5,4]      [7,6,1,2]
  /   \          /   \
[8,3] [5,4]     [7,6] [1,2]
 / \   / \       / \   / \
[8][3][5][4]   [7][6][1][2]

MERGE (combine back in sorted order):
[8][3] -> [3,8]        [5][4] -> [4,5]
[3,8][4,5] -> [3,4,5,8]
[7][6] -> [6,7]        [1][2] -> [1,2]
[6,7][1,2] -> [1,2,6,7]
[3,4,5,8][1,2,6,7] -> [1,2,3,4,5,6,7,8]
```

Final Answer: Sorted Array = 1, 2, 3, 4, 5, 6, 7, 8

Problem 6: Sort the array A = [22, 9, 33, 4, 15, 7] in ascending order using Shell Sort with gap sequence 3, then 1.

Solution:

```
Initial Array: 22, 9, 33, 4, 15, 7   (n = 6)

GAP = 3:
  i=3 (key=4) : A[0]=22>4 shift -> A[3]=22
                 j=-3 stop -> insert 4 at A[0]   -> 4,9,33,22,15,7
  i=4 (key=15): A[1]=9<=15, no shift              -> 4,9,33,22,15,7
  i=5 (key=7) : A[2]=33>7 shift -> A[5]=33
                 j=-1 stop -> insert 7 at A[2]   -> 4,9,7,22,15,33
  After gap=3 : 4, 9, 7, 22, 15, 33

GAP = 1 (standard Insertion Sort):
  i=1 (key=9) : 4<9, no shift
  i=2 (key=7) : 9>7 shift; 4<7 stop; insert@1     -> 4,7,9,22,15,33
  i=3 (key=22): 9<22, no shift
  i=4 (key=15): 22>15 shift; 9<15 stop; insert@3  -> 4,7,9,15,22,33
  i=5 (key=33): 22<33, no shift
```

Final Answer: Sorted Array = 4, 7, 9, 15, 22, 33

Problem 7: Sort the array A = [3, 19, 1, 14, 8, 7] in ascending order using Heap Sort. Show the Max Heap constructed and the extraction steps.

Solution:

```
Initial Array: 3, 19, 1, 14, 8, 7   (n = 6)

BUILD MAX HEAP (i from n/2-1 = 2 down to 0):
  i=2 (node=1, child=7): 7>1, swap A[2],A[5] -> 3,19,7,14,8,1
  i=1 (node=19, children 14,8): 19 already largest, no change
  i=0 (node=3, children 19,7): largest=19, swap A[0],A[1]
                              -> 19,3,7,14,8,1
     heapify index1 (node=3, children 14,8): largest=14, swap
                              -> 19,14,7,3,8,1
  Max Heap = 19, 14, 7, 3, 8, 1

EXTRACTION:
  swap A[0],A[5] -> 1,14,7,3,8,19 ; heapify(size5) -> 14,8,7,3,1,19
  swap A[0],A[4] -> 1,8,7,3,14,19 ; heapify(size4) -> 8,3,7,1,14,19
  swap A[0],A[3] -> 1,3,7,8,14,19 ; heapify(size3) -> 7,3,1,8,14,19
  swap A[0],A[2] -> 1,3,7,8,14,19 ; heapify(size2) -> 3,1,7,8,14,19
  swap A[0],A[1] -> 1,3,7,8,14,19 ; heap size1, done
```

Final Answer: Sorted Array = 1, 3, 7, 8, 14, 19

Problem 8: Sort the array A = [54, 26, 93, 17, 77, 31, 44, 55, 20] in ascending order using Radix Sort.

Solution:

```
Initial Array: 54, 26, 93, 17, 77, 31, 44, 55, 20
Maximum number = 93 (2 digits), so 2 passes are required.

PASS 1 (Units digit):
  Buckets: 0:[20] 1:[31] 3:[93] 4:[54,44] 5:[55] 6:[26] 7:[17,77]
  Collected -> 20, 31, 93, 54, 44, 55, 26, 17, 77

PASS 2 (Tens digit):
  Buckets: 1:[17] 2:[20,26] 3:[31] 4:[44] 5:[54,55] 7:[77] 9:[93]
  Collected -> 17, 20, 26, 31, 44, 54, 55, 77, 93
```

Final Answer: Sorted Array = 17, 20, 26, 31, 44, 54, 55, 77, 93

Problem 9: Sort the array A = [60, 35, 80, 20, 45, 70, 90] in ascending order using Binary Tree Sort. Show the Binary Search Tree formed and the traversal used.

Solution:

```
Insert elements in the given order 60,35,80,20,45,70,90 into a BST:

                60
               /  \
             35     80
            /  \   /  \
          20   45 70   90

Perform IN-ORDER traversal (Left - Root - Right):
  Visit 20 -> 35 -> 45 -> 60 -> 70 -> 80 -> 90
```

Final Answer: Sorted Array = 20, 35, 45, 60, 70, 80, 90

Section B – Searching (Binary Search)

Problem 10: Using Binary Search, find the key = 41 in the sorted array A = [4, 9, 13, 20, 28, 36, 41, 50, 58, 67, 74, 85]. Show low, high and mid at every step and the number of comparisons made.

Solution:

```
Array (index 0-11): 4,9,13,20,28,36,41,50,58,67,74,85

Step 1: low=0, high=11, mid=5 -> A[5]=36
        36 < 41  => low = 6

Step 2: low=6, high=11, mid=8 -> A[8]=58
        58 > 41  => high = 7

Step 3: low=6, high=7, mid=6 -> A[6]=41
        41 == 41 => KEY FOUND at index 6
```

Final Answer: Key 41 found at index 6 | Number of comparisons = 3

Problem 11: Using Binary Search on the same array A = [4, 9, 13, 20, 28, 36, 41, 50, 58, 67, 74, 85], search for key = 30, which is not present. Show all the steps and the result.

Solution:

```
Step 1: low=0, high=11, mid=5 -> A[5]=36
        36 > 30  => high = 4

Step 2: low=0, high=4, mid=2 -> A[2]=13
        13 < 30  => low = 3

Step 3: low=3, high=4, mid=3 -> A[3]=20
        20 < 30  => low = 4

Step 4: low=4, high=4, mid=4 -> A[4]=28
        28 < 30  => low = 5

Now low(5) > high(4)  => SEARCH UNSUCCESSFUL
```

Final Answer: Key 30 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 = 512 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 $2^9 = 512$, $\log_2(512) = 9$

Maximum comparisons = 9

Final Answer: Maximum comparisons required = 9

Problem 13: Find the total number of comparisons made by Bubble Sort (without early termination) in the worst case for an array of n = 10 elements.

Solution:

In the worst case, Bubble Sort compares every adjacent pair in every one of the (n-1) passes.

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

For n = 10: Total comparisons = 10 × 9 / 2 = 90 / 2 = 45

Final Answer: Total worst-case comparisons = 45

Problem 14: Heap Sort builds a heap in O(n) time and then performs n extraction steps, each costing O(log n). For n = 1024 elements, estimate the approximate total number of operations required.

Solution:

Total time complexity of Heap Sort = O(n) [build heap] + O(n log n) [extractions] = O(n log n)

For n = 1024: $\log_2(1024) = 10$ (since $2^{10} = 1024$)

Approximate operations ≈ $n \times \log_2 n = 1024 \times 10 = 10{,}240$

Final Answer: Approximate total operations ≈ 10,240

Problem 15: Compare the auxiliary space required by Merge Sort and Quick Sort for sorting n = 50,000 elements.

Solution:

Merge Sort requires O(n) auxiliary space, since it needs temporary arrays of the same total size as the input during merging.

For n = 50,000, Merge Sort therefore needs about 50,000 extra memory locations.

Quick Sort requires only O(log n) auxiliary space, used for the recursive call stack (assuming balanced partitions).

For n = 50,000: since $2^{15} = 32{,}768$ and $2^{16} = 65{,}536$, $\log_2(50{,}000) \approx 15.6$, so about 16 stack frames are needed.

Final Answer: Merge Sort needs about 50,000 extra memory locations (O(n)); Quick Sort needs only about 16 (O(log n)) – Quick Sort is far more space-efficient.

Problem 16: The worst-case running time of Quick Sort satisfies the recurrence T(n) = T(n-1) + O(n), with T(1) = O(1), which occurs when the partition is maximally unbalanced. Solve this recurrence to find the worst-case time complexity, and contrast it with the best-case recurrence T(n) = 2T(n/2) + O(n).

Solution:

Worst Case: T(n) = T(n-1) + cn

```
T(n) = T(n-1) + cn
     = T(n-2) + c(n-1) + cn
     = T(n-3) + c(n-2) + c(n-1) + cn
     = ...
     = T(1) + c(2 + 3 + ... + n)
     = O(1) + c * n(n+1)/2
     = O(n²)
```

Best Case: T(n) = 2T(n/2) + cn

This matches the Master Theorem with a=2, b=2, d=1, where $a = b^d$, giving T(n) = O(n log n).

Final Answer: Worst-case time complexity = $O(n^2)$; Best-case time complexity = O(n log n)