DATA STRUCTURE

Chapter 8

Searching & Sorting

Solved Problems – Exam Practice Set 5

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 1-4, for additional exam practice.

Section A – Sorting Algorithms

Problem 1: Sort the array A = [31, 8, 19, 2, 14] in ascending order using Bubble Sort. Show the array after every pass and find the total number of comparisons and swaps.

Solution:

```
Initial Array: 31, 8, 19, 2, 14

Pass 1 (4 comparisons):
  31,8  -> swap -> 8,31,19,2,14
  31,19 -> swap -> 8,19,31,2,14
  31,2  -> swap -> 8,19,2,31,14
  31,14 -> swap -> 8,19,2,14,31
  After Pass 1 : 8, 19, 2, 14, 31   (swaps = 4)

Pass 2 (3 comparisons):
  8,19 -> no swap
  19,2 -> swap -> 8,2,19,14,31
  19,14 -> swap -> 8,2,14,19,31
  After Pass 2 : 8, 2, 14, 19, 31   (swaps = 2)

Pass 3 (2 comparisons):
  8,2 -> swap -> 2,8,14,19,31
  8,14 -> no swap
  After Pass 3 : 2, 8, 14, 19, 31   (swaps = 1)

Pass 4 (1 comparison):
  2,8 -> no swap -- array already sorted, algorithm terminates.
```

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

Total swaps = 4 + 2 + 1 + 0 = 7

Final Answer: Sorted Array = 2, 8, 14, 19, 31 | Comparisons = 10, Swaps = 7

Problem 2: Sort the array A = [52, 19, 68, 7, 33, 41] in ascending order using Selection Sort. Show the array after each pass.

Solution:

```
Initial Array: 52, 19, 68, 7, 33, 41

Pass 1: minimum = 7 (index 3)  -> swap A[0],A[3] -> 7,19,68,52,33,41
Pass 2: minimum = 19 (already in place) -> no swap -> 7,19,68,52,33,41
Pass 3: minimum = 33 (index 4) -> swap A[2],A[4] -> 7,19,33,52,68,41
Pass 4: minimum = 41 (index 5) -> swap A[3],A[5] -> 7,19,33,41,68,52
Pass 5: minimum = 52 (index 5) -> swap A[4],A[5] -> 7,19,33,41,52,68
```

Final Answer: Sorted Array = 7, 19, 33, 41, 52, 68

Problem 3: Sort the array A = [85, 22, 63, 10, 47] in ascending order using Insertion Sort. Show the array after each element is inserted.

Solution:

```
Initial Array: 85, 22, 63, 10, 47

i=1, key=22: 85>22 shift; insert 22 at 0          -> 22,85,63,10,47
i=2, key=63: 85>63 shift; 22<63 stop; insert@1    -> 22,63,85,10,47
i=3, key=10: 85>10,63>10,22>10 shift; insert@0    -> 10,22,63,85,47
i=4, key=47: 85>47,63>47 shift; 22<47 stop; ins@2 -> 10,22,47,63,85
```

Final Answer: Sorted Array = 10, 22, 47, 63, 85

Problem 4: Sort the array A = [72, 15, 58, 4, 91, 26, 63] 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: 72, 15, 58, 4, 91, 26, 63 (pivot = 63, the last element)

```
i = -1
j=0: 72<=63? No
j=1: 15<=63  -> i=0, swap A[0],A[1]  -> 15,72,58,4,91,26,63
j=2: 58<=63  -> i=1, swap A[1],A[2]  -> 15,58,72,4,91,26,63
j=3: 4<=63   -> i=2, swap A[2],A[3]  -> 15,58,4,72,91,26,63
j=4: 91<=63? No
j=5: 26<=63  -> i=3, swap A[3],A[5]  -> 15,58,4,26,91,72,63

Place pivot: swap A[i+1] (A[4]) with A[high] (A[6])
                                      -> 15,58,4,26,63,72,91

Pivot 63 is now fixed at index 4.
Left sub-array  : [15,58,4,26] -> Quick Sort gives [4,15,26,58]
Right sub-array : [72,91] -> already in order [72,91]
```

Final Answer: Sorted Array = 4, 15, 26, 58, 63, 72, 91

Problem 5: Sort the array A = [19, 7, 3, 25, 14, 2, 17, 11] in ascending order using Merge Sort. Show the divide and merge steps.

Solution:

```
DIVIDE:
[19,7,3,25,14,2,17,11]
   /              \
[19,7,3,25]        [14,2,17,11]
  /   \              /    \
[19,7][3,25]        [14,2][17,11]
 / \   / \            / \   /  \
[19][7][3][25]      [14][2][17][11]

MERGE (combine back in sorted order):
[19][7] -> [7,19]        [3][25] -> [3,25]
[7,19][3,25] -> [3,7,19,25]
[14][2] -> [2,14]        [17][11] -> [11,17]
[2,14][11,17] -> [2,11,14,17]
[3,7,19,25][2,11,14,17] -> [2,3,7,11,14,17,19,25]
```

Final Answer: Sorted Array = 2, 3, 7, 11, 14, 17, 19, 25

Problem 6: Sort the array A = [64, 34, 25, 12, 22, 11, 90] in ascending order using Shell Sort with initial gap = n/2 = 3, then gap = 1.

Solution:

```
Initial Array: 64, 34, 25, 12, 22, 11, 90   (n = 7)

GAP = 3:
  i=3 (key=12): A[0]=64>12 shift -> A[3]=64
                j=-3 stop -> insert 12 at A[0]  -> 12,34,25,64,22,11,90
  i=4 (key=22): A[1]=34>22 shift -> A[4]=34
                j=-2 stop -> insert 22 at A[1]  -> 12,22,25,64,34,11,90
  i=5 (key=11): A[2]=25>11 shift -> A[5]=25
                j=-1 stop -> insert 11 at A[2]  -> 12,22,11,64,34,25,90
  i=6 (key=90): A[3]=64<=90, no shift
  After gap=3 : 12, 22, 11, 64, 34, 25, 90

GAP = 1 (standard Insertion Sort):
  i=1 (key=22): 12<22, no shift
  i=2 (key=11): 22>11,12>11 shift; insert@0        -> 11,12,22,64,34,25,90
  i=3 (key=64): 22<64, no shift
  i=4 (key=34): 64>34 shift; 22<34 stop; insert@3   -> 11,12,22,34,64,25,90
  i=5 (key=25): 64>25,34>25 shift; 22<25 stop; ins@3-> 11,12,22,25,34,64,90
  i=6 (key=90): 64<90, no shift
```

Final Answer: Sorted Array = 11, 12, 22, 25, 34, 64, 90

Problem 7: Sort the array A = [16, 4, 10, 14, 7, 9] in ascending order using Heap Sort. Show the Max Heap constructed and the extraction steps.

Solution:

```
Initial Array: 16, 4, 10, 14, 7, 9   (n = 6)

BUILD MAX HEAP (i from n/2-1 = 2 down to 0):
  i=2 (node=10, child=9): 10 already largest, no change
  i=1 (node=4, children 14,7): largest=14, swap A[1],A[3]
                               -> 16,14,10,4,7,9
  i=0 (node=16, children 14,10): 16 already largest, no change
  Max Heap = 16, 14, 10, 4, 7, 9

EXTRACTION:
  swap A[0],A[5] -> 9,14,10,4,7,16 ; heapify(size5) -> 14,9,10,4,7,16
  swap A[0],A[4] -> 7,9,10,4,14,16 ; heapify(size4) -> 10,9,7,4,14,16
  swap A[0],A[3] -> 4,9,7,10,14,16 ; heapify(size3) -> 9,4,7,10,14,16
  swap A[0],A[2] -> 7,4,9,10,14,16 ; heapify(size2) -> 7,4,9,10,14,16
  swap A[0],A[1] -> 4,7,9,10,14,16 ; heap size1, done
```

Final Answer: Sorted Array = 4, 7, 9, 10, 14, 16

Problem 8: Sort the array A = [482, 17, 326, 894, 52, 3, 619] in ascending order using Radix Sort.

Solution:

```
Initial Array: 482, 17, 326, 894, 52, 3, 619
Maximum number = 894 (3 digits), so 3 passes are required.
```

```
PASS 1 (Units digit):
  Buckets: 2:[482,52] 3:[3] 4:[894] 6:[326] 7:[17] 9:[619]
  Collected -> 482, 52, 3, 894, 326, 17, 619

PASS 2 (Tens digit):
  Buckets: 0:[3] 1:[17,619] 2:[326] 5:[52] 8:[482] 9:[894]
  Collected -> 3, 17, 619, 326, 52, 482, 894

PASS 3 (Hundreds digit):
  Buckets: 0:[3,17,52] 3:[326] 4:[482] 6:[619] 8:[894]
  Collected -> 3, 17, 52, 326, 482, 619, 894
```

Final Answer: Sorted Array = 3, 17, 52, 326, 482, 619, 894

Problem 9: Sort the array A = [42, 23, 58, 10, 30, 50, 70] in ascending order using Binary Tree Sort. Show the Binary Search Tree formed and the traversal used.

Solution:

```
Insert elements in the given order 42,23,58,10,30,50,70 into a BST:

                42
               /  \
             23     58
            /  \   /  \
          10   30 50   70

Perform IN-ORDER traversal (Left - Root - Right):
  Visit 10 -> 23 -> 30 -> 42 -> 50 -> 58 -> 70
```

Final Answer: Sorted Array = 10, 23, 30, 42, 50, 58, 70

Section B – Searching (Binary Search)

Problem 10: Using Binary Search, find the key = 56 in the sorted array A = [5, 10, 18, 27, 34, 41, 49, 56, 63, 72, 80]. Show low, high and mid at every step and the number of comparisons made.

Solution:

```
Array (index 0-10): 5,10,18,27,34,41,49,56,63,72,80

Step 1: low=0, high=10, mid=5 -> A[5]=41
        41 < 56  => low = 6

Step 2: low=6, high=10, mid=8 -> A[8]=63
        63 > 56  => high = 7

Step 3: low=6, high=7, mid=6 -> A[6]=49
        49 < 56  => low = 7

Step 4: low=7, high=7, mid=7 -> A[7]=56
        56 == 56 => KEY FOUND at index 7
```

Final Answer: Key 56 found at index 7 | Number of comparisons = 4

Problem 11: Using Binary Search on the same array A = [5, 10, 18, 27, 34, 41, 49, 56, 63, 72, 80], search for key = 38, which is not present. Show all the steps and the result.

Solution:

```
Step 1: low=0, high=10, mid=5 -> A[5]=41
        41 > 38  => high = 4

Step 2: low=0, high=4, mid=2 -> A[2]=18
        18 < 38  => low = 3

Step 3: low=3, high=4, mid=3 -> A[3]=27
        27 < 38  => low = 4

Step 4: low=4, high=4, mid=4 -> A[4]=34
        34 < 38  => low = 5

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

Final Answer: Key 38 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 = 256 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 $2^8 = 256$, $\log_2(256) = 8$

Maximum comparisons = 8

Final Answer: Maximum comparisons required = 8

Problem 13: An array of n = 6 elements is in completely reverse sorted order. Find the total number of swaps Bubble Sort will perform to sort this array, and explain why this represents the worst case for the number of swaps.

Solution:

When an array is in completely reverse sorted order, every single comparison made by Bubble Sort finds a pair that is out of order, so EVERY comparison results in a swap.

Total comparisons (and hence total swaps in this case) = (n-1) + (n-2) + ... + 1 = n(n-1) / 2

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

This is the maximum possible number of swaps for any arrangement of 6 elements, since no other arrangement can have more out-of-order adjacent pairs.

Final Answer: Total swaps = 15 (this is the absolute worst case for Bubble Sort's swap count)

Problem 14: Use the recursion-tree method to solve the Merge Sort recurrence T(n) = 2T(n/2) + n, with T(1) = O(1), and hence find the time complexity.

Solution:

```
Level 0:            n                              -> cost = n
Level 1:      n/2         n/2                       -> cost = n/2 + n/2 = n
Level 2:   n/4  n/4    n/4  n/4                      -> cost = 4 * (n/4) = n
   ...                                                  ...
Level k:  (2^k) sub-problems of size n/2^k            -> cost = n   (each level costs n)

The recursion stops when sub-problem size = 1, i.e. n / 2^k = 1  =>  k = log2(n)
So there are (log2 n + 1) levels in total, each contributing a cost of n.

Total cost = n * (log2 n + 1) = O(n log n)
```

Final Answer: T(n) = O(n log n)

Problem 15: A sorted array has n = 100 elements. Compare the worst-case number of comparisons required by Linear Search and Binary Search, and find approximately how many times faster Binary Search is.

Solution:

Linear Search (worst case): comparisons = n = 100

Binary Search (worst case): comparisons = $\lceil \log_2 n \rceil = \lceil \log_2 100 \rceil$

Since $2^6 = 64$ and $2^7 = 128$, $\log_2(100) \approx 6.64$, so $\lceil \log_2 100 \rceil = 7$

Speed-up factor = 100 / 7 ≈ 14.3

Final Answer: Linear Search = 100 comparisons; Binary Search = 7 comparisons; Binary Search is about 14 times faster in the worst case

Problem 16: For the array A = [4, 1, 3, 2], count the number of inversions (pairs i < j such that A[i] > A[j]). Hence predict the exact number of swaps Bubble Sort will perform on this array, and verify your answer by tracing the algorithm.

Solution:

An inversion is any pair of positions (i, j) with i < j but A[i] > A[j]. Every such pair must be corrected by exactly one adjacent swap during Bubble Sort, so the total number of swaps always equals the number of inversions in the array.

```
Array: 4, 1, 3, 2   (positions 0,1,2,3)

Inversions:
  (4,1) -> 4 > 1  : inversion
  (4,3) -> 4 > 3  : inversion
  (4,2) -> 4 > 2  : inversion
  (1,3) -> 1 < 3  : not an inversion
  (1,2) -> 1 < 2  : not an inversion
  (3,2) -> 3 > 2  : inversion

Total inversions = 4

VERIFY BY TRACING BUBBLE SORT:
Pass 1: 4,1 swap->1,4,3,2 ; 4,3 swap->1,3,4,2 ; 4,2 swap->1,3,2,4  (3 swaps)
Pass 2: 1,3 no swap ; 3,2 swap -> 1,2,3,4                          (1 swap)

Total swaps performed = 3 + 1 = 4
```

Final Answer: Number of inversions = 4, which exactly matches the 4 swaps performed by Bubble Sort