DATA STRUCTURE

Chapter 8

Searching & Sorting

Solved Problems – Exam Practice Set 2

For Engineering (B.Tech / Diploma) Students

Contents

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

Section A – Sorting Algorithms

Problem 1: Sort the array A = [5, 1, 4, 2, 8] in ascending order using Bubble Sort. Show the array after every pass and find the total number of comparisons and swaps.

Solution:

```
Initial Array: 5, 1, 4, 2, 8

Pass 1 (4 comparisons):
  5,1 -> swap -> 1,5,4,2,8
  5,4 -> swap -> 1,4,5,2,8
  5,2 -> swap -> 1,4,2,5,8
  5,8 -> no swap
  After Pass 1 : 1, 4, 2, 5, 8   (swaps = 3)

Pass 2 (3 comparisons):
  1,4 -> no swap
  4,2 -> swap -> 1,2,4,5,8
  4,5 -> no swap
  After Pass 2 : 1, 2, 4, 5, 8   (swaps = 1)

Pass 3 (2 comparisons):
  1,2 -> no swap
  2,4 -> no swap
  No swap occurred -- array already sorted, algorithm terminates.
```

Total comparisons = 4 + 3 + 2 = 9

Total swaps = 3 + 1 + 0 = 4

Final Answer: Sorted Array = 1, 2, 4, 5, 8 | Comparisons = 9, Swaps = 4

Problem 2: Sort the array A = [64, 25, 12, 22, 11] in ascending order using Selection Sort. Show the array after each pass.

Solution:

```
Initial Array: 64, 25, 12, 22, 11

Pass 1: Unsorted [64,25,12,22,11] -> minimum = 11 (index 4)
        swap A[0] and A[4] -> 11, 25, 12, 22, 64

Pass 2: Unsorted [25,12,22,64] -> minimum = 12 (index 2)
        swap A[1] and A[2] -> 11, 12, 25, 22, 64

Pass 3: Unsorted [25,22,64] -> minimum = 22 (index 3)
        swap A[2] and A[3] -> 11, 12, 22, 25, 64

Pass 4: Unsorted [25,64] -> minimum = 25 (already in place)
        no swap needed   -> 11, 12, 22, 25, 64
```

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

Problem 3: Sort the array A = [9, 5, 1, 4, 3] in ascending order using Insertion Sort. Show the array after each element is inserted.

Solution:

```
Initial Array: 9, 5, 1, 4, 3

i=1, key=5 : 9>5 shift; insert 5 at index 0       -> 5, 9, 1, 4, 3
i=2, key=1 : 9>1,5>1 shift both; insert 1 at 0     -> 1, 5, 9, 4, 3
i=3, key=4 : 9>4,5>4 shift; 1<4 stop; insert at 1  -> 1, 4, 5, 9, 3
i=4, key=3 : 9>3,5>3,4>3 shift; 1<3 stop; insert@1 -> 1, 3, 4, 5, 9
```

Final Answer: Sorted Array = 1, 3, 4, 5, 9

Problem 4: Sort the array A = [33, 10, 55, 71, 29, 3, 18] 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: 33, 10, 55, 71, 29, 3, 18 (pivot = 18, the last element)

```
i = -1
j=0: 33<=18? No
j=1: 10<=18  -> i=0, swap A[0],A[1]  -> 10,33,55,71,29,3,18
j=2: 55<=18? No
j=3: 71<=18? No
j=4: 29<=18? No
j=5: 3<=18   -> i=1, swap A[1],A[5]  -> 10,3,55,71,29,33,18

Place pivot: swap A[i+1] (A[2]) with A[high] (A[6])
                                      -> 10,3,18,71,29,33,55

Pivot 18 is now fixed at index 2.
Left sub-array  : [10,3]  -> Quick Sort gives [3,10]
Right sub-array : [71,29,33,55] -> Quick Sort gives [29,33,55,71]
```

Final Answer: Sorted Array = 3, 10, 18, 29, 33, 55, 71

Problem 5: Sort the array A = [12, 11, 13, 5, 6, 7] in ascending order using Merge Sort. Show the divide and merge steps.

Solution:

```
DIVIDE:
[12,11,13,5,6,7]
   /           \
[12,11,13]      [5,6,7]
  /    \          /   \
[12] [11,13]     [5]  [6,7]
       /  \             /  \
     [11] [13]         [6]  [7]

MERGE (combine back in sorted order):
[11][13]       -> [11,13]
[12][11,13]    -> [11,12,13]
[6][7]         -> [6,7]
```

```
[5][6,7]       -> [5,6,7]
[11,12,13][5,6,7] -> [5,6,7,11,12,13]
```

Final Answer: Sorted Array = 5, 6, 7, 11, 12, 13

Problem 6: Sort the array A = [35, 20, 45, 10, 15, 25] in ascending order using Shell Sort with gap sequence 3, then 1.

Solution:

```
Initial Array: 35, 20, 45, 10, 15, 25   (n = 6)

GAP = 3:
  i=3 (key=10): A[0]=35>10 shift -> A[3]=35
                j=-3 stop -> insert 10 at A[0]   -> 10,20,45,35,15,25
  i=4 (key=15): A[1]=20>15 shift -> A[4]=20
                j=-2 stop -> insert 15 at A[1]   -> 10,15,45,35,20,25
  i=5 (key=25): A[2]=45>25 shift -> A[5]=45
                j=-1 stop -> insert 25 at A[2]   -> 10,15,25,35,20,45
  After gap=3 : 10, 15, 25, 35, 20, 45

GAP = 1 (standard Insertion Sort):
  i=1 (key=15): 10<15, no shift
  i=2 (key=25): 15<25, no shift
  i=3 (key=35): 25<35, no shift
  i=4 (key=20): 35>20,25>20 shift; 15<20 stop; insert@2 -> 10,15,20,25,35,45
  i=5 (key=45): 35<45, no shift
```

Final Answer: Sorted Array = 10, 15, 20, 25, 35, 45

Problem 7: Sort the array A = [1, 12, 9, 5, 6, 10] in ascending order using Heap Sort. Show the Max Heap constructed and the extraction steps.

Solution:

```
Initial Array: 1, 12, 9, 5, 6, 10   (n = 6)

BUILD MAX HEAP (i from n/2-1 = 2 down to 0):
  i=2 (node=9, child=10): 10>9, swap A[2],A[5] -> 1,12,10,5,6,9
  i=1 (node=12, children 5,6): 12 already largest, no change
  i=0 (node=1, children 12,10): largest=12, swap A[0],A[1]
                                -> 12,1,10,5,6,9
     heapify index1 (node=1, children 5,6): largest=6, swap
                                -> 12,6,10,5,1,9
  Max Heap = 12, 6, 10, 5, 1, 9

EXTRACTION:
  swap A[0],A[5] -> 9,6,10,5,1,12 ; heapify(size5) -> 10,6,9,5,1,12
  swap A[0],A[4] -> 1,6,9,5,10,12 ; heapify(size4) -> 9,6,1,5,10,12
  swap A[0],A[3] -> 5,6,1,9,10,12 ; heapify(size3) -> 6,5,1,9,10,12
  swap A[0],A[2] -> 1,5,6,9,10,12 ; heapify(size2) -> 5,1,6,9,10,12
  swap A[0],A[1] -> 1,5,6,9,10,12 ; heap size1, done
```

Final Answer: Sorted Array = 1, 5, 6, 9, 10, 12

Problem 8: Sort the array A = [329, 457, 657, 839, 436, 720, 355] in ascending order using Radix Sort.

Solution:

```
Initial Array: 329, 457, 657, 839, 436, 720, 355
Maximum number = 839 (3 digits), so 3 passes are required.

PASS 1 (Units digit):
  Buckets: 0:[720] 5:[355] 6:[436] 7:[457,657] 9:[329,839]
  Collected -> 720, 355, 436, 457, 657, 329, 839

PASS 2 (Tens digit):
  Buckets: 2:[720,329] 3:[436,839] 5:[355,457,657]
  Collected -> 720, 329, 436, 839, 355, 457, 657

PASS 3 (Hundreds digit):
  Buckets: 3:[329,355] 4:[436,457] 6:[657] 7:[720] 8:[839]
  Collected -> 329, 355, 436, 457, 657, 720, 839
```

Final Answer: Sorted Array = 329, 355, 436, 457, 657, 720, 839

Problem 9: Sort the array A = [27, 14, 35, 10, 19, 31, 42] in ascending order using Binary Tree Sort. Show the Binary Search Tree formed and the traversal used.

Solution:

```
Insert elements in the given order 27,14,35,10,19,31,42 into a BST:

                27
               /  \
             14     35
            /  \   /  \
          10   19 31   42

Perform IN-ORDER traversal (Left - Root - Right):
  Visit 10 -> 14 -> 19 -> 27 -> 31 -> 35 -> 42
```

Final Answer: Sorted Array = 10, 14, 19, 27, 31, 35, 42

Section B – Searching (Binary Search)

Problem 10: Using Binary Search, find the key = 45 in the sorted array A = [2, 5, 8, 12, 16, 23, 38, 45, 56, 72, 91]. Show low, high and mid at every step and the number of comparisons made.

Solution:

```
Array (index 0-10): 2,5,8,12,16,23,38,45,56,72,91

Step 1: low=0, high=10, mid=5 -> A[5]=23
        23 < 45  => low = 6

Step 2: low=6, high=10, mid=8 -> A[8]=56
        56 > 45  => high = 7

Step 3: low=6, high=7, mid=6 -> A[6]=38
        38 < 45  => low = 7

Step 4: low=7, high=7, mid=7 -> A[7]=45
        45 == 45 => KEY FOUND at index 7
```

Final Answer: Key 45 found at index 7 | Number of comparisons = 4

Problem 11: Using Binary Search on the same array A = [2, 5, 8, 12, 16, 23, 38, 45, 56, 72, 91], search for key = 6, which is not present. Show all the steps and the result.

Solution:

```
Step 1: low=0, high=10, mid=5 -> A[5]=23
        23 > 6  => high = 4

Step 2: low=0, high=4, mid=2 -> A[2]=8
        8 > 6   => high = 1

Step 3: low=0, high=1, mid=0 -> A[0]=2
        2 < 6   => low = 1

Step 4: low=1, high=1, mid=1 -> A[1]=5
        5 < 6   => low = 2

Now low(2) > high(1)  => SEARCH UNSUCCESSFUL
```

Final Answer: Key 6 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 = 2048 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 $2^{11} = 2048$, $\log_2(2048) = 11$

Maximum comparisons = 11

Final Answer: Maximum comparisons required = 11

Problem 13: Find the total number of comparisons made by Insertion Sort in the worst case for an array of n = 8 elements (i.e. the array is in reverse sorted order).

Solution:

In the worst case (reverse sorted array), every new key must be compared with all previously sorted elements before insertion.

Total comparisons = 1 + 2 + 3 + ... + (n-1) = n(n-1) / 2

For n = 8: Total comparisons = 8 × 7 / 2 = 56 / 2 = 28

Final Answer: Total worst-case comparisons = 28

Problem 14: Estimate the approximate number of comparisons made by Quick Sort (average case) while sorting an array of n = 1000 elements.

Solution:

Average-case time complexity of Quick Sort is $O(n \log_2 n)$.

$\log_2(1000) \approx 9.97$

Approximate comparisons $\approx n \times \log_2 n = 1000 \times 9.97 \approx 9970$

Final Answer: Approximate number of comparisons ≈ 9970 (average case)

Problem 15: Merge Sort's running time satisfies the recurrence relation T(n) = 2T(n/2) + O(n), with T(1) = O(1). Solve this recurrence using the Master Theorem and state the resulting time complexity.

Solution:

The recurrence is of the standard form $T(n) = aT(n/b) + O(n^d)$, where a = 2, b = 2 and d = 1.

Compare a with b^d: $b^d = 2^1 = 2$, and a = 2, so $a = b^d$ (Case 2 of the Master Theorem).

When $a = b^d$, the Master Theorem gives: $T(n) = O(n^d \log n) = O(n \log n)$.

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

Problem 16: Given the array of (key, label) pairs A = [(4,A), (3,B), (4,C), (1,D)], sort it in ascending order of key using (a) Selection Sort and (b) Bubble Sort. In each case, state the final relative order of the two elements with key = 4, and hence identify which of the two algorithms is stable.

Solution:

(a) Selection Sort:

```
Initial: (4,A), (3,B), (4,C), (1,D)

Pass 1: scan for minimum -> (1,D) at index 3
        swap A[0],A[3] -> (1,D), (3,B), (4,C), (4,A)
Pass 2: subarray [(3,B),(4,C),(4,A)] -> minimum = (3,B), no swap
Pass 3: subarray [(4,C),(4,A)] -> minimum = (4,C) (first one), no swap

Final: (1,D), (3,B), (4,C), (4,A)
Relative order of key=4 elements is now C before A
(ORIGINAL order was A before C) -- ORDER REVERSED.
```

(b) Bubble Sort:

```
Initial: (4,A), (3,B), (4,C), (1,D)

Pass 1: (4,A)-(3,B) swap -> (3,B),(4,A),(4,C),(1,D)
        (4,A)-(4,C) equal, no swap
        (4,C)-(1,D) swap -> (3,B),(4,A),(1,D),(4,C)
Pass 2: (3,B)-(4,A) no swap
        (4,A)-(1,D) swap -> (3,B),(1,D),(4,A),(4,C)
        (4,A)-(4,C) equal, no swap
Pass 3: (3,B)-(1,D) swap -> (1,D),(3,B),(4,A),(4,C)
        (3,B)-(4,A) no swap
Pass 4: no swaps -- sorted

Final: (1,D), (3,B), (4,A), (4,C)
Relative order of key=4 elements is A before C
(same as the ORIGINAL order) -- ORDER PRESERVED.
```

Final Answer: Bubble Sort is stable (preserves relative order of equal keys); Selection Sort is NOT stable, since it can reverse the relative order of equal-key elements, as shown above.