UCS301-5

Q1. (a) Time complexity analysis:

Statement 1: 1

Statement 2: 1

Statement 3: n+1

Statement 4: n

Statement 5: n(n+1)/2

Statement 6: n(n+1)/2

Statement 7: n

Statement 8: 1

(b) To show $3n + 7 = \Theta(n)$:

$c_1 n \leq 3n + 7 \leq c_2 n$ for $n \geq n_0$

Let $c_1 = 1$ and $c_2 = 10$

$1n \leq 3n + 7 \leq 10n$

$n \geq 7/2$

Therefore, $n_0 = 4$, $c_1 = 1$, $c_2 = 10$ satisfy the condition.

Q2. (a) Address calculation:

Address = Base + [(i - LBi) * Cj * Ck + (j - LBj) * Ck + (k - LBk)] * size

= 300 + [(9 - 6) * 20 * 50 + (16 - (-2)) * 50 + (32 - (-5))] * 4

= 300 + [3000 + 900 + 185] * 4

= 300 + 16340

= 16640

(b) Sparse matrix transpose algorithms omitted due to length constraints.

Q3. Algorithm for creating LL3 with common nodes:

1. Initialize pointers p1 and p2 to heads of LL1 and LL2

2. Initialize LL3 as empty

```
3. While p1 and p2 are not null:
  a. If p1->data < p2->data, move p1 to next
  b. Else if p2->data < p1->data, move p2 to next
  c. Else (equal values):
    - Remove node from LL1/LL2 and add to LL3
    - Move p1 and p2 to next
4. Return LL3
```

Q4. (a) Infix to postfix conversion:

A+B*(C^D-E)^F+G*H-I

Postfix: ABCD^E-F^*+GH*+I-

(b) Stack-life calculations:

(i) m=1: (2n-2)X + (2n-1)Y

(ii) m=n: 0

(iii) ASL = [(n-1)X + nY](n+1)/2

Q5. (a) Circular queue operations (student's mistaken implementation):

(i) isFull(): return (front == (rear + 1) % n)

(ii) isEmpty(): return (front == -1 && rear == -1)

(iii-vi) Operations omitted due to length constraints.

(b) Queue reversal algorithm:

```
1. If Q is empty or has 1 element, return
2. Dequeue front element
3. Reverse remaining queue recursively
4. Enqueue dequeued element
```

Execution on Q = 4 6 1 9:

Step 1: 6 1 9, 4 (dequeued)

Step 2: 1 9, 6 4 (dequeued)

Step 3: 9, 1 6 4 (dequeued)

Step 4: 9 1 6 4 (final reversed queue)

UCS301-6

Q1. (a) BST drawing:

```
    5
   / \
  3   6
 / \   \
1   4   8
 \     /
  2   7
```

Postorder traversal: 2 1 4 3 7 8 6 5

(b) Maximum height of AVL tree with 7 nodes: 3

Q2. (a) Address of A[i][j] = 100 + (10i + j - 11)

(b) Linear search: O(n(1+2)) = O(3n) = O(n)

    Binary search: O(log(n(1+2))) = O(log(3n)) = O(log n)

Q3. (a) Max heap: A complete binary tree where each node is greater than or equal to its children.

Example:

```
    50
   / \
  30  20
 / \ /
```

```
10 5 15
```

(b) Max heap after inserting 32, 15, 20, 30, 12, 25, 16:

```
   32
  / \
 30  25
/ \ / \
15 12 20 16
```

Q4. (a) Hash table with linear probing:

0:	43
1:	-
2:	92
3:	36
4:	4
5:	71
6:	13
7:	14
8:	87
9:	-
10:	11

Q5. (a) Postfix equation result: Not provided in the question.

(b) Infix to postfix: a*b+(c-d) -> ab*cd-+

Q6. (a) AVL tree construction omitted due to length constraints.

Q7. (a) Dijkstra's algorithm execution omitted due to length constraints.

(b) Spanning tree: A subgraph that is a tree and connects all vertices of the graph.

Q8. (a) ENQUEUE using reverse: reverse, push, reverse

DEQUEUE: pop

Q9. (a) Ascending order: F3, F2, F4, F1

Q10. (a) Binary tree vs Binary search tree:

- Binary tree: Each node has at most two children

- BST: Left subtree contains only nodes with keys less than the node's key, right subtree only nodes with keys greater than the node's key

(b) Complete binary tree: All levels are filled except possibly the last, which is filled from left to right.

(c) 2 possible binary trees:

```
 A    A
/      \
B       B
\      /
 C    C
```

UCS301-7

Q1. Address of A = 4142 - (4*15 + 10)*2 = 3982

Q2. Postfix: AB$C+K#L+MN*-OP^W*U/+

Q3. (a) FRONT values: 3, 3, 3, 7, 7

(b) Circular queue: A linear queue connected end-to-end. Overflow: (rear+1)%size == front. Underflow: front == -1.

Q4. fun1() prints the linked list in order. Output: 1 2 3 4 5 6

Q5. Quick sort and Merge sort pseudocodes omitted due to length constraints. Both have average time complexity of O(n log n).

Q6. Explanations of tree types omitted due to length constraints.

Q7. BFS traversal: s, r, w, v, t, x, u, y

Q8. Bellman-Ford algorithm execution omitted due to length constraints.

Q9. Hash table with linear probing:

0: 11

1: 23

2: 92

3: 14

4: 4

5: 71

6: 36

7: 47

8: 58

9: 13

10: 43

UCS301-8

Q1. (a) Algorithm to check balanced parentheses omitted due to length constraints.

(b) Hash table insertions:

(i) Quadratic probing: 3, 6, 1, 3, 8

(ii) Double hashing: 3, 6, 1, 3, 8

(c) Heapsort execution omitted due to length constraints. Worst-case time complexity: O(n log n)

Q2. (a) BST operations omitted due to length constraints.

(b) AVL tree operations omitted due to length constraints.

(c) Sorted doubly linked list insertion algorithm omitted due to length constraints.

Q3. (a) XOR linked list search and delete algorithms omitted due to length constraints.

(b) Stock span algorithm omitted due to length constraints.

(c) Queue after fun() execution: 9, 8, 3, 1, 7, 5