UCS301-0

Q1. Infix to postfix conversion:

(A$B+C)#(K+L-M*N+O^P*W/U)

Postfix: AB$C+KL+MN*-OP^W*U/+#

Q2. Push and pop sequence for STABLE:

Push S, Push T, Push A, Push B, Pop B, Pop A, Pop T, Pop S, Push L, Push E, Pop E, Pop L

Q3. Time complexity analysis:

(a) O(n^3)

(b) O(n^2 log n)

(c) O(m log n)

Q4. (a) Address of A = 3982

(b) 4X + 5Y = 31

Q5. (a) Stack contents after each iteration:

1st: E

2nd: E, S

3rd: E, S, A

(b) The function reverses the order of elements in the queue.

Q6. Radix Sort steps:

Iteration 1 (LSD):

COUNT: [1, 1, 1, 2, 1, 1, 1, 1, 0, 0]

A: [690, 690, 234, 234, 435, 435, 567, 387, 892, 203]

Iteration 2:

COUNT: [1, 0, 0, 3, 1, 0, 1, 0, 1, 2]

A: [203, 234, 435, 441, 567, 387, 690, 892]

Iteration 3 (MSD):

COUNT: [0, 0, 2, 1, 1, 1, 1, 0, 2, 0]

A: [203, 234, 387, 435, 441, 567, 690, 892]

Q7. Algorithm to delete principal numbers:

1. Calculate the sum of all elements
2. Iterate from right to left:
   a. If current element > (sum / remaining elements), delete it
   b. Update sum and remaining elements count
3. Shift remaining elements to fill gaps

Q8. Round Robin CPU Scheduling pseudocode:

```
while (SCLL is not empty or LL is not empty)
  if (LL is not empty)
    move all processes from LL to SCLL

  for each process P in SCLL
    if (P.burst_time <= tq)
      execute P for P.burst_time
      remove P from SCLL
    else
      execute P for tq
      P.burst_time -= tq
      move P to end of SCLL

  update waiting times and turnaround times
```

```

## UCS301-9

Q1. (a) Time complexity analysis:

i) O(n^3)

ii) O(n^2 log n)

(b) Binary search recursive function:

```cpp
int binarySearch(int arr[], int l, int r, int x) {
    if (r >= l) {
        int mid = l + (r - l) / 2;
        if (arr[mid] == x) return mid;
        if (arr[mid] > x) return binarySearch(arr, l, mid - 1, x);
        return binarySearch(arr, mid + 1, r, x);
    }
    return -1;
}
```

Recursive calls for searching 8:

binarySearch(A, 0, 11, 8)

binarySearch(A, 6, 11, 8)

binarySearch(A, 6, 8, 8)

binarySearch(A, 6, 7, 8)

Q2. (a) Enqueue and Dequeue functions:

```cpp
void Enqueue(struct queue &Q, char x) {
    if ((Q.rear + 1) % MAXSIZE == Q.front) {
        cout << "Queue is full";

```
        return;
    }
    if (Q.front == -1) Q.front = MAXSIZE - 1;
    Q.rear = (Q.rear - 1 + MAXSIZE) % MAXSIZE;
    Q.arr[Q.rear] = x;
}

char Dequeue(struct queue &Q) {
    if (Q.front == -1) {
        cout << "Queue is empty";
        return '\0';
    }
    char x = Q.arr[Q.front];
    if (Q.front == Q.rear) Q.front = Q.rear = -1;
    else Q.front = (Q.front - 1 + MAXSIZE) % MAXSIZE;
    return x;
}
```

(b) Queue contents after first three iterations:

1: 'D', 'A', 'M', 'A', 'G', 'E', 'D' (front=4, rear=0)

2: 'A', 'M', 'A', 'G', 'E', 'D', 'A' (front=3, rear=5)

3: 'M', 'A', 'G', 'E', 'D', 'A', 'M' (front=2, rear=4)

Q3. Infix to postfix conversion:

A*(B&D/E)-F*(G%H/K)

Postfix: ABD&E/*F-GH%K/*-

Q4. (a) Quick Sort steps:

[8, 7, 6, 1, 0, 9, 2]

[2, 7, 6, 1, 0, 8, 9]

[2, 0, 1, 6, 7, 8, 9]

[0, 1, 2, 6, 7, 8, 9]

Worst case: Already sorted array, O(n^2)

(b) Modified partition for 6, 7, 8:

```cpp
void partition(int arr[], int n) {
    int low = 0, mid = 0, high = n - 1;
    while (mid <= high) {
        if (arr[mid] == 6) swap(arr[low++], arr[mid++]);
        else if (arr[mid] == 7) mid++;
        else swap(arr[mid], arr[high--]);
    }
}
```

Q5. (a) Sparse matrix multiplication:

Result: {{5,5,4},{0,1,25},{2,1,70},{3,1,75},{3,3,96}}

(b) B is a valid stack permutation of A. Sequence:

Push 5, Push 6, Push 7, Push 8, Push 9, Pop 9, Pop 8, Pop 7, Push 6, Push 5, Pop 6, Pop 5

Q6. (a) Fill in the blanks:

1: head = nn;

2: r->next != NULL

3: r = r->next

4: r->next = nn

5: nn->next = head

6: i < counter

7: p->next != p

8: j < k - 1

9: q = p

10: p = p->next

11: q->next = p->next

12: p

13: p = p->next

14: p->data

(b) Order of elimination: 5, 2, 4, 1

Child who becomes IT: 3

UCS301-10

Q1. BST construction and AVL tree steps omitted due to length constraints.

Q2. (a) Insertion Sort algorithm:

```
for i = 1 to n-1
  key = arr[i]
  j = i - 1
  while j >= 0 and arr[j] > key
    arr[j+1] = arr[j]
    j = j - 1
  arr[j+1] = key
```

Sorting steps: [11,12,13,5,6], [11,12,13,5,6], [11,12,13,5,6], [5,11,12,13,6], [5,6,11,12,13]

Worst case: O(n^2), Best case: O(n)

(b) Min heap operations:

Insert 318: [99,142,305,221,440,318]

Extract min: [142,221,305,318,440]

Insert 102: [102,142,305,221,440,318]

Extract min: [142,221,305,318,440]

Final level order: 142,221,305,318,440

Q3. (a) Doubly linked list operations omitted due to length constraints.

(b) Postfix evaluation:

Stack operations: 2, 2, 3, 6, 36, 7, 252, 5, 247

Q4. (a) Hash table insertions:

Quadratic probing: [-, 36, -, 72, 27, 55, 32, 43]

Double hashing: [-, 36, 27, 72, 55, 43, 32, 45]

(b) Time complexity:

(i) O(n)

(ii) O(log n)

(iii) O(n log n)

Q5. (a) DFS traversal from H: H, D, C, A, B, G, E, I, F

(b) Kruskal's algorithm steps:

BD(3), AB(4), CF(7), FE(8), BC(10), DE(20)

Total cost: 52

Q6. (a) Circular queue operations omitted due to length constraints.

(b) CLL structure after execution:

30 -> 20 -> 10 -> 30

Q7. (a) Algorithm to move negative elements:

```
```

```
i = 0
for j = 0 to n-1
    if arr[j] < 0
        swap(arr[i], arr[j])
        i++
```

(b) Missing lines:

1: q->next = NULL;

2: p->next = head;

3: head = p;

(c) Output: 1 4 7 1 4 7