Semester: 3rd.

Programme: B.Tech

Branch/Specialization: CSE, IT, CSSE, CSCE

AUTUMN END SEMESTER EXAMINATION-2023

3rd Semester, B.Tech

DATA STRUCTURES

CS 21001

(For 2022 Admitted Batches)

Time: 3 Hours **Full Marks: 50**

Answer any SIX questions.
Question paper consists of four SECTIONS i.e. A, B, C and D.
Section A is compulsory.
Attempt minimum one question each from Sections B, C, D.
The figures in the margin indicate full marks.
Candidates are required to give their answers in their own words as far as practicable and all parts of a question should be answered at one place only.

SECTION-A (Learning levels 1 and 2)

1. Answer the following questions.

(a) Write a function with arguments whose worst case time complexity is O(m^2+ n (log m)^2). 1

Ans:

```
void func(int m, int n) {
        int i, j, k;
        for(i=1; i<=m; i++)
                for(j=1; j<=m; j++)
                        statement1;
        for(i=1; i<=n; i++)
                for(j=1; j<=m; j=j*2)
                        for(k=1; k<=m; k=k*2)
                                statement2;
}
```

(b) An array X[-15…10, 15…40] requires two bytes of storage and is presented in row-major format. If the beginning location is 2500, determine the location of X[8][25]. 1

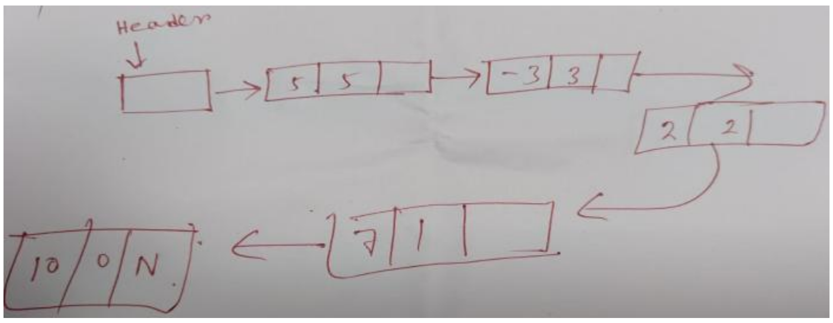
Ans:

Number or rows, m = [10 – (- 15)] +1 = 26
Number or columns, n = [40 – 15)] +1 = 26
Row-Major:
The given values are: B = 2500, W = 2 bytes, i = 8, j = 25, Lr = -15, Lc = 15, n = 26

Address of A [8][25] = BA + w * [n * (i – Lr) + (j – Lc)]
= 2500 + 2* [26 * (8 – (-15)) + (25 – 15)]
= 2500 + 2* [26 * 23 + 10] = 2500 + 2 * [598 + 10]
= 2500 + 2 * 608 = = 2500 + 1216 = 3716

(c) Let P(x) denote the polynomial: P(x) = $5x^5 – 3x^3 + 2x^2 + 7x +10$, having coefficient, exponent, and the formal parameter (i.e., x). Visually depict it using a header linked list by articulating the assumptions, if any. 1

Ans:



(d) Consider a deque of size 6. Illustrate the step-by-step process for insertion of elements 10, 20, 30, 50, 60, and 70 at front end, wherein each step has to capture the current state of the deque, front, and rear. 1

Ans:

Initial Step: Deque is empty.
Deque: [_, _, _, _, _, _]
Front: -1
Rear: -1

Step 1: Insert 10 at the Front
Deque: [10, _, _, _, _, _]
Front: 0
Rear: 0

Step 2: Insert 20 at the front end
Deque: [20, 10, _, _, _, _]
Front: 1
Rear: 0

Step 3: Insert 30 at the front end
Deque: [30, 20, 10, _, _, _]
Front: 2
Rear: 0

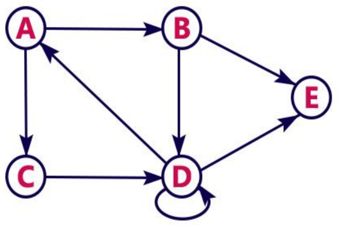
Step 4: Insert 50 at the front end
Deque: [50, 30, 20, 10, _, _]
Front: 3
Rear: 0

Step 5: Insert 60 at the front end
Deque: [60, 50, 30, 20, 10, _]
Front: 4
Rear: 0

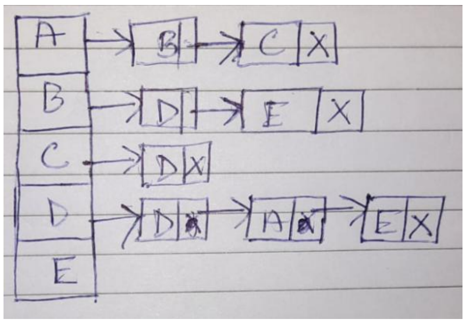
Step 6: Insert 70 at the front end
Deque: [70, 60, 50, 30, 20, 10]
Front: 5
Rear: 0

Now, the deque is full, and further insertions at the front end would result in an overflow.

(e) Let G (as shown below) be the unweighted directed graph with 5 vertices. Represent it using an adjacency list. 1



Ans:



(f) Given an array of characters. Write a function to sort the array using bubble sort. The character comparison should be made using ASCII values. 1

Ans:

```
void sort(char arr[], int n) {
        int i, j;
        char temp;
        for(i = 0; i < n-1; i++) {
                for (j = i+1; j < n; j++) {
                        if (arr[i] > arr[j]) {
                                temp = arr[i];
                                arr[i] = arr[j];
                                arr[j] = temp;
                        }
                }
        }
}
```

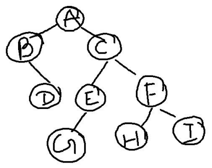
(g) Consider the following In-order and Post-order traversal of a binary tree. Find the preorder sequence by constructing the tree. 1

In-order : B → D → A → G → E → C → H → F → I

Post-order : D → B → G → E → H → I → F → C → A

Ans:

Binary Tree:

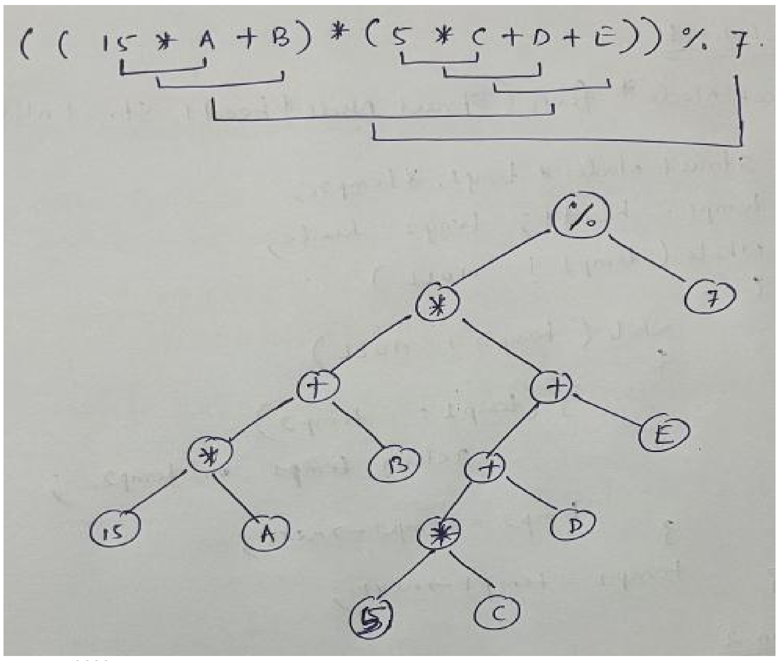


Preorder: A → B → D → C → E → G → F → H → I

(h) Draw the binary tree for the expression ((15*A+B)*(5*C+D+E))%7. 1

(The operators '%' and '*' have same precedence and higher than the precedence of '+')

Ans:



(i) Write a function to count the number of digits of a given integer using recursion. The signature of the function is int noOfDigits(int n), and the function should display an error message if the value of n is less than or equal to 0. 1

Ans: (Nonrecurssive function is also acceptable)

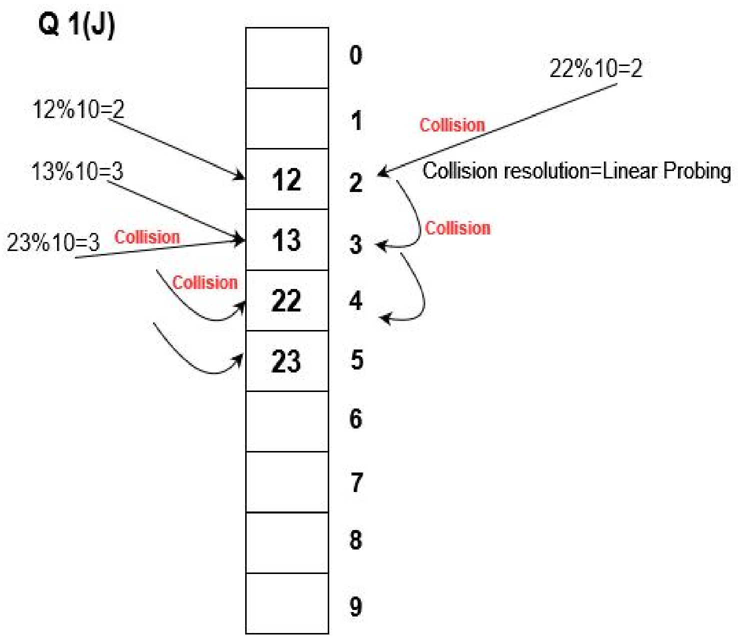
```
int noOfDigits(int n) {
        if(n < 0) {
                printf("Error: Input should be a non-negative integer.\n");
                return -1; // Indicate an error
        }
        // Base case: If n is 0, it has 1 digit
        if(n == 0)
                return 1;
        // Recursive case: Count digits of n/10 and add 1
        return 1 + noOfDigits(n / 10);
}
```

OR

```
int noOfDigits(int n) {
        int count = 0;
        if (n == 0) return 1;
        if (n < 0) n = -n;
        while(n != 0) {
                count++;
                n = n/10;
        }
        return count;
}
```

(j) The keys 12, 13, 22, and 23 are to be inserted into an initially empty hash table of length 10 using linear probing with the hash function h(k) = k mod 10. For each insertion, display the state of the hash table by capturing the key, address, address after linear probing, and the number of probes. 1

Ans:



SECTION-B (Learning levels 1,2, and 3)

2. (a) Write a function to subtract one sparse matrix from another, which is represented in row triplet form. [4]

Ans:

```
typedef struct {
        int *row;
        int *col;
        int *val;
        int nnz;  // Number of non-zero elements
} Sparse;

Sparse subtract(Sparse *mat1, Sparse *mat2) {
        int i, j, row1, col1, val1, val2, val3, flag;
        Sparse mat3;
        mat3.nnz=0;
        mat3.row=(int *)malloc((mat1->nnz + mat2->nnz)*sizeof(int));
        mat3.col=(int *)malloc((mat1->nnz + mat2->nnz)*sizeof(int));
        mat3.val=(int *)malloc((mat1->nnz + mat2->nnz)*sizeof(int));
        for(i = 0; i < mat1->nnz; i++) {
                row1 = mat1->row[i];
                col1 = mat1->col[i];
                val1 = mat1->val[i];
                for(j = 0; j < mat2->nnz; j++) {
                        if(mat2->row[j] == row1 && mat2->col[j] == col1)
                                break;
                }
                // If the element exists in both matrices,
                // subtract the values and add to the matrix3
                if(j < mat2->nnz) {
                        val2 = mat2->val[j];
                        val3 = val1 - val2;
                        // Only add to the matrix3 if the val3 is not zero
                        if(val3 != 0) {
                                mat3.row[mat3.nnz] = row1;
                                mat3.col[mat3.nnz] = col1;
                                mat3.val[mat3.nnz] = val3;
                                mat3.nnz++;
                        }
                }
                else {
                        // If the element only exists in matrix1,
                         // add it to the matrix3
                        mat3.row[mat3.nnz] = row1;
                        mat3.col[mat3.nnz] = col1;
                        mat3.val[mat3.nnz] = val1;
                        mat3.nnz++;
                }
        }
        // Iterate through the elements of matrix2 that do not exist in matrix1
        for(i = 0; i < mat2->nnz; i++) {
                row2 = mat2->row[i];
                col2 = mat2->col[i];
                // Check if the element exists in matrix1
                flag = 0;
                for(j = 0; j < mat1->nnz; j++) {
                        if(mat1->row[j] == row2 && mat1->col[j] == col2) {
                                flag = 1;
                                break;
                        }
                }
                // If the element only exists in matrix2,
                // add it to the matrix3 with negated value
                if(!flag) {
                        mat3.row[mat3.nnz] = row2;
                        mat3.col[mat3.nnz] = col2;
                        // Subtracting matrix2, so negate the value
                        mat3.val[mat3.nnz] = -mat2->val[i];
                        mat3.nnz++;
                }
        }
        return mat3;
}
```

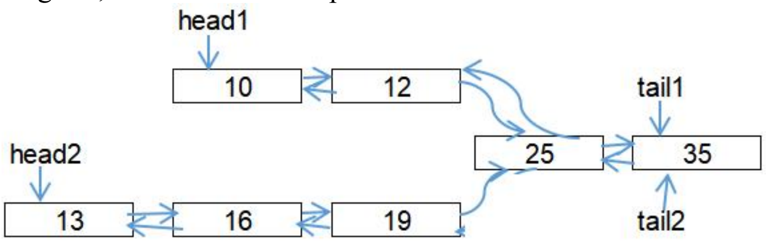
(b) Given a string. Write a function to reverse the order of the words in the given string. [4]
Example:
Input: "I love DS"
Output: "DS love I"
Hint: Use the **strtok** function, which splits a string array according to a given delimiter and returns the next token in the string. However, it needs to be called in a loop to get all tokens and return NULL when there are no more tokens.
Ans:

```
void reverse_words(char *str) {
        int len, beg, end, word_start, word_end;
        char tmp;
        len=strlen(str);
        // Reverse the entire string
        beg = 0;
        end = len - 1;
        while(beg < end) {
                tmp = str[beg];
                str[beg] = str[end];
                str[end] = tmp;
                beg++;
                end--;
        }
        // Reverse each word individually
        beg = 0;
        while(beg < len) {
                // Find the start and end indices of the current word
                while(beg < len && str[beg] == ' ')
                        beg++;
                word_start = beg;
                while(beg < len && str[beg] != ' ')
                        beg++;
                word_end = beg - 1;
                // Reverse the current word
                while(word_start < word_end) {
                        tmp = str[word_start];
                        str[word_start] = str[word_end];
                        str[word_end] = temp;
                        word_start++;
                        word_end--;
                }
        }
}
```

3. (a) Given a pointer to the header node of a singly linked list. Write a function to reverse the list by changing the links between nodes. [4]
Ans:

```
struct Node {
        int data;
        struct Node* next;
};
void reverseLinkedList(struct Node** head) {
        struct Node *prev = NULL, *curr = (*head)->next, *nextNode;
        while(curr != NULL) {
                nextNode=curr->next;
                curr->next=prev;
                prev=curr;
                curr=nextNode;
        }
        struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
        *head=newNode;
        newNode->next=prev;
}
```

(b) There are two doubly linked lists. By some programming error, the end node of one of the double linked lists got linked to the second list, forming an inverted Y-shaped list (see the below figure for reference). Write a function to display the value of the node where two double linked lists merge. As per the below diagram, the inverted Y-shape occurs at the node value 25. [4]



Ans:

```
typedef struct Node {
        int data;
        struct Node* prev;
        struct Node* next;
} Node;

// Function to find the merging point of two doubly linked lists
Node* findMergePoint(Node* head1, Node* head2) {
        // Function to get the length of a doubly linked list
        int getLength(Node* head) {
                int len = 0;
                while(head != NULL) {
                        len++;
                        head=head->next;
                }
                return len;
        }
        int len1, len2, diff
        len1 = getLength(head1);
        len2 = getLength(head2);
        diff = abs(len1 - len2);
        // Advance the longer list's pointer by the difference in lengths
        while(diff > 0) {
                if(len1 > len2)
                        head1 = head1->next;
                else
                        head2 = head2->next;
                diff--;
        }
        // Traverse both lists together until a common node is found
        while(head1 != NULL && head2 != NULL) {
                if(head1 == head2)
                        return head1;  // Merging point found
                head1 = head1->next;
                head2 = head2->next;
        }
        return NULL;  // No merging point found
}
```

SECTION-C (Learning Levels 3 and 4)

4. (a) Convert the following infix expression to its corresponding postfix expression using the suitable data structure. Illustrate neatly each step of the conversion. [4]

A + (B * C – (D / E ^ F) * G) * H

The precedence of ‘^’ is highest followed by the ‘*’ and ‘/’ which are same precedence. The precedence of ‘+’ and ‘-’ are same and it is lowest.

Ans:

Given infix expression is A + (B * C – (D / E ^ F) * G) * H

Expression will be scanned from *Left to Right*.

Scanned Symbol	Stack	Output Expression
A		A
+	+	A
(	+ (	A
B	+ (	A B
*	+ (*	A B
C	+ (*	A B C
-	+ (-	A B C *
(	+ (- (	A B C *
D	+ (- (	A B C * D
/	+ (- (/	A B C * D
E	+ (- (/	A B C * D E
^	+ (- (/ ^	A B C * D E
F	+ (- (/ ^	A B C * D E F
)	+ (-	A B C * D E F ^ /
*	+ (- *	A B C * D E F ^ /
G	+ (- *	A B C * D E F ^ / G
)	+	A B C * D E F ^ / G * -
*	+ *	A B C * D E F ^ / G * -
H	+ *	A B C * D E F ^ / G * - H
None	Empty Stack	A B C * D E F ^ / G * - H * +

Postfix expression is A B C * D E F ^ / G * - H * +

(b) Create two stacks using a static array i.e., both stacks will use the same array for storing elements, and each stack must grow towards each other. Write the following functions to implement two stacks: [4]

1. push1(int x) –> pushes x to the first stack.
2. push2(int x) –> pushes x to the second stack.
3. int pop1() –> pops an element from the first stack and returns the popped element.
4. int pop2() –> pops an element from the second stack and returns the popped element.

Ans:

```
int top1 = -1, top2 = n;
// Method to push an element x to stack1
void push1(int x) {
        if(top1 == top2-1) {
                printf("Stack Overflow...");
                return;
        }
        top1++;
        stack[top1] = x;
}

// Method to push an element x to stack2
void push2(int x) {
        if(top1 == top2-1) {
                printf("Stack Overflow...");
                return;
        }
        top2--;
        stack[top2] = x;
}
```

```
// Method to pop an element from first stack1
int pop1() {
        int x;
        if(top1 == -1)  {
                printf("Stack Underflow...");
                return -9999;
        }
        x = stack[top1];
        top1--;
        return(x);
}

// Method to pop an element from second stack2
int pop2() {
        int x;
        if(top2 == n)  {
                printf("Stack Underflow...");
                return -9999;
        }
        x = stack[top2];
        top2++;
        return(x);
}
```

5. (a) Write a function to determine whether the two binary trees are identical or not. Two binary trees are identical when they have the same data with exactly the same arrangement of data. [4]

Examples:

Input:

```
        1              1
       / \            / \
      2   3          2   3
     /              /
    4              4
```

Output: Both binary trees are identical

Input:

```
        1              1
       / \            / \
      5   3          2   3
     /              /
    4              4
```

Output: Binary trees are not identical

Ans:

```
int isIdentical(Node* root1, Node* root2) {
        // Check if both the trees are empty
        if(root1 == NULL && root2 == NULL)
                return 1;
        // If any one of the tree is non-empty and other is empty, return false
        else if(root1 == NULL || root2 == NULL)
                return 0;
        else {
        // Check if current data of both trees equal as well as both left and
        // right sub-trees are equal (recursively)
                if(root1->data == root2->data &&
                isIdentical(root1->left, root2->left)
                && isIdentical(root1->right, root2->right))
                        return 1;
                else
                        return 0;
        }
}
```

(b) Write the recursive functions for finding the minimum and subsequently, [4] maximum key in the binary search tree.

<u>Ans:</u>

```
// Recursive function to find the minimum key in the BST
int findMin(struct Node* root) {
        if (root == NULL) {
                printf("Error: Tree is empty\n");
                return -1; //value indicating an error
        }
        if (root->left == NULL)
                return root->key;
        return findMin(root->left);
}

// Recursive function to find the maximum key in the BST
int findMax(struct Node* root) {
        if (root == NULL) {
                printf("Error: Tree is empty\n");
                return -1; //value indicating an error
        }
        if (root->right == NULL)
                return root->key;
        return findMax(root->right);
}
```

6. (a) Why do we need an AVL tree? Insert the given keys in the order: 70, 65, 85, 60, 80, 90, 75, 95 into an empty AVL tree. Clearly Illustrate the AVL tree after each insertion, showing the resulting tree structure, the balance factor at each node and the rotations performed at each step to maintain AVL tree. [4]

<u>Ans:</u>

AVL trees are a type of self-balancing binary search tree that maintains a balance factor at each node. The balance factor is the difference between the heights of the left and right subtrees. AVL trees ensure that the balance factor of every node is within the range [-1, 0, 1], which guarantees a logarithmic height and efficient search, insertion, and deletion operations.

Here's the step-by-step illustration of inserting keys (70, 65, 85, 60, 80, 90, 75, 95) into an empty AVL tree:

Insert 70:

```
   70
```

Balance factor = 0 (node 70), so no rotations needed.

Insert 65:

```
     70
    /
   65
```

Balance factor = 0 (node 65), 1 (node 70), no rotations needed.

Insert 85:

```
     70
    /  \
   65   85
```

Balance factor = 0 (node 65), 0 (node 85), 0 (node 70), no rotations needed.

Insert 60:

```
      70
     /  \
    65   85
   /
  60
```

Balance factor = 0 (node 60), 1 (node 65), 0 (node 85), 1 (node 70), no rotations needed.

Insert 80:

```
          70
        /    \
      65     85
     /      /
   60     80
```

Balance factor = 0 (node 60), 1 (node 65), 0 (node 80), 1 (node 85), 0 (node 70), no rotations needed.

Insert 90:

```
          70
        /    \
      65     85
     /      /  \
   60     80  90
```

Balance factor = 0 (node 60), 1 (node 65), 0 (node 80), 0 (node 90), 0 (node 85), 0 (node 70), no rotations needed.

Insert 75:

```
          70
        /    \
       65     85
      /      /  \
    60    80    90
          /
        75
```

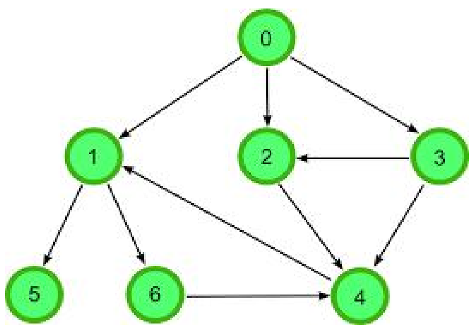
Balance factor = 0 (node 60), 1 (node 65), 0 (node 75), 1 (node 80), 0 (node 90), 1 (node 85), -1 (node 70), no rotations needed.

Insert 95:

```
          70
        /    \
       65     85
      /      /  \
    60    80    90
          /        \
        75          95
```

Balance factor = 0 (node 60), 1 (node 65), 0 (node 75), 1 (node 80), 0 (node 95), -1 (node 90), 0 (node 85), -1 (node 70), no rotations needed.

(b) Let G (as shown below) is the undirected graph. Show the step-by-step process to display the output sequence using breadth-first and depth-first searches, considering node with the value of 0 as the starting point. [4]



Ans:

Starting node: 0

Breadth-First-Search

The steps are as follow:

1. Initialize all nodes to the Ready state.

   Ready State: 0, 1, 2, 3, 4, 5, 6
   Waiting State: None
   Processed State: None
   Queue: Empty

2. Insert node 0 into the queue and change its status to waiting

   Ready State: 1, 2, 3, 4, 5, 6
   Waiting State: 0
   Processed State: None
   Queue: 0

3. Remove the front node 0, process it and change its status to the processed state and insert into queue all the neighbor nodes of 0 those are in Ready State.

   Ready State: 4, 5, 6
   Waiting State: 1, 2, 3
   Processed State: 0
   Queue: 1, 2, 3

4. Remove the front node 1 from the queue, process it and change its status to Processed State, and insert into queue all the neighbors of 1 those are in the Ready State

   Ready State: 4
   Waiting State: 2, 3, 5, 6
   Processed State: 0, 1
   Queue: 2, 3, 5, 6

5. Remove the front node 2 from the queue, process it and change its status to Processed State, and insert into queue all the neighbors of 2 those are in the Ready State.

   Ready State: None
   Waiting State: 3, 4, 5, 6
   Processed State: 0, 1, 2
   Queue: 3, 5, 6, 4

6. As there is no node in the Ready Queue, so subsequent nodes will be removed from the queue and will be processed sequentially.

   The sequence of traversal is same as of processing: 0, 1, 2, 3, 5, 6, and 4

<u>Depth-First-Search</u>

The steps are as follows:

1. Initialize all nodes to the Ready state.

   Ready State: 0, 1, 2, 3, 4, 5, 6
   Waiting State: None
   Processed State: None
   Stack: Empty

2. Push node 0 into the Stack and change its status to waiting

   Ready State: 1, 2, 3, 4, 5, 6
   Waiting State: 0
   Processed State: None
   Stack: 0

3. Pop node 0 from the top of the stack, process it, change its state to processed state, and push onto stack all the neighbors of 0 those are in

Ready State.

Ready State: 4, 5, 6
Waiting State: 1, 2, 3
Processed State: 0
Stack: 1, 2, 3

4. Pop node 3 from top of the stack, change its status to processed state, and push neighbors of node 3 onto stack, those are in the Ready State. So, node 4 is pushed onto the stack.

   Ready State: 5, 6
   Waiting State: 1, 2
   Processed State: 0, 3
   Stack: 1, 2, 4

5. Pop node 4 from the top of the stack, change its status to processed state, and push its neighbors onto stack, those are in the Ready State. No node is pushed

   Ready State: 5, 6
   Waiting State: 1, 2,
   Processed State: 0, 3, 4
   Stack: 1, 2

6. Pop node 2 from the top of the stack, change its status to processed state, and push its neighbors onto stack, those are in the Ready State. No node is pushed.

   Ready State: 5, 6
   Waiting State: 1
   Processed State: 0, 3, 4, 2
   Stack: 1

7. Pop node 1 from the top of the stack, change its status to processed state, and push its neighbors onto stack, those are in the Ready State. Node 5 and 6 are pushed onto the stack.

   Ready State: None
   Waiting State: 5, 6
   Processed State: 0, 3, 4, 2, 1
   Stack: 5, 6

Then node 6 and 5 will be popped from the stack, no node will be pushed onto the stack, and the final traversal sequence will be 0, 3, 4, 2, 1, 6, and 5.

SECTION-D (Learning levels 4,5,6)

7. (a) What constitutes a good hash function? Demonstrate the insertion of keys: 50, 700, 76, 85, 92, 73, 102, 150, 783, 23, 167, 92, and 23 into a hash table with the separate chaining collision resolution technique. The hash table has 10 slots, and it uses the division method. [4]

Ans:

A good hash function should have the following properties:

- Deterministic: For a given input, the hash function should always produce the same output.
- Efficient: It should be computationally efficient to compute the hash value for any given input.
- Uniformity: It should distribute the keys uniformly across the hash table, reducing the likelihood of collisions.
- Avalanche Effect: A small change in the input should produce a significantly different hash value.
- No Reversibility: It should be computationally infeasible to reverse the hash value to obtain the original input.

Insertion of keys: 50, 700, 76, 85, 92, 73, 102, 150, 783, 23, 167, 92, and 23 into a hash table with separate chaining collision resolution using the division method. The division method involves taking the remainder when dividing the key by the number of slots in the hash table.

(i) Insertion of 50

Hash (50) = 50 % 10 = 0

50 will be inserted into slot '0'.

(ii) Insertion of 700

Hash (700) = 700 % 10 = 0 (Collision)

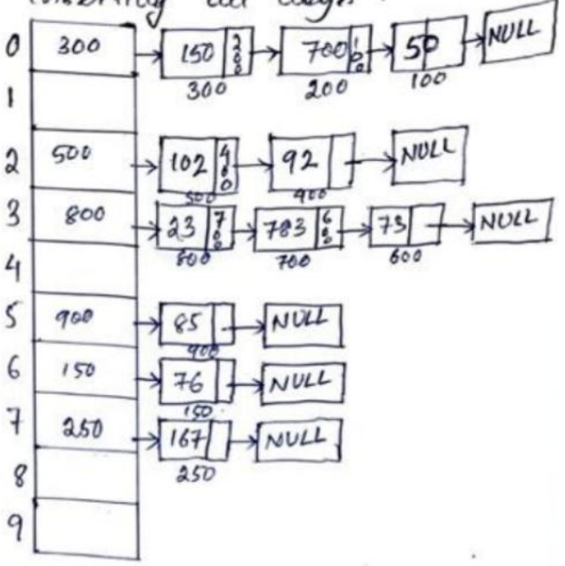
700 will be inserted into slot '0'.

(iii) Insertion of 76

Hash (76) = 76 % 10 = 6

76 will be inserted into slot 6.

The final hash table after inserting all keys.



(iv) Insertion of 85

Hash (85) = 85 % 10 = 5

85 will be inserted into slot 5

(v) Insertion of 92

Hash (92) = 92 % 10 = 2

92 will be inserted into slot 2.

(vi) Insertion of 73

Hash (73) = 73 % 10 = 3

73 will be inserted into slot 3

(vii) Insertion of 102

Hash (102) = 102 % 10 = 2 (Collision)

102 will be inserted into slot 2.

(viii) Insertion of 150

Hash (150) = 150 % 10 = 0 (Collision)

150 will be inserted into slot '0'

(ix) Insertion of 783

Hash(783) = 783 % 10 = 3 (Collision)

783 will be inserted into slot

(x) Insertion of 23 :

Hash (23) = 23 % 10 = 3 (Collision)

23 will be inserted into slot 3

(xi) Insertion of 167 : —

Hash(167) = 167 % 10 = 7

167 will be inserted into slot 7.

(xii) Insertion of 92 : —

92 is already inserted.

No need to insert again.

(xiii) Insertion of 23 : —

92 is already inserted.

No need to insert again.

(b) In reference to quadratic probing, write functions for insertion and search a key. [4]

Ans:

```
void hashing_insert(int table[], int size, int arr[], int n) {
        int i, j, ind, new_ind
        for(i = 0; i < n; i++) {
                ind = arr[i] % size;
                if(table[ind] == -1)
                        table[ind] = arr[i];
                else {
                        for(j = 0; j < size; j++) {
                                new_ind = (ind + j * j) % size;
                                if(table[new_ind] == -1) {
                                        table[new_ind] = arr[i];
                                        break;
                                }
                        }
                }
        }
        display(table, size);
}

void hashing_Search(int x, int table[], int size) {
        int j, ind, new_ind
        ind= x % size;
        if(table[ind]==x) {
                printf("Element is found ");
                break;
        }
        else  {
                for(j = 1; j < size; j++) {
                        new_ind = (ind + j * j) % size;
                        if(table[new_ind]==x) {
                                printf("Element is found");
                                break;
                        }
                }
        }
        printf("Element is not found");
}
void display(int arr[], int size) {
        int i;
        for(i = 0; i < size; i++)
                printf("%d\n", arr[i]);

}
```

8. (a) Describe an optimized version of the Bubble Sort algorithm that includes enhancements to reduce the number of unnecessary comparisons and swaps. Provide the modified algorithm and analyze the impact on its time complexity. [4]

Ans:

Bubble Sort is a simple and commonly used sorting algorithm. However, the most of the codes available exhibits higher time complexity. An optimized version of the Bubble Sort algorithm reduces the time complexity and improves overall performance. By understanding the algorithm's weaknesses and implementing efficient techniques, we can achieve better results.The Bubble Sort algorithm compares adjacent elements and swaps them if they are in the order in which the elements need to be sorted. This process is repeated until the entire array is sorted. While Bubble Sort is straightforward to implement, its efficiency can be improved.

```
void bubbleSort(int arr[], int size) {
        int itr, j, temp, flag;
        for(itr = 0; itr < (size-1); itr++) {
                flag=0;
                for(j = 0; j < (size-itr-1); j++) {
                        if(arr[j] > arr[j + 1]) {
                                temp = arr[j];
                                arr[j] = arr[j + 1];
                        arr[j + 1] = temp;
                        flag=1;
                    }
                }
                if (flag == 0)
                    break;
        }
}
```

Improving the Time Complexity: The existing code implements the Bubble Sort algorithm correctly, but it can be optimized to reduce the time complexity. Here are the modifications we can make to improve performance:

Advantages:

- ✓ Reduce the number of iterations
- ✓ Early termination
- ✓ These optimizations help improve the performance of the bubble sort algorithm, making it more efficient for sorting small-sized arrays or partially sorted arrays.

Conclusions: In the worst case, the algorithm requires nested loops. The outer loop runs for n iterations, and the inner loop also runs for n iterations in the first pass, then n-1 in the second pass, and so on until it runs for 1 iteration in the last pass. The number of comparisons and swaps in the worst case is proportional to the sum of the first n integers, which is (n * (n - 1)) / 2, leading to O(n^2) time complexity. While the algorithm has the potential to terminate early if the array is partially sorted, the worst-case time complexity remains the same as basic Bubble Sort. The algorithm may perform better for partially sorted or nearly sorted arrays. However, the average-case performance is improved, especially for partially sorted arrays. In the best-case scenario (when the array is already sorted), the time complexity is O(n), as the algorithm only makes one pass through the array without any swaps.

(b) What is the divide & conquer principle? Explain it in the context of Merge sort with a suitable example. [4]

Ans:

The divide and conquer is one of the problem-solving techniques that involves breaking down a complex problem into smaller sub-problems till smallest sub-problem reach. Then solving each sub-problem independently, and then combining their individual solutions to solve the original problem. In the context of Merge Sort, the divide and conquer technique is applied to sort an array by recursively dividing it into smaller halves till single element. Then sorting each half independently by merging the sorted halves to produce the final sorted array.

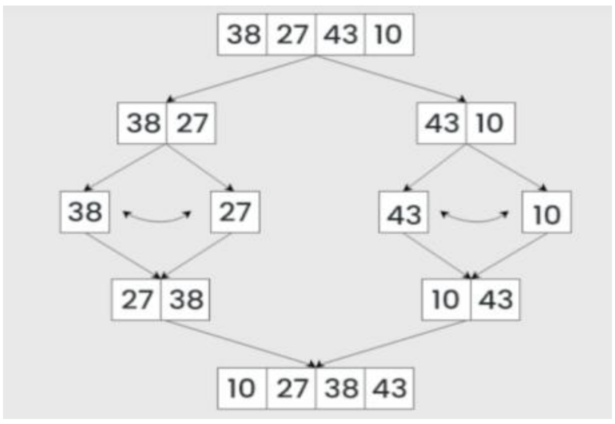
Example:

Consider an unsorted array: [38, 27, 43, 10].

i. Divide:
The array is divided into two halves: [38, 27] and [43, 10].
Further divisions occur until individual elements are considered sorted.

ii. Conquer:
Each subarray is sorted independently, yielding: [27, 38] and [10, 43].

iii. Combine (Merge):
The two sorted subarrays are merged to produce the final sorted array: [10, 27, 38, 43].


38 27 43 10
38 27
43 10
38
27
43
10
27 38
10 43
10 27 38 43

*****