Jorhat Engineering College

Laboratory Manual

Data Structures & Algorithms Lab

For

B.Tech III Semester

Department of Computer Science and Engineering

Department Vision

To become a prominent department of Computer Science and Engineering for producing quality human resources to meet the needs of the industry and society

Department Mission

- To impart quality education through well-designed curriculum and academic facilities to meet the computing needs of the industry and society
- To inculcate the spirit of creativity, team work, innovation, entrepreneurship and professional ethics among the students
- To facilitate effective interactions to foster networking with alumni, industries, institutions of learning and research and other stake-holders
- To promote research and continuous learning in the field of Computer Science and Engineering

PROGRAM OUTCOMES	
PO No.	**Program Outcome Description**
PO 1	**Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
PO 2	**Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
PO 3	**Design / Development of solution:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
PO 4	**Conduct investigation of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
PO 5	**Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
PO 6	**The engineer & society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
PO 7	**Environment & sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
PO 8	**Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
PO 9	**Individual & team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
PO 10	**Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
PO 11	**Project management & finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
PO 12	**Life long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Program Specific Outcomes	
PSO 1	Gain ability to employ modern computer languages, environments and platforms in creating innovative career paths
PSO 2	Achieve an ability to implement, test and maintain computer based system that fulfils the desired needs

Lab Outcomes	
LO1	write effective algorithms
LO2	Identify which data structure that efficiently model the information in a problem
LO3	Identify benefits of data structures and their applications.
LO4	demonstrate searching and sorting techniques
LO5	demonstrate concepts of Stack, Queue, Linked List, Tree, Graph

SUBJECT INDEX

DOs and DON'Ts in Laboratory:

1. Make entry in the Log Book as soon as you enter the Laboratory.

2. All the students are supposed to enter the terminal number in the log book.

3. Do not change the terminal on which you are working.

4. All the students are expected to get at least the algorithm of the program/concept to be implemented.

5. Strictly observe the instructions given by the teacher/Lab Instructor.

Lab Exercise 1 (2 Hours)

Title: To implement a character stack data type and use it to reverse a string.

Objectives: To study the stack datatype

Input: Read a string.

Output: Print String in reverse way.

Algorithm:

```
Step1: Input a String.
Step2: Calculate the length of a string.
Step3: Push all the characters one at a time into Stack.
          Continue step3 till length of a string
Step4: Pop all the characters one at a time from the stack into same string
          Continue step4 till length of a string.
Step5: Print
a string.
Step6: End.
```

Lab Exercise 2 (2 Hours)

Title: To write a program, using a queue data type, to simulate a bank where customers are served on a first-come- first-serve basis

Objectives: To study queue datatype

Note:- Insert all bank customers details at the start.

Algorithm: to add customers into

the queue Steps:

Step1:Start
Step2: Ask customer bank account
number. Step3: Insert into the queue
by incrementing rear. Step4: End

Algorithm: toprocess
customers Steps:

Step1: Start.
Step2: Get the account number from queue by incrementing front to
process. Step3: End

Algorithm: to deposit Steps:

Step1: Start.

Step2: Enter the amount to be deposited. Step3: update balance set bal=bal-amt;
Step4: End

Algorithm: to withdraw Steps:

Step1: Start.

Step2: Enter the amount to be withdrawn. Step3: update balance set bal=bal+amt;
Step4: End

Lab Exercise 3 (2 Hours)

Title: Program for implementing singly linked list operations.

Concatenate two linked list and create third one

Free all nodes In a linked list

Reverse a linked list

Given two linked list, create a third list which is intersection of elements in the two.

Objectives: To study Link list operations

Algorithm: Creating first linked

list Steps:

```
Step1 :Start
Step2: Allocate memory for the new node.
                  Temp=malloc()
Step3: Assign the value to the data field of the new node.
                   Temp->info=ele
Step4: Make the link field of the new node to point to the
                   starting node of the linked list.
                          Temp-
>next=NULL       Step4:
Chk if head is NULL.
             If  yes  then  say  head
=   temp   Otherwise   perform
setp 5.
Step5: Go till r->next is
            Not NULL.
            Then r->next =
            temp
Step6: End
```

Algorithm: Creating second linked

list Steps:

```
Step1 :Start
Step2: Allocate memory for the new node.
                  Temp=malloc()
Step3: Assign the value to the data field of the new node.
                   Temp->info=ele
Step4: Make the link field of the new node to point to the
```

```
                  starting node of the linked list.
                        Temp-
>next=NULL       Step4:
Chk if head1 is NULL.
            If yes then say head1 =
temp Otherwise perform setp 5.
Step5: Go till r->next is
           Not NULL.
           Then r->next =
           temp
Step6: End
```

Algorithm: To Concatenate above two

linked list Steps:

```
Step1:Set head2=NULL,
temp=head Step2:Go till temp is
Not NULL
          Allocate memory for the new
          node. Temp1=malloc()
         Chk head2 is NULL
         If yes then
head2=temp1;
Otherwise perform setp
3.
Step3: Go till r->next is
           Not NULL. Then
           r->next =
           temp1
Step4: Assign
temp=head1 Step5:Go
till temp is Not NULL
          Allocate memory for the new
          node. Temp1=malloc()
         Chk head2 is NULL
         If yes then
head2=temp1;
Otherwise perform
step3
Step6: Go till r->next is
           Not NULL. Then
           r->next =
           temp1
Step7:End
```

Algorithm: To delete all node from linked list Steps:

Step1:Start

Step2: Go till ((r=head)!=NULL) Say head=head->next;

f ree(r);

Step3:End

Algorithm: To reverse a linked list Steps:

Step1: Start

Step2: Set cur = head; prev = NULL;

nxt= NULL;

Step3: Go till cur != NULL

// Store next

nxt = cur->next;

// Reverse current node's pointer cur->next = prev;

// Move pointers one position ahead. prev =

cur; cur = nxt; Step4:Set head = prev;

Algorithm: Intersection of two linked lists

Note:- pass head of both linked list as input struct Node *getIntersection(struct Node *head1,

struct Node *head2)

{

struct Node *result = NULL; struct Node *t1 = head1;

// Traverse list1 and search each element of it in

// list2. If the element is present in list 2, then

// insert the element to result while (t1 != NULL)

```
{

if (isPresent(head2, t1->data)) // to find out common element push (&result, t1->data);

t1 = t1->next;

}

return result;

}

void push (struct Node** head_ref, int new_data)

{

/* allocate memory for new node */ struct Node* new_node =

(struct Node*) malloc(sizeof(struct Node));

/* put in the data */ new_node->data = new_data;

/* link the old list off the new node */ new_node->next = (*head_ref);

/* move the head to point to the new node */ (*head_ref) = new_node;

}

bool isPresent (struct Node *head, int data)

{

struct Node *t = head; Go till (t != NULL)

{

Check is t->data = data return 1;

t = t->next;

}

return 0;

}
```

Lab Exercise 4 (2 Hours)

Title : Program for deleting every third element from the linked list.

Algorithm:

```
Steps:
Step1:Start
Step2: if head is NULL return NULL;
Step3: set ptr = head, prev = NULL; Step4:
// Traverse list and delete every 3rd node int count = 0;
Go till (ptr != NULL)
{
// increment Node count count++;
// check if count is equal to k
// if yes, then delete current Node if (count==3)
{
// put the next of current Node in
// the next of previous Node delete(prev->next);
prev->next = ptr->next;
// set count = 0 to reach further count = 0;
}
// update prev if count is not 0  if (count !=  0)
prev = ptr;
ptr = prev->next;
}
Step5: return head;
Step6: End
```

<u>Lab Exercise 5 (2 Hours)</u>

Title : Programs to implement queue using doubly linked list.

Objective : To study doubly link list

Algorithm: Insert a node in liked list. Steps:

Step1: Allocate memory for the new node.

Temp=malloc()

Step2: Assign the value to the data field of the new

node. Temp->info

Step3: Chk if head=NULL then head=temp Otherwise

r=head;

while(r->next!=NULL) r=r->next;

r->next=temp;

Step4:End

Algorithm: Delete a node from linked list.

Step1: Start

Step2: If list is empty then display list is empty Otherwise

r=head; no=r->no;

head=head->next; free(r);

Step3:End

Algorithm: To display a linked list Step1:Start

Step2: Chk list is empty then display list is empty Otherwise

r=head; while(r!=NULL)

{

cout<<r->no<<endl; r=r->next;

}

<u>Lab Exercise 6 (2 Hours)</u>

Title : To create a Tree and implement traversal techniques (In-order, Pre-order, Post-order)

Objective : To study Tree traversal technique

Algorithm: Inorder

Algorithm Inorder(tree)

1. Traverse the left Subtree, i.e., call Inorder(left-subtree)

2. Visit the root.

3. Traverse the right subtree, i.e., call Inorder(right-subtree)

Algorithm:Preorder

Algorithm Preorder(tree)

1. Visit the root.

2. Traverse the left subtree, i.e., call Preorder(left-subtree)

3. Traverse the right subtree, i.e., call Preorder(right-subtree)

Algorithm:Postorder

Algorithm Postorder(tree)

1. Traverse the left subtree, i.e., call Postorder(left-subtree)

2. Traverse the right subtree, i.e., call Postorder(right-subtree)

3. Visit the root.

Lab Exercise 7 (2 Hours)

Title : To create a Graph and implement traversal techniques (BFS & DFS).

Objective : To study Graph and implement traversal techniques

Algorithm

We use the following steps to implement DFS traversal...

- Step 1 - Define a Stack of size total number of vertices in the graph.
- Step 2 - Select any vertex as starting point for traversal. Visit that vertex and push it on to the Stack.
- Step 3 - Visit any one of the non-visited adjacent vertices of a vertex which is at the top of stack and push it on to the stack.
- Step 4 - Repeat step 3 until there is no new vertex to be visited from the vertex which is at the top of the stack.
- Step 5 - When there is no new vertex to visit then use back tracking and pop one vertex from the stack.
- Step 6 - Repeat steps 3, 4 and 5 until stack becomes Empty.
- Step 7 - When stack becomes Empty, then produce final spanning tree by removing unused edges from the graph

We use the following steps to implement BFS traversal...

- Step 1 - Define a Queue of size total number of vertices in the graph.
- Step 2 - Select any vertex as starting point for traversal. Visit that vertex and insert it into the Queue.
- Step 3 - Visit all the non-visited adjacent vertices of the vertex which is at front of the Queue and insert them into the Queue.
- Step 4 - When there is no new vertex to be visited from the vertex which is at front of the Queue then delete that vertex.
- Step 5 - Repeat steps 3 and 4 until queue becomes empty.
- Step 6 - When queue becomes empty, then produce final spanning tree by removing unused edges from the graph

Lab Exercise 8 (2 Hours)

Title : To implement Bubble, Selection and Insertion sort.

Objective : To study sorting technique

Algorithm:

```
//selection sort
public static void selectionSort(int[] arr)
{
            // find the smallest element starting from position i
             for (int i = 0; i < arr.length - 1; i++)
             {
                int min = i;  // record the position of the smallest
                for (int j = i + 1; j < arr.length; j++)
                {
                    // update min when finding a smaller element
                    if (arr[j] < arr[min])
                        min = j;
                }
                 // put the smallest element at position i
                swap(arr, i, min);
             }
}
public static void swap (int[] arr, int i, int j)
{
             int temp = arr[i];
             arr[i] = arr[j];
             arr[j] = temp;
}
//insertion sort
public static void insertionSort(int[] arr)
{
    for (int i = 1; i < arr.length; i++)
    {
        // a temporary copy of the current element
        int tmp = arr[i];
        int j;
         // find the position for insertion
        for (j = i; j > 0; j--)
        {
            if (arr[j - 1] < tmp)
                break;
            // shift the sorted part to right
            arr[j] = arr[j - 1];
        }
         // insert the current element
        arr[j] = tmp;
    }
}
//bubble sort
public static void bubbleSort (int[] data)
{
    for (int i = data.length - 1; i >= 0; i--)
    {
        // bubble up
        for (int j = 0; j <= i - 1; j++)
        {           if (data[j] > data[j + 1])
                swap(data, j, j + 1);          }
    } }
```

Lab Exercise 9 (2 Hours)

Title : To implement recursive Binary Search.
Objective : To study searching technique

Algorithm:

```
int binarySearch(int arr[], int l, int r, int x)

{

    if (r >= l) {

        int mid = l + (r - l) / 2;

          // If the element is present at the middle

        // itself

        if (arr[mid] == x)

            return mid;

          // If element is smaller than mid, then

        // it can only be present in left subarray

        if (arr[mid] > x)

            return binarySearch(arr, l, mid - 1, x);

          // Else the element can only be present

        // in right subarray

        return binarySearch(arr, mid + 1, r, x);

    }

      // We reach here when element is not

    // present in array

    return -1;

}
```

Lab Exercise : 10 (2 Hours)

Title: Programs for recursive functions for a singly linked NULL terminated list Insert() , traverse(), search() .

Objective: To understand the recursion technique

Note:- 2 parameters passed 1:-head 2:-no to be inserted node* newNode(int no)

{

node *new_node = new node; new_node->no = no; new_node->next = NULL; return new_node;

}

Algorithm: Insert newnode . Step1:Start

Step2: if s is NULL then return newNode(no); Otherwise

{

s->next= call same function by passing(s->next,no);

}

cout<<"node attached"<<endl; return head;

Step3:End

Algorithm: Traverse all node

Note:- head passed to the function as argument Step1:Start

Step2: if (s == NULL)

return ;

Otherwise

cout << s->no << " ";

call same function by passing(s->next);

Algorithm: Search a number in linked list

Note:- Head and number to search passed Steps:

```
Step1:Start

Step2: if (s == NULL) return 0 ;

else if (s->no==no) return 1;

else

call same function by passing  (s->next,no);
```