LAB NO: 1 **Date:**

REVIEW OF FUNDAMENTAL DATA STRUCTURES

Objectives:

In this lab, student will be able to:

- recall the concepts learnt in Data Structures Lab
- implement basic data structures

Description: Data Structures specify the structure of data storage in a program. Various data structures namely **arrays, stacks, queues, linked lists, trees** are used for storing data. Each data structure is different from the other in its fundamental way of storing. In **arrays**, a contiguous piece of memory is allocated for storing data. **Static** and **Dynamic arrays** are two types of array which differ by the instance at which memory is allocated. In **static arrays**, memory is allocated at compile time of the program whereas in **Dynamic arrays** memory is allocated at run time. **Dynamic arrays** overcome the problem of unnecessary wastage of memory space. **Stack** is a data structure in which the insertion to the **Stack** (called as push) and removal from the **stack** (called as pop) operations are performed in Last In First Out (LIFO) order. LIFO specifies that the last item to be pushed is the first one to be popped. **Queue** is a data structure in which the insertion to the queue (enque) and removal of element from the queue (Dequeue) happen in the same order. This means it follows First In First Out (FIFO) order. Linked lists store data in non-linea manner. A node of a **linked list** is created at run time and is used to store data element. Nodes of a **linked list** will be allocated memory at run time and the nodes can be anywhere in memory. **Singly linked** list and **doubly linked** lists are two broad types of **linked lists**. **Single linked list** has a single pointer to the next node whereas doubly linked list has two pointers one to the left node and other to the right node. A special value NULL will be used to denote the non-existence of node. **Trees** are very useful in specific storage requirements of graphs, dictionaries etc. **Binary tree** is a special form of trees in which every node can have maximum two children.

I. SOLVED EXERCISE:

1) Write an algorithm and program to implement a doubly linked list which supports the following operations
   i. Create the list by adding each node at the front.
   ii. Insert a new node to the left of the node whose key value is read as an input.
   iii. Delete all occurrences of a given key, if it is found. Otherwise, display appropriate message.
   iv. Search a node based on its key value.
   v. Display the contents of the list.

Description : Doubly linked list is a data structure in which the data elements are stored in nodes and the nodes are connected by two links. Out of two links one link points to the neighboring node in the left direction and the other link to the node in the right direction. Addresses of nodes will be used to represent the node. A special value NULL is used to represent the absence of a node. Creating the doubly linked list, insertion of an element

to the left/right of any node, deletion of all nodes with specific node content and displaying all nodes are the operations commonly performed on a doubly linked list. Each of the operations consumes certain amount of time and memory. Hence they differ in time and space efficiency.

Algorithm: Doubly Linked list
Define a structure to hold list node
Define two links within the node one for left link and the other for rlink

```
CreateList(int val)
begin
        if head == NULL then
                node = allocate memory for a Node
                node ->llink = node->rlink = NULL
                node->val = val
                head=node
        else
                print "List is already created …"
        end if
end

insertIntoList(int before, int val)
begin
        node=head
        while node->val != before
                node = node->rlink
        end while
        if node != NULL then
                newNode = allocate memory for a node
                newNode->val = val
                if node->llink != NULL then
                        node->llink->rlink = newNode
                        newNode->llink = node->llink
                        newNode->rlink = node
                        node->llink = newNode
                else
                        newNode->rlink = node
                        node->llink = newNode
                             head = newNode
                end if
        else
                print "Unable to insert, node with value " val "not found"
                return
        end if
end

deleteAll(int delVal)
```

```
begin
        node = head
        while node != NULL
                if node->val == delVal
                        if node->llink != NULL then
                                node -> llink ->rlink = node -> rlink
                                if node->rlink != NULL then
                                        node->rlink->llink = node->llink
                                        node = node->rlink
                                else
                                        node->llink->rlink = NULL
                                        node=NULL
                                end if
                         else
                                if node->rlink != NULL then
                                        node ->rlink->llink = NULL
                                        head = node->rlink
                                        node = head
                                        release memory for node
                                else
                                        head = node = NULL
                                        release memory for node
                                end if
                        end if
                else
                        node = node->rlink
                end if
        end while
end
```

searchNode(int searchVal)

```
begin
        node=head
        while node != NULL
        do
                if node->val == searchVal then
                        print “Node is found with key “, searchVal
                end if
                node = node->rlink
        end do
end
```

displayAll()

```
begin
```

```
        node = head
        while node != NULL
        do
                print "Node    with val  ", node->val
                node = node->rlink
        end do
end
```

Time Complexity:

For creating list θ(1) is the time complexity.

For Insertion, Search, Delete, Display All operations the complexity is O(n) where n is the number of nodes.

Program

```
#include<stdio.h>
#include<stdlib.h>

struct node {
int val;
struct node *llink,*rlink;
};

typedef struct node *NODE;
NODE head=NULL;
void CreateList(int val)
{
        NODE nd;
        if (head == NULL) {
                nd = (NODE) malloc(sizeof(struct node));
                nd->llink = nd->rlink = NULL;
                nd->val = val;
                head=nd;
        }
        else {
                printf("List is already created ….\n");
        }
}

void insertIntoList(int before, int val)
{
        NODE nd, newnd;
        nd=head;
        while (nd != NULL && nd->val != before)
```

```
                nd = nd->rlink;
        if (nd != NULL) {
                newnd = (NODE)malloc(sizeof(struct node));
                newnd->llink = newnd->rlink = NULL;
                newnd->val = val;
                if (nd->llink != NULL) {
                        nd->llink->rlink = newnd;
                        newnd->llink = nd->llink;
                        newnd->rlink = nd;
                        nd->llink = newnd;
                }
                else {
                        newnd->rlink = nd;
                        nd->llink = newnd;
                        head=newnd;
                }
        }
        else
                printf( "Unable to insert, node with value  %d not found", val);

}
void deleteAll(int delVal)
{
        NODE nd,nxtNode;
        nd = head;

        while (nd != NULL) {
                if (nd->val == delVal) {
                        if (nd->llink != NULL) {
                                nd-> llink ->rlink = nd -> rlink;
                                if (nd->rlink != NULL) {
                                        nd->rlink->llink = nd->llink;
                                        nxtNode = nd->rlink;
                                        free(nd);
                                        nd = nxtNode;
                                }
                                else {
                                        nd->llink->rlink = NULL;
                                        free(nd);
                                        nd=NULL;
                                }

                        }
                        else {
                                if (nd->rlink != NULL) {
                                        nd ->rlink->llink = NULL;
```

```
                                        head = nd->rlink;
                                        free(nd);
                                        nd = head;

                                }
                                else {
                                        free(nd);
                                        head = nd = NULL;
                                }
                        }
                }
                else
                        nd = nd->rlink;
        }
}


void searchNode(int searchVal) {
        NODE nd;
        nd=head;
        while (nd != NULL) {
                if (nd->val == searchVal)
                        printf( "Node is found with key %d\n", searchVal);
                nd = nd->rlink;
        }
}

void displayAll()
{
        NODE nd;
        nd = head;
        while (nd != NULL) {
                printf("Node    with val  %d\n", nd->val);
                nd = nd->rlink;
        }
}
int main() {
        int choice, val,before;
        do {
                 printf("1. Create List\n");
                 printf("2. Insert into List\n");
                 printf("3. Delete all by value\n");
                 printf("4. Search by value\n");
                 printf("5. Display all\n");
                 printf("6. Exit\n");
                 printf("Enter your choice    :");
```

```
                scanf("%d", &choice);
                switch(choice) {
                        case 1: printf("Please enter the node value");
                                 scanf("%d", &val);
                                 CreateList(val);
                                break;
                        case 2: printf("Please enter the node value to insert ");
                                scanf("%d", &val);
                                printf("Please enter the node value before which new node
has to be inserted ");
                                scanf("%d", &before);
                                insertIntoList(before, val);
                                break;
                        case 3:printf("Enter the node value to be deleted ");
                                scanf("%d", &val);
                                deleteAll(val);
                                break;
                        case 4:printf("Enter the node value to be searched ");
                                scanf("%d", &val);
                                searchNode(val);
                                break;
                        case 5:displayAll();
                                break;
                        case 6:
                                break;
                        default:printf("Invalid choice ");
                                 break;
                         }
          }while(choice != 6);
          return 0;
}
```

Sample Input and Output:

```
1. Create List
2. Insert into List
3. Delete all by value
4. Search by value
5. Display all
6. Exit
Enter your choice    :1
Please enter the node value5
1. Create List
2. Insert into List
3. Delete all by value
4. Search by value
5. Display all
```

```
6. Exit
Enter your choice    :2
Please enter the node value to insert 3
Please enter the node value before which new node has to be inserted 5
1. Create List
2. Insert into List
3. Delete all by value
4. Search by value
5. Display all
6. Exit
Enter your choice    :3
Enter the node value to be deleted 3
1. Create List
2. Insert into List
3. Delete all by value
4. Search by value
5. Display all
6. Exit
Enter your choice    :4
Enter the node value to be searched 5
Node is found with key 5
1. Create List
2. Insert into List
3. Delete all by value
4. Search by value
5. Display all
6. Exit
Enter your choice    :5
Node    with val  5
1. Create List
2. Insert into List
3. Delete all by value
4. Search by value
5. Display all
6. Exit
Enter your choice    :6
```

II. LAB EXERCISES

1). Write a program to construct a binary tree to support the following operations.
Assume no duplicate elements while constructing the tree.

i. Given a key, perform a search in the binary search tree. If the key is found then display "key found" else insert the key in the binary search tree.

ii. Display the tree using inorder, preorder and post order traversal methods

2). Write a program to implement the following graph representations and display them.

i. Adjacency list

ii. Adjacency matrix

III. ADDITIONAL EXERCISES:

1). Solve the problem given in solved exercise using singly linked list.
2). Write a program to implement Stack and Queue using circular doubly linked list.

[OBSERVATION SPACE – LAB1]

LAB NO: 2 **Date:**

FUNDAMENTALS OF ALGORITHMIC PROBLEM SOLVING

Objectives:

In this lab, student will be able to:

- Familiarize with fundamentals of problem solving with the help of algorithms.
- Realize that for a problem there can be multiple solutions with different complexities.
- Determine the time complexity associated with algorithms.

Description: A solution to the problem is obtained after understanding clearly the problem, nature of input and output. The detailed step by step solution to the problem is called an algorithm. For a single problem, there can be multiple ways in which solution is found. Hence, a problem may be solvable using multiple algorithms. To measure the efficiency of an algorithm, measurement along time required by the algorithm and space required is used. To measure time, an operation called basic operation is identified.

I. SOLVED EXERCISE:

1) Write an algorithm for finding the Greatest Common Divisor (GCD) of two numbers using Euclid's algorithm and implement the same. Determine the time complexity of the algorithm.

Description: Greatest Common Divisor(GCD) of two numbers is the largest divisor which divides the two numbers. E.g. GCD(36,8) = 4. Euclid's algorithm is one of the oldest algorithm for calculating GCD. The algorithm is significant because the solution is obtained by reducing the problem space with irregular count. We take the modulus of first number when divided by the second number and we shrink the first number with the second number and the second number with the modulus. This we continue until the second number is not zero. When the second number is zero, the GCD is the first number. Time complexity of this algorithm is in log(n) where n is the second number.

ALGORITHM *EuclidGCD(m, n)*
//Computes gcd*(m, n)* by Euclid's algorithm
//Input: Two nonnegative, not-both-zero integers *m* and *n*
//Output: Greatest common divisor of *m* and *n*
while $n \neq 0$ **do**
    $r \leftarrow m \bmod n$
    $m \leftarrow n$
    $n \leftarrow r$
return *m*

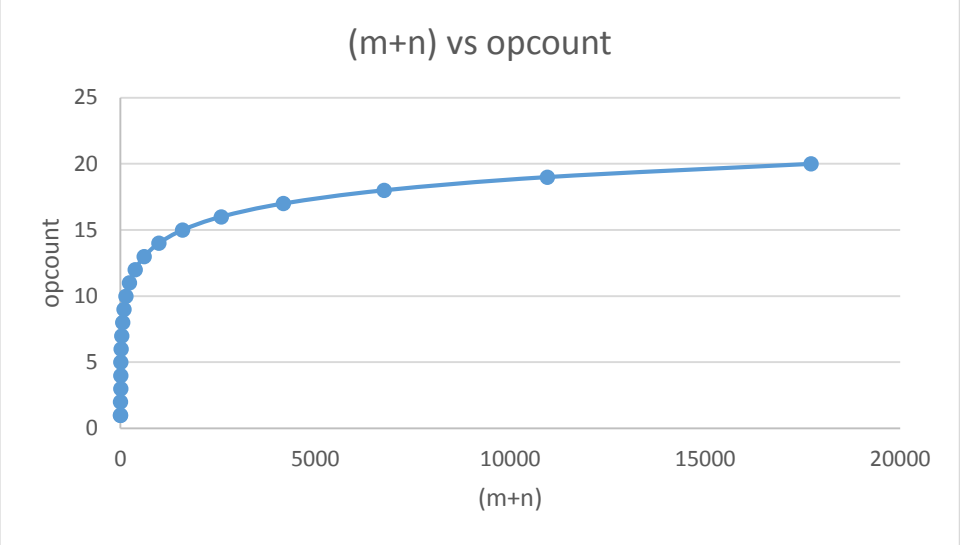
Time Complexity: O(log n). The worst case for this algorithm is when the inputs are two consecutive Fibonacci numbers. We can plot the graph of (m+n) vs the step count (opcount is shown in the sample code), where m and n are the two inputs to function EuclidGCD.

Program

```
#include<stdio.h>
unsigned int EuclidGCD(unsigned int m, unsigned int n) {
    unsigned int r;
    int  opcount = 0; // variable to count how many times the basic operation executes.
    while(n!=0) {
            opcount++;
             r = m %n;
            m = n;
            n=r;
    }
    printf("Operation count= %d\n", opcount);
    return m;
}
int  main() {
            unsigned int m,n;
            printf("Enter the two numbers whose GCD has to be calculated");
            scanf("%d", &m);
            scanf("%d", &n);
            printf("GCD of two numbers using Euclid's method is %d",
                    EuclidGCD(m,n));  return 0;
}
```

Sample Input and Output:
Enter the two numbers whose GCD has to be calculated 8 13
Operation count= 6
GCD of two numbers using Euclids method is 1
Tabulate the values of (m+n), opcount as shown below and plot the graph.

(m+n)	1	8	13	21	.....	6765	10946	17711
opcount	1	4	5	6	....	18	19	20



II. LAB EXERCISES

1). Write a program to find GCD using consecutive integer checking method and analyze its time efficiency.
2). Write a program to find GCD using middle school method and analyze its time efficiency.

III. ADDITIONAL EXERCISES

1) Write a program for computing $\lfloor \sqrt{n} \rfloor$ for any positive integer and analyze its time efficiency. Besides assignment and comparison, your program may only use the four basic arithmetic operations.
2) Write a program to implement recursive solution to the Tower of Hanoi puzzle and analyze its time efficiency.
3) Write a program to compute the n^{th} Fibonacci number recursively and analyze its time efficiency.

[OBSERVATION SPACE – LAB2]

LAB NO: 3 **Date:**

BRUTE FORCE TECHNIQUE - I

Objectives:

In this lab, student will be able to:

- Understand brute-force design technique
- Apply this technique for sorting, searching etc.
- Determine the time complexity associated with brute-force algorithms

Description: Brute force is a straightforward approach to solving a problem, usually directly based on the problem statement and definitions of the concepts involved. Brute force is applicable to a very wide variety of problems. For some important problems e.g. sorting, searching, matrix multiplication, string matching the brute-force approach yields reasonable algorithms of value with no limitation on instance size. The expense of designing a more efficient algorithm may be unjustifiable if only a few instances of a problem need to be solved and a brute-force algorithm can solve those instances with acceptable speed.

I. SOLVED EXERCISE:

1) Write a program to sort a set of integers using selection sort algorithm and analyze its time efficiency. Obtain the experimental result of order of growth. Plot the result for the best and worst case.

Description: In selection sort, we scan the entire given list to find its smallest element and exchange it with the first element, putting the smallest element in its final position in the sorted list. Then we scan the list, starting with the second element, to find the smallest among the last n-1 elements and exchange it with the second element, putting the second smallest element in its final position. This is repeated for all positions.

ALGORITHM *SelectionSort(A*[0..$n-1$]*)*
//Sorts a given array by selection sort
//Input: An array $A[0..n-1]$ of orderable elements
//Output: Array $A[0..n-1]$ sorted in nondecreasing order
for $i \leftarrow 0$ **to** $n-2$ **do**
  $min \leftarrow i$
  for $j \leftarrow i+1$ **to** $n-1$ **do**
    if $A[j] < A[min]$
      $min \leftarrow j$
    end if
  end for
  swap $A[i]$ and $A[min]$
end for

Time Complexity:

$C_{worst}(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1$

The worst case occurs when elements are given in decreasing order.

$C_{worst}(n) = \sum_{i=0}^{n-2} n-1-i-1+1$

$= \sum_{i=0}^{n-2} n-i-1$

$=$ n-1 + n-2 … +1

$= \frac{(n-1)(n-1+1)}{2}$

$= \frac{(n-1)(n)}{2}$

$= \theta(n^2)$

This can be observed by repeating the experiment with the worst case inputs for different array sizes say 10, 15, 20, 35, 100. A plot of number of elements in the array vs opcount (opcount is shown in the sample code) gives a quadratic curve.

Program

```
#include<stdio.h>
#include<stdlib.h>
void selectionSort(int *a, unsigned int n)
{
  unsigned int i,j,min;
  int temp;
  int opcount=0; // introduce opcount
  for(i= 0;i<n-1;i++)
  {
       min=i;
       for(j=i + 1;j<n;j++)
       {
          ++opcount;       // increment opcount for every comparison
          if(a[j ]<a[min])
             min=j;
       }
       //swap A[i] and A[min]
       temp = a[i];
       a[i] = a[min];
       a[min]=temp;
   }
   printf("\nOperation Count %d\n",opcount);
}
int main() {
       int *a;
       int i,n = 5;
       // generate worst case input of different  input size
       for (int j=0; j < 4; j++) // repeat experiment for 4 trials
       {
            a = (int *)malloc(sizeof(int)*n);
            for (int k=0; k< n; k++)
                a[k] = n-k; // descending order list
            printf("Elements are ");
            for(i=0;i<n;i++)
```

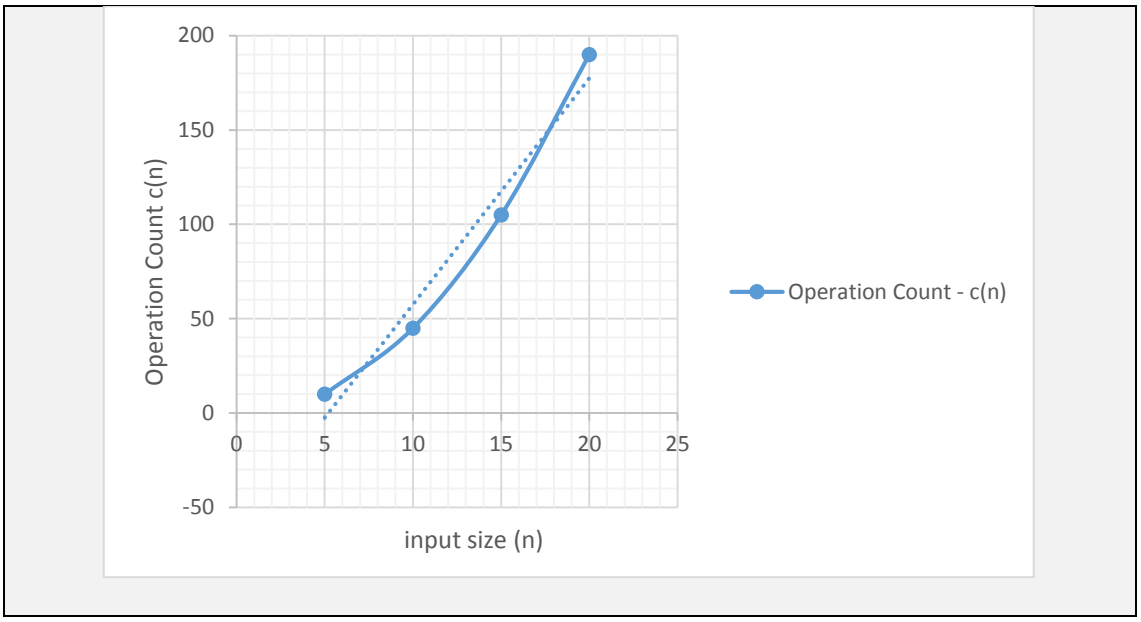
```
                printf("%d   ",a[i]);
        selectionSort(a,n);
        printf("Elements after selection sort ");
        for(i=0;i<n;i++)
                printf("%d   ",a[i]);
        printf("\n");
        free(a);
        n = n + 5;  // try with a new input size
    }
    return 0;
}
```

Sample Input and Output :

Elements are 5 4 3 2 1
Operation Count 10
Elements after selection sort 1 2 3 4 5
Elements are 10 9 8 7 6 5 4 3 2 1
Operation Count 45
Elements after selection sort 1 2 3 4 5 6 7 8 9 10
Elements are 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
Operation Count 105
Elements after selection sort 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Elements are 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
Operation Count 190
Elements after selection sort 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

Tabulate the values of n and opcount as shown below and plot the graph.

input size - (n)	5	10	15	20
Operation Count - c(n)	10	45	105	190



II. LAB EXERCISES

1). Write a program to sort set of integers using bubble sort. Analyze its time efficiency. Obtain the experimental result of order of growth. Plot the result for the best and worst case.

2). Write a program to implement brute-force string matching. Analyze its time efficiency.

3). Write a program to implement solution to partition problem using brute-force technique and analyze its time efficiency theoretically. A partition problem takes a set of numbers and finds two disjoint sets such that the sum of the elements in the first set is equal to the second set. [Hint: You may generate power set]

III. ADDITIONAL EXERCISES

1). Write a program to implement matrix multiplication using brute-force technique and analyze its time efficiency. Obtain the experimental result of order of growth. Plot the result for the best and worst case.

2). Write a program in C for finding maximal clique in a graph by brute-force approach. Clique is a maximal complete subgraph in a graph.

3). Write a program to implement solution to partition problem using recursion.