SRI CHANDRASEKHARENDRA SARASWATHI VISWA MAHAVIDYALAYA

(UNIVERSITY ESTABLISHED UNDER SECTION 3 OF UGC ACT 1956)
ENATHUR, KANCHIPURAM – 631 561

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Name : SK.MOHAMMED IRFAN

Reg. No : 11249A342

Class : II B.E. (CSE)

Course Code:

Course Name:

SRI CHANDRASEKHARENDRA SARASWATHI VISWA MAHAVIDYALAYA

(UNIVERSITY ESTABLISHED UNDER SECTION 3 OF UGC ACT 1956)
ENATHUR, KANCHIPURAM – 631 561

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

BONAFIDE CERTIFICATE

This is to certify that this is the bonafide record of work done by Mr/Ms. **S.K MOHAMMED IRFAN** ,

with **Reg. No 11249A342** of **II-B.E.(CSE)** during the academic year 2025 – 2026.

Station:

Date:

Staff-in-charge **Head of the Department**

Submitted for the Practical examination held on_____.

Examiner-1 **Examiner-2**

INDEX

Exercise: 1(A)	Linear search
Date:08-08-25	

Aim: To find the element in an array using linear search

Algorithm:

1. Start
2. Read n and array elements
3. Read search element key
4. Traverse array and compare each element with key
5. If found, print position and stop
6. Else print “Not found”
7. Stop

Program:

```
#include <stdio.h>
int main() {
    int n, key, i, flag = 0;
    scanf("%d", &n);
    int a[n];
    for(i=0;i<n;i++) scanf("%d", &a[i]);
    scanf("%d", &key);
    for(i=0;i<n;i++) {
        if(a[i]==key) {
            printf("Found at position %d", i+1);
            flag=1;
            break;
        }
    }
    if(!flag) printf("Not found");
    return 0;
}
```

Output:

5 10 20 30 40 50
30 Found at position 3

Result:

Thus, a program that finds the element in an array using linear search is written and executed successfully

Exercise: 1(B)	Binary search
Date:08-08-25	

Aim: To search an element in a sorted array using binary search.

Algorithm:

1. Start
2. Input sorted array and key
3. Set low=0, high=n-1
4. Repeat while low<=high
5. Find mid=(low+high)/2
6. If a[mid]==key, print found
7. If a[mid]>key, set high=mid-1, else low=mid+1
8. Stop

Program:

```
#include <stdio.h>
int main() {
    int n, key, i, low, high, mid, flag=0;
    scanf("%d", &n);
    int a[n];
    for(i=0;i<n;i++) scanf("%d", &a[i]);
    scanf("%d", &key);
    low=0; high=n-1;
    while(low<=high) {
        mid=(low+high)/2;
        if(a[mid]==key) {
            printf("Found at position %d", mid+1);
            flag=1;
            break;
        } else if(a[mid]>key)
            high=mid-1;
        else
            low=mid+1;
    }
    if(!flag) printf("Not found");
    return 0;
}
```

Output:

```
10 20 30 40 50
40
Found at position 4
```

Result: Thus, a program that searches an element in a sorted array using binary search is written and executed successfully

Exercise: 2	Implementation of stack
Date:16-08-25	

Aim: To implement stack using array.

Algorithm:

1. Start
2. Initialize top = -1
3. Push adds element if not full
4. Pop removes element if not empty
5. Display prints all elements
6. Stop

Program:

```
#include <stdio.h>
#define MAX 5
int stack[MAX], top=-1;
void push(int x){ if(top<MAX-1) stack[++top]=x; }
void pop(){ if(top>=0) top--; }
void display(){ for(int i=top;i>=0;i--) printf("%d ",stack[i]); }
int main(){
    push(10); push(20); push(30);
    display();
    pop();
    printf("\n");
    display();
    return 0;
}
```

Output:

```
30 20 10
20 10
```

Result: Thus , a program that implements stack using array is written and executed successfully

Exercise:3	**Implementation of application of stack**
Date:16-08-25	

Aim: To reverse a string using stack.

Algorithm:

1. Start
2. Read string
3. Push all characters into stack
4. Pop and print each character
5. Stop

Program:

```
#include <stdio.h>
#include <string.h>
#define MAX 100
char stack[MAX];
int top=-1;
void push(char c){stack[++top]=c;}
char pop(){return stack[top--];}
int main(){
    char str[100];
    scanf("%s",str);
    for(int i=0;i<strlen(str);i++) push(str[i]);
    for(int i=0;i<strlen(str);i++) printf("%c",pop());
    return 0;
}
```

Output:

```
hello
olleh
```

Result: Thus a program that reverses a string using stack is written and executed successfully

Exercise:4	Implementation of queue
Date:22-08-25	

Aim: To implement a queue using an array.

Algorithm:

1. start
2. Initialize front=0, rear=-1
3. Enqueue inserts at rear
4. Dequeue removes from front
5. Stop

Program:

```
#include <stdio.h>
#define MAX 5
int queue[MAX], front=0, rear=-1;
void enqueue(int x){ if(rear<MAX-1) queue[++rear]=x; }
void dequeue(){ if(front<=rear) front++; }
void display(){ for(int i=front;i<=rear;i++) printf("%d ",queue[i]); }
int main(){
    enqueue(10); enqueue(20); enqueue(30);
    display();
    printf("\n");
    dequeue();
    display();
    return 0;
}
```

Output:

```
10 20 30
20 30
```

Result: Thus the program that implementation of queue using array is written and executed successfully

Exercise:5	Implementation of Singly Linked List
Date:22-08-25	

Aim: To implement a singly linked list.

Algorithm:

1. start
2. Create nodes dynamically
3. Link them sequentially
4. Traverse to display elements
5. Stop

Program:

```
#include <stdio.h>
#include <stdlib.h>
struct node{int data;struct node*next;};
int main(){
    struct node *head=NULL,*temp,*newnode;
    for(int i=0;i<3;i++){
        newnode=(struct node*)malloc(sizeof(struct node));
        scanf("%d",&newnode->data);
        newnode->next=NULL;
        if(head==NULL) head=temp=newnode;
        else{temp->next=newnode;temp=newnode;}
    }
    temp=head;
    while(temp){printf("%d ",temp->data);temp=temp->next;}
    return 0;
}
```

Output:

```
10 20 30
10 20 30
```

Result : Thus the program that implementation of singly linked list is written and executed successfully

Exercise:6	Implementation of Doubly Linked List
Date:05-09-25	

Aim: To implement a doubly linked list.

Algorithm:

1. Start
2. Create doubly linked nodes
3. Connect prev and next pointers
4. Traverse forward to display
5. Stop

Program:

```
#include <stdio.h>
#include <stdlib.h>
struct node{int data;struct node *prev,*next;};
int main(){
   struct node *head=NULL,*temp,*newnode;
   for(int i=0;i<3;i++){
      newnode=(struct node*)malloc(sizeof(struct node));
      scanf("%d",&newnode->data);
      newnode->next=NULL;
  if(head==NULL){newnode->prev=NULL;head=temp=newnode;}
else{newnode->prev=temp;temp->next=newnode;temp=newnode;}
   }
   temp=head;
   while(temp){printf("%d ",temp->data);temp=temp->next;}
   return 0;
}
```

Output:

10 20 30
10 20 30

Result: Thus a program that implements a doubly linked list is written and executed successfully

Exercise:7	Perform Traversal on a Binary Search Tree
Date:05-09-25	

Aim: To create and traverse a Binary Search Tree.

Algorithm:

1. Start
2. Insert elements following BST rule
   Perform inorder traversal
3. Stop

Program:

```
#include <stdio.h>
#include <stdlib.h>
struct node{int data;struct node*left,*right;};
struct node*insert(struct node*r,int val){
    if(r==NULL){r=(struct node*)malloc(sizeof(struct
node));r->data=val;r->left=r->right=NULL;}
    else if(val<r->data)r->left=insert(r->left,val);
    else r->right=insert(r->right,val);
    return r;
}
void inorder(struct node*r){if(r){inorder(r->left);printf("%d
",r->data);inorder(r->right);}}
int main(){
    struct node*root=NULL;int n,x;
    scanf("%d",&n);
    for(int i=0;i<n;i++){scanf("%d",&x);root=insert(root,x);}
    inorder(root);
    return 0;
}
```

Output:

```
40 20 60 10 30
10 20 30 40 60
```

Result: Thus , a program that performs creation and traversion in a Binary Search Tree is written and executed successfully

Exercise:8	Implementation of graph search algorithms
Date:12-09-25	

Aim: To perform BFS and DFS traversal on a graph

Algorithm:

1. Start
2. Input adjacency matrix
3. Use queue for BFS, recursion for DFS
4. Stop

Program:

```
#include <stdio.h>

int n,a[10][10],visited[10];

void dfs(int v){int i;visited[v]=1;printf("%d
",v);for(i=0;i<n;i++)if(a[v][i]&&!visited[i])dfs(i);}

void bfs(int v){

    int q[10],f=0,r=-1,i;

    visited[v]=1; q[++r]=v;

    while(f<=r){

        v=q[f++]; printf("%d ",v);

        for(i=0;i<n;i++) if(a[v][i]&&!visited[i]){q[++r]=i;visited[i]=1;}

    }

}

int main(){

    scanf("%d",&n);

    for(int i=0;i<n;i++)for(int j=0;j<n;j++)scanf("%d",&a[i][j]);

    for(int i=0;i<n;i++)visited[i]=0;

    dfs(0);

    printf("\n");

    for(int i=0;i<n;i++)visited[i]=0;
```

```
    bfs(0);
    return 0;
}
```

Output:

```
0 1 1 0
1 0 1 1
1 1 0 0
0 1 0 0
0 1 2 3
0 1 2 3
```

Result: Thus, a program that performs BFS and DFS traversal on a graph is written and executed successfully

Exercise:9(A)	Sort Given Numbers using Selection Sort
Date:10-10-25	

Aim: To sort an array using selection sort.

Algorithm:

1. Start
2. Repeat for i=0 to n-1
3. Find min element index
4. Swap with a[i]
5. Stop

Program:

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

```
int main(){
    int n,i,j,min,temp;
    scanf("%d",&n);
    int a[n];
    for(i=0;i<n;i++) scanf("%d",&a[i]);
    for(i=0;i<n-1;i++){
        min=i;
        for(j=i+1;j<n;j++) if(a[j]<a[min]) min=j;
        temp=a[i];a[i]=a[min];a[min]=temp;
    }
    for(i=0;i<n;i++) printf("%d ",a[i]);
    return 0;
}
```

Output:

```
64 25 12 22 11
11 12 22 25 64
```

Result: Thus,a program that sorts using array .selection sort is written and executed successfully

Exercise:9(b)	Heap sort
Date:10-10-25	

Aim: To sort array using heap sort

Algorithm:

1. Start
2. Swap root with last element
3. Heapify reduced heap
4. stop

Program:

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

```
void heapify(int a[], int n, int i){
    int largest=i,l=2*i+1,r=2*i+2,temp;
    if(l<n&&a[l]>a[largest]) largest=l;
    if(r<n&&a[r]>a[largest]) largest=r;

if(largest!=i){temp=a[i];a[i]=a[largest];a[largest]=temp;heapify(a,n,large
st);}
}
void heapSort(int a[], int n){
    for(int i=n/2-1;i>=0;i--) heapify(a,n,i);
    for(int i=n-1;i>=0;i--){int t=a[0];a[0]=a[i];a[i]=t;heapify(a,i,0);}
}
int main(){
    int n;scanf("%d",&n);int a[n];
    for(int i=0;i<n;i++) scanf("%d",&a[i]);
    heapSort(a,n);
    for(int i=0;i<n;i++) printf("%d ",a[i]);
    return 0;
}
```

Output:

```
5
4 10 3 5 1
1 3 4 5 10
```

Result: Thus a program that sorts arrays using heap sort is written and executed successfully

Exercise:9(c)	Quick sort
Date:17-10-25	

Aim: To sort an array using quick sort.

Algorithm:

1. Start
2. Choose pivot
3. Partition array
4. Recursively sort left and right
5. Stop

Program:

```
#include <stdio.h>
void swap(int*a,int*b){int t=*a;*a=*b;*b=t;}
int partition(int a[],int low,int high){
    int pivot=a[high],i=low-1;
    for(int j=low;j<high;j++) if(a[j]<pivot){i++;swap(&a[i],&a[j]);}
    swap(&a[i+1],&a[high]);return i+1;
}
void quickSort(int a[],int low,int high){
    if(low<high){int
pi=partition(a,low,high);quickSort(a,low,pi-1);quickSort(a,pi+1,high);}
}
int main(){
    int n;scanf("%d",&n);int a[n];
    for(int i=0;i<n;i++) scanf("%d",&a[i]);
    quickSort(a,0,n-1);
    for(int i=0;i<n;i++) printf("%d ",a[i]);
    return 0;
}
```

Output:

```
5
64 25 12 22 11
11 12 22 25 64
```

Result: Thus the program that sorts arrays using quick sort is written and executed successfully

Exercise:9(d)	Merge sort
Date:17-10-25	

Aim: To sort an array using merge sort.

Algorithm:

1. Start
2. Divide array into halves
3. Recursively sort halves
4. Merge sorted halves
5. Stop

Program:

```
#include <stdio.h>
void merge(int a[],int l,int m,int r){
    int n1=m-l+1,n2=r-m,i,j,k;
    int L[n1],R[n2];
    for(i=0;i<n1;i++)L[i]=a[l+i];
    for(j=0;j<n2;j++)R[j]=a[m+1+j];
    i=j=0;k=l;
    while(i<n1&&j<n2) a[k++]=(L[i]<=R[j])?L[i++]:R[j++];
    while(i<n1)a[k++]=L[i++];
    while(j<n2)a[k++]=R[j++];
}
void mergeSort(int a[],int l,int r){
    if(l<r){int
m=(l+r)/2;mergeSort(a,l,m);mergeSort(a,m+1,r);merge(a,l,m,r);}
}
int main(){
    int n;scanf("%d",&n);int a[n];
    for(int i=0;i<n;i++) scanf("%d",&a[i]);
    mergeSort(a,0,n-1);
```

```
  for(int i=0;i<n;i++) printf("%d ",a[i]);
    return 0;
}
```

Output:

5

64 25 12 22 11

11 12 22 25 64

Result: Thus the program that sorts arrays using merge sort written and executed successfully

Exercise:10	Implementation of Hashing
Date:07-11-25	

Aim: To implement hashing using linear probing.

Algorithm:

1. Start
2. Initialize hash table with -1
3. For each key, compute index=key%size
4. If occupied, probe next index

Program:

```
#include <stdio.h>
#define SIZE 10
int main(){
    int hash[SIZE];for(int i=0;i<SIZE;i++)hash[i]=-1;
    int n,key,index;
    scanf("%d",&n);
    for(int i=0;i<n;i++){
        scanf("%d",&key);
        index=key%SIZE;
        while(hash[index]!=-1) index=(index+1)%SIZE;
        hash[index]=key;
    }
    for(int i=0;i<SIZE;i++) printf("%d ",hash[i]);
    return 0;
}
```

Output:

12 22 32 42 52

-1 12 22 32 42 52 -1 -1 -1 -1

Result: Thus the program to implement of hashing is written and executed successfully