SRI CHANDRASEKHARENDRA SARASWATHI VISWA MAHAVIDYALAYA

(UNIVERSITY ESTABLISHED UNDER SECTION 3 OF UGC Acr 1956)
ENATHUR, KANCHIPURAM -631 561

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Name : T.Neelesh

Reg. No: 11249A386

Class • 11 B.E. (CSE)

Subject Code: BCSF183P60
Subject Name.• DATA STRUCTURE AND ALGORITHMS LAB

# SRI CHANDRASEKHARENDRA SARASWATHI VISWA MAHAVIDYALAYA

(UNIVERSITY ESTABLISHED UNDER SECTION 3 OF UGC Acr 1956)
ENATIWR, KANCHIPURAM - 631 561

## DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

**BONAFIDE CERTIFICATE**

This is to certify that this is the bonafide record of wcxkdone by

Mr.,'Ms. Tulasi manikanta Neelesh

with Reg. No 11249A386 of Il-B.E.(CSE) in the DATA STRUCTURE AND ALGORITHMS LAB(BCSF183P60) during the academic year 2025 - 2026.

Station:Enathur

Date:

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

Submitted for the Practical examination held on _____.

Examiner-t Examirrr.2

# INDEX

| | |
|---|---|
| Exercise: I(A) | Linear search |
| Date:08-08-25 | |

Aim: To write a linear search program

Algorithm: 1.start the program

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 the program

# Program:

```
#include <stdio.h>

int main() {
   int arr[100], n, key, i, found = 0;

   // Input: size of array
   printf("Enter number of elements: ");
   scanf("%d", &n);

   // Input: array elements
   printf("Enter %d elements:\n", n);
   for (i = 0; i < n; i++) {
      scanf("%d", &arr[i]);
   }

   // Input: element to search
   printf("Enter element to search: ");
   scanf("%d", &key);

   // Linear search
   for (i = 0; i < n; i++) {
      if (arr[i] == key) {
         printf("Element found at position %d\n", i + 1);
         found = 1;
         break;
      }
   }

   if (!found) {
      printf("Element not found in the array.\n");
   }

   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

Date: 08/08/25 Binary search

Exercise: 1(b)

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

## Algorithm:

1. Start the program
2. Input sorted array and key
3. Set low=0, high=n-l
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-l, else low=mid+l

Porgram:

```
#include <stdio.h>



Int main() {

   Int arr[100], n, key, low, high, mid;
```

Stop

# Program:

```
// Input size
Printf("Enter number of elements: ");
Scanf("%d", &n);
// Input sorted array
Printf("Enter %d elements (in sorted order):\n", n);
For (int I = 0; I < n; i++) {
    Scanf("%d", &arr[i]);
}
// Key to search
Printf("Enter element to search: ");
Scanf("%d", &key);
Low = 0;
High = n – 1;
// Binary Search
While (low <= high) {
```

```
    Mid = (low + high) / 2;


    If (arr[mid] == key) {
       Printf("Element found at position %d\n", mid + 1);
       Return 0;
    }
    Else if (arr[mid] < key) {
       Low = mid + 1;
    }
    Else {
       High = mid – 1;
    }
  }
  Printf("Element not found in the array.\n");
 Return 0;
}
```

Stop

# Program:

Exercise no: 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 the program

Program : #include <stdio.h>

#define MAX 5

Int stack[MAX];

Int top = -1;

// Function declarations

Void push();

```
Void pop();

Void peek();

Void display();


Int main() {

   Int choice;


   While (1) {

      Printf("\n--- STACK MENU ---\n");

      Printf("1. Push\n");

      Printf("2. Pop\n");

      Printf("3. Peek\n");

      Printf("4. Display\n");

      Printf("5. Exit\n");


       Stop
```

Program:

```
    Printf("Enter your choice: ");
    Scanf("%d", &choice);

    Switch (choice) {
        Case 1: push(); break;
        Case 2: pop(); break;
        Case 3: peek(); break;
        Case 4: display(); break;
        Case 5: return 0;
        Default: printf("Invalid choice! Try again.\n");
    }
}

Return 0;
```

```
}

Void push() {
   Int value;

   If (top == MAX – 1) {
      Printf(“Stack Overflow! Cannot push.\n”);
   } else {
      Printf(“Enter value to push: “);
      Scanf(“%d”, &value);
      Stack[++top] = value;
      Printf(“%d pushed into stack.\n”, value);
   }
}
```

Stop

Program:

```
Void pop() {

   If (top == -1) {

      Printf(“Stack Underflow! Nothing to pop.\n”);

   } else {

      Printf(“%d popped from stack.\n”, stack[top--]);

   }

}
Void peek() {

   If (top == -1) {

      Printf(“Stack is empty.\n”);

   } else {

      Printf(“Top element = %d\n”, stack[top]);
   }

}
```

```
Void display() {
   If (top == -1) {
      Printf(“Stack is empty.\n”);
   } else {
      Printf(“Stack elements:\n”);
      For (int I = top; I >= 0; i--) {
         Printf(“%d\n”, stack[i]);
      }
   }
}
```

Stop

Program:

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) {

   If (top == MAX – 1) {

      Printf(“Stack Overflow\n”);

   } else {

      Stack[++top] = c;

   }
```

and executed successfully

```
}
Char pop() {

   If (top == -1) {

      Return ‘\0’;  // Empty

   } else {

      Return stack[top--];
   }

}
Int isMatchingPair(char open, char close) {

   If (open == ‘(‘ && close == ‘)’) return 1;
```

5. Stop

```
    If (open == '{' && close == '}') return 1;

    If (open == '[' && close == ']') return 1;

    Return 0;

}
Int isBalanced(char exp[]) {

    For (int I = 0; I < strlen(exp); i++) {

        Char c = exp[i];

      If (c == '(' || c == '{' || c == '[') {

            Push©;

        }
```

and executed successfully

```
      Else if (c == ')' || c == '}' || c == ']') {

         Char popped = pop();

         If (!isMatchingPair(popped, c)) {

            Return 0;   // Not balanced

         }
     }
}
If (top == -1)

      Return 1;

   Return 0;
}
```

5. Stop

```
Int main() {

   Char expression[MAX];
  Printf(“Enter an expression: “);

   Scanf(“%s”, expression);
   If (isBalanced(expression))
 Printf(“Expression is Balanced\n”);
  Else
   Printf(“Expression is NOT Balanced\n”);
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