    }

Program No. 1(a)

(a) Program for Recursive Linear Search.

1.1 Algorithm

```
LinearSearch (array, index, key):
    if index < 0:
        return -1;
    if item = key:
        return index
    return LinearSearch (array, index-1, key)
```

1.2 Program

```
#include <stdio.h>
int linearSearch(int arr[], int size, int key)
{
        if (size == 0) {
        return -1;
    }
    if (arr[size - 1] == key) {
           return size - 1;
    }
      return linearSearch(arr, size - 1, key);
}
int main()
{
    int arr[] = { 5, 15, 6, 9, 4 };
    int key = 4;
    int index
        = linearSearch(arr, sizeof(arr) / sizeof(int), key);
    if (index == -1) {
        printf("Key not found in the array.\n");
    }
    else {
        printf("The element %d is found at %d index of the "
                "given array \n",
                key, index);
    }
    return 0;
}
```

1.3 Output

```
The element 4 is found at 4 index of the given array.
```

(b) Program for Binary Search.

1.1 (b) Algorithm

```
binarySearch(arr, x, low, high)
    if low > high
        return False
    else
        mid = (low + high) / 2
        if x == arr[mid]
            return mid
        else if x > arr[mid]        // x is on the right side
            return binarySearch(arr, x, mid + 1, high)
        else                        // x is on the left side
            return binarySearch(arr, x, low, mid - 1)
```

1.2 (b) Program

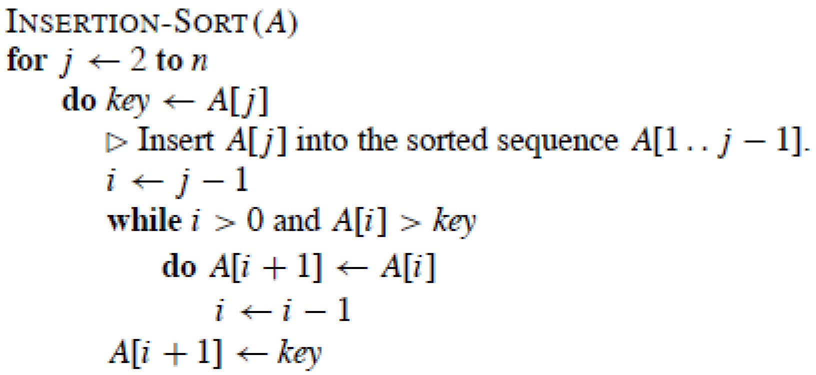
```
#include <stdio.h>
int binarySearch(int array[], int x, int low, int high)
{

  while (low <= high) {

    int mid = low + (high - low) / 2;

    if (array[mid] == x)

      return mid;

    if (array[mid] < x)

      low = mid + 1;

    else

      high = mid - 1;

  }

  return -1;

}
```

```
int main(void) {
  int array[] = {3, 4, 5, 6, 7, 8, 9};
  int n = sizeof(array) / sizeof(array[0]);
  int x = 4;
  int result = binarySearch(array, x, 0, n - 1);
  if (result == -1)
    printf("Not found");
  else
    printf("Element is found at index %d", result);
  return 0;
}
```

Program No. 2

Program for Insertion Sort in ascending order.

2.1 Algorithm



```
INSERTION-SORT(A)
for $j \leftarrow 2$ to $n$
    do $key \leftarrow A[j]$
        ▷ Insert $A[j]$ into the sorted sequence $A[1 \,.\,.\, j-1]$.
        $i \leftarrow j - 1$
        while $i > 0$ and $A[i] > key$
            do $A[i+1] \leftarrow A[i]$
                $i \leftarrow i - 1$
        $A[i+1] \leftarrow key$
```

2.2 Program

```
#include<stdio.h>

#include<conio.h>

void INSERTION(int a[],int n);// Function declaration

void main()

{

inti,a[10],n;

clrscr();

printf("\nProgram for INSERTION SORT\n");

printf("Enter no. of elements:\n");// Input  section

scanf("%d",&n);

for(i=1;i<=n;i++)

 {

printf("Enter element%d: ",i);

scanf("%d",&a[i]);
```

```
 }
INSERTION(a,n);// Calling function in main
printf("Sorted array:\n");// Output  section
for(i=1;i<=n;i++)
 {
printf("%d",a[i]);
 }
getch();
}
void INSERTION(int a[],int n)// Function definition
{
inti,j,key;
for(j=2;j<=n;j++)                                   // Sorting  section
 {
key=a[j];
i=j-1;
while((i>0)&&(a[i]>key))                    // Comparison
  {
a[i+1]=a[i];
i=i-1;
  }
a[i+1]=key;
 }
      }
```

2.3 Output

```
Program for INSERTION SORT

Enter no. of elements: 5

Enter element: 2

4

1

6

3

Sorted array: 1   2   3   4    6
```

Program No. 3

Program for Insertion Sort in descending order.

3.1 Algorithm

Insertion_Sort (A)   // in Descending Oredr

```
for j=2 to length[A]
    key=A[j]
    // Insert A[j] into sorted sequence A[1..j-1]
    i=j-1
    whilei>0 and A[i]<key   // Change
        A[i+1]=A[i]
        i=i-1
    A[i+1]=key
```

3.2 Program

```
#include<stdio.h>
#include<conio.h>
void INSERTION(int a[],int n);// Function declaration
void main()
{
inti,a[10],n;
clrscr();
printf("\nProgram for INSERTION SORT\n");
printf("Enter no. of elements:\n");// Input  section
scanf("%d",&n);
for(i=1;i<=n;i++)
```

```
 {
printf("Enter element%d: ",i);
scanf("%d",&a[i]);
 }
INSERTION(a,n);// Calling function in main
printf("Sorted array:\n");// Output  section
for(i=1;i<=n;i++)
 {
printf("%d",a[i]);
 }
getch();
}
void INSERTION(int a[],int n)// Function definition
{
inti,j,key;
for(j=2;j<=n;j++)                      // Sorting  section
 {
key=a[j];
i=j-1;
while((i>0)&&(a[i]<key))              // Comparison
  {
a[i+1]=a[i];
i=i-1;
  }
a[i+1]=key;
```

```
 }
    }
```

2.3 Output

```
Program for INSERTION SORT

Enter no. of elements: 5

Enter element: 2

4

1

6

3

Sorted array: 6   4   3     2    1
```

Program No. 4

Program for Merge Sort

4.1 Algorithm

Merge_Sort(A,p,r)

```
if p<r
    q=floor((p+r)/2)
    Merge_Sort(A,p,q)
    Merge_Sort(A,q+1,r)
    Merge(A,p,q,r)
```

Merge(A,p,q,r)

```
n1=q-p+1
n2=r-q
//Create array L[1..n1+1] and R[1..n2+1]
fori=1 to n1
    L[i]=A[p+i-1]
for j=1 to n2
    R[j]=A[q+j]
L[n1+1]=INFINITY
R[n2+1]=INFINITY
i=1
j=1
for k=p to r
    if L[i]<R[j]
        A[k]=L[i]
```

```
  i=i+1
else A[k]=R[j]
  j=j+1
```

4.2 Program

```
#include<stdio.h>
#include<conio.h>
voidMerge_Sort(int a[],intn,intp,int r);\\ Function declaration
void Merge(int a[],intp,intq,int r);
void main()
{
inti,n,a[10];
clrscr();
printf("\nProgram for MERGE SORT \n");
printf("Enter no. of elements:\n");\\ Input  section
scanf("%d",&n);
for(i=1;i<=n;i++)
 {
printf("Enter element%d: ",i);
scanf("%d",&a[i]);
 }
Merge_Sort(a,1,n);\\ Calling function
printf("Sorted array:\n");\\ Output section
for(i=1;i<=n;i++)
 {
printf("%d",a[i]);
```

```
 }
getch();
}

Void Merge_Sort(int a[],,int p,int r)\\ Function definition
{
int q;
if(p<r)
 {
  q=(p+r)/2;
Merge_Sort(a,p,q);                  \\  Function calling itself
Merge_Sort(a,q+1,r);                \\  Function calling itself
Merge(a,p,q,r);                       \\ Function calling another function
 }
}

void Merge(int a[], int p,int q,int r)\\ Function definition
{
int n1,n2,i,j,k,L[5],R[5];
 n1=q-p+1;
 n2=r-q;
for(i=1;i<n=1;i++)                                   \\  Creating arrays L & R
 {
  L[i]=a[p+i-1];
 }
```

```
for(j=1;j<=n2;j++)
 {
  R[j]=a[q+j];
 }
L[n1]=32767;
R[n2]=32767;
i=1;
 j=1;
for(k=p;k<=r;k++)                          \\ Sorting section
 {
if(L[i]<=R[j])
  {
a[k]=L[i];
i=i+1;
  }
else
  {
a[k]=R[j];
   j=j+1;
  }
 }
}
```

4.3 OUTPUT:

Program for MERGE SORT

```
Enter no. of elements: 6

Enter element0: 5

Enter element1: 8

Enter element2: 7

Enter element3: 2

Enter element4: 4

Enter element5: 3

Sorted array:

2     3     4     5     7     8
```

Program No. 5

Program for Quick Sort.

5.1 Algorithm

Quick_Sort(A,p,r)

```
if p<r
  q=Partition(A,p,r)
  Quick_Sort(A,p,q-1)
  Quick_Sort(A,q+1,r)
```

Partition(A,p,r)

```
x=A[r]
i=p-1
for j=p to r-1
   if A[j]<=x
      i=i+1
      swap(A[i],A[j])
A[i+1]=A[r]
return i+1
```

5.2 Program

```
#include<stdio.h>
#include<conio.h>
Void Quick_Sort(int a[],int p,int r);\\ Function declaration
int Partition(int a[],int p,int r);
void main()
```

```
{
int a[10],i,n;
clrscr();
printf("\nProgram for QUICK SORT\n");
printf("Enter no. of elements:\n");\\ Input  section
scanf("%d",&n);
for(i=1;i<=n;i++)
 {
printf("Enter element%d: ",i);
scanf("%d",&a[i]);
 }
Quick_Sort(a,1,n);\\ Calling function
printf("Sorted array:\n");\\ Output section
for(i=1;i<=n;i++)
 {
printf("%d",a[i]);
 }
getch();
}
Void Quick_Sort(int a[],int p,int r)\\ Function  definition
{
int q;
if(p<r)
 {
  q=Partition(a,p,r);                    \\  Function calling another function
Quick_Sort(a,p,q-1);              \\  Function calling itself

Quick_Sort(a,q+1,r);                  \\ Function calling itself

 }

}


int Partition(int a[],int p,int r)\\  Function definition

{

Int i,x,j,t;

 x=a[r];                                          \\ Choosing pivot element

i=p-1;

for(j=p;j<=r;j++)

 {

if(a[j]<=x)\\  Comparison

  {

i++;

   t=a[i];\\ Swapping  values

a[i]=a[j];

a[j]=t;

  }

 }

 t=a[i+1]; \\ Swapping  values

a[i+1]=a[r];

a[r]=t;

return(i+1);

}
```

5.3 OUTPUT:

```
Program for QUICK SORT

Enter no. of elements:

8

Enter element0: 2

Enter element1: 8

Enter element2: 7

Enter element3: 1

Enter element4: 3

Enter element5: 5

Enter element6: 6

Enter element7: 4

Sorted array:

    1    2    3    4    5    6    7    8
```

Program No. 6

Program for Heap Sort.

6.1 Algorithm for HEAP SORT

Heap_Sort(A)

```
BuildMax_Heap(A)
fori=length[A] down to 2
    swap(A[1],A[i])
    heapsize[A]=heapsize[A]-1
    Maxheapify(A,1)
```

BuildMax_Heap(A)

```
heapsize[A]=length[A]
fori=floor(length[A]/2) down to 1
    Maxheapify(A,i)
```

Maxheapify(A,i)

```
l=LEFT(i)
r=RIGHT(i)
if l<=heapsize[a] and A[l]>A[i]
   largest=l
else largest=i
if r<=heapsize[A] and A[r]>A[largest]
   largest=r
if largest != i
   swap(A[i],A[largest])
```

```
10.  Maxheapify(A,largest)
```

6.2 Program

```
#include<stdio.h>
#include<conio.h>
int h;  \\  Global variable declaration
void HeapSort(int a[],int n);  \\  Function declaration
void BuildMax_Heap(int a[],int h);
void Maxheapify(int a[],inti);
void main()
{
int a[50],i,length;
clrscr();
printf("\nProgram for HEAP SORT\n");
printf("Enter no. of elements:");  \\  Input  section
scanf("%d",&h);
length=h;
for(i=1;i<=h;i++)
{
printf("\nEnterelement%d: ",i);
scanf("%d",&a[i]);
}
HeapSort(a,h); \\  Calling function
printf("\nSorted array:\n");\\ Output section
for(i=1;i<=length;i++)
{
printf("%6d",a[i]);
}
getch();
}

void HeapSort(int a[],int n)\\  Function definition
{
intt,i,j,length;
length=n;
BuildMax_Heap(a,n);\\ Function calling another function
for(i=length;i>1;i--)
{
t=a[1];\\  Swapping values
a[1]=a[i];
a[i]=t;
h--;
Maxheapify(a,1);\\ Function calling another function
printf("\nAfteriteration%d: ",i);  \\ Demonstration of sorting
for(j=1;j<=length;j++)
{
printf("%6d",a[j]);
}
}
}
```

```
void BuildMax_Heap(int a[],int h)    \\ Function definition

{

inti;

int length=h;

for(i=(length/2);i>0;i--)

{

Maxheapify(a,i);  \\  Function calling another function

}

printf("\nMaxHeap:          ");

for(i=1;i<=length;i++)

printf("%6d",a[i]); \\ Maxheap

}


void Maxheapify(int a[],inti)       \\  Function definition

{

intl,r,largest,t;

l=(2*i);

r=(2*i)+1;

if((l<=h)&&(a[l]>a[i]))                    \\ Checking for largest element

{

largest=l;

}

else

{
```

```
largest=i;
}
if((r<=h)&&(a[r]>a[largest]))
{
largest=r;
}
if(largest!=i)
{
t=a[i]; \\  Swapping values
a[i]=a[largest];
a[largest]=t;
Maxheapify(a,largest);\\  Function calling itself
}
}
```

6.3 OUTPUT:

```
Program for HEAP SORT
Enter no. of elements : 10
Enter element1 : 4
Enter element2 : 1
Enter element3 : 3
Enter element4 : 2
Enter element5 : 16
Enter element6 : 9
Enter element7 : 10
Enter element8 : 14

Enter element9 : 8

Enter element10 : 7


MaxHeap:           16   14   10   8   7   9   3   2   4   1


After iteration1 : 14   8    10   4   7   9   3   2   1   16

After iteration2 : 10   8    9    4   7   1   3   2   14  16

After iteration3 : 9    8    3    4   7   1   2   10  14  16

After iteration4 : 8    7    3    4   2   1   9   10  14  16

After iteration5 : 7    4    3    1   2   8   9   10  14  16

After iteration6 : 4    2    3    1   7   8   9   10  14  16

After iteration7 : 3    2    1    4   7   8   9   10  14  16

After iteration8 : 2    1    3    4   7   8   9   10  14  16

After iteration9 : 1    2    3    4   7   8   9   10  14  16


Sorted array :

     1    2    3    4    7    8    9    10    14    16
```

Program No. 7

Program for Selection Sort.

7.1 Algorithm

```
selectionSort(array, size)
  repeat (size - 1) times
  set the first unsorted element as the minimum
  for each of the unsorted elements
    if element < currentMinimum
      set element as new minimum
  swap minimum with first unsorted position
end selectionSort
```

7.2 Program

```
#include <stdio.h>
// function to swap the the position of two elements
void swap(int *a, int *b) {
  int temp = *a;
  *a = *b;
  *b = temp;
}

void selectionSort(int array[], int size) {
  for (int step = 0; step < size - 1; step++) {
    int min_idx = step;
    for (int i = step + 1; i < size; i++) {
      if (array[i] < array[min_idx])
        min_idx = i;
    }
    swap(&array[min_idx], &array[step]);
  }
}
// function to print an array
void printArray(int array[], int size) {
  for (int i = 0; i < size; ++i) {
    printf("%d  ", array[i]);
  }
  printf("\n");
}
int main() {
  int data[] = {20, 12, 10, 15, 2};
  int size = sizeof(data) / sizeof(data[0]);
  selectionSort(data, size);
  printf("Sorted array in Acsending Order:\n");
  printArray(data, size);
}
```

7.3 Output:

```
Sorted array in Acsending Order: 2    10    12      15     20
```

Program No. 8

Program for Counting Sort.

8.1 Algorithm for COUNTING SORT

COUNTING(A,B,k)

```
1. fori=0 to k
2.    C[i]=0
3. for j=1 to length[A]
4.    C[A[j]]=C[A[j]]+1
5. fori=1 to k
6.    C[i]=C[i]+C[i-1]
7. for j=length[A] down to 1
8.    B[C[A[j]]]=A[j]
9.    C[A[j]]=C[A[j]]-1
```

8.2 Program

```
#include<stdio.h>
#include<conio.h>
voidCount_Sort(inta[],int n,int c[],int k);  \\ Function declaration
void main()
{
int a[10],n,k,i,b[10];
clrscr();
printf("\nProgram for COUNTING SORT\n");
printf("\nEnter no. of elements: ");   \\ Input section
scanf("%d",&n);
```

```
for(i=1;i<=n;i++)
 {
printf("\nEnterelement%d: ",i);
scanf("%d",&a[i]);
 }
 k=a[n];        \\ Finding largest element of the array
for(i=1;i<=n;i++)
 {
if(a[i]>a[n])
  k=a[i];
 }
printf("Maximum element: %d",k);
Count_Sort(a,n,b,k);          \\ Calling function
printf("\nSorted array:\n");   \\ Output  section
for(i=1;i<=n;i++)
 {
printf("%6d",b[i]);
 }
getch();
}

voidCount_Sort(int a[],int n,int b[],int k)  \\ Function definition
{
Int i,j,c[10];
for(i=0;i<=k;i++)
```

```
 {
c[i]=0;
}
for(j=1;j<=n;j++)
 {
c[a[j]]=c[a[j]]+1;
 }
for(i=1;i<=k;i++)
 {
c[i]=c[i]+c[i-1];
 }
printf("\n\nNewArray C:\n");
for(i=0;i<=k;i++)
 {
printf("%2d",c[i]);
 }
for(j=n;j>0;j--)           \\ Sorting Section
 {
b[c[a[j]]]=a[j];
c[a[j]]=c[a[j]]-1;
 }
printf("\n\nArray B:\n");   \\ Array B after sorting
for(i=1;i<=n;i++)
 {
printf("%2d",b[i]);
```

```
 }
}
```

8.3 OUTPUT:

```
Program for  COUNTING SORT

Enter no. of elements : 9

Enter element1 : 2

Enter element2 : 5

Enter element3 : 3

Enter element4 : 0

Enter element5 : 1

Enter element6 : 2

Enter element7 : 3

Enter element8 : 0

Enter element9 : 3


Maximum element : 5

Array B :

0  0  1  2  2 3 3 3  5

Sorted array :

        0     0     1     2     2     3     3     3     5
```

Program No. 9

Program for Counting Sort (change in while Loop).

9.1 Algorithm for COUNTING SORT

COUNTING(A,B,k)

```
fori=0 to k
    C[i]=0
for j=1 to length[A]
    C[A[j]]=C[A[j]]+1
fori=1 to k
    C[i]=C[i]+C[i-1]
for j= 1to length[A]   // Changes
    B[C[A[j]]]=A[j]
    C[A[j]]=C[A[j]]-1
```

9.2 Program

```
#include<stdio.h>
#include<conio.h>
voidCount_Sort(inta[],int n,int c[],int k);  \\ Function declaration
void main()
{
int a[10],n,k,i,b[10];
clrscr();
printf("\nProgram for COUNTING SORT\n");
printf("\nEnter no. of elements: ");   \\ Input section
scanf("%d",&n);
```

```
for(i=1;i<=n;i++)
 {
printf("\nEnterelement%d: ",i);
scanf("%d",&a[i]);
 }
 k=a[n];         \\ Finding largest element of the array
for(i=1;i<=n;i++)
 {
if(a[i]>a[n])
  k=a[i];
 }
printf("Maximum element: %d",k);
Count_Sort(a,n,b,k);           \\ Calling function
printf("\nSorted array:\n");   \\ Output  section
for(i=1;i<=n;i++)
 {
printf("%6d",b[i]);
 }
getch();
}

voidCount_Sort(int a[],int n,int b[],int k)  \\ Function definition
{
Int i,j,c[10];
for(i=0;i<=k;i++)
```

```
 {
c[i]=0;
}
for(j=1;j<=n;j++)
 {
c[a[j]]=c[a[j]]+1;
 }
for(i=1;i<=k;i++)
 {
c[i]=c[i]+c[i-1];
 }
printf("\n\nNewArray C:\n");
for(i=0;i<=k;i++)
 {
printf("%2d",c[i]);
 }
for(j=1;j<=n; j++)           \\ Sorting Section
 {
b[c[a[j]]]=a[j];
c[a[j]]=c[a[j]]-1;
 }
printf("\n\nArray B:\n");   \\ Array B after sorting
for(i=1;i<=n;i++)
 {
printf("%2d",b[i]);
```

```
 }
}
```

8.3 OUTPUT:

```
Program for  COUNTING SORT

Enter no. of elements : 9

Enter element1 : 2

Enter element2 : 5

Enter element3 : 3

Enter element4 : 0

Enter element5 : 1

Enter element6 : 2

Enter element7 : 3

Enter element8 : 0

Enter element9 : 3


Maximum element : 5

Array B :

0  0  1  2  2 3 3 3  5

Sorted array :

     0     0     1     2     2     3     3     3     5
```

Program No. 10

Program for Shell Sort.

10.1 Algorithm for SHELL SORT

Shell_Sort(A)

```
1. incr=n/2
2. whileincr>0
3.   fori=incr+1 to n
4.   j=i-incr
5.   while j>0
6.     if A[j]>A[j+incr]
7.     swap(A[j],A[j+incr])
8.       j=j-incr
9.    else j=0
10. incr=incr/2
```

10.2 Program

```
#include<stdio.h>
#include<conio.h>
voidShell_Sort(int a[],int n);\\  Function  declaration
void main()
{
inti,a[10],n;
clrscr();
printf("\nProgram for SHELL SORT\n");
printf("Enter no. of elements:\n");\\  Input  section
```

```
scanf("%d",&n);
for(i=1;i<=n;i++)
 {
printf("Enter element%d: ",i);
scanf("%d",&a[i]);
 }
Shell_Sort(a,n);\\  Calling  function
printf("Sorted array:\n");\\  Output  section
for(i=1;i<=n;i++)
 {
printf("%6d",a[i]);
 }
getch();
}

voidShell_Sort(int a[],int n)\\  Function  definition
{
intincr,i,j,t;
incr=n/2;
while(incr>0)
 {
for(i=(incr+1);i<=n;i++)
  {
   j=i-incr;
while(j>0)
```

```
   {
if(a[j]>a[j+incr])                        \\ Comparison
    {
     t=a[j];\\ Swapping  values
a[j]=a[j+incr];
a[j+incr]=t;
     j=j-incr;
    }
else
    {
     j=0;
    }
   }
  }
incr=incr/2;
 }
}
```

10.3 OUTPUT:

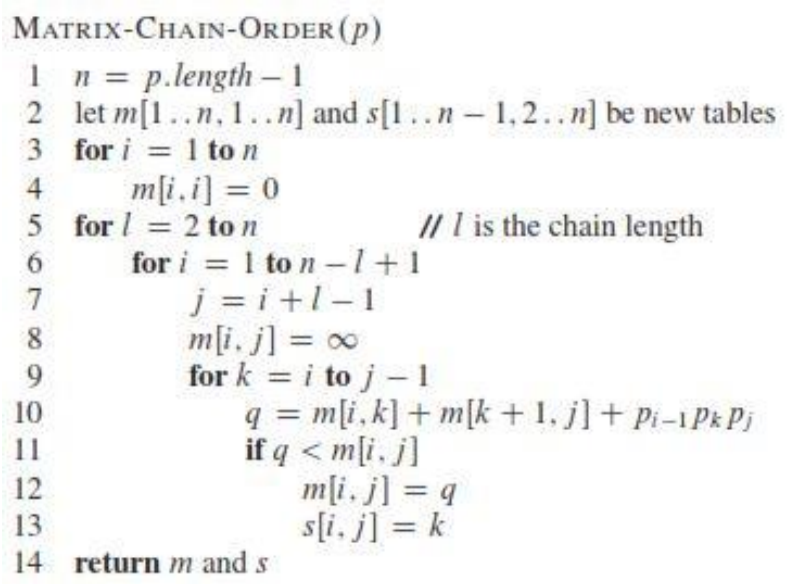
```
Program for SHELL SORT
Enter no. of elements:
8
Enter element1: 10
Enter element2: 2
Enter element3: 8
Enter element4: 12

Enter element5: 14

Enter element6: 6

Enter element7: 4

Enter element8: 16

Sorted array:

        2       4       6       8       10      12      14      16
```

Program No. 11

Program for matrix chain multiplication.

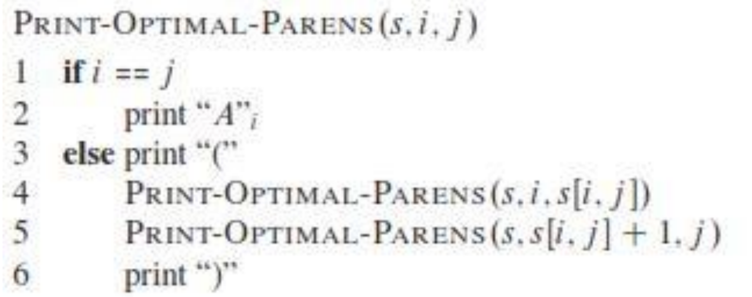
11.1 Algorithm

Matrix-chain-order (p)



```
MATRIX-CHAIN-ORDER(p)
n = p.length − 1
let m[1..n, 1..n] and s[1..n − 1, 2..n] be new tables
for i = 1 to n
    m[i, i] = 0
for l = 2 to n                // l is the chain length
    for i = 1 to n − l + 1
        j = i + l − 1
        m[i, j] = ∞
        for k = i to j − 1
            q = m[i, k] + m[k + 1, j] + $p_{i-1} p_k p_j$
            if q < m[i, j]
                m[i, j] = q
                s[i, j] = k
return m and s
```

Print-optimal-parantasization ( )



```
PRINT-OPTIMAL-PARENS(s, i, j)
if i == j
    print "A"$_i$
else print "("
    PRINT-OPTIMAL-PARENS(s, i, s[i, j])
    PRINT-OPTIMAL-PARENS(s, s[i, j] + 1, j)
    print ")"
```

11.2 Program

```
#include<stdio.h>
#include<conio.h>
int m[20][20],s[20][20];
int i,j,k,a,l,n,p[20];
```

```
void matrix_chain_order(int p[]);
//void print_optimal_parens(int s[][] ,int i ,int j);
void main()
 {
  clrscr();
  printf("\nEnter the size of p array") ;
  scanf("%d",&a);
  for(i=0;i<a;i++)
                {
                 printf("\nEnter the element p%d=\t",i);
                 scanf("%d",&p[i]);
                }
   matrix_chain_order(p);
   printf("\n\n");
   for(i=1;i<a;i++)
                 {
                  for(j=1;j<a;j++)
                              {
                               printf("%d\t",m[i][j]);
                              }
                              printf("\n");
                 }
                 printf("\n\n");
                 for(i=1;i<a-1;i++)
                 {
                  for(j=2;j<a;j++)
                              {
                               printf("%d\t",s[i][j]);
                              }
                              printf("\n");
                 }
   //          print_optimal_parens(s,1,n);

  getch();
 }
 void matrix_chain_order(int p[])
```

```
           {
            int q;
            n=a-1;
            for(i=1;i<=n;i++)
                     m[i][i]=0;
            for(l=2;l<=n;l++)
             {
                     for(i=1;i<=n-l+1;i++)
                       {
                        j=i+l-1;
                        m[i][j]=32000;
                       for(k=i;k<=j-1;k++)
                                {
                                 q=m[i][k]+m[k+1][j]+(p[i-1]*p[k]*p[j]);
                                 if(q<m[i][j])
                                  {        m[i][j]=q;
                                                     s[i][j]=k;
                                  }

                                }

                       }
             }
           }
/*   void print_optimal_parens(int s[][],int i,int j)
            {
            if(i==j)
                     {
                      printf("A%d",i);
                     }
            else
                     {
                      printf("(");
                      print_optimal_parens(s,i,s[i][j]);
                      print_optimal_parens(s,s[i][j]+1,j);
                      printf(")");
```

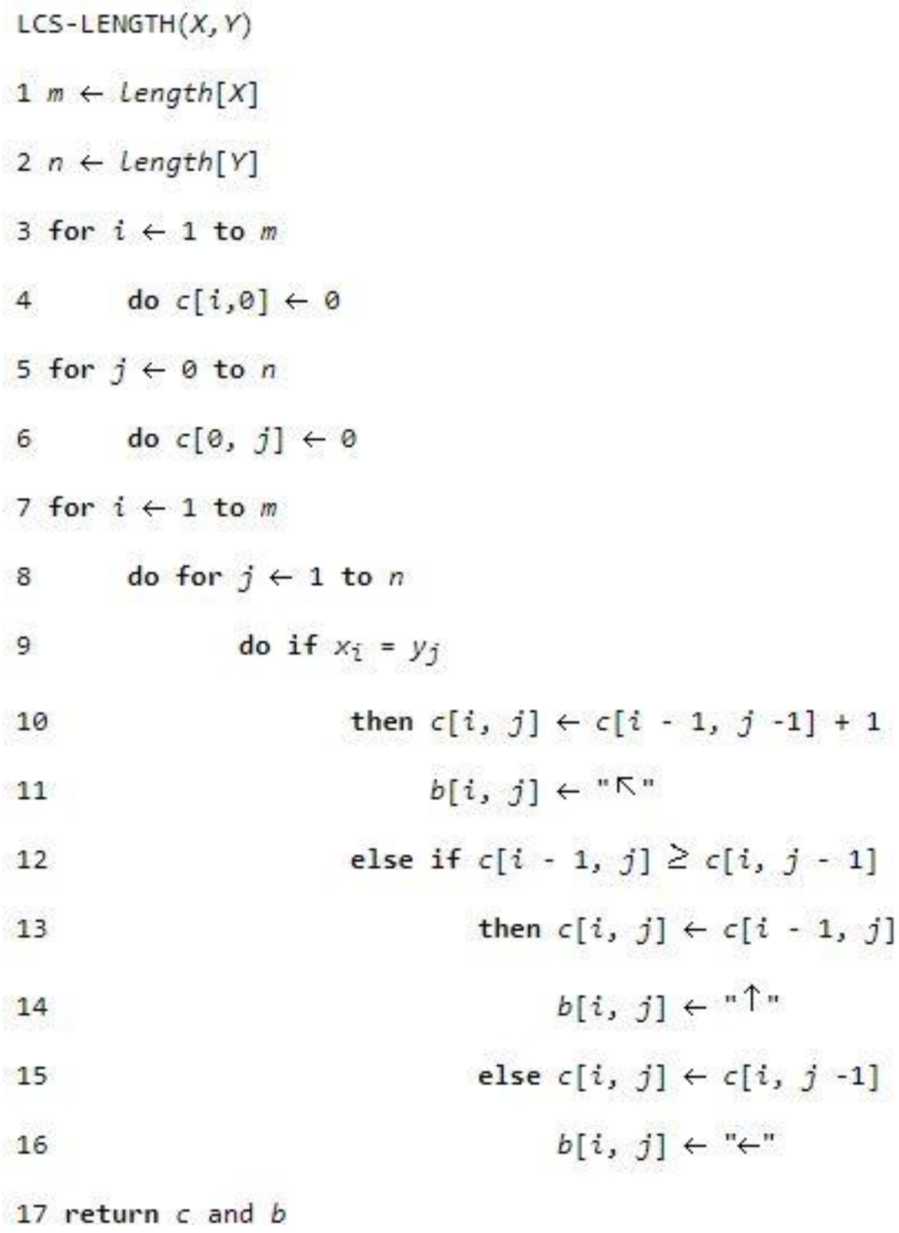
```
                }
        }    */
```

11.3 Output

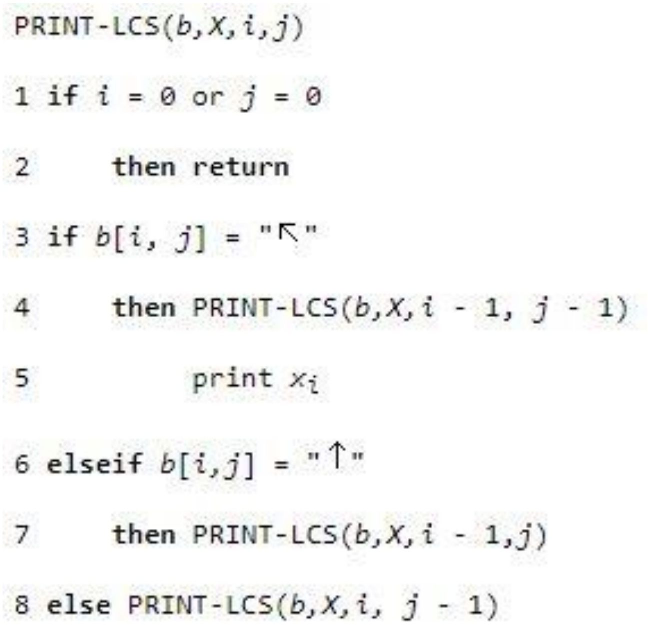
Program No. 12

Program for longest common sub-sequences

12.1 Algorithm



```
LCS-LENGTH($X, Y$)
$m \leftarrow length[X]$
$n \leftarrow length[Y]$
for $i \leftarrow 1$ to $m$
    do $c[i,0] \leftarrow 0$
for $j \leftarrow 0$ to $n$
    do $c[0, j] \leftarrow 0$
for $i \leftarrow 1$ to $m$
    do for $j \leftarrow 1$ to $n$
        do if $x_i = y_j$
            then $c[i, j] \leftarrow c[i - 1, j -1] + 1$
                $b[i, j] \leftarrow$ "↖"
            else if $c[i - 1, j] \geq c[i, j - 1]$
                then $c[i, j] \leftarrow c[i - 1, j]$
                    $b[i, j] \leftarrow$ "↑"
                else $c[i, j] \leftarrow c[i, j -1]$
                    $b[i, j] \leftarrow$ "←"
return $c$ and $b$
```



```
PRINT-LCS(b,X,i,j)
if i = 0 or j = 0
    then return
if b[i, j] = "↖"
    then PRINT-LCS(b,X,i - 1, j - 1)
         print $x_i$
elseif b[i,j] = "↑"
    then PRINT-LCS(b,X,i - 1,j)
else PRINT-LCS(b,X,i, j - 1)
```

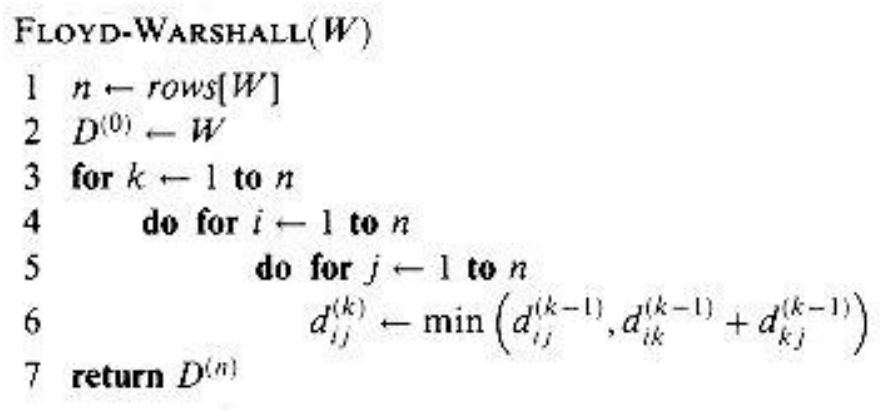
12.3 Program

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

int i, j, m, n, LCS_table[20][20];
char S1[20] = "ACADB", S2[20] = "CBDA", b[20][20];

void lcsAlgo() {
  m = strlen(S1);
  n = strlen(S2);

  // Filling 0's in the matrix
  for (i = 0; i <= m; i++)
    LCS_table[i][0] = 0;
  for (i = 0; i <= n; i++)
    LCS_table[0][i] = 0;


  // Building the mtrix in bottom-up way
  for (i = 1; i <= m; i++)
    for (j = 1; j <= n; j++) {
      if (S1[i - 1] == S2[j - 1]) {
        LCS_table[i][j] = LCS_table[i - 1][j - 1] + 1;
      } else if (LCS_table[i - 1][j] >= LCS_table[i][j - 1]) {
        LCS_table[i][j] = LCS_table[i - 1][j];
      } else {
        LCS_table[i][j] = LCS_table[i][j - 1];
      }
    }


  int index = LCS_table[m][n];
  char lcsAlgo[index + 1];
  lcsAlgo[index] = '\0';


  int i = m, j = n;
  while (i > 0 && j > 0) {
    if (S1[i - 1] == S2[j - 1]) {
      lcsAlgo[index - 1] = S1[i - 1];
      i--;
      j--;
      index--;
    }

    else if (LCS_table[i - 1][j] > LCS_table[i][j - 1])
      i--;
    else
      j--;
  }

  // Printing the sub sequences
  printf("S1 : %s \nS2 : %s \n", S1, S2);
  printf("LCS: %s", lcsAlgo);
}

int main() {
  lcsAlgo();
  printf("\n");
}
```

Program No. 13

Implement All-Pairs Shortest Paths problem using Floyd's algorithm.

13.1 Algorithm



```
FLOYD-WARSHALL($W$)
$n \leftarrow rows[W]$
$D^{(0)} \leftarrow W$
for $k \leftarrow 1$ to $n$
    do for $i \leftarrow 1$ to $n$
        do for $j \leftarrow 1$ to $n$
            $d_{ij}^{(k)} \leftarrow \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right)$
return $D^{(n)}$
```

13.2 Program

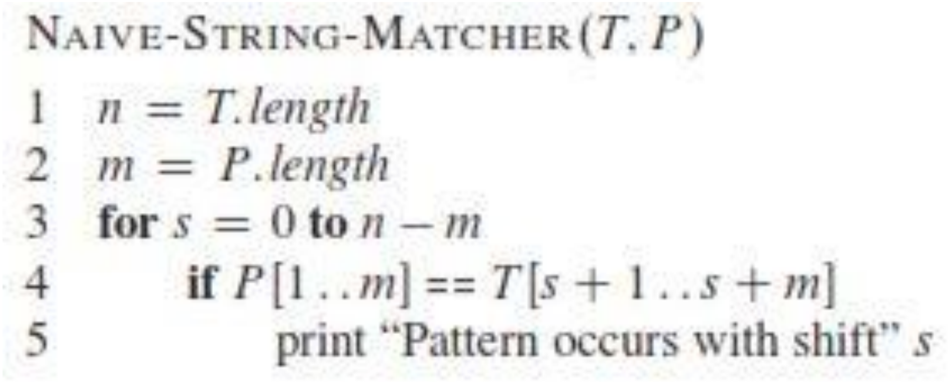
```
#include <stdio.h>
#define nV 4
#define INF 999
void printMatrix(int matrix[][nV]);
// Implementing floyd warshall algorithm
void floydWarshall(int graph[][nV]) {
  int matrix[nV][nV], i, j, k;
  for (i = 0; i < nV; i++)
    for (j = 0; j < nV; j++)
      matrix[i][j] = graph[i][j];
  // Adding vertices individually
  for (k = 0; k < nV; k++) {
    for (i = 0; i < nV; i++) {
      for (j = 0; j < nV; j++) {
        if (matrix[i][k] + matrix[k][j] < matrix[i][j])
```

```
          matrix[i][j] = matrix[i][k] + matrix[k][j];
      }
    }
  }
  printMatrix(matrix);
}
void printMatrix(int matrix[][nV]) {
  for (int i = 0; i < nV; i++) {
    for (int j = 0; j < nV; j++) {
      if (matrix[i][j] == INF)
        printf("%4s", "INF");
      else
        printf("%4d", matrix[i][j]);
    }
    printf("\n");
  }
}
int main( ) {
  int graph[nV][nV] = {{0, 3, INF, 5},
             {2, 0, INF, 4},
             {INF, 1, 0, INF},
             {INF, INF, 2, 0}};
  floydWarshall(graph);
}
```

Program No. 14

Program for naïve string matching.

14.1 Algorithm



```
NAIVE-STRING-MATCHER(T, P)
n = T.length
m = P.length
for s = 0 to n − m
    if P[1 . . m] == T[s + 1 . . s + m]
        print "Pattern occurs with shift" s
```

14.2 Program

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

void search(char* pat, char* txt)
{
    int M = strlen(pat);
    int N = strlen(txt);

    /* A loop to slide pat[] one by one */
    for (int i = 0; i <= N - M; i++) {
        int j;

        /* For current index i, check for pattern match */
        for (j = 0; j < M; j++)
            if (txt[i + j] != pat[j])
                break;

        if (j
            == M) // if pat[0...M-1] = txt[i, i+1, ...i+M-1]
            printf("Pattern found at index %d \n", i);
    }
}

// Driver's code
int main()
{
    char txt[] = "AABAACAADAABAAABAA";
    char pat[] = "AABA";

      // Function call
    search(pat, txt);
    return 0;
}
```

14.2 Output

```
Pattern found at index 0

Pattern found at index 9

Pattern found at index 13
```