University of Petroleum

&

Energy Studies

SCHOOL OF COMPUTER SCIENCE

Name: Akshat Agarwal

Course: BTech CSE

SAP ID: 500118953

BATCH: 1

PRESENTED TO: Dr. Rahul Kumar Singh

Semester: 3

GitHub Link
For all the Experiments

https://github.com/Akshat29065/DAA_Lab.git

Experiment 1: Patterns

- ## Star Patterns

1. Diamond:

Input:

```c
#include <stdio.h>

int main(){
    int i, j, k, n, m;
    printf("Enter the Number of rows: ");
    scanf("%d", &n);
    m = n;

    for(i=1; i<=n; i++){
        for(j=1; j<= m-1; j++){
            printf(" ");
        }
        for(k=1; k<=2*i-1; k++){
            printf("*");
        }
        m--;
        printf("\n");
    }

    for(i=n; i>=1; i--){
        for(j=1; j<m; j++){
            printf(" ");
        }
        for(k=1; k<=2*i-1; k++){
            printf("*");
        }
        m++;
        printf("\n");
    }

}
```

Output:

```
PS D:\UPES\2nd Year\Semester
Enter the Number of rows: 5
        *
       ***
      *****
     *******
    *********
    *********
     *******
      *****
       ***
        *
```

2. Floyd's Triangle:

Input:

```c
#include <stdio.h>

int main()
{
   int n;
   printf("Enter the number of rows: ");
   scanf("%d", &n);
   int num = 1;
   for (int i = 1; i <= n; i++)
   {
       for (int j = 1; j <= i; j++)
       {
           printf("%d ", num);
           num++;
       }
       printf("\n");
   }
   return 0;
}
```

Output:

```
PS D:\UPES\2nd Year\Semester
\Floyd_triangle }
Enter the number of rows: 5
1
2 3
4 5 6
7 8 9 10
11 12 13 14 15
```

3. Half Diamond:

Input:

```c
#include<stdio.h>
int main(){
    int i, j, n, m;

    printf("Enter number of columns:");
    scanf("%d",&n);
    m=1;
    for(i=1;i<n*2;i++){
        for(j=1; j<=m; j++){
            printf("*");
        }
        if(i < n){
            m++;
        }
        else{
            m--;
        }
        printf("\n");
    }
    return 0;
}
```

Output:

```
PS D:\UPES\2nd Year\Semest
f_diamond }
Enter number of columns:5
*
**
***
****
*****
****
***
**
*
```

4. Inverted Pyramid:

Input:

```c
#include <stdio.h>

int main()
{
    int i, j, k, n, m = 1;
    printf("Enter the number of rows: ");
    scanf("%d",&n);

    for(i=n; i>=1; i--){
        for(j=1; j<m; j++){
            printf(" ");
        }
        for(k=1; k<=2*i-1; k++){
            printf("*");
        }
        m++;
        printf("\n");
    }

}
```

Output:

```
PS D:\UPES\2nd Year\Semester
 { .\Inverted_pyramid }
Enter the number of rows: 5
*********
 *******
  *****
   ***
    *
```

5. Inverted Right Pyramid:

Input:

```c
#include <stdio.h>

int main(){
    int i, j, k, n;
    printf("Enter the number of rows: ");
    scanf("%d", &n);

    for(i=1; i<=n; i++){
        for(j=n; j>=i; j--){
            printf("*");
        }
        printf("\n");
    }
}
```

Output:

```
PS D:\UPES\2nd Year\Semester
le } ; if ($?) { .\Inverted_
Enter the number of rows: 5
*****
****
***
**
*
```

6. Left Arrow:

Input:

```c
#include <stdio.h>

int main(){
    int i, j, k, n;
    printf("Enter the number of rows: ");
    scanf("%d", &n);
    for (i = 1; i <= n; i++){
        for(j=n;j>=i;j--){
            printf("*");
        }
        printf("\n");
    }
    for (i = 1; i <= n; i++){
        for(j=1;j<=i;j++){
            printf("*");
        }
        printf("\n");
    }

}
```

Output:

```
PS D:\UPES\2nd Year\Semester
row }
Enter the number of rows: 5
*****
****
***
**
*
*
**
***
****
*****
```

7. Mirror Inverted Right Triangle:

Input:

```c
#include <stdio.h>

int main(){
    int i, j, k, n;
    printf("Enter the Number of Rows: ");
    scanf("%d", &n);

    for(i=n; i>=0; i--){
        for(j=n; j>=i; j--){
            printf(" ");
        }
        for(k=0; k<=i; k++){
            printf("*");
        }
        printf("\n");
    }
}
```

Output:

```
PS D:\UPES\2nd Year\Semester
d_right_triangle } ; if ($?)
Enter the Number of Rows: 5
 ******
  *****
   ****
    ***
     **
      *
```

8. Mirror Rhombus:

Input:

```c
#include <stdio.h>

int main(){
    int i, j, k, n;
    printf("Enter the Number of Rows: ");
    scanf("%d", &n);

    for(i=1; i<=n; i++){
        for(k=0; k<i; k++){
            printf(" ");
        }
        for(j=0; j<n; j++){
            printf("*");
        }
        printf("\n");
    }
}
```

Output:

```
PS D:\UPES\2nd Year\Semester
\Mirror_rhombus }
Enter the Number of Rows: 5
 *****
  *****
   *****
    *****
     *****
```

9. Mirror Right Angle Triangle:

Input:

```c
#include <stdio.h>

int main(){
    int i, j, k, n, m=1;
    printf("Enter the Number of Rows: ");
    scanf("%d", &n);

    for(i = n; i >= 1; i--){
        for(j = 1; j < i-1; j++){
            printf(" ");
        }
        for(k = 1; k<=m; k++){
            printf("*");
        }
        printf("\n");
        m++;
    }
    return 0;

}
```

Output:

```
PS D:\UPES\2nd Year\Semester
e_triangle } ; if ($?) { .\
Enter the Number of Rows: 5
    *
   **
  ***
 ****
*****
```

10. Pascal's Triangle:

Input:

```c
#include<stdio.h>
int main(){
    int n;
    printf("Enter the number of rows: ");
    scanf("%d",&n);
    for (int i = 1; i <= n; i++)    {
        for(int j = 1; j <= n-i; j++){
            printf("  ");
        }
        int C = 1;
        for (int j = 1; j <= i; j++){
            printf("%d   ", C);
            C = C * (i - j) / j;
        }
        printf("\n");
    }
    return 0;
}
```

Output:

```
PS D:\UPES\2nd Year\Semester
 .\Pascal_triangle }
Enter the number of rows: 5
        1
      1   1
    1   2   1
  1   3   3   1
1   4   6   4   1
```

11. Pyramid:

Input:

```c
#include <stdio.h>
int main(){
    int i, j, k, n, m;
    printf("Enter the number of rows: ");
    scanf("%d",&n);
    m=n;
    for(i=1; i<=n; i++){
        for(j=1; j<= m-1; j++){
            printf(" ");
        }
        for(k=1; k<=2*i-1; k++){
            printf("*");
        }
        m--;
        printf("\n");
    }
}
```

Output:

```
PS D:\UPES\2nd Year\Semester
Enter the number of rows: 5
    *
   ***
  *****
 *******
*********
```

12. Rhombus:

Input:

```c
#include <stdio.h>
int main(){
    int i, j, k, n;
    printf("Enter the number of rows: ");
    scanf("%d",&n);
    for(i=n;i>=1;i--){
        for(j=1;j<=i-1;j++){
            printf(" ");
        }
        for(k=1;k<=n;k++){
            printf("*");
        }
        printf("\n");
    }
    return 0;
}
```

Output:

```
PS D:\UPES\2nd Year\Semester
Enter the number of rows: 5
    *****
   *****
  *****
 *****
*****
```

13. Right Angle Triangle:

Input:

```c
#include <stdio.h>

int main(){
    int i, j, n;
    printf("Enter the Number of Rows: ");
    scanf("%d", &n);
    for(i=0;i<n;i++){
        for(j=0;j<=i;j++){
            printf("*");
        }
        printf("\n");
    }
}
```

Output:

```
PS D:\UPES\2nd Year\Semester
 if ($?) { .\Right_angle_tria
Enter the Number of Rows: 5
*
**
***
****
*****
```

14. Right Arrow:

Input:

```c
#include <stdio.h>
int main(){
  int i, j, k, n;
  printf("Enter the number of columns");
  scanf("%d",&n);
 for(i=0;i<n;i++){
   for(j=0;j<i;j++){
      printf(" ");
   }
   for(k=1;k<=n-i;k++){
     printf("*");
   }
   printf("\n");
 }
for(i=1;i<n;i++){
  for(j=1;j<n-i;j++){
    printf(" ");
  }
  for(k=1;k<=i+1;k++){
    printf("*");
  }
  printf("\n");
}
  return 0;
}
```

Output:

```
PS D:\UPES\2nd Year\Semester 3\
_arrow }
Enter the number of columns: 5
*****
 ****
  ***
   **
    *
   **
  ***
 ****
*****
```

15. Square:

Input:

```c
#include <stdio.h>
int main(){
    int i, j, n;
    printf("Enter the Number of Rows: ");
    scanf("%d", &n);
    for(i = 0; i < n; i++){
        for(j = 0; j < n; j++){
            printf("*");
        }
        printf("\n");
    }
}
```

Output:

```
PS D:\UPES\2nd Year\Semester
Enter the Number of Rows: 5
*****
*****
*****
*****
*****
```

- **Number**

1. Diamond Shape:

Input:

```c
#include <stdio.h>
int main() {
    int i, j, k, n = 5;
    for(i = 1; i <= n; i++) {
        for(j = i; j < n; j++){
            printf(" ");
        }
        for(k = 1; k <= (2*i - 1); k++){
            printf("%d", i);
        }
        printf("\n");
    }
    for(i = n-1; i >= 1; i--){
        for(j = n; j > i; j--){
            printf(" ");
        }
        for(k = 1; k <= (2*i - 1); k++){
            printf("%d", i);
        }
        printf("\n");
    }
    return 0;
}
```

Output:

```
PS D:\UPES\2nd Yea
.\Diamond_Shape }
    1
   222
  33333
 4444444
555555555
 4444444
  33333
   222
    1
```

2. Hollow Triangle:

Input:

```c
#include <stdio.h>
int main() {
    int i, j, k, n = 5;
    for(i = 1; i <= n; i++){
        for(j = i; j < n; j++){
            printf(" ");
        }
        for(k = 1; k <= (2*i - 1); k++){
            if(k == 1 || k == (2*i - 1))
                printf("*");
            else
                printf(" ");
        }
        printf("\n");
    }
    for(i = n-1; i >= 1; i--){
        for(j = n; j > i; j--){
            printf(" ");
        }
        for(k = 1; k <= (2*i - 1); k++){
            if(k == 1 || k == (2*i - 1))
                printf("*");
            else
                printf(" ");
        }
        printf("\n");
    }
    return 0;
}
```

Output:

```
PS D:\UPES\2
{ .\Hollow_D
    *
   * *
  *   *
 *     *
*       *
 *     *
  *   *
   * *
    *
```

3. Inverted Pyramid:

Input:

```c
#include <stdio.h>
int main(){
    int i, j, k, n = 5;
    for(i = n; i >= 1; i--){
        for(j = n; j > i; j--){
            printf(" ");
        }
        for(k = 1; k <= (2*i - 1); k++){
            printf("%d", i);
        }
        printf("\n");
    }
    return 0;
}
```

Output:

```
PS D:\UPES\
$?) { .\Inv
555555555
 4444444
  33333
   222
    1
```

4. Inverted Right Angle Pyramid:

Input:

```c
#include <stdio.h>
int main(){
    int i, j, n = 5;
    for(i = n; i >= 1; i--){
        for(j = 1; j <= i; j++){
            printf("%d ", j);
        }
        printf("\n");
    }
    return 0;
}
```

Output:

```
PS D:\UPES
ght_Angled
1 2 3 4 5
1 2 3 4
1 2 3
1 2
1
```

5. Left Aligned Pyramid:

Input:

```c
#include <stdio.h>
int main(){
    int i, j, k, n = 5;
    for(i = 1; i <= n; i++){
        for(k = 1; k <= i; k++){
            printf("%d ", k);
        }
        printf("\n");
    }
    return 0;
}
```

Output:

```
PS D:\UPES\
} ; if ($?)
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

6. Number Diamond:

Input:

```c
#include <stdio.h>
int main(){
    int i, j, k, n = 5;
    for(i = 1; i <= n; i++){
        for(j = i; j < n; j++){
            printf(" ");
        }
        for(k = 1; k <= (2*i - 1); k++){
            printf("%d", i);
        }
        printf("\n");
    }
    for(i = n-1; i >= 1; i--){
        for(j = n; j > i; j--){
            printf(" ");
        }
        for(k = 1; k <= (2*i - 1); k++){
            printf("%d", i);
        }
        printf("\n");
    }
    return 0;
}
```

Output:

```
PS D:\UPES
{ .\Number
    1
   222
  33333
 4444444
555555555
 4444444
  33333
   222
    1
```

7. Number Square:

Input:

```c
#include <stdio.h>
int main(){
    int i, j, n = 5;
    for(i = 1; i <= n; i++){
        for(j = 1; j <= n; j++){
            printf("%d ", i);
        }
        printf("\n");
    }
    return 0;
}
```

Output:

```
PS D:\UPES
.\Number_S
1 1 1 1 1
2 2 2 2 2
3 3 3 3 3
4 4 4 4 4
5 5 5 5 5
```

8. Number Triangle:

Input:

```c
#include <stdio.h>
int main() {
    int i, j, n = 5;
    for(i = 1; i <= n; i++){
        for(j = 1; j <= i; j++){
            printf("%d ", i);
        }
        printf("\n");
    }
    return 0;
}
```

Output:

```
PS D:\UPES\
) { .\Numbe
1
2 2
3 3 3
4 4 4 4
5 5 5 5 5
```

9. Pascal's Triangle:

Input:

```c
#include <stdio.h>
int main(){
    int i, j, n = 5, coef = 1;
    for(i = 0; i < n; i++){
        for(j = 0; j < n - i; j++){
            printf(" ");
        }
        for(j = 0; j <= i; j++){
            if(j == 0 || j == i){
                coef = 1;
            } else{
                coef = coef * (i - j + 1) / j;
            }
            printf("%d ", coef);
        }
        printf("\n");
    }
    return 0;
}
```

Output:

```
PS D:\UPES\2
) { .\Pascal
     1
    1 1
   1 2 1
  1 3 3 1
 1 4 6 4 1
```

10. Pyramid Triangle:

Input:

```c
#include <stdio.h>
int main(){
    int i, j, k, n = 5;
    for(i = 1; i <= n; i++){
        for(j = i; j < n; j++){
            printf(" ");
        }
        for(k = 1; k <= (2*i - 1); k++){
            printf("%d", k);
        }
        printf("\n");
    }
    return 0;
}
```

Output:

```
PS D:\UPES
) { .\Pyra
     1
    123
   12345
  1234567
 123456789
```

11. Pyramid:

Input:

```c
#include <stdio.h>
int main(){
    int i, j, k, n = 5;
    for(i = 1; i <= n; i++){
        for(j = i; j < n; j++){
            printf(" ");
        }
        for(k = 1; k <= (2*i - 1); k++){
            printf("%d", i);
        }
        printf("\n");
    }
    return 0;
}
```

Output:

```
PS D:\UPES
    1
   222
  33333
 4444444
555555555
```

12. Reverse Number:

Input:

```c
#include <stdio.h>
int main(){
    int i, j, n = 5;
    for(i = n; i >= 1; i--){
        for(j = 1; j <= i; j++){
            printf("%d ", j);
        }
        printf("\n");
    }
    return 0;
}
```

Output:

```
PS D:\UPES\
{ .\Reverse
1 2 3 4 5
1 2 3 4
1 2 3
1 2
1
```

13. Right Aligned Pyramid:

Input:

```c
#include <stdio.h>
int main(){
    int i, j, n = 5;
    for(i = 1; i <= n; i++){
        for(j = i; j < n; j++){
            printf(" ");
        }
        for(j = 1; j <= i; j++){
            printf("%d ", j);
        }
        printf("\n");
    }
    return 0;
}
```

Output:

```
PS D:\UPES
d } ; if (
    1
   1 2
  1 2 3
 1 2 3 4
1 2 3 4 5
```

14. Right Angled Pyramid:

Input:

```c
#include <stdio.h>
int main(){
    int i, j, n = 5;
    for(i = 1; i <= n; i++){
        for(j = 1; j <= i; j++){
            printf("%d ", j);
        }
        printf("\n");
    }
    return 0;
}
```

Output:

```
PS D:\UPES\
e } ; if ($
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

Experiment 2: Problems

1. Find sum of all array elements using recursion.

Input:

```c
#include <stdio.h>
int SumArray(int A[], int N){
    if (N <= 0)
        return 0;
    return (SumArray(A, N - 1) + A[N - 1]);
}
int main(){
    int A[] = { 1, 2, 3, 4, 5 };
    int N = sizeof(A) / sizeof(A[0]);
    printf("%d", SumArray(A, N));
    return 0;
}
```

Output:

```
PS D:\
15
```

2. Create an array ‘a1’ with ‘n’ elements. Insert an element in ith position of ‘a1’ and also delete an element from jth position of ‘a1’.

Inserting Input:

```c
#include <stdio.h>
int main(){
    int a[] = {1,2,4,5};
    int n = sizeof(a)/sizeof(a[0]);
    int position;
    int element;
    printf("Enter the position of the element you want to Insert: ");
    scanf("%d",&position);
    printf("Enter the element: ");
    scanf("%d", &element);

    for(int i = n-1; i >= position - 1; i--){
        a[i+1] = a[i];
    }
    a[position - 1] = element;
    printf("Array after insertion: \n");
    for(int j = 0; j<n ; j++)
        printf("%d\n", a[j]);
}
```

Output:

```
PS D:\UPES\2nd Year\Se
Enter the position of
Enter the element: 3
Array after insertion:
1
2
3
4
5
```

Deleting Input:

```c
#include <stdio.h>
int main(){
    int a[] = {1,2,3,4,5,6};
    int n = sizeof(a)/sizeof(a[0]);
    int position;
    printf("Enter the position you want to delete: ");
    scanf("%d", &position);
    for (int i = position -1; i < n -1; i++){
        a[i] = a[i+1];
    }
    printf("Array after deletion: ");
    for(int i = 0; i < n-1; i++){
        printf("%d ", a[i]);
    }
}
```

Output:

```
PS D:\UPES\2nd Year\Semester 3\DAA\Lab> c
Enter the position you want to delete: 3
Array after deletion: 1 2 4 5 6
```

3. Convert uppercase string to lowercase using for loop.

Input:

```c
#include <stdio.h>
#include <conio.h>
int main (){
    char str[30];
    int i;
    printf ("Enter the string: ");
    fgets(str, sizeof(str), stdin);
    for ( i = 0; i <= strlen(str); i++){  // The ASCII value of A is 65 and Z is 90
        if (str[i] >= 65 && str[i] <= 90)
        str[i] = str[i] + 32; /* add 32 to string character to convert into lower case. */
    }
    printf ("Upper Case to Lower case string is: %s", str);
    return 0;
}
```

Output:

```
PS D:\UPES\2nd Year\Semester 3\DAA\Lab> cd "d:\UPES\2nd Year\Semester 3\DAA\Lab\Experiment 2\Ex
3.c: In function 'main':
3.c:9:23: warning: implicit declaration of function 'strlen' [-Wimplicit-function-declaration]
     for ( i = 0; i <= strlen(str); i++){  // The ASCII value of A is 65 and Z is 90
                       ^~~~~~
3.c:9:23: warning: incompatible implicit declaration of built-in function 'strlen'
3.c:9:23: note: include '<string.h>' or provide a declaration of 'strlen'
Enter the string: AKSHAT AGARWAL
Upper Case to Lower case string is: akshat agarwal
PS D:\UPES\2nd Year\Semester 3\DAA\Lab\Experiment 2\Experiment 2>
```

4. Find the sum of rows and columns of matrix of given order (row x column).

Input:

```c
#include <stdio.h>
int main(){
    int array[10][10];
    int i, j, m, n, sum = 0;
    printf("Enter the order of the matrix\n");
    scanf("%d %d", &m, &n);
    printf("Enter the co-efficients of the matrix\n");
    for (i = 0; i < m; ++i){
        for (j = 0; j < n; ++j){
            scanf("%d", &array[i][j]);
        }
    }
    for (i = 0; i < m; ++i){
        for (j = 0; j < n; ++j){
            sum = sum + array[i][j] ;
        }
        printf("Sum of the %d row is = %d\n", i, sum);
        sum = 0;
    }
    sum = 0;
    for (j = 0; j < n; ++j){
        for (i = 0; i < m; ++i)
        {
            sum = sum + array[i][j];
        }
        printf("Sum of the %d column is = %d\n", j, sum);
        sum = 0;
    }
}
```

Output:

```
PS D:\UPES\2nd Year\Semester 3\DAA\Lab
Enter the order of the matrix
2 2
Enter the co-efficients of the matrix
1 2
3 4
Sum of the 0 row is = 3
Sum of the 1 row is = 7
Sum of the 0 column is = 4
Sum of the 1 column is = 6
```

5. Find the product of two matrices using pointers.

Input:

```c
#include <stdio.h>
#define ROW 3
#define COL 3
void matrixInput(int mat[][COL]);
void matrixPrint(int mat[][COL]);
void matrixMultiply(int mat1[][COL], int mat2[][COL], int res[][COL]);
int main(){
    int mat1[ROW][COL];
    int mat2[ROW][COL];
    int product[ROW][COL];
    printf("Enter elements in first matrix of size %dx%d\n", ROW, COL);
    matrixInput(mat1);
    printf("Enter elements in second matrix of size %dx%d\n", ROW, COL);
    matrixInput(mat2);
    matrixMultiply(mat1, mat2, product);
    printf("Product of both matrices is : \n");
    matrixPrint(product);
    return 0;
}
void matrixInput(int mat[][COL]){
    int row, col;

    for (row = 0; row < ROW; row++){
        for (col = 0; col < COL; col++){
            scanf("%d", (*(mat + row) + col));
        }
    }
}
```

```c
void matrixPrint(int mat[][COL]){
    int row, col;

    for (row = 0; row < ROW; row++){
        for (col = 0; col < COL; col++){
            printf("%d ", *(*(mat + row) + col));
        }
        printf("\n");
    }
}
void matrixMultiply(int mat1[][COL], int mat2[][COL], int res[][COL]){
    int row, col, i;
    int sum;
    for (row = 0; row < ROW; row++){
        for (col = 0; col < COL; col++){
            sum = 0;

            for (i = 0; i < COL; i++){
                sum += (*(*(mat1 + row) + i)) * (*(*(mat2 + i) + col));
            }
            *(*(res + row) + col) = sum;
        }
    }
}
```

Output:

```
PS D:\UPES\2nd Year\Semester 3\DAA\Lab> cd "
Enter elements in first matrix of size 3x3
1 2 3
4 5 6
7 8 9
Enter elements in second matrix of size 3x3
1 2 3
4 5 6
7 8 9
Product of both matrices is :
30 36 42
66 81 96
102 126 150
```

6. Store ‘n’ numbers (integers or real) in an array. Conduct a linear search for a given number and report success or failure in the form of a suitable message.

Input:

```c
#include <stdio.h>
int main() {
    int n, search, i;
    int found = 0;
    printf("Enter number of elements: ");
    scanf("%d", &n);
    int arr[n];
    printf("Enter %d numbers:\n", n);
    for(i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }
    printf("Enter number to search: ");
    scanf("%d", &search);
    for(i = 0; i < n; i++) {
        if(arr[i] == search) {
            printf("Number %d found at position %d.\n", search, i + 1);
            found = 1;
            break;
        }
    }
    if (!found) {
        printf("Number %d not found in the array.\n", search);
    }
    return 0;
}
```

Output:

```
PS D:\UPES\2nd Year\Semester 3\
Enter number of elements: 5
Enter 5 numbers:
1 2 3 4 5
Enter number to search: 2
Number 2 found at position 2.
```

7. Write a program to reverse an array.

Input:

```c
#include<stdio.h>
int main(){
    int i, n, temp;
    printf("Enter the size of the array: ");
    scanf("%d", &n);
    int arr[n];
    printf("Enter the elements: ");
    for(i = 0; i < n; i++){
        scanf("%d", &arr[i]);
    }
    int end = n - 1;
    for(i = 0; i < n/2; i++){
        temp = arr[i];
        arr[i] = arr[end];
        arr[end] = temp;
        end--;
    }
    printf("The reversed array: ");
    for(i = 0; i < n; i++){
        printf("%d ", arr[i]);
    }
}
```

Output:

```
PS D:\UPES\2nd Year\Semester 3\D
Enter the size of the array: 5
Enter the elements: 1 2 3 4 5
The reversed array: 5 4 3 2 1
```

8. Find the largest three distinct elements in an array: Input: arr[] = {10, 4, 3, 50, 23, 90} Output: 90, 50, 23

Input:

```c
#include <stdio.h>
int print3largest(int arr[], int arr_size){
    int i, first, second, third;
    first = second = third = arr[0];
    for (i = 1; i < arr_size; i++){
        if (arr[i] > first){
            third = second;
            second = first;
            first = arr[i];
        }
        else if (arr[i] > second && arr[i] != first){
            third = second;
            second = arr[i];
        }
        else if (arr[i] > third && arr[i] != second && arr[i] != first){
            third = arr[i];
        }
    }
    printf("Three largest elements are %d, %d, %d\n", first, second, third);
}
int main(){
    int arr[] = {10, 4, 3, 50, 23, 90};
    int n = sizeof(arr) / sizeof(arr[0]);
    print3largest(arr, n);
    return 0;
}
```

Output:

```
PS D:\UPES\2nd Year\Semester 3\DAA\Lab
Three largest elements are 90, 50, 23
PS D:\UPES\2nd Year\Semester 3\DAA\Lab
```

9. Move all zeroes to end of array

Input:

```c
#include <stdio.h>
int move_zeros(int A[], int n){
    int B[n];
    int j = 0;
    int count = 0;
    for (int i = 0; i < n; i++){
        if (A[i] != 0){
            B[j] = A[i];
            j++;
        }
        else {
            count++;
        }
    }
    while (count > 0){
        B[j] = 0;
        count--;
        j++;
    }
    for (int i = 0; i < n; i++){
        A[i] = B[i];
    }
    printf("array after shifting zeros to right side:\n");
    for (int i = 0; i < n; i++) {
        printf("%d ", A[i]);
    }
}
int main(){
    int A[] = {4,0,9,0,0,5,6,2,0};
    int n = sizeof(A) / sizeof(A[0]);
    move_zeros(A, n);
    return 0;
}
```

Output:

```
PS D:\UPES\2nd Year\Semester 3\DAA\Lab> cd
array after shifting zeros to right side:
4 9 5 6 2 0 0 0 0
```

10. Rearrange an array in maximum minimum form using Two Pointer Technique. Input: arr[] = {1, 2, 3, 4, 5, 6, 7} Output: arr[] = {7, 1, 6, 2, 5, 3, 4}.

Input:

```c
#include <stdio.h>
int rearrange(int arr[], int n){
    int result[n];
    int i = 0;
    int j = n - 1;
    int k = 0;
    while (i <= j){
        if (k % 2 == 0) {
            result[k] = arr[j];
            j--;
        } else {
            result[k] = arr[i];
            i++;
        }
        k++;
    }
    for (i = 0; i < n; i++){
        arr[i] = result[i];
    }
}
int main(){
    int arr[] = {1, 2, 3, 4, 5, 6, 7};
    int n = sizeof(arr) / sizeof(arr[0]);
    rearrange(arr, n);
    printf("Rearranged array: ");
    for (int i = 0; i < n; i++){
        printf("%d ", arr[i]);
    }
    return 0;
}
```

Output:

```
PS D:\UPES\2nd Year\Semester 3\D
Rearranged array: 7 1 6 2 5 3 4
```

11. Print all Distinct (Unique) Elements in given Array: Input: arr[] = {12, 10, 9, 45, 2, 10, 10, 45} Output: 12, 10, 9, 2

Input:

```c
#include <stdio.h>
void printDistinct(int arr[], int n) {
    for (int i = 0; i < n; i++) {
        int j;
        for (j = 0; j < i; j++) {
            if (arr[i] == arr[j]) {
                break;
            }
        }
        if (i == j) {
            printf("%d ", arr[i]);
        }
    }
}
int main() {
    int arr[] = {12, 10, 9, 45, 2, 10, 10, 45};
    int n = sizeof(arr) / sizeof(arr[0]);
    printf("Distinct elements are: ");
    printDistinct(arr, n);
    return 0;
}
```

Output:

```
PS D:\UPES\2nd Year\Semester 3\DAA\La
Distinct elements are: 12 10 9 45 2
```

12. Write a program to count the total number of nonzero elements in a two-dimensional array.

Input:

```c
#include <stdio.h>
int countNonZeroElements(int arr[][3], int rows, int cols){
    int count = 0;
    for (int i = 0; i < rows; i++){
        for (int j = 0; j < cols; j++){
            if (arr[i][j] != 0){
                count++;
            }
        }
    }
    return count;
}
int main() {
    int arr[3][3] = { {1, 0, 3}, {0, 5, 6}, {0, 0, 9}};
    int rows = 3;
    int cols = 3;
    int result = countNonZeroElements(arr, rows, cols);
    printf("Total number of non-zero elements: %d\n", result);
    return 0;
}
```

Output:

```
PS D:\UPES\2nd Year\Semester 3\DAA\La
Total number of non-zero elements: 5
```

13. Write a program using pointers to interchange the second biggest and the second smallest number in the array.

Input:

```c
#include <stdio.h>
#include <limits.h>
void interchangeSecondBiggestSmallest(int *arr, int size) {
    if (size < 4) {
        printf("Array should have at least 4 elements.\n");
        return;
    }
    int firstSmallest, secondSmallest;
    int firstBiggest, secondBiggest;
    firstSmallest = secondSmallest = INT_MAX;
    firstBiggest = secondBiggest = INT_MIN;
    for (int i = 0; i < size; i++){
        if (arr[i] < firstSmallest){
            secondSmallest = firstSmallest;
            firstSmallest = arr[i];
        } else if (arr[i] < secondSmallest && arr[i] != firstSmallest){
            secondSmallest = arr[i];
        }

        if (arr[i] > firstBiggest){
            secondBiggest = firstBiggest;
            firstBiggest = arr[i];
        } else if (arr[i] > secondBiggest && arr[i] != firstBiggest){
            secondBiggest = arr[i];
        }
    }
    if (secondSmallest == INT_MAX || secondBiggest == INT_MIN){
        printf("Second smallest or second biggest not found.\n");
        return;
    }
    for (int i = 0; i < size; i++){
        if (arr[i] == secondSmallest){
            arr[i] = secondBiggest;
        } else if (arr[i] == secondBiggest){
            arr[i] = secondSmallest;
        }
    }
}
int main(){
    int arr[] = {5, 1, 9, 3, 7, 6, 2, 8};
    int size = sizeof(arr) / sizeof(arr[0]);
    printf("Original array: ");
    for (int i = 0; i < size; i++){
        printf("%d ", arr[i]);
    }
    printf("\n");
    interchangeSecondBiggestSmallest(arr, size);
    printf("Array after interchanging second biggest and second smallest: ");
    for (int i = 0; i < size; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
    return 0;
}
```

Output:

```
PS D:\UPES\2nd Year\Semester 3\DAA\Lab> cd "d:\UPES\2nd Year\Semester 3\DAA\Lab
Original array: 5 1 9 3 7 6 2 8
Array after interchanging second biggest and second smallest: 5 1 9 3 7 6 8 2
```

Experiment 3: Problems in Binary Search

1. Find the number of rotations in a circularly sorted array

Input:

```c
#include <stdio.h>
int findRotations(int arr[], int n){
    int low = 0, high = n - 1;
    while (low <= high){
        if (arr[low] <= arr[high]){
            return low;
        }
        int mid = low + (high - low) / 2;
        int next = (mid + 1) % n;
        int prev = (mid - 1 + n) % n;
        if (arr[mid] <= arr[next] && arr[mid] <= arr[prev]){
            return mid;
        }
        if (arr[mid] >= arr[low]){
            low = mid + 1;
        } else{
            high = mid - 1;
        }
    }
    return -1;
}
int main(){
    int arr[] = {15, 18, 2, 3, 6, 12};
    int n = sizeof(arr) / sizeof(arr[0]);
    int rotations = findRotations(arr, n);
    printf("Number of rotations: %d\n", rotations);
    return 0;
}
```

Output:

```
PS D:\UPES\2nd Year\Sem
Number of rotations: 2
```

2. Search an element in a circularly sorted array

Input:

```c
#include <stdio.h>
int searchInRotatedArray(int arr[], int n, int key){
    int low = 0, high = n - 1;
    while (low <= high){
        int mid = low + (high - low) / 2;
        if (arr[mid] == key){
            return mid;
        }
        if (arr[low] <= arr[mid]){
            if (key >= arr[low] && key < arr[mid]){
                high = mid - 1;
            } else{
                low = mid + 1;
            }
        } else{
            if (key > arr[mid] && key <= arr[high]){
                low = mid + 1;
            } else{
                high = mid - 1;
            }
        }
    }
    return -1;
}
int main(){
    int arr[] = {15, 18, 2, 3, 6, 12};
    int n = sizeof(arr) / sizeof(arr[0]);
    int key = 6;
    int result = searchInRotatedArray(arr, n, key);
    if (result != -1){
        printf("Element %d found at index %d\n", key, result);
    } else{
        printf("Element %d not found in the array\n", key);
    }
    return 0;
}
```

Output:

```
PS D:\UPES\2nd Year\Semeste
Element 6 found at index 4
```

3. Find the first or last occurrence of a given number in a sorted array

Input:

```c
#include <stdio.h>
int findFirstOccurrence(int arr[], int n, int key){
    int low = 0, high = n - 1;
    int result = -1;
    while (low <= high){
        int mid = low + (high - low) / 2;
        if (arr[mid] == key){
            result = mid;
            high = mid - 1;
        } else if (arr[mid] < key){
            low = mid + 1;
        } else{
            high = mid - 1;
        }
    }
    return result;
}
int findLastOccurrence(int arr[], int n, int key) {
    int low = 0, high = n - 1;
    int result = -1;
    while (low <= high) {
        int mid = low + (high - low) / 2;
        if (arr[mid] == key) {
            result = mid;
            low = mid + 1;
        } else if (arr[mid] < key) {
            low = mid + 1;
        } else {
            high = mid - 1;
        }
    }
    return result;
}
```

```c
int main() {
    int arr[] = {1, 2, 2, 2, 3, 4, 5};
    int n = sizeof(arr) / sizeof(arr[0]);
    int key = 2;
    int first = findFirstOccurrence(arr, n, key);
    int last = findLastOccurrence(arr, n, key);
    if (first != -1){
        printf("First occurrence of %d is at index %d\n", key, first);
    } else{
        printf("%d is not present in the array\n", key);
    }
    if (last != -1){
        printf("Last occurrence of %d is at index %d\n", key, last);
    } else{
        printf("%d is not present in the array\n", key);
    }
    return 0;
}
```

Output:

```
PS D:\UPES\2nd Year\Semester 3\DAA\La
First occurrence of 2 is at index 1
Last occurrence of 2 is at index 3
PS D:\UPES\2nd Year\Semester 3\DAA\La
```

4. Count occurrences of a number in a sorted array with duplicates

Input:

```c
#include <stdio.h>
int findFirstOccurrence(int arr[], int n, int key){
    int low = 0, high = n - 1, result = -1;
    while (low <= high){
        int mid = low + (high - low) / 2;
        if (arr[mid] == key){
            result = mid;
            high = mid - 1;
        } else if (arr[mid] < key){
            low = mid + 1;
        } else{
            high = mid - 1;
        }
    }
    return result;
}
int findLastOccurrence(int arr[], int n, int key){
    int low = 0, high = n - 1, result = -1;
    while (low <= high){
        int mid = low + (high - low) / 2;
        if (arr[mid] == key){
            result = mid;
            low = mid + 1;
        } else if (arr[mid] < key){
            low = mid + 1;
        } else{
            high = mid - 1;
        }
    }
    return result;
}
```

```c
int countOccurrences(int arr[], int n, int key){
    int first = findFirstOccurrence(arr, n, key);
    int last = findLastOccurrence(arr, n, key);
    if (first == -1 || last == -1){
        return 0;
    }
    return last - first + 1;
}
int main() {
    int arr[] = {1, 2, 2, 2, 3, 4, 5};
    int n = sizeof(arr) / sizeof(arr[0]);
    int key = 2;
    int count = countOccurrences(arr, n, key);
    if (count > 0){
        printf("Element %d occurs %d times in the array\n", key, count);
    } else{
        printf("Element %d is not present in the array\n", key);
    }
    return 0;
}
```

Output:

```
PS D:\UPES\2nd Year\Semester 3\DAA\Lab
Element 2 occurs 3 times in the array
```

5. Find the smallest missing element from a sorted array

Input:

```c
#include <stdio.h>
int findSmallestMissing(int arr[], int n){
    int low = 0, high = n - 1;
    while (low <= high) {
        int mid = low + (high - low) / 2;
        if (arr[mid] != mid) {
            high = mid - 1;
        } else {
            low = mid + 1;
        }
    }
    return low;
}
int main(){
    int arr[] = {0, 1, 2, 3, 5, 6, 7};
    int n = sizeof(arr) / sizeof(arr[0]);
    int missing = findSmallestMissing(arr, n);
    printf("The smallest missing element is: %d\n", missing);
    return 0;
}
```

Output:

```
PS D:\UPES\2nd Year\Semester 3\DAA\L
The smallest missing element is: 4
```

6. Find floor and ceil of a number in a sorted integer array

Input:

```c
#include <stdio.h>
int findFloor(int arr[], int n, int key){
    int low = 0, high = n - 1;
    int floor = -1;
    while (low <= high){
        int mid = low + (high - low) / 2;
        if (arr[mid] == key){
            return arr[mid];
        } else if (arr[mid] < key){
            floor = arr[mid];
            low = mid + 1;
        } else{
            high = mid - 1;
        }
    }
    return floor;
}
int findCeil(int arr[], int n, int key){
    int low = 0, high = n - 1;
    int ceil = -1;
    while (low <= high){
        int mid = low + (high - low) / 2;
        if (arr[mid] == key){
            return arr[mid];
        } else if (arr[mid] < key){
            low = mid + 1;
        } else{
            ceil = arr[mid];
            high = mid - 1;
        }
    }
    return ceil;
}
```

```c
int main(){
    int arr[] = {1, 2, 4, 6, 10, 12, 14};
    int n = sizeof(arr) / sizeof(arr[0]);
    int key = 5;
    int floor = findFloor(arr, n, key);
    int ceil = findCeil(arr, n, key);
    if (floor != -1){
        printf("Floor of %d is %d\n", key, floor);
    } else{
        printf("Floor of %d does not exist\n", key);
    }
    if (ceil != -1){
        printf("Ceil of %d is %d\n", key, ceil);
    } else{
        printf("Ceil of %d does not exist\n", key);
    }
    return 0;
}
```

Output:

```
PS D:\UPES\2nd Y
Floor of 5 is 4
Ceil of 5 is 6
```

7. Search in a nearly sorted array in logarithmic time

Input:

```c
#include <stdio.h>
int searchInNearlySortedArray(int arr[], int n, int key) {
    int low = 0, high = n - 1;
    while (low <= high){
        int mid = low + (high - low) / 2;
        if (arr[mid] == key){
            return mid;
        } else if (mid > low && arr[mid - 1] == key){
            return mid - 1;
        } else if (mid < high && arr[mid + 1] == key){
            return mid + 1;
        }
        if (key < arr[mid]){
            high = mid - 2;
        } else{
            low = mid + 2;
        }
    }
    return -1;
}
int main(){
    int arr[] = {10, 3, 40, 20, 50, 80, 70};
    int n = sizeof(arr) / sizeof(arr[0]);
    int key = 40;
    int result = searchInNearlySortedArray(arr, n, key);
    if (result != -1){
        printf("Element %d found at index %d\n", key, result);
    } else{
        printf("Element %d not found in the array\n", key);
    }
    return 0;
}
```

Output:

```
PS D:\UPES\2nd Year\Semester
Element 40 found at index 2
```

8. Find the number of 1's in a sorted binary array

Input:

```c
#include <stdio.h>
int findFirstOne(int arr[], int n){
    int low = 0, high = n - 1;
    int result = -1;
    while (low <= high){
        int mid = low + (high - low) / 2;
        if (arr[mid] == 1){
            result = mid;
            high = mid - 1;
        } else{
            low = mid + 1;
        }
    }
    return result;
}
int countOnes(int arr[], int n){
    int firstOneIndex = findFirstOne(arr, n);
    if (firstOneIndex == -1){
        return 0;
    }
    return n - firstOneIndex;
}
int main(){
    int arr[] = {0, 0, 0, 1, 1, 1, 1};
    int n = sizeof(arr) / sizeof(arr[0]);
    int count = countOnes(arr, n);
    printf("The number of 1's in the array is: %d\n", count);
    return 0;
}
```

Output:

```
PS D:\UPES\2nd Year\Semester 3\DAA\La
The number of 1's in the array is: 4
```

9. Find the peak element in an array

Input:

```c
#include <stdio.h>
int findPeak(int arr[], int n){
    int low = 0, high = n - 1;
    while (low <= high){
        int mid = low + (high - low) / 2;
        if ((mid == 0 || arr[mid] >= arr[mid - 1]) && (mid == n - 1 || arr[mid] >= arr[mid + 1])){
            return mid;
        }
        if (mid > 0 && arr[mid] < arr[mid - 1]){
            high = mid - 1;
        }
        else{
            low = mid + 1;
        }
    }
    return -1;
}
int main(){
    int arr[] = {1, 3, 20, 4, 1};
    int n = sizeof(arr) / sizeof(arr[0]);
    int peakIndex = findPeak(arr, n);
    if (peakIndex != -1){
        printf("Peak element is %d at index %d\n", arr[peakIndex], peakIndex);
    } else{
        printf("No peak element found\n");
    }
    return 0;
}
```

Output:

```
PS D:\UPES\2nd Year\Semester 3
Peak element is 20 at index 2
```

10. Find the missing term in a sequence in logarithmic time

Input:

```c
#include <stdio.h>
int find_missing(int arr[], int n){
    int low = 0, high = n - 1;
    while (low <= high){
        int mid = (low + high) / 2;
        if (arr[mid] == mid + 1){
            low = mid + 1;
        } else{
            high = mid - 1;
        }
    }
    return low + 1;
}
int main(){
    int arr[] = {1, 2, 3, 5};
    int n = sizeof(arr) / sizeof(arr[0]);
    printf("The missing number is: %d\n", find_missing(arr, n));
    return 0;
}
```

Output:

```
PS D:\UPES\2nd Year\Semes
The missing number is: 4
```

11. Find floor and ceil of a number in a sorted array (Recursive solution)

Input:

```c
#include <stdio.h>
int findFloor(int arr[], int low, int high, int x){
    if (high < low){
        return -1;
    }
    int mid = low + (high - low) / 2;
    if (arr[mid] == x){
        return arr[mid];
    }
    if (arr[mid] < x){
        int floor = findFloor(arr, mid + 1, high, x);
        return (floor != -1) ? floor : arr[mid];
    }
    return findFloor(arr, low, mid - 1, x);
}
int findCeil(int arr[], int low, int high, int x){
    if (high < low){
        return -1;
    }
    int mid = low + (high - low) / 2;
    if (arr[mid] == x){
        return arr[mid];
    }
    if (arr[mid] > x){
        int ceil = findCeil(arr, low, mid - 1, x);
        return (ceil != -1) ? ceil : arr[mid];
    }
    return findCeil(arr, mid + 1, high, x);
}
```

```c
void findFloorAndCeil(int arr[], int n, int x){
    int floor = findFloor(arr, 0, n - 1, x);
    int ceil = findCeil(arr, 0, n - 1, x);
    if (floor != -1){
        printf("Floor of %d is %d\n", x, floor);
    } else{
        printf("No floor exists for %d\n", x);
    }
    if (ceil != -1){
        printf("Ceil of %d is %d\n", x, ceil);
    } else{
        printf("No ceil exists for %d\n", x);
    }
}
int main(){
    int arr[] = {1, 2, 8, 10, 10, 12, 19};
    int n = sizeof(arr) / sizeof(arr[0]);
    int x = 5;
    findFloorAndCeil(arr, n, x);
    return 0;
}
```

Output:

```
PS D:\UPES\2nd Y
Floor of 5 is 2
Ceil of 5 is 8
```

12. Find the frequency of each element in a sorted array containing duplicates

Input:

```c
#include <stdio.h>
void findFrequencies(int arr[], int n){
    int count = 1;
    for (int i = 1; i < n; i++){
        if (arr[i] == arr[i - 1]){
            count++;
        } else{
            printf("Element %d appears %d times\n", arr[i - 1], count);
            count = 1;
        }
    }
    printf("Element %d appears %d times\n", arr[n - 1], count);
}
int main(){
    int arr[] = {1, 1, 2, 2, 2, 3, 3, 4, 5, 5};
    int n = sizeof(arr) / sizeof(arr[0]);
    findFrequencies(arr, n);
    return 0;
}
```

Output:

```
PS D:\UPES\2nd Year\Semest
Element 1 appears 2 times
Element 2 appears 3 times
Element 3 appears 2 times
Element 4 appears 1 times
Element 5 appears 2 times
```

13. Find the square root of a number using a binary search

Input:

```c
#include <stdio.h>
#include <math.h>
double squareRoot(double N){
    if (N == 0) return 0;
    double low = 0, high = N;
    double mid;
    double precision = 1e-6;
    while ((high - low) > precision){
        mid = low + (high - low) / 2;
        if (mid * mid == N){
            return mid;
        }
        if (mid * mid < N){
            low = mid;
        } else{
            high = mid;
        }
    }
    return mid;
}
int main(){
    double N = 16.0;
    double result = squareRoot(N);
    printf("Square root of %.6f is approximately %.6f\n", N, result);
    return 0;
}
```

Output:

```
PS D:\UPES\2nd Year\Semester 3\DAA\Lab> cd "d:\UPES\
Square root of 16.000000 is approximately 4.000000
```

14. Division of two numbers using binary search algorithm

Input:

```c
#include <stdio.h>
#include <math.h>
int divide(int A, int B) {
    if (B == 0){
        printf("Division by zero is not allowed.\n");
        return -1;
    }
    int negativeResult = 0;
    if ((A < 0 && B > 0) || (A > 0 && B < 0)){
        negativeResult = 1;
    }
    A = abs(A);
    B = abs(B);
    int low = 0, high = A;
    int result = 0;
    while (low <= high){
        int mid = low + (high - low) / 2;
        if (B * mid == A){
            result = mid;
            break;
        }
        else if (B * mid < A){
            result = mid;
            low = mid + 1;
        }
        else{
            high = mid - 1;
        }
    }
    return (negativeResult) ? -result : result;
}
int main(){
    int A = 10, B = 3;
    int quotient = divide(A, B);
    if (quotient != -1){
        printf("The quotient of %d / %d is: %d\n", A, B, quotient);
    }
    return 0;
}
```

Output:

```
PS D:\UPES\2nd Year\Semester 3\DAA\Lab> cd "d:\UPES\2nd Year\Semester 3\DAA\Lab\Experiment 3
14.c: In function 'divide':
14.c:12:9: warning: implicit declaration of function 'abs' [-Wimplicit-function-declaration]
     A = abs(A);
         ^~~
The quotient of 10 / 3 is: 3
```

15. Find the odd occurring element in an array in logarithmic time

Input:

```c
#include <stdio.h>
int findOddOccurringElement(int arr[], int low, int high){
    if (low == high){
        return arr[low];
    }
    int mid = low + (high - low) / 2;
    if (mid % 2 == 1){
        mid--;
    }
    if (arr[mid] == arr[mid + 1]){
        return findOddOccurringElement(arr, mid + 2, high);
    } else{
        return findOddOccurringElement(arr, low, mid);
    }
}
int main(){
    int arr[] = {1, 1, 2, 2, 3, 3, 5, 5, 7};
    int n = sizeof(arr) / sizeof(arr[0]);
    int result = findOddOccurringElement(arr, 0, n - 1);
    printf("The odd occurring element is: %d\n", result);
    return 0;
}
```

Output:

```
PS D:\UPES\2nd Year\Semester 3\DAA
The odd occurring element is: 7
```

16. Find pairs with difference k in an array | Constant Space Solution

Input:

```c
#include <stdio.h>
#include <stdlib.h>
int compare(const void *a, const void *b){
    return (*(int*)a - *(int*)b);
}
void findPairsWithDifferenceK(int arr[], int n, int k){
    qsort(arr, n, sizeof(int), compare);
    int i = 0, j = 1;
    while (j < n){
        if (arr[j] - arr[i] == k){
            printf("Pair found: (%d, %d)\n", arr[i], arr[j]);
            i++;
            j++;
        }
        else if (arr[j] - arr[i] < k){
            j++;
        }
        else{
            i++;
        }
        if (i == j){
            j++;
        }
    }
}
int main(){
    int arr[] = {5, 20, 3, 2, 50, 80, 60};
    int n = sizeof(arr) / sizeof(arr[0]);
    int k = 20;
    findPairsWithDifferenceK(arr, n, k);
    return 0;
}
```

Output:

```
PS D:\UPES\2nd Year\Sem
Pair found: (60, 80)
```

17. Find k closest elements to a given value in an array

Input:

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
int compare(const void *a, const void *b){
    return (*(int*)a - *(int*)b);
}
void findKClosestElements(int arr[], int n, int target, int k){
    qsort(arr, n, sizeof(int), compare);
    int low = 0, high = n - 1;
    int mid = -1;
    while (low <= high){
        mid = low + (high - low) / 2;
        if (arr[mid] == target){
            break;
        } else if (arr[mid] < target){
            low = mid + 1;
        } else{
            high = mid - 1;
        }
    }
    int left = mid - 1;
    int right = mid + 1;
    printf("The %d closest elements to %d are: ", k, target);
    printf("%d ", arr[mid]);
    k--;
    while (k > 0){
        if (left >= 0 && right < n){
            if (abs(arr[left] - target) <= abs(arr[right] - target)){
                printf("%d ", arr[left]);
                left--;
            } else{
                printf("%d ", arr[right]);
                right++;
            }
        } else if (left >= 0){
            printf("%d ", arr[left]);
            left--;
        } else if (right < n){
            printf("%d ", arr[right]);
            right++;
        }
        k--;
    }
    printf("\n");
}
```

```c
int main(){
    int arr[] = {1, 5, 9, 8, 10, 3, 7, 6};
    int n = sizeof(arr) / sizeof(arr[0]);
    int target = 7;
    int k = 3;
    findKClosestElements(arr, n, target, k);
    return 0;
}
```

Output:

```
PS D:\UPES\2nd Year\Semester 3\DAA\Lab>
The 3 closest elements to 7 are: 7 6 8
```

Experiment 4: Patterns

1. Implement the activity selection problem to get a clear understanding of greedy approach.

Input:

```c
#include <stdio.h>
#include <stdlib.h>
typedef struct{
    int start;
    int finish;
} Activity;
int compare(const void *a, const void *b){
    Activity *activityA = (Activity *)a;
    Activity *activityB = (Activity *)b;
    return activityA->finish - activityB->finish;
}
void activitySelection(Activity activities[], int n){
    qsort(activities, n, sizeof(Activity), compare);
    printf("Selected activities:\n");
    printf("Start: %d, Finish: %d\n", activities[0].start, activities[0].finish);
    int lastFinishTime = activities[0].finish;
    for (int i = 1; i < n; i++){
        if (activities[i].start >= lastFinishTime) {
            printf("Start: %d, Finish: %d\n", activities[i].start, activities[i].finish);
            lastFinishTime = activities[i].finish;
        }
    }
}
int main(){
    Activity activities[] = {
        {1, 4}, {3, 5}, {0, 6}, {5, 7}, {8, 9}, {5, 9}
    };
    int n = sizeof(activities) / sizeof(activities[0]);
    activitySelection(activities, n);
    return 0;
}
```

Output:

```
PS D:\UPES\2nd Year\S
Selected activities:
Start: 1, Finish: 4
Start: 5, Finish: 7
Start: 8, Finish: 9
```

2. Get a detailed insight of dynamic programming approach by the implementation of Matrix Chain Multiplication problem and see the impact of parenthesis positioning on time requirements for matrix multiplication.

Input:

```c
#include <stdio.h>
#include <limits.h>
void matrixChainOrder(int p[], int n){
    int m[n][n];
    for (int i = 1; i < n; i++){
        m[i][i] = 0;
    }
    for (int l = 2; l < n; l++){
        for (int i = 1; i < n - l + 1; i++){
            int j = i + l - 1;
            m[i][j] = INT_MAX;

            for (int k = i; k < j; k++){
                int q = m[i][k] + m[k+1][j] + p[i-1] * p[k] * p[j];
                if (q < m[i][j]) {
                    m[i][j] = q;
                }
            }
        }
    }
    printf("Minimum number of multiplications is: %d\n", m[1][n-1]);
}
int main(){
    int p[] = {10, 20, 30, 40, 30};
    int n = sizeof(p) / sizeof(p[0]);
    matrixChainOrder(p, n);
    return 0;
}
```

Output:

```
PS D:\UPES\2nd Year\Semester 3\DAA\Lab> cd "
Minimum number of multiplications is: 30000
```

3. Compare the performance of Dijkstra and Bellman ford algorithm for the single source shortest path problem.

Comparison of Dijkstra and Bellman-Ford Algorithms:

Aspect	Dijkstra	Bellman-Ford
Graph Type	Works only for graphs with non-negative weights.	Works for graphs with negative weights.
Time Complexity	O(V+ElogV) with a priority queue.	O(V·E)
Optimality	Always finds the shortest path if conditions are met.	Handles graphs with negative weight edges correctly.
Ease of Implementation	Slightly more complex due to priority queue.	Simpler to implement iteratively.
Use Case	Faster for dense graphs with positive weights.	Essential when negative weights are present.
Edge Relaxation	Relaxes each edge once during iteration.	Relaxes edges V−1 times.

4. Through 0/1 Knapsack problem, analyze the greedy and dynamic programming approach for the same dataset.

Greedy Approach

Input:

```c
#include <stdio.h>
#include <stdlib.h>
typedef struct {
    int value;
    int weight;
    float ratio;
} Item;
int compare(const void* a, const void* b) {
    Item* item1 = (Item*)a;
    Item* item2 = (Item*)b;
    if (item1->ratio < item2->ratio) return 1;
    if (item1->ratio > item2->ratio) return -1;
    return 0;
}
float knapsackGreedy(Item items[], int n, int capacity) {
    qsort(items, n, sizeof(Item), compare);
    int totalWeight = 0;
    float totalValue = 0.0;
    for (int i = 0; i < n; i++) {
        if (totalWeight + items[i].weight <= capacity) {
            totalWeight += items[i].weight;
            totalValue += items[i].value;
        } else {
            int remainingWeight = capacity - totalWeight;
            totalValue += items[i].value * ((float)remainingWeight / items[i].weight);
            break;
        }
    }
    return totalValue;
}
```

Output:

```
PS D:\UPES\2nd Year\Semester 3\DAA\Lab>
Greedy approach: Maximum value = 240.00
```

Dynamic Programming

Input:

```c
#include <stdio.h>
int knapsackDP(int values[], int weights[], int n, int capacity){
    int dp[n + 1][capacity + 1];
    for (int i = 0; i <= n; i++){
        for (int w = 0; w <= capacity; w++){
            if (i == 0 || w == 0){
                dp[i][w] = 0;
            } else if (weights[i - 1] <= w){
                dp[i][w] = (values[i - 1] + dp[i - 1][w - weights[i - 1]] > dp[i - 1][w]) ?
                           (values[i - 1] + dp[i - 1][w - weights[i - 1]]) : dp[i - 1][w];
            } else {
                dp[i][w] = dp[i - 1][w];
            }
        }
    }
    return dp[n][capacity];
}
int main(){
    int values[] = {60, 100, 120};
    int weights[] = {10, 20, 30};
    int n = sizeof(values) / sizeof(values[0]);
    int capacity = 5
    int result = knapsackDP(values, weights, n, capacity);
    printf("Dynamic Programming approach: Maximum value = %d\n", result);
    return 0;
}
```

Output:

```
PS D:\UPES\2nd Year\Semester 3\DAA\Lab> cd "d:\UPES
Dynamic Programming approach: Maximum value = 220
```

5. Implement the sum of subset and N Queen problem.

Sum of Subset

Input:

```c
#include <stdio.h>
#include <stdbool.h>
void printSubset(int subset[], int size){
    for (int i = 0; i < size; i++){
        printf("%d ", subset[i]);
    }
    printf("\n");
}
void findSubset(int set[], int n, int target, int subset[], int index, int subsetSize){
    if (target == 0){
        printSubset(subset, subsetSize);
        return;
    }
    if (n == 0 || target < 0){
        return;
    }
    subset[subsetSize] = set[index];
    findSubset(set, n - 1, target - set[index], subset, index + 1, subsetSize + 1);
    findSubset(set, n - 1, target, subset, index + 1, subsetSize);
}
int main(){
    int set[] = {3, 34, 4, 12, 5, 2};
    int target = 9;
    int n = sizeof(set) / sizeof(set[0]);
    int subset[n];
    printf("Subsets with sum %d are:\n", target);
    findSubset(set, n, target, subset, 0, 0);
    return 0;
}
```

Output:

```
PS D:\UPES\2nd Year\Sem
Subsets with sum 9 are:
3 4 2
4 5
```

N Queen problem

Input:

```c
#include <stdio.h>
#include <stdbool.h>
#define N 4
void printSolution(int board[N][N]){
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++){
            printf("%d ", board[i][j]);
        }
        printf("\n");
    }
}
bool isSafe(int board[N][N], int row, int col){
    for (int i = 0; i < row; i++){
        if (board[i][col] == 1){
            return false;
        }
    }
    for (int i = row, j = col; i >= 0 && j >= 0; i--, j--){
        if (board[i][j] == 1){
            return false;
        }
    }
    for (int i = row, j = col; i >= 0 && j < N; i--, j++){
        if (board[i][j] == 1){
            return false;
        }
    }
    return true;
}
```

```c
bool solveNQueens(int board[N][N], int row){
    if (row >= N){
        return true;
    }
    for (int col = 0; col < N; col++){
        if (isSafe(board, row, col)){
            board[row][col] = 1;
            if (solveNQueens(board, row + 1)){
                return true;
            }
            board[row][col] = 0;
        }
    }
    return false;
}
int main() {
    int board[N][N] = {0};
    if (solveNQueens(board, 0)){
        printSolution(board);
    } else{
        printf("Solution does not exist\n");
    }
    return 0;
}
```

Output:

```
PS D:\UPE
0 1 0 0
0 0 0 1
1 0 0 0
0 0 1 0
```

6. Compare the Backtracking and Branch & Bound Approach by the implementation of 0/1 Knapsack problem. Also compare the performance with dynamic programming approach.

Backtracking

Input:

```c
#include <stdio.h>
int max(int a, int b){
    return (a > b) ? a : b;
}
int knapsackBacktracking(int values[], int weights[], int n, int capacity, int currentValue, int currentWeight, int index){
    if (currentWeight > capacity){
        return 0;
    }
    if (index == n){
        return currentValue;
    }
    int includeItem = knapsackBacktracking(values, weights, n, capacity, currentValue + values[index], currentWeight + weights[index], index + 1);
    int excludeItem = knapsackBacktracking(values, weights, n, capacity, currentValue, currentWeight, index + 1);
    return max(includeItem, excludeItem);
}
int main(){
    int values[] = {60, 100, 120};
    int weights[] = {10, 20, 30};
    int n = sizeof(values) / sizeof(values[0]);
    int capacity = 50;
    int result = knapsackBacktracking(values, weights, n, capacity, 0, 0, 0);
    printf("Backtracking approach: Maximum value = %d\n", result);
    return 0;
}
```

Output:

```
PS D:\UPES\2nd Year\Semester 3\DAA\Lab> cd
Backtracking approach: Maximum value = 220
PS D:\UPES\2nd Year\Semester 3\DAA\Lab\Expe
```

Branch & Bound

Input:

```c
#include <stdio.h>
#include <stdlib.h>
typedef struct{
    int weight;
    int value;
} Item;
typedef struct{
    int level;
    int profit;
    int weight;
    float bound;
} Node;
int compare(const void *a, const void *b){
    float r1 = ((Item*)a)->value / (float)((Item*)a)->weight;
    float r2 = ((Item*)b)->value / (float)((Item*)b)->weight;
    if (r1 > r2) return -1;
    if (r1 < r2) return 1;
    return 0;
}
float calculateBound(Node u, int n, int W, Item items[]){
    if (u.weight >= W) return 0;
    float bound = u.profit;
    int j = u.level + 1;
    int totalWeight = u.weight;
    while (j < n && totalWeight + items[j].weight <= W){
        totalWeight += items[j].weight;
        bound += items[j].value;
        j++;
    }
    if (j < n){
        bound += (W - totalWeight) * (items[j].value / (float)items[j].weight);
    }
    return bound;
}
```

```
int knapsack(int n, int W, Item items[]){
    qsort(items, n, sizeof(Item), compare);
    Node queue[100];
    int front = 0, rear = 0;
    Node u = { -1, 0, 0, 0.0 };
    queue[rear++] = u;
    int maxProfit = 0;
    while (front < rear){
        u = queue[front++];
        if (u.level == n - 1) continue;
        Node v = u;
        v.level = u.level + 1;
        v.weight = u.weight + items[v.level].weight;
        v.profit = u.profit + items[v.level].value;
        if (v.weight <= W && v.profit > maxProfit){
            maxProfit = v.profit;
        }
        v.bound = calculateBound(v, n, W, items);
        if (v.bound > maxProfit) {
            queue[rear++] = v;
        }
        v.weight = u.weight;
        v.profit = u.profit;
        v.bound = calculateBound(v, n, W, items);
        if (v.bound > maxProfit){
            queue[rear++] = v;
        }
    }
    return maxProfit;
}
int main() {
    int n = 4;
    int W = 7;
    Item items[] = {
        {1, 1},
        {3, 4},
        {4, 5},
        {5, 7}
    };
    int maxProfit = knapsack(n, W, items);
    printf("Maximum profit: %d\n", maxProfit);
    return 0;
}
```

Output:

```
PS D:\UPES\2nd Year
Maximum profit: 9
```

Dynamic Programming

Input:

```c
#include <stdio.h>
int max(int a, int b){
    return (a > b) ? a : b;
}
int knapsackDP(int values[], int weights[], int n, int capacity){
    int dp[n+1][capacity+1];
    for (int i = 0; i <= n; i++){
        for (int w = 0; w <= capacity; w++){
            if (i == 0 || w == 0){
                dp[i][w] = 0;
            } else if (weights[i-1] <= w){
                dp[i][w] = max(dp[i-1][w], dp[i-1][w - weights[i-1]] + values[i-1]);
            } else{
                dp[i][w] = dp[i-1][w];
            }
        }
    }
    return dp[n][capacity];
}
int main(){
    int values[] = {60, 100, 120};
    int weights[] = {10, 20, 30};
    int n = sizeof(values) / sizeof(values[0]);
    int capacity = 50;
    int result = knapsackDP(values, weights, n, capacity);
    printf("Dynamic Programming approach: Maximum value = %d\n", result);
    return 0;
}
```

Output:

```
PS D:\UPES\2nd Year\Semester 3\DAA\Lab> cd "d:\UPE
Dynamic Programming approach: Maximum value = 220
```