1. Insertion Sort

```
#include <stdio.h>

void insertionSort(int arr[], int n) {

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

        int key = arr[i];

        int j = i - 1;

        while (j >= 0 && arr[j] > key) {

            arr[j + 1] = arr[j];

            j = j - 1;

        }

        arr[j + 1] = key;

    }

}


void printArray(int arr[], int n) {

    for (int i = 0; i < n; i++)

        printf("%d ", arr[i]);

    printf("\n");

}


int main() {

    int arr[] = {12, 11, 13, 5, 6};

    int n = sizeof(arr) / sizeof(arr[0]);


    insertionSort(arr, n);
    printArray(arr, n);


    return 0;
}
```

2. Shell Sort

```
#include <stdio.h>

void shellSort(int arr[], int n) {

    for (int gap = n / 2; gap > 0; gap /= 2) {

        for (int i = gap; i < n; i++) {

            int temp = arr[i];

            int j;

            for (j = i; j >= gap && arr[j - gap] > temp; j -= gap)

                arr[j] = arr[j - gap];

            arr[j] = temp;

        }

    }

}


void printArray(int arr[], int n) {

    for (int i = 0; i < n; i++)

        printf("%d ", arr[i]);

    printf("\n");

}


int main() {
    int arr[] = {12, 34, 54, 2, 3};
    int n = sizeof(arr) / sizeof(arr[0]);

    shellSort(arr, n);
    printArray(arr, n);

    return 0;
}
```

3. Merge Sort

```
#include <stdio.h>
void merge(int arr[], int l, int m, int r) {
    int n1 = m - l + 1;
    int n2 = r - m;
    int L[n1], R[n2];

    for (int i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for (int j = 0; j < n2; j++)
        R[j] = arr[m + 1 + j];

    int i = 0, j = 0, k = l;
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        } else {
            arr[k] = R[j];
            j++;
        }
        k++;
    }

    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }

    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }
}

void mergeSort(int arr[], int l, int r) {
    if (l < r) {
        int m = l + (r - l) / 2;
        mergeSort(arr, l, m);
        mergeSort(arr, m + 1, r);
        merge(arr, l, m, r);
    }
}

void printArray(int arr[], int n) {
    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

int main() {
    int arr[] = {12, 11, 13, 5, 6, 7};
    int n = sizeof(arr) / sizeof(arr[0]);

    mergeSort(arr, 0, n - 1);
    printArray(arr, n);

    return 0;
}
```

4. Radix Sort

```
#include <stdio.h>
#include <stdlib.h>
int getMax(int arr[], int n) {
    int max = arr[0];
    for (int i = 1; i < n; i++)
```

```
        if (arr[i] > max)
            max = arr[i];
    return max;
}

void countSort(int arr[], int n, int exp) {
    int output[n];
    int i, count[10] = {0};

    for (i = 0; i < n; i++)
        count[(arr[i] / exp) % 10]++;

    for (i = 1; i < 10; i++)
        count[i] += count[i - 1];

    for (i = n - 1; i >= 0; i--) {
        output[count[(arr[i] / exp) % 10] - 1] = arr[i];
        count[(arr[i] / exp) % 10]--;
    }

    for (i = 0; i < n; i++)
        arr[i] = output[i];
}

void radixSort(int arr[], int n) {
    int m = getMax(arr, n);

    for (int exp = 1; m / exp > 0; exp *= 10)

        countSort(arr, n, exp);

}


void printArray(int arr[], int n) {

    for (int i = 0; i < n; i++)

        printf("%d ", arr[i]);

    printf("\n");

}


int main() {

    int arr[] = {170, 45, 75, 90, 802, 24, 2, 66};

    int n = sizeof(arr) / sizeof(arr[0]);


    radixSort(arr, n);

    printArray(arr, n);


    return 0;

}
```

5. Heap Sort

```
#include <stdio.h>


void heapify(int arr[], int n, int i) {

    int largest = i;
    int left = 2 * i + 1;

    int right = 2 * i + 2;


    if (left < n && arr[left] > arr[largest])

        largest = left;


    if (right < n && arr[right] > arr[largest])

        largest = right;


    if (largest != i) {

        int temp = arr[i];

        arr[i] = arr[largest];

        arr[largest] = temp;


        heapify(arr, n, largest);

    }

}


void heapSort(int arr[], int n) {

    for (int i = n / 2 - 1; i >= 0; i--)

        heapify(arr, n, i);


    for (int i = n - 1; i > 0; i--) {

        int temp = arr[0];

        arr[0] = arr[i];
        arr[i] = temp;

        heapify(arr, i, 0);
    }
}

void printArray(int arr[], int n) {
    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

int main() {
    int arr[] = {12, 11, 13, 5, 6, 7};
    int n = sizeof(arr) / sizeof(arr[0]);

    heapSort(arr, n);
    printArray(arr, n);

    return 0;
}
```

6. Linear Search

```
#include <stdio.h>
int linearSearch(int arr[], int n, int x) {
    for (int i = 0; i < n; i++)
```

```
        if (arr[i] == x)
            return i;
    return -1;
}

int main() {
    int arr[] = {2, 3, 4, 10, 40};
    int x = 10;
    int n = sizeof(arr) / sizeof(arr[0]);

    int result = linearSearch(arr, n, x);
    if (result == -1)
        printf("Element is not present in array\n");
    else
        printf("Element is present at index %d\n", result);

    return 0;
}
```

7. Binary Search

```
#include <stdio.h>
int binarySearch(int arr[], int l, int r, int x) {
    while (l <= r) {
        int mid = l + (r - l) / 2;

        if (arr[mid] == x)
```

```
                return mid;

            if (arr[mid] < x)
                l = mid + 1;
            else
                r = mid - 1;
        }

        return -1;
}

int main() {
        int arr[] = {2, 3, 4, 10, 40};
        int x = 10;
        int n = sizeof(arr) / sizeof(arr[0]);

        int result = binarySearch(arr, 0, n - 1, x);
        if (result == -1)
            printf("Element is not present in array\n");
        else
            printf("Element is present at index %d\n", result);

        return 0;
}
```

8. implementation of quick sort

```
#include<stdio.h>

#include<conio.h>

void quick_sort(int [],int,int);

int split(int [],int,int);

void main()

{

int a[100],n,i;

clrscr();

printf("\nenter the number of elements");

scanf("%d",&n);

printf("\nenter the array values");

for(i=0;i<n;i++)

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

printf("\narray before sorting");

for(i=0;i<n;i++)

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

quick_sort(a,0,n-1);

printf("\narray after sorting");

for(i=0;i<n;i++)

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

getch();

}

void quick_sort(int a[],int lower,int upper)

{

int i;
```

```
if(lower<upper)

{

i=split(a,lower,upper);

quick_sort(a,lower,i-1);

quick_sort(a,i+1,upper);

}}

int split(int a[],int lower,int upper)

{

int new_lower,new_upper,pivot,temp;

new_lower=lower+1;

new_upper=upper;

pivot=a[lower];

while(new_lower<=new_upper)

{

while(a[new_lower]<pivot)

new_lower++;

while(a[new_upper]>pivot)

new_upper--;

if(new_lower<new_upper)

{

temp=a[new_lower];

a[new_lower]=a[new_upper];

a[new_upper]=temp;

}

}
```

```
temp=a[lower];

a[lower]=a[new_upper];

a[new_upper]=temp;

return new_upper;

}
```