Linear search:

```
/*write a program in c to create a linear search*/
#include<stdio.h>
int main(){
   int arr[]= {10,20,30,21,22,20};
   int searchfor=20;
   for(int i =0; i<5; i++){
      if(arr[i]==searchfor){
         printf("element found at position %d", i);
      }
   }
}
```

BUBBLE SORT:

```
/*write a program in c to perform bubble sort*/
#include <stdio.h>
int main()
{
   int arr[] = {1555, 1, 90, 12, 234, 3222};
   int n = 6;
   for (int i = 0; i < n; i++)
   {
      for (int j = 0; j < n - i - 1; j++)
      {
         if (arr[j] > arr[j + 1])
         {
            int temp = arr[j];
            arr[j] = arr[j + 1];
            arr[j + 1] = temp;
            for(int k= 0; k<n; k++){
               printf("%d\t", arr[k]);
            }
            printf("\n");
         }
      }
   }
   for (int i = 0; i < 6; i++)
   {
      printf("%d\t", arr[i]);
```

```
    }
}
```

SELECTION SORT:

```
/*wap to implement selection sort*/
#include <stdio.h>
int main()
{
   int arr[] = {10, 20, 21, 1, 2, -4};
   int n = 6;
   for (int i = 0; i < n; i++)
   {
      int minindex = i;
      for (int j = i + 1; j < n; j++)
      {
         if (arr[j] < arr[minindex])
         {
            minindex = j;
         }
      }
      int temp = arr[i];
      arr[i] = arr[minindex];
      arr[minindex] = temp;
   }
   for (int i = 0; i < n; i++)
   {
      printf("%d\t", arr[i]);
   }
}
```

Min max:

```
/*write a program to find the min max element using cout comprasion*/
#include <stdio.h>
int main()
{
   int arr[] = {1, 2, 1, -10, 100};
   int smallest = arr[0], greatest = arr[0];
   for (int i = 0; i < 5; i++)
   {
      if (arr[i] < smallest)
      {
         smallest = arr[i];
      }
      if (arr[i] > greatest)
      {
         greatest = arr[i];
      }
   }
   printf("Largest number is %d\n", greatest);
   printf("smallest number is %d\n", smallest);
}
```

Binary search:

```
/*wap to implement a binary search with number of comprassion*/
#include <stdio.h>

int main()
{
   int arr[] = {10, 20, 30, 40, 50, 60};
   int n = 6;
   int low = 0, high = n - 1;
   int comparisons = 0;
   int searchelement = 50;

   while (low <= high)
   {
      int mid = (low + high) / 2;
      comparisons++;

      if (arr[mid] == searchelement)
      {
         printf("Element found at position %d\n", mid);
         printf("Comparisons = %d", comparisons);
         return 0;
      }
      else if (arr[mid] < searchelement)
      {
         low = mid + 1;
```

```
      }
      else
      {
         high = mid - 1;
      }
   }

   printf("Element not found\n");
   printf("Comparisons = %d", comparisons);

   return 0;
}
```

BRUTE FORCE

```
/*write a program to implement a bruteforce string matching*/
#include <stdio.h>
#include <string.h>
int main()
{
   char str[] = "Cfding is fun until you find a bug";
   char match[] = "fun";
   int n = strlen(str);
   char m = strlen(match);
   for (int i = 0; i <= n - m; i++)
   {
      int j;
      for (j = 0; j < m; j++)
      {
         if (str[i + j] != match[j])
         {
            break;
         }
      }
      if (j == m)
      {
         printf("Pattern found at index %d\n", i);
      }
   }
}
```

Reverse binary search:

```
#include <stdio.h>

int binary_search(int arr[], int low, int high, int search)
{
   if (low > high)
   {
      return -1;
   }

   int mid = (low + high) / 2;

   if (arr[mid] == search)
   {
      return mid;
   }
   else if (search < arr[mid])
   {
      return binary_search(arr, low, mid - 1, search);
   }
   else
   {
      return binary_search(arr, mid + 1, high, search);
   }
}
```

```
int main()
{
   int arr[] = {10, 20, 30, 40, 50, 60, 70};
   int search = 50;
   int n = 7;

   int result = binary_search(arr, 0, n - 1, search);

   if (result != -1)
      printf("Element found at index %d\n", result);
   else
      printf("Element not found\n");

   return 0;
}
```

Merge sort:

```
// Merge Sort in C
#include <stdio.h>

void merge(int arr[], int low, int mid, int high)
{
  int i = low;
  int j = mid + 1;
  int k = low;
  int temp[100];

  while (i <= mid && j <= high)
  {
    if (arr[i] < arr[j])
    {
      temp[k] = arr[i];
      i++;
    }
    else
    {
      temp[k] = arr[j];
      j++;
    }
    k++;
  }
```

```
    while (i <= mid)
    {
        temp[k] = arr[i];
        i++;
        k++;
    }

    while (j <= high)
    {
        temp[k] = arr[j];
        j++;
        k++;
    }

    for (int x = low; x <= high; x++)
    {
        arr[x] = temp[x];
    }
}

void mergeSort(int arr[], int low, int high)
{
    if (low < high)
    {
        int mid = (low + high) / 2;
```

```
        mergeSort(arr, low, mid);
        mergeSort(arr, mid + 1, high);
        merge(arr, low, mid, high);
    }
}

int main()
{
    int arr[] = {1, 2, 13, 12, 23, 4, 21};
    int n = 7;

    mergeSort(arr, 0, n - 1);

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

    return 0;
}int main()
{
    int arr[] = {1, 2, 13, 12, 23, 4, 21};
    int n = 7;

}
```

Quick sort:

```
#include <stdio.h>

// Function to swap two elements
void swap(int *a, int *b) {
   int temp = *a;
   *a = *b;
   *b = temp;
}

// Partition function
int partition(int arr[], int low, int high) {
   int pivot = arr[high];   // choosing last element as pivot
   int i = low - 1;

   for (int j = low; j < high; j++) {
      if (arr[j] < pivot) {
         i++;
         swap(&arr[i], &arr[j]);
      }
   }

   swap(&arr[i + 1], &arr[high]);
   return i + 1;
}
```

```
// Quick Sort function
void quickSort(int arr[], int low, int high) {
   if (low < high) {
      int pi = partition(arr, low, high);

      quickSort(arr, low, pi - 1);  // left side
      quickSort(arr, pi + 1, high); // right side
   }
}

// Main function
int main() {
   int arr[] = {10, 7, 8, 9, 1, 5};
   int n = sizeof(arr) / sizeof(arr[0]);

   quickSort(arr, 0, n - 1);

   printf("Sorted array: ");
   for (int i = 0; i < n; i++) {
      printf("%d ", arr[i]);
   }

   return 0;
}
```

DFS:

```
#include <stdio.h>

#define MAX 10

int adj[MAX][MAX];
int visited[MAX];
int n;

// DFS function
void dfs(int v) {
   printf("%d ", v);
   visited[v] = 1;

   for (int i = 0; i < n; i++) {
      if (adj[v][i] == 1 && visited[i] == 0) {
         dfs(i);
      }
   }
}

int main() {
   printf("Enter number of vertices: ");
   scanf("%d", &n);

   printf("Enter adjacency matrix:\n");
```

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

    dfs(0); // start from node 0
    return 0;
}
```

BFS:

```
#include <stdio.h>

#define MAX 10

int adj[MAX][MAX];
int visited[MAX];
int queue[MAX];
int front = -1, rear = -1;
int n;

// enqueue
void enqueue(int v) {
   if (rear == MAX - 1) return;
   if (front == -1) front = 0;
   queue[++rear] = v;
}

// dequeue
int dequeue() {
   return queue[front++];
}

// BFS function
void bfs(int start) {
   enqueue(start);
```

```
    visited[start] = 1;

    while (front <= rear) {
        int v = dequeue();
        printf("%d ", v);

        for (int i = 0; i < n; i++) {
            if (adj[v][i] == 1 && visited[i] == 0) {
                enqueue(i);
                visited[i] = 1;
            }
        }
    }
}

int main() {
    printf("Enter number of vertices: ");
    scanf("%d", &n);

    printf("Enter adjacency matrix:\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            scanf("%d", &adj[i][j]);
        }
    }
```

```
    bfs(0); // start from node 0
    return 0;
}
```

TOPOLOGICAL SORT:

```
#include <stdio.h>

#define MAX 10

int adj[MAX][MAX];
int visited[MAX];
int stack[MAX];
int top = -1;
int n;

// DFS function
void dfs(int v) {
  visited[v] = 1;

  for (int i = 0; i < n; i++) {
    if (adj[v][i] == 1 && !visited[i]) {
      dfs(i);
    }
  }

  stack[++top] = v; // push to stack
}

// Topological sort
void topoSort() {
```

```
    for (int i = 0; i < n; i++) {
        if (!visited[i]) {
            dfs(i);
        }
    }

    printf("Topological Order: ");
    for (int i = top; i >= 0; i--) {
        printf("%d ", stack[i]);
    }
}

int main() {
    printf("Enter number of vertices: ");
    scanf("%d", &n);

    printf("Enter adjacency matrix:\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            scanf("%d", &adj[i][j]);
        }
    }

    topoSort();
    return 0;
}
```

MAX HEAP:

```
#include <stdio.h>

#define MAX 100

int heap[MAX];
int size = 0;

// Swap function
void swap(int *a, int *b) {
   int temp = *a;
   *a = *b;
   *b = temp;
}

// Insert into heap
void insert(int value) {
   size++;
   heap[size] = value;

   int i = size;

   // Heapify up
   while (i > 1 && heap[i] > heap[i / 2]) {
      swap(&heap[i], &heap[i / 2]);
      i = i / 2;
```

```
    }
}

// Delete root (max element)
void deleteRoot() {
    if (size == 0) {
        printf("Heap is empty\n");
        return;
    }

    printf("Deleted: %d\n", heap[1]);

    heap[1] = heap[size];
    size--;

    int i = 1;

    // Heapify down
    while (2 * i <= size) {
        int largest = i;
        int left = 2 * i;
        int right = 2 * i + 1;

        if (left <= size && heap[left] > heap[largest])
            largest = left;
```

```
    if (right <= size && heap[right] > heap[largest])
      largest = right;

    if (largest != i) {
      swap(&heap[i], &heap[largest]);
      i = largest;
    } else {
      break;
    }
  }
}

// Display heap
void display() {
  for (int i = 1; i <= size; i++) {
    printf("%d ", heap[i]);
  }
  printf("\n");
}

int main() {
  insert(10);
  insert(20);
  insert(5);
  insert(30);
```

```
    printf("Heap: ");
    display();

    deleteRoot();

    printf("After deletion: ");
    display();

    return 0;
}
```

MIN HEAP:

```
#include <stdio.h>

#define MAX 100

int heap[MAX];
int size = 0;

// Swap function
void swap(int *a, int *b) {
   int temp = *a;
   *a = *b;
   *b = temp;
}

// Insert into Min Heap
void insert(int value) {
   size++;
   heap[size] = value;

   int i = size;

   // Heapify up
   while (i > 1 && heap[i] < heap[i / 2]) {
      swap(&heap[i], &heap[i / 2]);
      i = i / 2;
```

```
    }
}

// Delete root (minimum element)
void deleteRoot() {
    if (size == 0) {
        printf("Heap is empty\n");
        return;
    }

    printf("Deleted: %d\n", heap[1]);

    heap[1] = heap[size];
    size--;

    int i = 1;

    // Heapify down
    while (2 * i <= size) {
        int smallest = i;
        int left = 2 * i;
        int right = 2 * i + 1;

        if (left <= size && heap[left] < heap[smallest])
            smallest = left;
```

```
    if (right <= size && heap[right] < heap[smallest])
      smallest = right;

    if (smallest != i) {
      swap(&heap[i], &heap[smallest]);
      i = smallest;
    } else {
      break;
    }
  }
}

// Display heap
void display() {
  for (int i = 1; i <= size; i++) {
    printf("%d ", heap[i]);
  }
  printf("\n");
}

int main() {
  insert(30);
  insert(10);
  insert(20);
  insert(5);
```

```
    printf("Min Heap: ");
    display();

    deleteRoot();

    printf("After deletion: ");
    display();

    return 0;
}
```