Queue using Array

```
#include <stdio.h>
#include <stdlib.h>
struct Queue
{
    int size;
    int front;
    int rear;
    int *Q;
};

void create(struct Queue *q,int size)
{
    q->size=size;
    q->front=q->rear=-1;
    q->Q=(int *)malloc(q->size*sizeof(int));
}

void enqueue(struct Queue *q,int x)
{
    if(q->rear==q->size-1)
        printf("Queue is Full");
    else
    {
        q->rear++;
        q->
        Q[q->rear]=x;
    }
}

int dequeue(struct Queue *q)
{
    int x=-1;

    if(q->front==q->rear)
        printf("Queue is Empty\n");
    else
    {
```

```c
        q->front++;
        x=q->Q[q->front];
    }
    return x;
}

void Display(struct Queue q)
{
    int i;

    for(i=q.front+1;i<=q.rear;i++)
        printf("%d ",q.Q[i]);
    printf("\n");
}

int main()
{
    struct Queue q;
    create(&q,5);

    enqueue(&q,10);
    enqueue(&q,20);
    enqueue(&q,30);

    Display(q);

    printf("%d ",dequeue(&q));


    return 0;
}
```

Queue using CPP

```
#include <iostream>
using namespace std;

class Queue
{
private:
    int front;
    int rear;
    int size;
    int *Q;
public:
    Queue(){front=rear=-1;size=10;Q=new int[size];}
    Queue(int size){front=rear=-1;this-
>size=size;Q=new int[this->size];}
    void enqueue(int x);
    int dequeue();
    void Display();
};

void Queue::enqueue(int x)
{
    if(rear==size-1)
        printf("Queue Full\n");
    else
    {
        rear++;
        Q[rear]=x;
    }
}

int Queue::dequeue()
{
    int x=-1;
    if(front==rear)
        printf("Queue is Empty\n");
    else
    {
        x=Q[front+1];
        front++;
    }
    return x;
}

void Queue::Display()
{
    for(int i=front+1;i<=rear;i++)
        printf("%d ",Q[i]);
    printf("\n");
}

int main()
{
    Queue q(5);

    q.enqueue(10);
    q.enqueue(20);
    q.enqueue(30);

    q.Display();

    return 0;
}
```

Circular Queue

```
#include <stdio.h>
#include <stdlib.h>
struct Queue
{
    int size;
    int front;
    int rear;
    int *Q;
};

void create(struct Queue *q,int size)
{
    q->size=size;
    q->front=q->rear=0;
    q->Q=(int *)malloc(q->size*sizeof(int));
}

void enqueue(struct Queue *q,int x)
{
    if((q->rear+1)%q->size==q->front)
        printf("Queue is Full");
    else
    {
        q->rear=(q->rear+1)%q->size;
        q->Q[q->rear]=x;
    }
}

int dequeue(struct Queue *q)
{
    int x=-1;

    if(q->front==q->rear)
        printf("Queue is Empty\n");
    else
    {
        q->front=(q->front+1)%q->size;
        x=q->Q[q->front];
    }
    return x;
}

void Display(struct Queue q)
{
    int i=q.front+1;

    do
    {

        printf("%d ",q.Q[i]);
        i=(i+1)%q.size;
    }while(i!=(q.rear+1)%q.size);

    printf("\n");
}

int main()
{
    struct Queue q;
    create(&q,5);

    enqueue(&q,10);
    enqueue(&q,20);
    enqueue(&q,30);
    enqueue(&q,40);
    enqueue(&q,50);
    enqueue(&q,60);

    Display(q);

    printf("%d ",dequeue(&q));


    return 0;
}
```

Queue using Linked List

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

struct Node
{
    int data;
    struct Node *next;

}*front=NULL,*rear=NULL;

void enqueue(int x)
{
    struct Node *t;
    t=(struct Node*)malloc(sizeof(struct Node));
    if(t==NULL)
        printf("Queue is FUll\n");
    else
    {
        t->data=x;
        t->next=NULL;
        if(front==NULL)
            front=rear=t;
        else
        {
            rear->next=t;
            rear=t;
        }
    }

}

int dequeue()
{
    int x=-1;
    struct Node* t;

    if(front==NULL)
```

```c
        printf("Queue is Empty\n");
    else
    {
        x=front->data;
        t=front;
        front=front->next;
        free(t);
    }
    return x;
}

void Display()
{
    struct Node *p=front;
    while(p)
    {
        printf("%d ",p->data);
        p=p->next;
    }
    printf("\n");
}

int main()
{
    enqueue(10);
    enqueue(20);
    enqueue(30);
    enqueue(40);
    enqueue(50);

    Display();

    printf("%d ",dequeue());

    return 0;
}
```

Bubble Sort

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

void swap(int *x,int *y)
{
    int temp=*x;
    *x=*y;
    *y=temp;
}

void Bubble(int A[],int n)
{
    int i,j,flag=0;

    for(i=0;i<n-1;i++)
    {
        flag=0;
        for(j=0;j<n-i-1;j++)
        {
            if(A[j]>A[j+1])
            {
                swap(&A[j],&A[j+1]);
                flag=1;
            }
        }
        if(flag==0)
            break;
    }

}


int main()
{
   int A[]={11,13,7,12,16,9,24,5,10,3},n=10,i;

    Bubble(A,n);
    for(i=0;i<10;i++)
        printf("%d ",A[i]);
    printf("\n");

    return 0;
}
```

Insertion Sort

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

void swap(int *x,int *y)
{
    int temp=*x;
    *x=*y;
    *y=temp;
}

void Insertion(int A[],int n)
{
    int i,j,x;

    for(i=1;i<n;i++)
    {
        j=i-1;
        x=A[i];
        while(j>-1 && A[j]>x)
        {
            A[j+1]=A[j];
            j--;
        }
        A[j+1]=x;
    }
}


int main()
{
   int A[]={11,13,7,12,16,9,24,5,10,3},n=10,i;

    Insertion(A,n);

    for(i=0;i<10;i++)
        printf("%d ",A[i]);
    printf("\n");
    return 0;
}
```

Selection Sort

```c
#include <stdio.h>
#include<stdlib.h>

void swap(int *x,int *y)
{
    int temp=*x;
    *x=*y;
    *y=temp;
}

void SelectionSort(int A[],int n)
{
    int i,j,k;

    for(i=0;i<n-1;i++)
    {
        for(j=k=i;j<n;j++)
        {
            if(A[j]<A[k])
                k=j;
        }
        swap(&A[i],&A[k]);
    }
}


int main()
{
   int A[]={11,13,7,12,16,9,24,5,10,3},n=10,i;

    SelectionSort(A,n);

    for(i=0;i<10;i++)
        printf("%d ",A[i]);
    printf("\n");

    return 0;
}
```

Quick Sort

```c
#include <stdio.h>
#include<stdlib.h>

void swap(int *x,int *y)
{
    int temp=*x;
    *x=*y;
    *y=temp;
}

int partition(int A[],int l,int h)
{
    int pivot=A[l];
    int i=l,j=h;

    do
    {
        do{i++;}while(A[i]<=pivot);
        do{j--;}while(A[j]>pivot);

        if(i<j)swap(&A[i],&A[j]);
    }while(i<j);

    swap(&A[l],&A[j]);
    return j;
}

void QuickSort(int A[],int l,int h)
{
    int j;

    if(l<h)
    {
        j=partition(A,l,h);
        QuickSort(A,l,j);
        QuickSort(A,j+1,h);
    }
}


int main()
{
   int A[]={11,13,7,12,16,9,24,5,10,3},n=10,i;

    QuickSort(A,n);

    for(i=0;i<10;i++)
        printf("%d ",A[i]);
    printf("\n");

    return 0;
}
```

Merge Sort

```c
#include <stdio.h>
#include<stdlib.h>

void swap(int *x,int *y)
{
    int temp=*x;
    *x=*y;
    *y=temp;
}

void Merge(int A[],int l,int mid,int h)
{
    int i=l,j=mid+1,k=l;
    int B[100];

    while(i<=mid && j<=h)
    {
        if(A[i]<A[j])
            B[k++]=A[i++];
        else
            B[k++]=A[j++];
    }
    for(;i<=mid;i++)
        B[k++]=A[i];
    for(;j<=h;j++)
        B[k++]=A[j];

    for(i=l;i<=h;i++)
        A[i]=B[i];
}

void IMergeSort(int A[],int n)
{
    int p,l,h,mid,i;

    for(p=2;p<=n;p=p*2)
    {
        for(i=0;i+p-1<=n;i=i+p)
        {
            l=i;
            h=i+p-1;
            mid=(l+h)/2;
            Merge(A,l,mid,h);
        }
    }
    if(p/2<n)
        Merge(A,0,p/2-1,n);

}


int main()
{
   int A[]={11,13,7,12,16,9,24,5,10,3},n=10,i;

    IMergeSort(A,n);

    for(i=0;i<10;i++)
        printf("%d ",A[i]);
    printf("\n");

    return 0;
}
```

Recursive Merge Sort

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

void swap(int *x,int *y)
{
    int temp=*x;
    *x=*y;
    *y=temp;
}

void Merge(int A[],int l,int mid,int h)
{
    int i=l,j=mid+1,k=l;
    int B[100];

    while(i<=mid && j<=h)
    {
        if(A[i]<A[j])
            B[k++]=A[i++];
        else
            B[k++]=A[j++];
    }
    for(;i<=mid;i++)
        B[k++]=A[i];
    for(;j<=h;j++)
        B[k++]=A[j];

    for(i=l;i<=h;i++)
        A[i]=B[i];
}

void MergeSort(int A[],int l,int h)
{
    int mid;
    if(l<h)
    {
        mid=(l+h)/2;
        MergeSort(A,l,mid);
        MergeSort(A,mid+1,h);
        Merge(A,l,mid,h);

    }
}


int main()
{
   int A[]={11,13,7,12,16,9,24,5,10,3},n=10,i;

    MergeSort(A,n);

    for(i=0;i<10;i++)
        printf("%d ",A[i]);
    printf("\n");

    return 0;
}
```

Count Sort

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

void swap(int *x,int *y)
{
    int temp=*x;
    *x=*y;
    *y=temp;
}

int findMax(int A[],int n)
{
    int max=INT32_MIN;
    int i;
    for(i=0;i<n;i++)
    {
        if(A[i]>max)
            max=A[i];
    }
    return max;
}


void CountSort(int A[],int n)
{
    int i,j,max,*C;

    max=findMax(A,n);
    C=(int *)malloc(sizeof(int)*(max+1));

    for(i=0;i<max+1;i++)
    {
        C[i]=0;
    }
    for(i=0;i<n;i++)
    {
        C[A[i]]++;
```

```c
    }

    i=0;j=0;
    while(j<max+1)
    {
        if(C[j]>0)
        {
            A[i++]=j;
            C[j]--;
        }
        else
            j++;
    }
}




int main()
{
   int A[]={11,13,7,12,16,9,24,5,10,3},n=10,i;

    CountSort(A,n);

    for(i=0;i<10;i++)
        printf("%d ",A[i]);
    printf("\n");

    return 0;
}
```

Shell Sort

```c
#include <stdio.h>
#include<stdlib.h>

void swap(int *x,int *y)
{
    int temp=*x;
    *x=*y;
    *y=temp;
}

void ShellSort(int A[],int n)
{
    int gap,i,j,temp;

    for(gap=n/2;gap>=1;gap/=2)
    {
        for(i=gap;i<n;i++)
        {
            temp=A[i];
            j=i-gap;
            while(j>=0 && A[j]>temp)
            {
                A[j+gap]=A[j];
                j=j-gap;
            }
            A[j+gap]=temp;

        }
    }

}




int main()
{
   int A[]={11,13,7,12,16,9,24,5,10,3},n=10,i;

    SellSort(A,n);

    for(i=0;i<10;i++)
        printf("%d ",A[i]);
    printf("\n");

    return 0;
}
```