Sorting

1. Bubble Sort

```
void bubbleSort(int *arr,int size)
{
int e,f,m,g;
m=size-2;
while(m>=0)
{
e=0;
f=e+1;
while(e<=m)
{
if(arr[f]<arr[e])
{
arr[e]=arr[e]+arr[f];
arr[f]=arr[e]-arr[f];
arr[e]=arr[e]-arr[f];
}
e++;
f++;
}
m--;
}
```

2. Insertion Sort:-

```
void insertionSort(int *arr,int size)
{
int p,j,num,y;
int lb,ub;
lb=0;
ub=size-1;
y=lb+1;
while(y<=ub)
{
p=y;
num=arr[p];
j=p-1;
while(j>=lb)
{
if(num>=arr[j])
{
break;
}
arr[j+1]=arr[j];
j--;
p--;
}
arr[p]=num;
y++;
}
}
```

3. Selection Sort:-

```
void selectionSort(int *arr,int size)
{
int e,f,g,m;
e=0;
while(e<=size-2)
{
f=e+1;
m=e;
while(f<=size-1)
{
if(arr[f]<arr[m])
{
m=f;
}
f++;
}
g=arr[e];
arr[e]=arr[m];
arr[m]=g;
printArray(arr,size);
e++;
}
```

4. Quick Sort:-

```
void quickSort(int *arr,int size)
{
int stack[2][5];
int top=5;
int lb,ub;
int e,f,pp,g;
top--;
```

```
stack[0][top]=0;
stack[1][top]=size-1;
while(top!=5)
{
lb=stack[0][top];
ub=stack[1][top];
top++;
e=lb;
f=ub;
while(1)
{
while(e!=ub && arr[e]<=arr[lb])
{
e++;
}
while(f!=lb && arr[f]>=arr[lb])
{
f--;
}
if(e<f)
{
g=arr[e];
arr[e]=arr[f];
arr[f]=g;
}
else
{
g=arr[f];
arr[f]=arr[lb];
arr[lb]=g;
break;
}
}//inner loop
pp=f;
if(pp-1>lb)
{
top--;
stack[0][top]=lb;
stack[1][top]=pp-1;
}
if(pp+1<ub)
{
top--;
stack[0][top]=pp+1;
stack[1][top]=ub;
}
}//Exterior loop
printArray(arr,size);
}
```

5. Merge Sort :-

```
void mergeSort(int *arr,int size)
{
int stack1[2][size],stack2[2][size];
int top1,top2;
int low,high,mid;
top1=top2=size;
top1--;
stack1[0][top1]=0;
stack1[1][top1]=size-1;
while(top1!=size)
{
low=stack1[0][top1];
high=stack1[1][top1];
top1++;
mid=(low+high)/2;
top2--;
stack2[0][top2]=low;
stack2[1][top2]=high;
if(low<mid)
{
top1--;
stack1[0][top1]=low;
stack1[1][top1]=mid;
}
if(mid+1<high)
{
top1--;
stack1[0][top1]=mid+1;
stack1[1][top1]=high;
}
```

```
}
int e,f,g,lb,ub;
while(top2!=size)
{
lb=stack2[0][top2];
ub=stack2[1][top2];
top2++;
e=lb;
while(e<ub)
{
f=e+1;
while(f<=ub)
{
if(arr[f]<arr[e])
{
arr[e]=arr[e]+arr[f];
arr[f]=arr[e]-arr[f];
arr[e]=arr[e]-arr[f];
}
f++;
}
e++;
}
}
printArray(arr,size);
}
```

6. Heap Sort :-

```
void buildHeap(int *arr,int size)
{
for(int i=1;i<size;i++)
{
if(arr[i]>arr[(i-1)/2])
{
int j=i;
while(arr[j]>arr[(j-1)/2])
{
swap(arr[j],arr[(j-1)/2]);
j=(j-1)/2;
}
}
}
void heapSort(int *arr,int size)
{
buildHeap(arr,size);
for(int i=size-1;i>0;i--)
{
swap(arr[0],arr[i]);
int j=0,index;
do
{
index=(2*j+1);
if(arr[index]<arr[index+1] && index<(i-1))
index++;
if(arr[j]<arr[index] && index<i)
swap(arr[j],arr[index]);
j=index;
}while(index<i);
}
}
```

7. Radix Sort:-

```
int largest(int *arr,int size)
{
int large=arr[0];
for(int i=0;i<size;i++)
{
if(large<arr[i])
{
large=arr[i];
}
}
return large;
}
void radixSort(int *arr,int size)
{
int bucket[10][10],bucketCount[10];
int i,j,k,divisor,remainder,large,pass;
int nop=0;
large=largest(arr,size);
divisor=1;
```

```
while(large>0)
{
nop++;
large/=10;
}
for(pass=0;pass<nop;pass++)
{
for(i=0;i<10;i++)
{
bucketCount[i]=0;
}
for(i=0;i<size;i++)
{
remainder=(arr[i]/divisor)%10;
bucket[remainder][bucketCount[remainder
]]=arr[i];
bucketCount[remainder]+=1;
}
i=0;
for(k=0;k<10;k++)
{
for(j=0;j<bucketCount[k];j++)
{
arr[i]=bucket[k][j];
i++;
}
}
divisor*=10;
}
}
```

8. Shell Sort:-

```
void shellSort(int *arr,int size)
{
int i,j,k,tmp;
for(i=size/2;i>0;i=i/2)
{
for(j=i;j<size;j++)
{
for(k=j-i;k>=0;k=k-i)
{
if(arr[k+i]>=arr[k])
{
break;
}
else
{
tmp=arr[k];
arr[k]=arr[k+i];
arr[k+i]=tmp;
}
}
}
}
}
```