Bubble Sort

```
#include<stdio.h>
int main()
{
int x[5];
int y,e,f,g,m;
for(y=0;y<=4;y++)
{
printf("Enter a number : ");
scanf("%d",&x[y]);
}
m=3;
while(m>=0)
{
e=0;
f=1;
while(e<=m)
{
if(x[f]<x[e])
{
g=x[e];
x[e]=x[f];
x[f]=g;
}
e++;
f++;
}
m--;
}
for(y=0;y<=4;y++)
{
printf("%d\n",x[y]);
}
return 0;
}
```

Linear Sort

```
#include<stdio.h>
int main()
{
int x[10],y,e,f,g,size,oep,iep;
size=10;
for(y=0;y<size;y++)
{
printf("Enter a number : ");
scanf("%d",&x[y]);
}
oep=size-2;
iep=size-1;
e=0;
while(e<=oep)
{
f=e+1;
while(f<=iep)
{
if(x[f]<x[e])
{
g=x[e];
x[e]=x[f];
x[f]=g;
}
f++;
}
e++;
}
for(y=0;y<size;y++)
{
printf("%d\n",x[y]);
}
return 0;
}
```

Selection sort

```
#include<stdio.h>
int main()
{
int x[10],y,e,f,g,si;
y=0;
while(y<=9)
{
printf("Enter a number : ");
scanf("%d",&x[y]);
y++;
}
e=0;
while(e<=8)
{
si=e;
f=e+1;
while(f<=9)
{
if(x[f]<x[si])
{
si=f;
}
f++;
}
g=x[si];
x[si]=x[e];
x[e]=g;
e++;
}
y=0;
while(y<=9)
{
printf("%d\n",x[y]);
y++;
}
return 0;
}
```

Insertion Sort

```
#include<stdio.h>
int main()
{
int x[10],y,num,z;
for(y=0;y<10;y++)
{
printf("Enter a number : ");
scanf("%d",&x[y]);
}
y=1;
while(y<=9)
{
num=x[y];
z=y-1;
while(z>=0 && x[z]>num)
{
x[z+1]=x[z];
z--;
}
x[z+1]=num;
y++;
}
for(y=0;y<10;y++)
{
printf("%d\n",x[y]);
}
return 0;
}
```

Quick Sort (Iterative)

```
#include<stdio.h>
int main()
{
int x[10],y,e,f,g,pp,stack[10][2],top,lowerBound,upperBound,lb,ub;
for(y=0;y<=9;y++)
{
printf("Enter a number : ");
scanf("%d",&x[y]);
}
lowerBound=0;
upperBound=9;
top=10;
// push lowerBound , upperBound on stack
top--;
```

```
stack[top][0]=lowerBound;
stack[top][1]=upperBound;
while(top!=10)    // enter the loop construct if the stack is not empty
{
// pop
lb=stack[top][0];
ub=stack[top][1];
top++;
e=lb;
f=ub;
while(e<f)
{
// move e forward
while(e<ub && x[e]<=x[lb]) e++;
// move f backward
while(x[f]>x[lb]) f--;
if(e<f)
{
g=x[e];
x[e]=x[f];
x[f]=g;
}
else
{
g=x[lb];
x[lb]=x[f];
x[f]=g;
pp=f;
}
}
if(pp+1<ub)
{
top--;
stack[top][0]=pp+1;
stack[top][1]=ub;
}
if(lb<pp-1)
{
top--;
stack[top][0]=lb;
stack[top][1]=pp-1;
}
}
for(y=0;y<=9;y++)
{
printf("%d\n",x[y]);
}
```

```
return 0;
}
```

Quick Sort(Recursive)

```
#include<stdio.h>
int findPartitionPoint(int *x,int lb,int ub)
{
int e,f,g;
e=lb;
f=ub;
while(e<f)
{
while(e<ub && x[e]<=x[lb])
{
e++;
}
while(x[f]>x[lb])
{
f--;
}
if(e<f)
{
g=x[e];
x[e]=x[f];
x[f]=g;
}
else
{
g=x[lb];
x[lb]=x[f];
x[f]=g;
}
}
return f;
}
void quickSort(int *x,int lb,int ub)
{
int pp;
if(ub<=lb) return;
pp=findPartitionPoint(x,lb,ub);
quickSort(x,lb,pp-1);
quickSort(x,pp+1,ub);
}
int main()
{
int x[10],y;
```

```
for(y=0;y<=9;y++)
{
printf("Enter a number : ");
scanf("%d",&x[y]);
}
quickSort(x,0,9);
for(y=0;y<=9;y++)
{
printf("%d\n",x[y]);
}
return 0;
}
```

Merge Sort

```
#include<stdio.h>
int main()
{
int x[10],tmp[10],i1,i2,i3,a,b,mid,lb1,lb2,lb3,ub1,ub2,ub3,top1,top2,y;
int stack1[10][2],stack2[10][2];
top1=10;
top2=10;
for(y=0;y<=9;y++)
{
printf("Enter a number : ");
scanf("%d",&x[y]);
}
// push lowerbound,upperbound on stack1
top1--;
stack1[top1][0]=0;
stack1[top1][1]=9;
while(top1!=10) // stack1 should not be empty
{
// pop from stack1 into a,b
a=stack1[top1][0];
b=stack1[top1][1];
top1++;
// push a,b on stack2
top2--;
stack2[top2][0]=a;
stack2[top2][1]=b;
// calculate mid
mid=(a+b)/2;
if(a<mid) // push on stack1
{
top1--;
stack1[top1][0]=a;
stack1[top1][1]=mid;
```

```
}
if(mid+1<b) // push on stack1
{
top1--;
stack1[top1][0]=mid+1;
stack1[top1][1]=b;
}
}
while(top2!=10) // stack2 should not be empty
{
// pop from stack2
lb1=stack2[top2][0];
ub2=stack2[top2][1];
top2++;
ub1=(lb1+ub2)/2;
lb2=ub1+1;
// logic to merge starts here
lb3=lb1;
ub3=ub2;
i1=lb1;
i2=lb2;
i3=lb3;
while(i1<=ub1 && i2<=ub2)
{
if(x[i1]<x[i2])
{
tmp[i3]=x[i1];
i1++;
}
else
{
tmp[i3]=x[i2];
i2++;
}
i3++;
}
while(i1<=ub1)
{
tmp[i3]=x[i1];
i1++;
i3++;
}
while(i2<=ub2)
{
tmp[i3]=x[i2];
i2++;
i3++;
```

```
}
// copy back
i3=lb3;
while(i3<=ub3)
{
x[i3]=tmp[i3];
i3++;
}
}
for(y=0;y<=9;y++)
{
printf("%d\n",x[y]);
}
return 0;
}
```

Merge Sort (Recursive)

```
#include<stdio.h>
#include<stdlib.h>
void merge(int *x,int lb,int mid,int ub)
{
int i1,i2,i3,tmpSize,lb1,lb2,lb3,ub1,ub2,ub3;
int *tmp;
tmpSize=(ub-lb)+1;
tmp=(int *)malloc(sizeof(int)*tmpSize);
lb1=lb;
ub1=mid;
lb2=mid+1;
ub2=ub;
lb3=0;
ub3=tmpSize-1;
i1=lb1;
i2=lb2;
i3=lb3;
while(i1<=ub1 && i2<=ub2)
{
if(x[i1]<x[i2])
{
tmp[i3]=x[i1];
i1++;
}
else
{
tmp[i3]=x[i2];
i2++;
}
i3++;
```

```
}
while(i1<=ub1)
{
tmp[i3]=x[i1];
i1++;
i3++;
}
while(i2<=ub2)
{
tmp[i3]=x[i2];
i2++;
i3++;
}
i1=lb1;
i3=0;
while(i1<=ub2)
{
x[i1]=tmp[i3];
i3++;
i1++;
}
free(tmp);
}
void mergeSort(int *x,int low,int high)
{
int mid;
if(low<high)
{
mid=(low+high)/2;
mergeSort(x,low,mid);
mergeSort(x,mid+1,high);
merge(x,low,mid,high);
}
}
int main()
{
int x[10],y;
for(y=0;y<=9;y++)
{
printf("Enter a number : ");
scanf("%d",&x[y]);
}
mergeSort(x,0,9);
for(y=0;y<=9;y++)
{
printf("%d\n",x[y]);
}
```

```
return 0;
}
```

Heap Sort

```
#include<stdio.h>
int main()
{
int x[10],y,ci,ri,lci,rci,g,swi;
y=0;
while(y<=9)
{
printf("Enter a number : ");
scanf("%d",&x[y]);
y++;
}
// logic to convert the contents of the array to heap
y=1;
while(y<=9)
{
ci=y;
while(ci>0)
{
ri=(ci-1)/2;
if(x[ci]>x[ri])
{
g=x[ci];
x[ci]=x[ri];
x[ri]=g;
ci=ri;
}
else
{
break;
}
}
y++;
}
// implementing heap sort
y=9;
while(y>0)
{
g=x[0];
x[0]=x[y];
x[y]=g;
y--;
ri=0;
```

```
while(ri<y)
{
lci=(ri*2)+1;
if(lci>y) break;
rci=lci+1;
if(rci>y)
{
swi=lci;
}
else
{
if(x[lci]>x[rci])
{
swi=lci;
}
else
{
swi=rci;
}
}
if(x[swi]>x[ri])
{
g=x[swi];
x[swi]=x[ri];
x[ri]=g;
ri=swi;
}
else
{
break;
}
}
}
for(y=0;y<=9;y++)
{
printf("%d\n",x[y]);
}
return 0;
}
```

Radix sort

```
#include<stdio.h>
#include<stdlib.h>
typedef struct _queue_node
{
int num;
struct _queue_node *next;
}QueueNode;
typedef struct _queue
{
QueueNode *start;
QueueNode *end;
int size;
}Queue;
void initQueue(Queue *queue)
{
queue->start=NULL;
queue->end=NULL;
queue->size=0;
}
int isQueueEmpty(Queue *queue)
{
return queue->size==0;
}
void addToQueue(Queue *queue,int num)
{
QueueNode *t;
t=(QueueNode *)malloc(sizeof(QueueNode));
t->num=num;
t->next=NULL;
if(queue->start==NULL)
{
queue->start=t;
queue->end=t;
}
else
{
queue->end->next=t;
queue->end=t;
}
queue->size++;
}
int removeFromQueue(Queue *queue)
{
int num;
QueueNode *t;
num=queue->start->num;
```

```
t=queue->start;
queue->start=queue->start->next;
if(queue->start==NULL) queue->end=NULL;
free(t);
queue->size--;
return num;
}
void clearQueue(Queue *queue)
{
QueueNode *t;
while(queue->start!=NULL)
{
t=queue->start;
queue->start=queue->start->next;
queue->size--;
free(t);
}
queue->end=NULL;
queue->size=0;
}
int main()
{
int x[10],y,e,f,i,num,largest,dc,k;
Queue queues[10];
for(i=0;i<=9;i++)
{
initQueue(&queues[i]);
}
for(y=0;y<=9;y++)
{
printf("Enter a number : ");
scanf("%d",&x[y]);
}
largest=x[0];
for(y=1;y<=9;y++)
{
if(x[y]>largest) largest=x[y];
}
dc=1;
num=largest;
while(num>9)
{
num=num/10;
dc++;
}
e=10;
f=1;
```

```
k=1;
while(k<=dc)
{
// spread out in 10 queues according to digit at kth place
y=0;
while(y<=9)
{
num=x[y];
i=(num%e)/f;
addToQueue(&queues[i],num);
y++;
}
// collect all numbers from 10 queues and keep it in array
i=0;
y=0;
while(y<=9)
{
while(!isQueueEmpty(&queues[y]))
{
num=removeFromQueue(&queues[y]);
x[i]=num;
i++;
}
y++;
}
e=e*10;
f=f*10;
k++;
}
for(y=0;y<=9;y++)
{
printf("%d\n",x[y]);
}
return 0;
}
```

Shell sort

```
#include<stdio.h>
int main()
{
int x[10],y,z,num,size,diff,lb,ub;
lb=0;
ub=9;
for(y=0;y<=9;y++)
{
printf("Enter a number : ");
scanf("%d",&x[y]);
```

```
}
size=(ub-lb)+1;
diff=size/2;
while(diff>=1)
{
y=lb+diff;
while(y<=ub)
{
num=x[y];
z=y-diff;
while(z>=0 && x[z]>num)
{
x[z+diff]=x[z];
z=z-diff;;
}
x[z+diff]=num;
y=y+diff;
}
diff=diff/2;
}
for(y=0;y<=9;y++) printf("%d\n",x[y]);
return 0;
}
```

Sorting

Algorithm	Data Structure	Time Complexity			Worst Case Auxiliary Space Complexity
		Best	Average	Worst	Worst
Quicksort	Array	O(n log(n))	O(n log(n))	O(n^2)	O(n)
Mergesort	Array	O(n log(n))	O(n log(n))	O(n log(n))	O(n)
Heapsort	Array	O(n log(n))	O(n log(n))	O(n log(n))	O(1)
Bubble Sort	Array	O(n)	O(n^2)	O(n^2)	O(1)
Insertion Sort	Array	O(n)	O(n^2)	O(n^2)	O(1)
Select Sort	Array	O(n^2)	O(n^2)	O(n^2)	O(1)
Bucket Sort	Array	O(n+k)	O(n+k)	O(n^2)	O(nk)
Radix Sort	Array	O(nk)	O(nk)	O(nk)	O(n+k)