INSERTION\_SORT

void insertion\_sort(int arr[],int n)

{

int i,j,k;

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

{

k=arr[i];

j=i-1;

while(j>=0 && arr[j]>k)

{

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

j=j-1;

}

arr[j+1]=k;

}

}

SELECTION\_SORT

void selection\_sort(int arr[],int n)

{

int i,j,min;

for(i=0;i<n-1;i++)

{

min=i;

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

{

if(arr[j]<arr[min])

{

min=j;

}

swap(&arr[min] , &arr[i]);

}

}

}

BUBBLE\_SORT

void bubble\_sort(int arr[],int n)

{

int i,j;

for(i=0;i<n-1;i++)

{

for(j=0;j<n-1-i;j++)

{

if(arr[j]>arr[j+1])

{

swap(&arr[j],&arr[j+1]);

}

}

}

}

SHELL\_SORT

void shell\_sort(int arr[],int n)

{

int temp,gap,j;

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

{

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

{

temp=arr[i];

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

{

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

}

arr[j]=temp;

}

}

}

COMB\_SORT

int nextgap(int k)

{

k=(k\*10)/13;

if(k<1)

{

return 1;

}

return k;

}

void comb\_sort(int arr[],int n)

{

int i,j,k;

k=n;

bool s=true;

while(k!=1||s==true)

{

k=nextgap(k);

s=false;

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

{

if(arr[i]>arr[i+k])

{

swap(&arr[i],&arr[i+k]);

s=true;

}

}

}

}

COCKTAIL\_SORT

void cocktail\_sort(int arr[],int n)

{

bool swapped = true;

int start=0;

int end=n-1;

while(swapped)

{

for(int i=start;i<end;i++)

{

if(arr[i]>arr[i+1])

{

swap(&arr[i],&arr[i+1]);

}

swapped=true;

}

if(!swapped)

{

break;

}

swapped = false;

end--;

for(int i=end-1;i>=start;i--)

{

if(arr[i]>arr[i+1])

{

swap(&arr[i],&arr[i+1]);

}

swapped=true;

}

start++;

}

}

MERGE\_SORT

void Merge(int \*array , int left, int mid, int right)

{

int temp[right-left+1] , k=0 , i=left , j=mid+1 ;

while(i<=mid && j<=right)

{

if(array[i]<array[j])

{

temp[k]=array[i];

i++;

k++;

}

else

{

temp[k]=array[j];

j++;

k++;

}

}

while(i<=mid)

{

temp[k]=array[i];

i++;

k++;

}

while(j<=right)

{

temp[k]=array[j];

j++;

k++;

}

for(i=left; i <=right ; i++)

{

array[i]=temp[i-left];

}

}

void MergeSort(int \*array ,int left ,int right)

{

if(right>left)

{

int mid = (left + right)/2;

MergeSort(array,left,mid);

MergeSort(array,mid+1,right);

Merge(array,left,mid,right);

}

}

QUICK\_SORT

int partition(int arr[],int low,int high)

{

int pivot =arr[high];

int i=(low-1);

for(int j=low;j<=high-1;j++)

{

if(arr[j]<pivot)

{

i++;

swap(&arr[i],&arr[j]);

}

}

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

return (i+1);

}

void quicksort(int arr[],int low ,int high)

{

if(low<high)

{

int p=partition(arr,low,high);

quicksort(arr,p+1,high);

quicksort(arr,low,p-1);

}

}

RADIX\_SORT

int getmax(int arr[],int n)

{

int max=arr[0];

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

{

if(arr[i]>max)

{

max=arr[i];

}

}

return max;

}

void countsort(int arr[],int n,int exp)

{

int output[n];

int count[10]={0};

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

{

count[(arr[i]/exp)%10]++;

}

for(int i=1;i<10;i++)

{

count[i]+=count[i-1];

}

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

{

output[count[ (arr[i]/exp)%10] -1]=arr[i];

count[(arr[i]/exp)%10]--;

}

for(int 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);

}

}

HEAP\_SORT

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

{

int largest=i;

int l=2\*i+1;

int r=2\*i+2;

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

{

largest = l;

}

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

{

largest = r;

}

if(largest != i)

{

swap(&arr[i],&arr[largest]);

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--)

{

swap(arr[0], arr[i]);

heapify(arr, i, 0);

}

}