SORTING IN C

Bubble Sort

NON – ADAPTIVE→

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

// For printing the array
void show_array(int * arr, int n)
{
   int i;
   for(i = 0; i<n; i++)
   {
      printf("%d",arr[i]);
   }
   printf("\n");

}
// Function to do sorting
void calculate_bubble_sort(int * arr, int n)
{
  int i,j,temp;
  for(i = 0; i<n-1; i++)   // For number of pass
  {
     for(j = 0; j<n-1-i; j++)  // for comparison in each pass
     {
        if(arr[j] > arr[j+1])
        {
           temp = arr[j];
           arr[j] = arr[j+1];
```

```
            arr[j+1] = temp;
          }
       }
    }
}

void main()
{

   int arr[40],i;
   printf("Enter the number of elements:");
   scanf("%d",&n);

   printf("Enter the array elements:");
   for(i = 0; i<n; i++)
   {
      scanf("%d",&arr[i]);
   }

   show_array(arr,n); // print the array before sorting
   calculate_bubble_sort(arr,n); // call the function to sort the array
   show_array(arr,n); // print the array after sorting
}
```

ADAPTIVE→

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

// For printing the array
```

```
void show_array(int * arr, int n)
{
   int i;
   for(i = 0; i<n; i++)
   {
      printf("%d",arr[i]);
   }
   printf("\n");

}
// Function to do sorting
void calculate_bubble_sort(int * arr, int n)
{
  int i,j,temp;
  int flag = 0;
  for(i = 0; i<n-1; i++)   // For number of pass
  {
     flag = 1;
     for(j = 0; j<n-1-i; j++)  // for comparison in each pass
     {
        if(arr[j] > arr[j+1])
        {
           temp = arr[j];
           arr[j] = arr[j+1];
           arr[j+1] = temp;
           flag = 0;
        }
     }
```

```
      if(flag == 1)
      {
         return;
      }
   }
}

void main()
{

   int arr[40],i;
   printf("Enter the number of elements:");
   scanf("%d",&n);

   printf("Enter the array elements:");
   for(i = 0; i<n; i++)
   {
      scanf("%d",&arr[i]);
   }

   show_array(arr,n); // print the array before sorting
   calculate_bubble_sort(arr,n); // call the function to sort the array
   show_array(arr,n); // print the array after sorting
}
```

Selection Sort

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

```
// For printing the array
void show_array(int * arr, int n)
{
   int i;
   for(i = 0; i<n; i++)
   {
      printf("%d",arr[i]);
   }
   printf("\n");

}
// Function to do sorting
void calculate_selection_sort(int * arr, int n)
{
   int i,j;
   int minindex,temp;
   for(i = 0; i<n-1; i++)
   {
      minindex = i;
      for(j = i+1; j<n-1; j++)
      {
         if(arr[j]<arr[minindex]
           {
              minindex = j;
           }
      }

      temp = arr[i];
      arr[i] = arr[minindex];
```

```
        arr[minindex] = temp;
    }
}

void main()
{

    int arr[40],i;
    printf("Enter the number of elements:");
    scanf("%d",&n);

    printf("Enter the array elements:");
    for(i = 0; i<n; i++)
    {
        scanf("%d",&arr[i]);
    }

    show_array(arr,n); // print the array before sorting
    calculate_selection_sort(arr,n); // call the function to sort the array
    show_array(arr,n); // print the array after sorting
}
```

Insertion Sort

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

void show_array(int *arr, int n);
void calculate_insertion_sort(int *arr, int n);
```

```
int main()
{
   int arr[40],i,n;

   printf("Enter the array size:");
   scanf("%d",&n);

   printf("Enter the array elements:");
   for(i=0;i<n;i++)
   {
      scanf("%d",&arr[i]);
   }

   show_array(arr,n);
   calculate_insertion_sort(arr,n);
   show_array(arr,n);

   return 0;
}

void show_array(int * arr, int n)
{
   int i;
   for(i = 0; i<n ; i++)
   {
      printf("%d",arr[i]);
   }
   printf("\n");
```

```
}

void calculate_insertion_sort(int *arr, int n)
{
   int i, j,store;
   for(i=1 ; i<n; i++)
   {
      store = arr[i];

      j = i-1;

      while(j>=0 && a[j]>store)
      {
         a[j+1] = a[j];
         j--;
      }
      a[j+1] = store;
   }
}
```

Quick Sort

```
#include <stdio.h>
#include <stdlib.h>
void show_array(int *arr, int n);
void calculate_quick_sort(int *arr, int na);
int partition(int *arr, int low, int high);
int main()
{
   int arr[40],n,i;
```

```
    printf("Enter the size of the array:");
    scanf("%d",&n);

    printf("Enter the array element:");
    for(i = 0; i<n; i++)
    {
        scanf("%d",&arr[i]);
    }

    show_array(arr,n);
    calculate_quick_sort(arr, 0, n-1);
    show_array(arr,n);
    return 0;
}

void show_array(int *arr, int n)
{
    int i;
    for(i = 0;i<n i++)
    {
        printf("%d",arr[i]);
    }
    printf("\n");
}

int partition(int *arr, int low, int high)
{
    int pivot,temp;
    pivot = arr[low];
```

```
int i = low+1;
int j = high;

do{
while(arr[i] <= pivot)
{
   i++;
}

while(arr[j] > pivot)
{
   j--;
}

if(i<j)
{
   temp = arr[i];
   arr[i] = arr[j];
   arr[j] = temp;
}
} while(i < j);

// swap arr[low] and arr[j]
temp = arr[low];
arr[low] = arr[j];
arr[j] = temp;

return j;
```

```
}

void calculate_quick_sort(int *arr, int low,int high)
{

   int partition_index; // it holds the index of pivot after calling the partition function
   if(low<high)
   {



   partition_index = partition(arr, low, high);

   calculate_quick_sort(arr, low, partition_index-1); // for sorting left array
   calculate_quick_sort(arr, partition_index+1, high); // for sorting right array

   }
}
```

Merge Sort

```
#include <stdio.h>
#include <stdlib.h>
void show_array(int *arr, int n);
void calculate_Merge_sort(int *arr, int low, int high);
void merge_procedure(int *arr, int low, int mid, int high);
int main()
{
   int arr[40],n,i;
   printf("Enter the size of the array:");
```

```
    scanf("%d",&n);

    printf("Enter the array element:");
    for(i = 0; i<n; i++)
    {
        scanf("%d",&arr[i]);
    }

    show_array(arr,n);
    calculate_Merge_sort(arr, 0, n-1);
    show_array(arr,n);
    return 0;
}

void show_array(int *arr, int n)
{
    int i;
    for(i = 0;i<n i++)
    {
        printf("%d",arr[i]);
    }
    printf("\n");
}

void merge_procedure(int *arr, int low, int mid, int high)
{
    int i,j,k, arr_new[200];
    i = low;
    j = mid+1;
```

```
k = low;

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

while(i<=mid )
{
   arr_new[k] = arr[i];
   k++;
   i++;

}

while(j< = high)
{
   arr_new[k] = arr[j];
```

```
        k++;
        j++;
    }

    for(i = low ; i<= high; i++)
    {
        arr[i] = arr_new[i];
    }




}

void calculate_Merge_sort(int *arr, int low, int high)
{
    int mid;
    if(low<high)
    {
        mid = (low+high)/2;
        calculate_Merge_sort(arr, low, mid);
        calculate_Merge_sort(arr, mid+1, high);
        merge_procedure(arr, low, mid, high);
    }
}
```