# Chapter 8

# Sorting

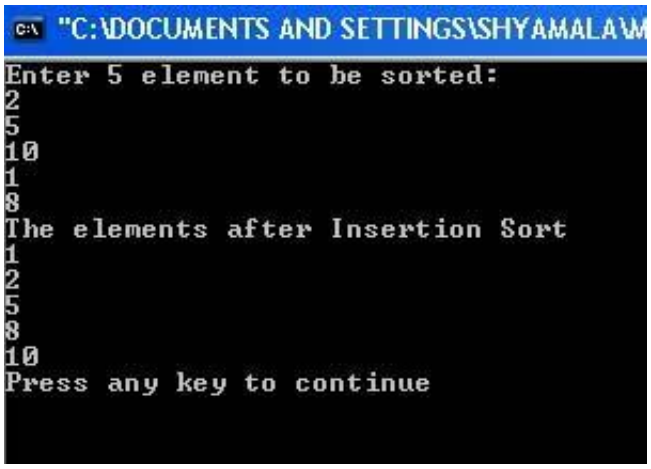
## 8.1 Introduction

- Sorting is the problem of obtaining and maintaining a file or records in some sort of order.
- If the record key is numeric then the order is likely to need to be in numerically ascending order.
- For example, employee work number, product code, international standard book code (ISBN).
- If the records key is alphabetic then the order is likely to be alphabetic e.g. Customer Name, Product Name
- If the record key is alphanumeric then the order is likely to be alphanumeric (and depend on the computer representation of characters) e.g. Car number plates.
- Normally, in practice, we would have to sort full records but in the classroom we will only consider record keys. (The identifier of a record).
- There are many algorithms for sorting, some simple but perhaps inefficient others complicated but efficient.
- We will look at a few algorithms for sorting.

## 8.2 Insertion Sort

- This is the way that most people sort playing cards, particularly clear if they are picking up one by one and putting them in place.
- The program for insertion sort is as follows:

```
#include <iostream.h>
#include <conio.h> #define MAX 5
void main() {     int array[MAX], i,
j, current;


      //Input 5 numbers to be sorted      cout<<"Enter
"<<MAX<<" element to be sorted:"<<endl;   for(i=0; i<MAX;
i++)
      {
            cin>>array[i];
      }

      //Insertion Sort
      for(i=1; i<MAX; i++)
      {
            current = array[i];
            j = i;
            while ( (j>0) && (current<array[j-1]) )
            {
                  array[j] = array[j-1];
                  j--;
                  array[j] = current;
            }
      }

      //Elements after being sorted       cout<<"The
elements after Insertion Sort"<<endl;     for(i=0;
i<MAX; i++)
      {
            cout<<array[i]<<endl;
      }
}
```



```
Enter 5 element to be sorted:
2
5
10
1
8
The elements after Insertion Sort
1
2
5
8
10
Press any key to continue
```

## 8.3 Selection Sort

- The approach is rather different to the insertion sort in the sense that you are "selection" an element to put into a particular position.
- A description might be: Starting from position 0, find the smallest and then exchange it with the element in position 0. Now, starting from position 1, find the smallest and exchange with position 1. Now from 2 etc etc
- A slightly more formal description might be:

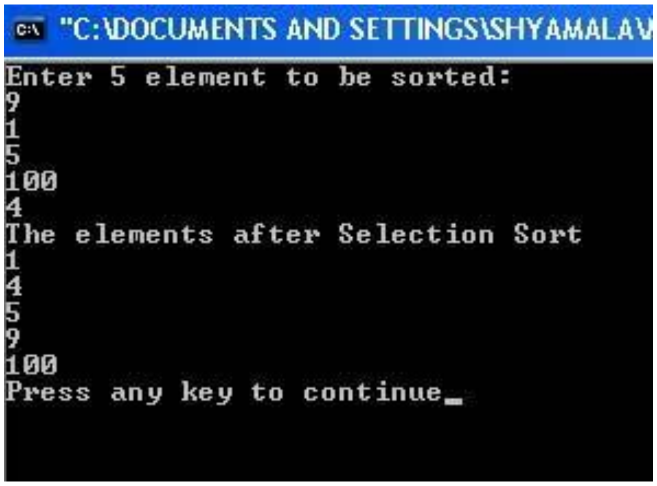
```
For position i = 0 to max-1 Find smallest from position i Exchange with position i
```

 Expending the above might give the following program:

```
#include <iostream.h>
#include <conio.h>
#define MAX 5 int
Min(int a[], int pos)
{      int i, m, index = pos;
       m = a[pos];
       for(i=pos+1; i<MAX;
i++)
```

```
            {
                        if(a[i]<m)
                  {
                  m = a[i];
            index = i;
                  }
            }
                  return index;
} void
main()
{     int array[MAX], i, k, temp;

      //Input 5 numbers to be sorted      cout<<"Enter
"<<MAX<<" element to be sorted:"<<endl;   for(i=0; i<MAX;
i++)
            {
                        cin>>array[i];
            }

      //Selection Sort
      for(i=0; i<MAX; i++)
            {
                        k = Min(array,i);

                        if(array[k]<array[i])
                  {
                  temp = array[k];
            array[k] = array[i];
      array[i] = temp;
                  }
            }

                        //Elements after being sorted
      cout<<"The elements after Selection Sort"<<endl;
      for(i=0; i<MAX; i++)
            {
                        cout<<array[i]<<endl;
            }
```

```
}
```



```
"C:\DOCUMENTS AND SETTINGS\SHYAMALAV
Enter 5 element to be sorted:
9
1
5
100
4
The elements after Selection Sort
1
4
5
9
100
Press any key to continue_
```

## 8.4 Bubble Sort

- Sometimes also called the ‘Mickey Mouse’ sort, this algorithm is very simple but inefficient.
- It works by comparing neighbours in the array and exchanging them if necessary to put smaller of the pair first. On each pass through the array another element ‘bubbles’ up into place.
- In its crudest form the algorithm is:

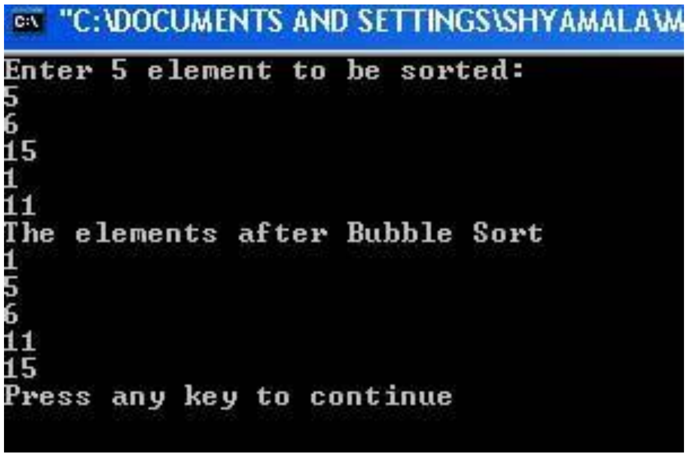
```
for (i=0; i<max; i++)
{
      for (j=0; j<max-1; j++)
      {
            if(array[j] > array[j+1])
            {
                  save = array[j];
      array[j] = array[j+1];
array[j+1] = save;
            }
      }
}
```

- (You might actually consider; if the largest is sinking to the bottom – if so, just turn it upside down!!!)

- An improvement can be made if you realise that on each pass another element has moved into its final position so line 3 course be replaced by '**`for(j=0; j<max-i-1; j++)`**'

- A further improvement might be made if you test to see if an element has already been sorted or otherwise. We need to realise that once the array is sorted no further exchanges is necessary. To ensure this just put in the exchange routine '**`sorted = false`**' initially and setting it to true once it has been exchanged.

```
#include <iostream.h>
#include <conio.h>
#define MAX 5
 void main()
{     int array[MAX], i, j, temp, sorted = 0;


      //Input 5 numbers to be sorted      cout<<"Enter
"<<MAX<<" element to be sorted:"<<endl;   for(i=0; i<MAX;
i++)
         {
                cin>>array[i];
         }

      //Bubble Sort
      i = 0;
         while ( (i<MAX) && (sorted==0) )
         {
                sorted = 1;
                for (j=0; j<MAX-i-1; j++)
                {
                      if(array[j] > array[j+1])
                      {
                           temp = array[j];
                           array[j] = array[j+1];
                           array[j+1] = temp;
                           sorted = 0;
                }    }
                i++;


         }



         //Elem
                ents after being sorted
         cout<<"The elements after Bubble Sort"<<endl;
         for(i=0; i<MAX; i++)
         {
                cout<<array[i]<<endl
         ;
         }
   }
```



```
"C:\DOCUMENTS AND SETTINGS\SHYAMALA\W
Enter 5 element to be sorted:
5
6
15
1
11
The elements after Bubble Sort
1
5
6
11
15
Press any key to continue
```

## 8.5 Merge Sort

- We are now looking at the ‘divide and conquer’ category of algorithms.
- Basically this just means splitting a problem up into simple problems until we get to a stage where we can solve the smaller problem.
- The basis here is that in order to sort an array we divide it into two halves and sort each half. How do we sort an half? – by dividing it into 2 halves etc.

- Once we get down to a manageable size we merge the 2 halves and return through the recursion merging gradually larger ad larger numbers

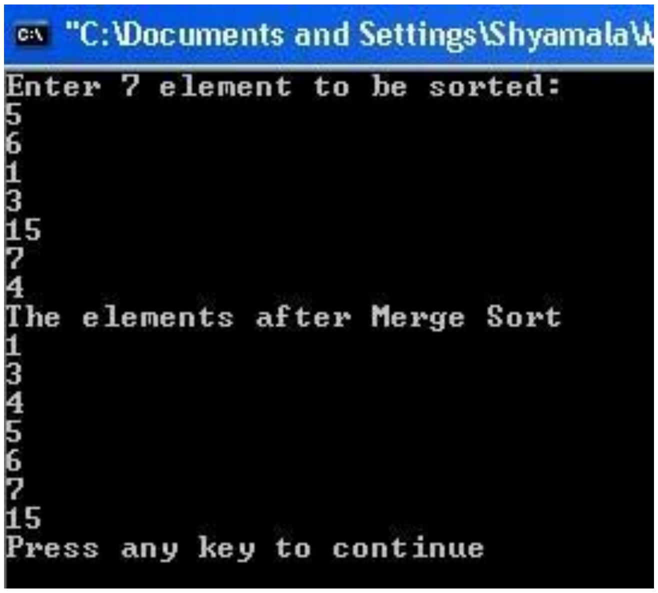
- Example:

```
8    7    6    5    4    3    2    1
8    7    6    5              4    3    2    1
8    7       6    5          4    3       2    1

 merge:
7    8
             5    6          3    4       1    2
5    6    7    8             1    2    3    4
1    2    3    4    5    6    7    8
```

- Here is the full program

```
#include <iostream.h>
#include <conio.h> #define MAX 7 int
array[MAX];       //global variables
 void Merge(int lpos, int rpos, int
rend)
{  int i, lend, numelements, tmppos, TmpArray[MAX];


      lend = rpos - 1;
tmppos = lpos;
     numelements = rend - lpos + 1;
      while( lpos<=lend && rpos <= rend )
      {
            if( array[lpos] <= array[rpos] )
      TmpArray[tmppos++] = array[lpos++];
            else
```

```
                    TmpArray[tmppos++] = array[rpos++];
         }
      while (lpos <= lend)
         {

                TmpArray[tmppos++] = array[lpos++];
         }
      while (rpos <= rend)
         {
                TmpArray[tmppos++] = array[rpos++];
         }
      for (i = 0 ; i < numelements; i++, rend--)
         {
                array[rend]=TmpArray[rend];
         }
}  void MergeSort (int left, int
right)
{
      int center;
      if(left < right)
         {
             center = (left + right)/2;
      MergeSort(left, center);
                MergeSort(center+1, right);
                Merge(left, center+1, right);
         }
} void main()
{
         int i;

         //Input 7 numbers to be sorted
      cout<<"Enter "<<MAX<<" element to be sorted:"<<endl;
      for(i=0; i<MAX; i++)
         {
                cin>>array[i];
         }

         //Merge Sort
         MergeSort(0, MAX-1);

         //Elements after being sorted cout<<"The
         elements after Merge Sort"<<endl; for(i=0;
         i<MAX; i++)
         {       cout<<array[i]<<endl;
         }



 }
```



"C:\Documents and Settings\Shyamala\W

```
Enter 7 element to be sorted:
5
6
1
3
15
7
4
The elements after Merge Sort
1
3
4
5
6
7
15
Press any key to continue
```

## 8.6 Quick Sort

- The basic algorithm was invented in 1960 by Hoarse. It is the very popular sorting algorithm and is perhaps the most widely used of all sorting algorithms.

- It has been extensively analysed and is well understood. Its advantages are that it is in-place (i.e. sorts within the array, does not use auxiliary memory), and is relatively simple to implement.

- It is a divide and conquer algorithm, to some extend similar to the merge sort except that the division of the table is not binary (i.e. does not simple divide the table into halves) but is based on the characteristics of the data. The division process is called "partitioning".

- The basic algorithm is:

```
void QuickSort(int left, int right, int a[])
{
      int pivotpos;
      if(left < right)
      {
            pivotpos = Partition(left, right, a);
            QuickSort(left, pivotpos-1, a);
            QuickSort(pivotpos+1, right, a);
      }
}
```

- The most important aspect of the algorithm is "how do we choose pivot?" As you can see from the above algorithm because we are recursively calling quicksort for two tables, left of pivot and right of pivot, (i.e. excluding the pivot element) this must mean that the pivot element and all elements to the right are greater than the pivot element.

- Consider the following:

```
A S O R T I N G E X A M P L E
```

- Start with the rightmost element (E) and scan to the left while the elements are greater (that E). Scan from the left while the elements are less (than E).

```
A S                 A M P L E
```

- Exchange the elements where the scan stops.

```
A A                 S M P L E
```

- Repeat this process:

```
A A O          E X S M P L E
```

- Exchange:

```
A A E          O X S M P L E
```

- Repeat:

A A E **R** T I N G O X S M P L **E**

 The scan cross at element 4 (the R) so now exchange the rightmost element with the 4th.

```
A A E E T I N G O X S M P L R
```

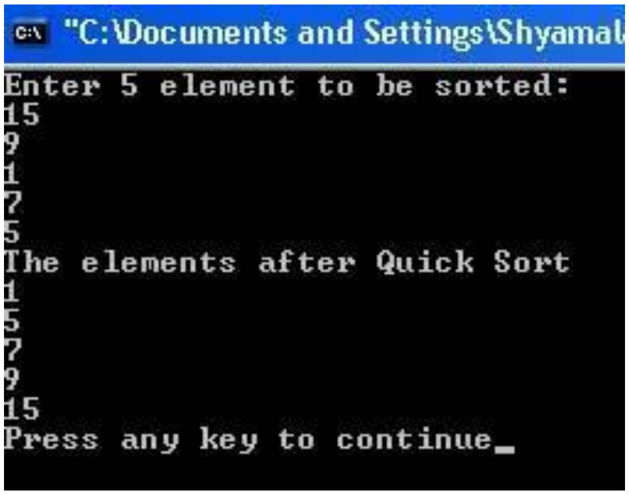
- Now the 4th element is in its final position, all elements to the left are not greater than the 4th and all elements to the right are not less than the 4th.
- So the partitioning process has found **pivotpos** and made some changes in the table.
- Now what need to be done are a quicksort on element 1 to 3 and a quicksort on element 5 to 15.
- The Partition function is as follows:

```
void swap(int x, int y, int a[])
{
      int temp;
      temp = a[x];
      a[x] = a[y];
      a[y] = temp;
}  int Partition(int left, int right, int
a[])
{
      char pivot;        int i,
lastsmall, pivotpos;

      Swap(left, (left+right)/2,
a);    pivot = a[left];
      pivotpos = left;
      for(i=left+1; i<=right; i++)
      {
            if(a[i] < pivot)
            {
                  //move large value right, small to left
                  Swap(++pivotpos, i, a);
            }
      Swap(left, pivotpos, a);
      return pivotpos;
}
```

- The full program for QuickSort is as follows:

```
#include <iostream.h>
#include <conio.h>
#define MAX 5
void Swap(int x, int y, int a[])
{
        int temp;
        temp = a[x];
        a[x] = a[y];
        a[y] = temp;
}
```

```
int Partition(int left, int right, int a[])
{
      char pivot;       int i,
lastsmall, pivotpos;


         Swap(left, (left+right)/2, a);


      pivot = a[left];
      pivotpos = left;
      for(i=left+1; i<=right; i++)
         {
               if( a[i] < pivot )
                     //move large entry to right and small to left
                     Swap(++pivotpos, i, a);
         }
         Swap(left, pivotpos, a);


         return pivotpos;
}  void QuickSort(int left, int right, int
a[])
{
      int pivotpos;
      if( left<right )
         {
               pivotpos = Partition(left, right, a);
               QuickSort(left, pivotpos-1, a);
               QuickSort(pivotpos+1, right, a);
         }
} void main()
{     int array[MAX], i;


      //Input 5 numbers to be sorted      cout<<"Enter
"<<MAX<<" element to be sorted:"<<endl;   for(i=0; i<MAX;
i++)
         {
               cin>>array[i];
         }

         //Quick Sort
         QuickSort(0, MAX-1, array);

         //Elements after being sorted cout<<"The
         elements after Quick Sort"<<endl; for(i=0;
         i<MAX; i++)
         {       cout<<array[i]<<endl;
         }



 }
```



```
"C:\Documents and Settings\Shyamal
Enter 5 element to be sorted:
15
9
1
7
5
The elements after Quick Sort
1
5
7
9
15
Press any key to continue_
```