Sorting

- Sorting and searching are among the most common programming processes.

- We want to keep information in a sensible order.
- – alphabetical order
- – ascending/descending order
- – order according to names, ids, years, departments etc.

- The aim of sorting algorithms is to put unordered information in an ordered form.

- There are many sorting algorithms, such as:
  - – Selection Sort
  - – Bubble Sort
  - – Insertion Sort
  - – Merge Sort
  - – Quick Sort

- The first three are the foundations for faster and more efficient algorithms.

Selection Sort

⇨ The list is divided into two sublists, *sorted* and *unsorted*, which are divided by an imaginary wall.

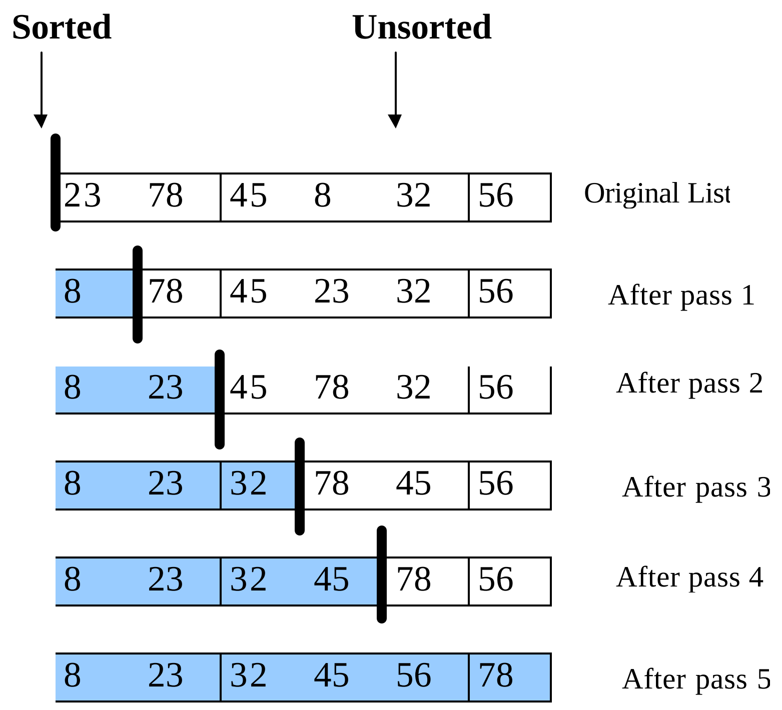
⇨ We find the smallest element from the unsorted sublist and swap it with the element at the beginning of the unsorted data.

⇨ After each selection and swapping, the imaginary wall between the two sublists move one element ahead, increasing the number of sorted elements and decreasing the number of unsorted ones.

⇨ Each time we move one element from the unsorted sublist to the sorted sublist, we say that we have completed a sort pass.

⇨ A list of *n* elements requires *n-1* passes to completely rearrange the data.

Selection Sort Example



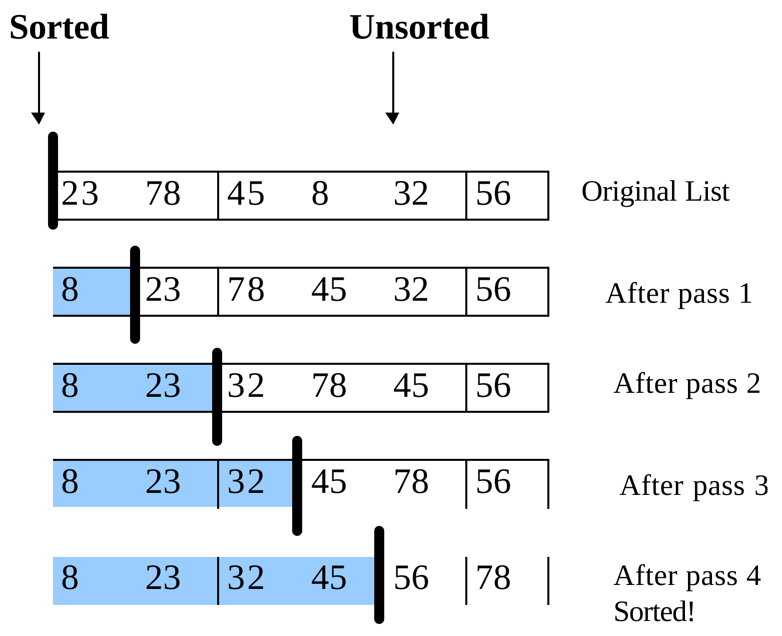
Selection Sort Algorithm

```
/* Sorts by selecting smallest element in unsorted
   portion of array and exchanging it with element
   at the beginning of the unsorted list.
   Pre  list must contain at least one item
        last contains index to last element in list
   Post list is rearranged smallest to largest
*/
void selectionSort(int list[], int last)
{
   int current, walker, smallest, tempData;

   for (current = 0; current <= last; current ++){
      smallest = current;
      for (walker=current+1; walker <= last; walker++)
         if(list[walker] < list[smallest])
            smallest = walker;

      // Smallest selected; swap with current element
      tempData  = list[current];
      list[current] = list[smallest];
      list[smallest] = tempData;
   }
}
```

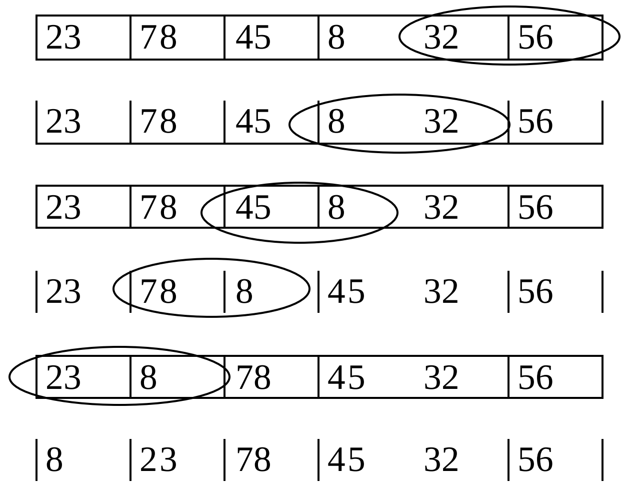
Bubble Sort

- ⇨ The list is divided into two sublists: sorted and unsorted.
- ⇨ The smallest element is bubbled from the unsorted list and moved to the sorted sublist.
- ⇨ After that, the wall moves one element ahead, increasing the number of sorted elements and decreasing the number of unsorted ones.
- ⇨ Each time an element moves from the unsorted part to the sorted part one sort pass is completed.
- ⇨ Given a list of n elements, bubble sort requires up to n-1 passes to sort the data.
- ⇨ Bubble sort was originally written to "bubble up" the highest element in the list. From an efficiency point of view it makes no difference whether the high element is bubbled or the low element is bubbled.

Bubble Sort Example



Trace of 1st pass of Bubble Sort Example:



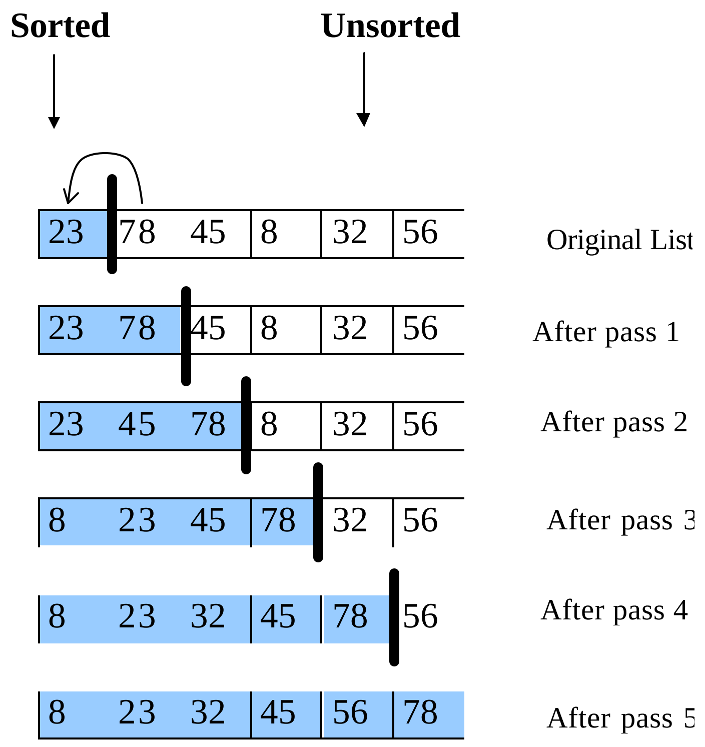
Bubble Sort Algorithm

```
/* Sorts list using bubble sort. Adjacent elements
   are compared and exchanged until list is
   completely ordered.
   Pre  list must contain at least one item
        last contains index to last element in list
   Post list is rearranged smallest to largest
*/
void bubbleSort(int list[], int last)
{
   int current, walker, temp;

   for (current = 0; current <= last; current++){
      for (walker=last; walker > current; walker--)
         if(list[walker] < list[walker - 1]){
            temp = list[walker];
            list[walker] = list[walker – 1];
            list[walker-1] = temp;
         }
   }
   return;
}
```

Insertion Sort

- ⇨Most common sorting technique used by card players.
- ⇨Again, the list is divided into two parts: sorted and unsorted.
- ⇨In each pass, the first element of the unsorted part is picked up, transferred to the sorted sublist, and inserted at the appropriate place.
- ⇨A list of *n* elements will take at most *n-1* passes to sort the data.

Insertion Sort Example



Insertion Sort Algorithm

```
/* With each pass, first element in unsorted
   sublist is inserted into sorted sublist.
   Pre  list must contain at least one item
        last contains index to last element in list
   Post list has been rearranged
*/

void insertionSort(int list[], int last)
{
   int current, located, temp, walker;

   for (current = 1; current <= last; current++){
      located = 0;
      temp = list[current];
      for (walker=current-1; walker >= 0 && !located;)
         if(temp < list[walker]){
            list[walker + 1]= list[walker];
            walker--;
         }
         else
            located = 1;
      list[walker + 1] = temp;
   }
   return;
}
```