Selection Sort Algorithm

☞ Order an array in descending order

- find the largest element in array [0 .. size-1] and exchange it with element 0;
- find the largest element in array [1 .. size-1] and exchange it with element 0;

  • • •
- find the largest element in array [size-2 .. size-1] and exchange it with element size-2;

```
1. for Index = 0 to size-2 do
    2. Find the position of the largest
       element in subarray [Index..size-1].
    3. if Index is not the position of the
       largest element then
       4. Switch the largest element with
          the one at position Index.
```

Refinement for step 2:

```
 2.1 Initialize the position of the largest
       element to Index.
 2.2 for Next = Index+1 to size-2 do
     2.3 if the element at Next > the largest
            element so far then
         2.4 set the position of the largest
              element found so far to Next.
```

Selection Sort

18	35	22	97	84	55	61	10	47	unsorted array
97	35	22	18	84	55	61	10	47	
97	84	22	18	35	55	61	10	47	
97	84	61	18	35	55	22	10	47	
97	84	61	55	35	18	22	10	47	
97	84	61	55	47	18	22	10	35	
97	84	61	55	47	35	22	10	18	
97	84	61	55	47	35	22	10	18	
97	84	61	55	47	35	22	18	10	sorted array

```
// Sort an array of integers in descending order
#include <iostream.h>
#include <iomanip.h>

void selectsort(int data[], // input/output: array
                int size)   // input: array size
{   int smallest_idx, // smallest element position
        tmp;          // for exchange

    for (int idx=0; idx<size-1; idx++) {
        // for each subarray [idx..size-1]
        // find index of smallest element
        smallest_idx = idx;
        for (int subidx=idx+1; subidx<size; subidx++)
            if (data[subidx] > data[smallest_idx])
                smallest_idx = subidx;

        if (smallest_idx > idx) {
            // exchange with smallest element
            tmp = data[idx];
            data[idx] = data[smallest_idx];
            data[smallest_idx] = tmp;
        }
    }
} // selectsort

void main() {
    int A[10] = { 5,2,4,6,9,2,25,5,18,3 };

    for (int i=0; i<10; i++)
        cout << setw(5) << A[i];

    selectsort(A, 10);

    cout << endl;
    for (int i=0; i<10; i++)
        cout << setw(5) << A[i];

    return;
}
```

Insertion Sort Algorithm

The array is divided into two parts. The left part of it is sorted while the right part is not.

Initially, the left part consists of one element.

For each ‘first’ component of the unsorted part of the array

```
1.temporarily remove it;

2.find its proper place in the sorted part
  of the array;

3.slide smaller values one component to the
  right;

4.insert the ‘first’ component into its new
  position;
```

Insertion Sort

```
// Sort an array of integers in descending order
void insertsort(int data[], //input/output: array
                int size)  //input: array size

{
   int current_item, // item under consideration
       insert_point; // insertion point

   for (int current=1; current<size; current++) {

      current_item = data[current];
      // find insertion point
      insert_point = current;
      while ((insert_point != 0) &&
            (data[insert_point-1]>current_item)){
         // shift item right to make room
         data[insert_point]=data[insert_point-1];
         insert_point--;
      }

      // insert item
      data[insert_point] = current_item;
   }
} // insertsort
```

Insertion Sort

18	35	22	97	84	55	61	10	47	unsorted array
35	18	22	97	84	55	61	10	47	
35	22	18	97	84	55	61	10	47	
97	35	22	18	84	55	61	10	47	
97	84	35	22	18	55	61	10	47	
97	84	55	35	22	18	61	10	47	
97	84	61	55	35	22	18	10	47	
97	84	61	55	35	22	18	10	47	
97	84	61	55	47	35	22	18	10	sorted array

Bubble Sort Algorithm

The array is divided into two parts. The right part of the array is sorted while the left part is not.

for every ‘last’ component in the unsorted part of array

    for every component from the first to the ‘last’

        - compare that component to with the neighboring component to the right;

        - exchange them if necessary;

Bubble Sort:

```
// Sort an array of integers in descending order
void bubble(int data[], // input/output: array
            int size)   // input: array size

{
   int temp;   // for swap

   for (int rightmost=size-1; rightmost>0;
            rightmost--)

      // bubble up the largest item
      for (int current=0; current<rightmost;
               current++)
         if (data[current] > data[current+1]) {
            // swap the two array elements
            temp = data[current];
            data[current] = data[current+1];
            data[current+1] = temp;
         }


} // bubble
```

Bubble Sort

18	35	22	97	84	55	61	10	47	unsorted array
35	18	22	97	84	55	61	10	47	
35	22	18	97	84	55	61	10	47	
35	22	97	84	55	61	18	47	10	
35	97	84	55	61	22	47	18	10	
97	84	55	61	35	47	22	18	10	
97	84	61	55	47	35	22	18	10	sorted array

Advantages of each Sorting Algorithm

- ☞ The selection sort is simple because it does not require any wholesale moves; only two values are exchanged on any trip through the array.

- ☞ The insertion sort minimizes unnecessary travel through the array. If the values are sorted to begin with, a single trip through the array establishes that fact. In contrast, selection sort requires the same number of trips no matter how well-organized the array is.

- ☞ The bubble sort is worse than selection sort for a jumbled array and will require many more component exchanges. Unlike the insertion sort, it's utterly incapable of minimizing travel through the array. We use it because it is easy to write correctly!