Sorting Algorithms in FORTRAN

Dr. Ugur GUVEN

Importance of Sorting

- When you are doing scientific programming in FORTRAN, one of the most important things is to be able to sort the numerical data so that the data can go from raw form to a more usable one.
- Usually, we will need to sort the data to either ascending order or descending order to see the magnitudes of the results.
- It may also be required to sort only portions of the data so selective sorting algorithms will also need to be applied.

Bubble Sort

- The most easier type of sorting in FORTRAN programming is the Bubble Sort. This is especially an efficient type of programming when there are only limited number of data as it can be programmed very easily and does not require much computational power.
- In a bubble sort the first two elements are examined and swapped if the second is greater than the first (assuming sort from high to low value). Next the second and third elements are examined and swapped if the third is greater than the second. This proceeds until each consecutive pair has been considered and if necessary, swapped. The end result is that the smallest value in the array has "bubbled" to the last element in the array.
- These passes continue until either no swaps occur during an entire pass through the array.

Bubble Sort

- Pass 1

Base array : 1 5 4 2 3
Swap : [5 1] 4 2 3
Swap : 5 [4 1] 2 3
Swap : 5 4 [2 1] 3
Swap : 5 4 2 [3 1]

Pass 2

No Swap : [5 4] 2 3 1
No Swap : 5 [4 2] 3 1
Swap : 5 4 [3 2] 1

Pass 3

No Swap : [5 4] 3 2 1
No Swap : 5 [4 3] 2 1

Pass is done because we know the last two elements are in the right positions. **No swaps in pass 3, sort completed**

Selection Sort

- The Selection Sort is probably the simplest available, and worth remembering for those emergencies when you must patch something together quickly. For a basic example, assume that we have an array "x" containing "n" elements, that we want to sort from high to low by value. Find the element with the maximum value (maxloc function is a good choice for this job), and swap the contents of this element with those of the first element.
- Now that the first element contains the largest value in the array, scan the remaining elements (2 through "n") for the element with the largest value. Switch the contents of that element with the contents of the second element in the array. Move on to repeat this sequence of search and swap operations, until you are just checking and possibly swapping just the last two elements of the array.
- To sort in increasing order, use the same process with a minimum check

Selection Sort

- Here is an example

Beginning Array : 2 3 1 7 9 5
First Pass : [9] 3 1 7 [2] 5
Second Pass : 9 [7] 1 [3] 2 5
Third Pass : 9 7 [5] 3 2 [1]

Two more checks on maximum value follow, but no more swaps result.

Insertion Sort

- The first step in an insertion sort is to put the first two elements in proper order. For a high to low sort, this involves swapping the first two if the second element has a higher value.
- Next, the third element is checked. If it is less than the second element, it is left in place. If not, the elements before it are checked for the first one that has a lower value.
- The value of the third element is inserted before this lower valued one and values of elements are shifted up in the array to produce an expanded ordered list.
- This pattern of checking for order, and shifting values as needed continues until all elements in the array are checked.

Importance of Sorting

- Here is a simple example of an insertion sort. In each step square brackets surround the portion of the array that is shifted, and parentheses surround the value that is inserted.

Base array : 1 3 4 2 5
Shift : (3)[1] 4 2 5
Shift : (4)[3 1] 2 5
Shift : 4 3 (2)[1] 5
Shift : (5)[4 2 3 1]

- In each step square brackets surround the portion of the array that is shifted, and parentheses surround the value that is inserted.

Circle Sort

- Sort an array of integers (of any convenient size) into ascending order using Circlesort.
- In short, compare the first element to the last element, then the second element to the second last element, etc.
- Then split the array in two and recurse until there is only one single element in the array, like this:

```
Before: 6 7 8 9 2 5 3 4 1
After:  1 4 3 5 2 9 8 7 6
```

- Repeat this procedure until quiescence (i.e. until there are no swaps).

Sorting Homework

- Please create a file of array of numbers with at least 30 numbers in unsorted order.
- Sort the array in ascending order using Bubble Sort, Selection Method, Insertion Method and Circle Method
- Count the number of steps required for sorting
- Compare the number of steps of each method
- Change the array by putting the last number as the highest and then the lowest
- Try the methods above and compare the steps when the last number is highest and lowest. Do you see a pattern?