3 An inefficient sorting routine in a short program

Here is a handy method of sorting an array, though inefficient.

The most important part of the code is the following routine `ooo()` (which stands for 'out of order').

```
int ooo ( int n, SORTABLE a[] )
{
  int found = 0;
  for (int i=0; i<n-1 && ! found; ++i)
  {
    if ( threeway ( a[i], a[i+1] ) > 0 )
    {
      SORTABLE x = a[i]; a[i] = a[i+1]; a[i+1] = x;
      found = 1;
    }
  }
  return found;
}
```

Full program.

```
#include <stdio.h>
typedef int SORTABLE;

int threeway ( SORTABLE x, SORTABLE y )
{ return x-y; }

int ooo ( int n, SORTABLE a[] ) ... // as above

void sort ( int n, SORTABLE a[] )
{ while ( ooo ( n, a ) ) {} }

int main()
{ int a[9] = {3,1,4,1,5,9,2,6,5};
  int n = 9;

  sort (n,a);

  for (int i=0; i<n; ++i) printf(" %d", a[i]);
  printf("\n");

  return 0;
}

gcc -std=gnu99 sort.c
```

```
a.out
 1 1 2 3 4 5 5 6 9
```

3.1 Remarks

- This works, though it is not obvious that it works. The idea is that if the routine `ooo()` finds two adjacent items which are out of order, it swaps them and returns 1; if it finds none then the array is sorted and it returns 0.
- It is not obvious what its runtime is; it is not even obvious that it always halts. It is certainly inefficient, but it is a short and simple routine.