Sorting, Ranking, Indexing, Selecting

Sorting, Ranking, Indexing, Selecting

The definitions of sorting ranking and indexing should be clear from the following example:

Original	Indexed	Ranked	Sorted
3	6	4	0
1	2	2	1
2	3	3	2
6	1	6	3
4	5	5	4
0	4	1	6

If we can index and sort, we can rank

İf v is the original array,

vi is the indexed array,

vr is the ranked array, and

vs is the sorted array, then indexing vi gives vr

```
[vs, vi] = sort(v);
[x, vr] = sort(vi);
```

Note that vs = v(vi) and vs(vr) = v. Also, x contains the integers 1,2, . . . (up to the length of x).

Writing your own SORT function

```
function [B] = mysort(A,x)
   B = [];
   for i =1:length(A)
      minElem = min(A);
      numOcc = length(find(A == minElem));
      for j = 1:numOcc
         if x == 0
               B = [B minElem];
         elseif x == 1
               B = [minElem B];
         end
       end
      A(find(A == minElem)) = [];
   end
end
```

Selection

Selection is the problem of finding the mth largest element in v.

For instance, to find the median we perform selection
with m = [n/2] (if n is odd and we index from zero).

Selection from an array that has already been sorted is trivial.

There are efficient ways to do selection that do not require complete sorting.

Finding extreme values

Finding extreme values (i.e. the maximum and the minimum) is a special case of selection.

Extreme values can be located by cycling once through the array values. Thus the complexity is linear in the array length, and the dominant operation is the arithmetic comparison (< or >).

This is an in-place calculation. In Octave these are built-in:

```
## a is the maximum value and b is the index of a in v.
[a,b] = max(v);
## a is the minimum value and b is the index of a in v.
[a,b] = min(v);
```

What If Matlab/Octave Didn't Have built-in max and min functions?

```
function [a,b] = Max(v)
  a = v(1);
  b = 1;
  for i=2:length(v)
    if (v(i) > a)
      a = v(i);
      b = i;
    endif
  endfor
endfunction
```

```
function [a,b] = Min(v)
  a = v(1);
  b = 1;
  for i=2:length(v)
    if (v(i) < a)
      a = v(i);
      b = i;
    endif
  endfor
endfunction
```

Straight Insertion

Straight insertion is a simple but slow in-place sorting algorithm.

It should only be used with very small lists (a rule of thumb is to use it if there are at most 8 values).

The basic idea is to move from left to right, at each step locating the smallest value to the right of the current location, and then swap with the value in the current location.

Straight Insertion

```
3 2 1 8 4 5 7 Position 1 (swap 3&1)
1 2 3 8 4 5 7 Position 2 (no swap)
1 2 3 8 4 5 7 Position 3 (no swap)
1 2 3 8 4 5 7 Position 4 (swap 8&4)
1 2 3 4 8 5 7 Position 5 (swap 8&5)
1 2 3 4 5 8 7 Position 6 (swap 8&7)
1 2 3 4 5 7 8 Sorted Array
```