8 Sorting

Functions:

piksrt: sort an array by straight insertion
piksr2: sort two arrays by straight insertion
shell: sort an array by Shell's method
sort1: sort an array by the heapsort method
sort2: sort two arrays the heapsort method
indexx: sort and construct an index for an array
sort3: sort, use and index to sort 3 or more arrays
rank: sort, construct a rank table for an array
qcksrt: sort an array by the quicksort method
eclass: determine equivalence classes
eclazz: determine equivalence classes

---

piksrt

Description:

This function sorts a list of floating point numbers by direct insertion in the underlying array.

piksrt(arr) $\longrightarrow$ arr

Where:

arr on input is a list of real numbers and on output the list sorted into ascending numerical order.

Example:

A list is sorted into ascending order using `piksrt`.

```
-> piksrt([4,1,3,2,6,5])?
[1.00, 2.00, 3.00, 4.00, 5.00, 6.00]
-> piksrt([6,5,5,4,2,2])?
[2.00, 2.00, 4.00, 5.00, 5.00, 6.00]
```

---

piksr2

Description:

This function sorts a list of floating point numbers by direct insertion in the underlying array and makes the corresponding changes in another list.

piksr2(arr, brr) $\longrightarrow$ [arr, brr]

Where:

arr on input is a list of real numbers and on output the list sorted into increasing numerical order,
brr on input is a list of real numbers and on output the list rearranged in the same way that *arr* has been rearranged.

Example:

A list is sorted into ascending order and the corresponding changes made to a second list by `piksrt2`.

```
-> piksr2([3,1,4,2,6,5],[1,2,3,4,5,6])?
[[1.00, 2.00, 3.00, 4.00, 5.00, 6.00], [2.00, 4.00, 1.00, 3.00, 6.00, 5.00]]
-> piksr2(last(%),first(%))?
[[1.00, 2.00, 3.00, 4.00, 5.00, 6.00], [3.00, 1.00, 4.00, 2.00, 6.00, 5.00]]
```

---

shell

Description:

This funtion sorts a list of real numbers using Shell's method.

shell(arr) $\longrightarrow$ arr

Where:

arr on input is a list of real numbers and on output the list sorted into increasing numerical order by the Shell-Mezgar algorithm.

Pre-variables:

shell_aln2i [1.442695d0] a floating point number,
shell_tiny [1.0d-5] a floating point number.

Example:

A list is sorted into ascending order using `shell`.

```
-> shell([10,3,6,2,56,9])?
[2.00, 3.00, 6.00, 9.00, 10.00, 56.00]
```

---

sort1

Description:

This function sorts a list of floating point numbers using the heapsort algorithm. Note that the corresponding Numerical Recipes function is called *sort*.

sort1(ra) ⟶ ra

Where:

ra on input is a list of numbers to be sorted and on output the list sorted into increasing numerical order.

Example:

A list is sorted into ascending order using `sort1`.

```
-> sort1([10,3,6,2,56,9])?
[2.00, 3.00, 6.00, 9.00, 10.00, 56.00]
```

---

sort2

Description:

This function sorts a list of floating point numbers using the heapsort algorithm and makes the corresponding changes in a second list.

sort2(ra, rb) ⟶ [ra, rb]

Where:

ra on input is a list of real numbers and on output the list sorted into increasing numerical order,
rb on input is a list of real numbers and on output the list sorted using the same transformations as applied to *ra*.

Example:

A list is sorted into ascending order and the corresponding changes made to a second list by `sort2`.

```
-> sort2([3,1,4,2,6,5],[1,2,3,4,5,6])?
[[1.00, 2.00, 3.00, 4.00, 5.00, 6.00], [2.00, 4.00, 1.00, 3.00, 6.00, 5.00]]
-> sort2(last(%),first(%))?
[[1.00, 2.00, 3.00, 4.00, 5.00, 6.00], [3.00, 1.00, 4.00, 2.00, 6.00, 5.00]]
```

---

indexx

Description:

This function sorts a list of real numbers by making an index for the list.

indexx(arrin) ⟶ indx

Where:

arrin is a list of real numbers,
indx is a list of integers such that the set $arrin(indx(j))$ is increasing with j.

Example:

An list of rankings is computed for a list of numbers by `indexx`.

```
-> indexx([7,4,8,34])?
[2, 1, 3, 4]
```

---

sort3

Description:

This function sorts three lists by making an index from the first and using it to sort the remaining two lists.

sort3(ra, rb, rc) ⟶ [ra, rb, rc]

Where:

ra on input is a list of real numbers and on output the list sorted into increasing numerical order,
rb is a real list which is sorted using the same transformations as applied to ra,
rc is a real list which is sorted using the same transformations as applied to ra.

Example:

A list of numbers is sorted by `sort3` and the corresponding changes is made to two other lists.

```
-> sort3([3,7,1,89,32],[1,2,3,4,5],[5,4,3,2,1])?
[[1.00, 3.00, 7.00, 32.00, 89.00], [3.00, 1.00, 2.00, 5.00, 4.00],
[3.00, 5.00, 4.00, 1.00, 2.00]]
```

---

rank

Description:

This function computes a list of ranks for a list of real numbers.

$$\mathbf{rank(indx)} \longrightarrow \mathbf{irank}$$

Where:

indx is a list of integers as output by *indexx*,
irank is a list of integers representing the ranks.

Example:

The function `rank` is used to compute a list of ranks.

```
-> indexx([6,1,4,2,5,3])?
[2, 4, 6, 3, 5, 1]
-> rank(%)?
[2, 4, 6, 3, 5, 1]
```

qcksrt

Description:

This function sorts a list of real numbers using the quicksort algorithm.

$$\mathbf{qcksrt(arr)} \longrightarrow \mathbf{arr}$$

Where:

arr on input is a list of real numbers and on output the list sorted into increasing numeric order.

Pre-variables:

qcksrt_m [7] an integer being the size of the subarrays sorted by straight insertion,
qcksrt_nstack [50] an integer being the size of the auxiliary storage,
qcksrt_fm [7875.0d0] a floating point number being a constant used by the random number generating statements,
qcksrt_fa [211.0d0] a floating point number being a constant used by the random number generating statements,
qcksrt_fc [1663.0d0] a floating point number being a constant used by the random number generating statements,
qcksrt_fmi [1.2698413d-4] a floating point number being a constant used by the random number generating statements.

Example:

Two lists are sorted into ascending order using `qcksrt`.

```
-> qcksrt([10,2,6,4,8,2,6])?
[2.00, 2.00, 4.00, 6.00, 6.00, 8.00, 10.00]
-> qcksrt([45,6721,2,87,93])?
[2.00, 45.00, 87.00, 93.00, 6721.00]
```

eclass

Description:

This function determines a set of equivalence classes of integers $S = \{1, ..., n\}$.

eclass(n, lista, listb) $\longrightarrow$ **nf**

Where:

lista is a list of integers of length $m \leq n$ with elements being from the set S,
listb is a list of integers of length m with elements from the set S such that in S $lista_j$ is equivalent to $listb_j$ for $1 \leq j \leq m$,
nf is a list giving in order the index number of the equivalence class of each of the n elements.

Example:

A list of 11 elements is sorted into 4 equivalence classes.

```
-> lista : [1,1,5,2,6,2,7,11,3,4,12]?
[1, 1, 5, 2, 6, 2, 7, 11, 3, 4, 12]
-> listb : [5,9,13,6,10,14,3,7,15,8,4]?
[5, 9, 13, 6, 10, 14, 3, 7, 15, 8, 4]
-> eclass(15,lista,listb)?
[13, 14, 15, 8, 13, 14, 15, 8, 13, 14, 15, 8, 13, 14, 15]
```

eclazz

Description:

This function determines a set of equivalence classes described by a function using a method of Eardley.

eclazz(n, equiv) ⟶ nf

Where:

n is an integer giving the range 1 through n of numbers under consideration,
equiv is a Senac function of two integer variables returning true when the numbers are equivalent and false otherwise,
nf is a list giving the index number of the equivalence class of each of the n elements.

Example:

A list of length 20 is sorted into 9 equivalence classes.

```
-> f(x,y) := if x/y = 2 then true else false;
-> Any pair, p & q, from an equiv. class of f can be written p = q*2^n,
   for some integer n                #
-> listn(1,20)?
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
-> eclazz(20,f)?
[16, 16, 12, 16, 20, 12, 14, 16, 18, 20, 11, 12, 13, 14, 15, 16, 17, 18,
19, 20]
```