OS20-Systemcalls

Nico Max

COLLABORATORS			
	TITLE : OS20-Systemcalls		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Nico Max	April 16, 2022	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

OS20-Systemcalls

1.1 VisualSort User Manual

-- VisualSort V1.15 --

User Manual

Have fun fith it!

1.2 VisualSort - Program Requirements

VisualSort needs at least Kick 2.0. I only can strongly advice

Kick 1.2/1.3 owners: UPGRADE!

It wouldn't be bad having reqtools.library in your LIBS: directory.

With it the program gives you the possibility to change the screen-

mode into a desired one and some other fancy things :-)

1.3 VisualSort - History

1.0 -: first version

1.01 -: keyboard control added

-: little bug fixed while drawing GUI

1.13 -: line drawing method added

-: bug in InsertSort and ShellSort fixed

-: Arexx port added

-: some functions added

-: several internal enhancements

1.15 -: Selectsort modified to work faster for most statistical cases

-: freehand added

-: GUI localized

1.4 VisualSort - Contacting the author

If you want to write me some postcards or if you want to send anything

here's my address:

Nico Max

Gerüstbauerring 15

18109 Rostock

Germany

or email to: max@informatik.uni-rostock.de

1.5 VisualSort - Greetings

Oil Hog (cool :-) TG, Rene,

Bundy (Die Weisheit jagt dich, aber du bist schneller!),

steeple, Silke F., Silke G. and a whole lot more...

1.6 Known Bugs

I completely don't know, why MergeSort works so ugly if line mode

are set. Please let me know if somebody has an idea about that.

1.7 VisualSort - The implemented algorithms

short briefing about the algorithms:

BubbleSort ShakeSort InsertSort SelectSort ShellSort

MergeSort QuickSort HeapSort CountSort HybridSort

BubbleSort

A very simple one. BubbleSort starts at the beginning of the array. It

looks at the first pair and checks the relation. If its the right one

the next pair is taken. If not both elements get swapped and BubbleSort
takes the next pair. This way it will do until the end of the array is
reached. Now BubbleSort asks itself if a swapping has been done so far.
If no the array is well sorted. If yes the flag gets changed to false
and all starts from the beginning.

ShakeSort

An improved version. ShakeSort starts "left" and goes the same way like
BubbleSort through the array until the end. But now ShakeSort goes from
the end through the array until the beginning. So far so good. But
ShakeSort memorizes both borders since the array is well sorted. So the
way beetwen the borders gets smaller and smaller.

InsertSort

The algorithm works like the following example:

the unsorted array

```
3 1 6 7 2
| | | | |
3 <----------- | | | | "3" gets inserted into the sorted area
1 3 <------------- | | | "1" falls down" until the final position
1 3 6 <--------------- | | ..and so on...
1 3 6 7 <----------------- |
1 2 3 6 7 <-------------------
```

sorted

InsertSort "inserts" each element into the sorted area with checking
the relation.

SelectSort

Very simple too. Lets have an example:

the array: 5 1 9 3 2; starting at element "5"

1. step: find the smallest element from position -> 1 (wow :-))
2. step: swap element at position with the smallest -> 1 5 9 3 2
3. step: increment position and goto 1. step

This is done until position reaches the end. From the beginning the
array changed like the following way:

5 1 9 3 2, minimum -> 1, swap 5 and 1

1 5 9 3 2, minimum -> 2; swap 5 and 2

1 2 9 3 5, minimum -> 3; swap 9 and 3

1 2 3 9 5, minimum -> 5; swap 9 and 5

1 2 3 5 9 -> sorted.

ShellSort

Like InsertSort but with variable steps. This means that ShellSort

doesn’t takes the closest element from another while comparing and
inserting but one depending from an internal variable witch gets
smaller and smaller in a special way. Have a look at the source.
MergeSort
MergeSort belongs to a group called: divide and conquer. MergeSort
takes the array and splits it. In a rekursive way MergeSort goes first
into the "left" part of the splitted area and does the same until the
area are one element large. From now MergeSort merges the left and the
right array together. This can be done very simple because you have the
met condition that the left and the right array are already sorted.
Lets have an example.
the array: 5 1 9 3 2
1. step: split 5 1 9 3 2 ,split point: arraylength div 2
|
/ \
5 1 9 3 2
2. step: split | \
/ \ \
5 1 |
3. step: merge \ / |
| |
1 5 9 3 2
4. step: split | |
| / \
| / \
| 9 3 2
5. step: split | | |
| | / \
| | 3 2
6. step: merge | | \ /
| | |
| 9 2 3
7. step: merge | \ /
| \ /
1 5 2 3 9
8. step: merge \ /
1 2 3 5 9
MergeSort are a good algorithm for comps with a large array to sort but
insufficient memory to put all elements in.

QuickSort

A Divide and Counquer algorithm too. Have a look at the example!

the array: 5 2 9 6 1 3 4

1. step: choose an pivot-element -> 5

2. step: put all elements < 5 at the left and all > 5 at the right

-> 2 1 3 4 5 9 6

3. rekursive step into the left part and goto step 1

-> 2 1 3 4 pivot: 2

-> 1 2 3 4 "1" is only one elemnt -> no further rekursion

to the left

go rekursive into the right part

-> 3 4 pivot: 3

-> 3 4 come out of the rekursion and put both already

sorted parts together

-> 1 2 3 4 take the right part from the first rekursion

-> 9 6 pivot: 9

-> 6 9 put both parts together

-> 1 2 3 4 5 6 9 voila

May look a bit confuse. Remember: first choose an element in any way.

Then re-arrange the array. Put all whats smaller to the left and all

the bigger ones to the right. Then go rekursive into both parts and to

the same until there is nothing more to go into. Then put the left and

the right part together and go the rekursion up.

5 2 9 6 1 3 4 -> 5

5

2 1 3 4 9 6

2 1 3 4 -> 2

2

1 3 4

1 2 3 4

9 6 -> 9

6 9

1 2 3 4 5 6 9

HeapSort

A tricky one. First HeapSort rearranges the array into something whats

called a heap. A heap is a datastructure defined in the following way:

k1 k[j/2] >= k[j]

/ \

k2 k3 A tree with a special relation between the

/ \ / \ fathers and the sons.

k4 k5 k6 k7

. .

. .

the array:

5 2 9 6 1 3 2 - linear representation

| 5 This isn't the heap but heapsort forces this

-> / \ one into a heap. Take the middle element "6"

2 9 and let it fall down. The relation is that the

/ \ / \ father is bigger than the son. "6" cant sink

6 1 3 2 much deeper. So take in the array the one left

beside "6" -> "9". "9" can't sink deeper too.

The next is "2". "6" is bigger than "2". "2" and "6" are change their

places.

-> 5 The "5" falls down too.

/ \

6 9 -> 9 Now "9" is the biggest one. Put

/ \ / \ / \ "9" into the sorted area and put

2 1 3 2 6 5 the deepest element at the right

/ \ / \ edge of the tree into the root

2 1 3 2 -> "2"

-> 2 Let "2" fall down.

/ \

6 5 -> 6 Now "6" is the biggest one in the

/ \ / / \ heap. Put "6" out and let "3" take

2 1 3 2 5 place of "6".

/ \ /

2 1 3

-> 3 Let "3" fall down.

/ \

2 5 -> 5 "5" is the biggest one. Put 'em out

/ \ / \ and "1" to the old place.

2 1 2 3

/ \

2 1

-> 1 .. fall down ..

/ \

2 3 -> 3 .. put "3" out and "2" in ..

/ / \

2 2 1

/

2

-> 2 put "2" out and "1" in, change both places, put "2"

/ \ out and "1" out and voila.

2 1

For large elements this method gets good run-times. This is only an example. There are of course other much clever methods to realize that. (bottom-up heapsort)

These are not all existing algorithms. Lets have for example one called Countsort. This algorithm counts for every element the inverse relations of all elements f.e.

array: 5 2 9 6 1 3

inverse relations: 3 1 5 4 0 2 (means: 3 elements are <= 5,1 elem. <= 2

1 element <= 2, and so on)

positions : 0 1 2 3 4 5

sorted area : 5 With the calculated inverse relation

2 you have the final positions for each

9 element.

1 3 6

1 2 3 5 6 9

But visualize that and you'll see nothing, because most work are done internally without changing elements but then all together.

Another one is Hybridsort. This algorithm consist of one of the algorithms we've already talked about. In particular look at an example:

array: 5 2 9 6 1 3 Now take an apropriate number of intervals, lets

say 2. <0..4> <5..9> are the ranges of the

intervals. Decide now for every element in witch

1. 2. interval it belongs. Sort both arrays with any

2 5 algorithm, put both together and voila.

1 9

3 6

1.8 VisualSort - Purpose of

While my computer science study I got the task to write a program for visualisation of sorting algorithms. I've done it but for MS-DOS machines. (Arrgghhh) So I decided to rewrite it for my Amiga.

VisualSort was written to show graphically how the different algorithms

work.

Features

-: all common algorithms implemented (see Algorithms)

-: fontsensitive and screenmode adaptive

-: fully mouse and keyboard controlled

-: Arexx port

1.9 VisualSort - Usage and Menus

getting started from cli:

Starting VisualSort from CLI you can pass some arguments with it.

ASCENDING/DESCENDING - means the type of pre-sorting of the used ares

DEGREE - means in percent the number of elements witch are

already pre-sorted

LINES - lets you choose line-drawing mode (points are

default)

just try VisualSort LINES D 88 to see what I mean...

starting fron WB:

Doubleclick the right icon.

the menus:

Program

Save Statistics... - saves some statistics about the algorithm work

About... - gives some informations about the program

Quit - leaves the program

Algorithms - contain all implemented algorithms

Statistics... - brings the Statisticsrequester up

Setup

Screenmode... - lets you change the screenmode

Degree.. - lets you change the degree

points/lines - lets you set the kind of drawing the array

random - choose arraycontent per random

ascending/descending - pre-sorts the array with regards to the choosen

degree

freehand - draw your own sorting area :-)

complete empty parts - completion of empty parts in drawn picture

Statistics immediately - opens Statisticsrequester immediately after an

algorithm ends

Pick one of the algorithms and the visualisation starts. Now the big

empty part of the screen gets filled with lots of points (or lines). And
while the algorithm works these points/lines get changed to a nice
ascending line. The interesting thing while this is done is the kind,
how it is done. This depends on the kind of the choosen algorithm.
The two gadgets at the lower right edge of the screen lets you stop or
break the visualisation. Either press the underlined buttons or Space
and ESC will do the same. The disabled scroller may let you set a time
delay in the future.

1.10 VisualSort - Usage via Arexx

You can reach VisualSorts Arexxport contacting a port named: VISUALSORT.
Several commands are offered as follows:
QUIT - leaves the program
POPUP/POPBACK - brings the program's screen to front/back
SCREENMODE - brings the Screenmoderequester up
(only if reqtools.library was found!)
LINES - select lines as visualisation method
POINTS - the same but with points
RANDOMIZE - creates arraycontent per random
ASCENDING/DESCENDING - the direction of pre-sorted elements
FREEHAND - self-draw-mode selection
DEGREE <num> - sets the number of elements which shall
be pre-sorted using degree of existing elements
COMPLETE ON/OFF - completion of empty parts in drawn picture
IMMEDIATE ON/OFF - lets the Statisticsrequester pop up after
an algorithms ends
STATISTICS - brings the Statisticsrequester up
SAVESTATISTICS <name> - saves statistics into a file
BubbleSort [WAIT] - These are all implemented algorithms. Each of these
ShakeSort [WAIT] commands let the right algorithm work.
InsertSort [WAIT] - The WAIT-keyword stops the script-execution util
SelectSort [WAIT] the algorithm ends
ShellSort [WAIT]
MergeSort [WAIT]
QuickSort [WAIT]
HeapSort [WAIT]

1.11 Copyright notices