bubblesort

Laurence R Taylor

2020/06/24

Abstract

This package is a LaTeX port of the sorting function bubble sort. Bubble sort is usually coded using arrays but it can be done without them and since LaTeX does not support arrays natively, bubble sort seemed like a good routine to port. It's lack of speed is not really a problem with the data sizes likely to be encountered in most LaTeX applications.

The objects to be sorted are either a stack or a list. A *stack* is a sequence of legal TeX code items enclosed in brace pairs,

`{`*item 1*`}{`*item 2*`}...{`*item n*`}`

while a *list* is a macro which expands to a stack. Fragile commands in an *item* may not work as expected. To sort the items one needs a notion of when one *item* is "less than" another.

The macro `\bubblesort` does the bubble sort. It takes three arguments: `\bubblesort[#1]{#2}{#3}`. The optional argument is a comparator macro telling when one item is smaller than another. If empty it assumes the stack is a stack of integers or real numbers and sorts it using the usual $<$. Argument `#2` is the stack or list to be sorted and argument `#3` a macro which will contain the sorted list in increasing order.

The only dependency is `etoolbox.sty`.

1 Introduction

There are two macros provided which implement the standard bubble sort algorithms on stacks or lists. See section 2 for a discussion of stacks or lists.

1.1 `\bubblesort`

`\bubblesort[#1]{#2}{#3}`: Argument `#2` is the stack or list to be sorted. Argument `#3` contains the sorted list after evaluation. Argument `#2` can be the same as argument `#3`. Argument `#3` can also be blank in which case the output string is inserted into the input stream. Optional argument `#1` is a comparator macro.

1.1.1 Comparator macros

A comparator macro can have any legal TEX name. It must have two arguments. When in use the comparator macro is evaluated on consecutive pairs of elements in the stack. If argument `#2` is "smaller" than argument `#1` the macro sets `\bubblesortflag` to `1` and sets it to `-1` otherwise. Two examples are supplied: `\realSort` on line 28 of the code below and `\alphSort` on line 30.

1.2 `\doublebubblesort`

`\doublebubblesort[#1]{#2}{#3}{#4}{#5}`:

The macro `\doublebubblesort` sorts two stack/lists. Arguments `#1`, `#2` and `#3` are identical to the corresponding arguments for `\bubblesort`. Argument `#4` is another stack or list of the same length or longer than stack/list `#2`. When expanded `\doublebubblesort` sorts `#2` just the same as `\bubblesort` would. However, every move made on stack `#2` is also done on stack `#4`. Argument `#5` is the name for the output of the result of this "sort" on `#4`. As usual `#5` can be blank or `#4` and `#5` can be the same macro. `#3` and `#5` can also be the same macro, in which case it contains the result of the sort on argument `#4`, *except* if both `#3` and `#5` are blank, `#3` is put into the input stream followed by `#5`.

An example to keep in mind is the following. One way to mimic a hash in LATEX is to have a list of keys and a second list of values with the value associated to a key being determined by position in the values list. If the key list is sorted it will be necessary to do the same interchanges on the values list to maintain the correspondence. See subsection 3.3 for another example using `\doublebubblsort`.

2 Stacks and lists

The lists here closely resemble Knuth's lists in the TEXbook, page 378, Appendix D, except they lack Knuth's initial `\\` prepended to each item. As discussed by Knuth, the `\\` can be used to process each item in the list. Sort algorithms require knowledge of pairs of items from the list so macros which only know about one item are not needed.

Other implementations of lists use a reserved character as a separator. Separators like commas or semicolons or whatever have the drawback that then the separator can not be used in the item text without additional coding.

TEX's brace-pair mechanism is quite robust. It handles nested brace pairs just fine. One word of warning: an empty brace pair `{}` is used as an end of list indicator. They are added to the stack/list arguments when needed and are not present at the end of lists produced by macros in this package so they rarely trouble the user. It does mean that there can be no empty (or even `white space`) brace pairs in the input list. Given how TEX discards white space, `{white space}` is probably not what is wanted anyway. Using `{{white space}}` will probably yield results closer to what was intended and this works.

3 Examples

Here are a few examples if things that can be done with sorts.

3.1 \refs

Given a stack of references defined from \label such as {ref 1}{ref 2}...{ref n} which expand to integers or real numbers try to define

\def\refs#1{\bubblesort[\refSort]{#1}{\answers}}
where \refSort#1#2{\realSort{\ref{#1}}{\ref{#2}}}.

The above does not work because \ref is protected and does not expand to just the number. Instead, the package refcount.sty supplies \getrefnumber which is expandable and does what is needed. Add \usepackage{refcount} to the preamble and define

```
\def\refSort#1#2{%
\edef\aa{\getrefnumber{#1} pt}\edef\bb{\getrefnumber{#2} pt}%
\ifdimless{\bb}{\aa}{\bubblesortflag=1}{\bubblesortflag=-1}%
}
```

Then\def\refs#1{\bubblesort[\refSort]{#1}{\ans}} sorts the numbers in increasing order and puts the answer in \ans.

3.2 Use \refs twice

Since \bubblesort is a stable sorting algorithm, it can be usefully used more than once on the same data. Suppose \refSort is defined as above and there is a list \def\LL{{ref 1}{ref 2}...{ref n}}. Then \bubblesort[\refSort]{\LL}{\LL} sorts the numbers in increasing order.

But some of them may be from definitions, others from theorems, or lemmas, etc. Suppose \refName#1 is a macro such that \refName{ref k} returns Definition, Theorem, Lemma, etc.. Then

\bubblesort[\refSort]{\LL}{\LL} \bubblesort[\alphSort]{\LL}{\LL}

returns \def\LL{{ref k_1}{ref k_2}...{ref k_n}} where the first set of references are Definition's, the second set of references are Lemma's and the third set of references are Theorem's. The set of numbers for the Definition's are increasing, the set of numbers for the Lemma's are increasing and the set of numbers for the Theorem's are increasing. With more complicated coding, the names can be added and the lists compactified to get something line Theorems 1-3 and 5, Lemma 9 and Definitions 6, 8 and 12.

3.3 \permute

Use \doublebubblesort to apply a permutation to a stack/list. A *permutation* is an ordered stack/list of symbols which also has a natural order. Popular symbol sets are positive integers and lowercase letters. The comparator macro needs to sort a list of symbols with no repetitions back to its natural order. A permutation π can be given by listing the values of π when applied to the symbols in their natural order. For example π={{5}{1}{2}{3}{4}} means π(1)=5, π(2)=1, π(3)=2, π(4)=3, π(5)=4.

At the end of line 2 of the displayed code below, \ott contains the symbols in their natural order so \permute does not need to be told the symbols in advance. At the end of line 3, \ott contains the inverse to the original permutation. At the end of line 4, #4 contains the permuted version of #3.

```
\newcommand{\permute}[4][\realSort]{%
\bubblesort[#1]{#2}{\ott}%
\doublebubblesort[#1]{#2}{\ott}{\ott}{\ott}%
\doublebubblesort[#1]{\ott}{\ott}{#3}{#4}%
}
```

If \def\LL{{${a_1}$}{${a_2}$}{${a_3}$}{${a_4}$}{${a_5}$}} then \permute{{5}{1}{2}{3}{4}}{\LL}{} yields

a_5 a_1 a_2 a_3 a_4 , which is $a_{\pi(1)}$ $a_{\pi(2)}$ $a_{\pi(3)}$ $a_{\pi(4)}$ $a_{\pi(5)}$ as desired.

\permute can also be used to multiply permutations:

- \permute{\pi_1}{\pi_2}{\ans} yields \pi_2∘\pi_1
- \permute{\pi_2}{\pi_1}{\ans} yields \pi_1∘\pi_2.

4 Implementation

Line numbers refer to the line numbers printed next to the code beginning with \NeedsTeXFormat here. There are only four public macros, \bubblesort on line 32 and \doublebubblesort on line 71 and comparator macros \realSort on line 28 and \alphSort on line 30. Private macros begin with \@bubblesort@ to minimize the chance of name conflicts with other packages. This makes reading the code somewhat tedious so in the descriptions of the code below, the initial \@bubblesort@ will be replaced by *.

4.1 \bubblesort

\bubblesort[#1]{#2}{#3}: The implementation of \bubblesort begins by defining an empty *rightList and putting the stack/list to be sorted, #2, into *workingList. Note #2 is now safe - unless #2 equals #3, #2 will never change.

Then \bubblesort removes the leftmost item in *workingList and saves it in *testItem. Then it moves on to \@bubblesort@S which does the recursion.

First, \bubblesort@S (line 39) sets an etoolbox boolean, did a flip, to false and sets *leftList to empty. Then the macro removes the leftmost element in *workingList and saves it in *nextItem. Then it enters the while loop (line 43). The exit condition is that *nextItem is empty. Then the comparator macro is evaluated on the ordered pair (*testItem , *nextItem). The smaller of the pair is append to *leftList on the right and the larger is set to *testItem. The loop continues until *workingList is empty. At this point *testItem contains the largest element in the original *workingList and it is added to *rightList at the left end. Hence *leftList followed by *rightList is the original list. Next *leftList is put into *workingList and then the leftmost element is removed and put into *testItem. Finally the while loop is reentered.

After k iterations of the while loop, *rightList has k elements which are the k largest elements in the original list in increasing order. When the boolean did a flip is false, *leftList is also ordered so *leftList followed by *rightList is the original list sorted and the while loop exits. The sorted list and macro #3 from \bubblesort are passed on to \@bubblesort@output which outputs the list as requested and \bubblesort is finished.

Remark: The reader who has worked through the discussion above will have noticed that this implementation of bubble sort uses about four times the storage as the C version which sorts the array in place. Stacks used in LaTeX, are probably small enough that this is not a problem.

Sorting a thousand item list of integers in reverse order (which is the worse case scenario) took less than sixty-five seconds running TexShop 4.44 on a Mac 2013 Powerbook using timing code from tex.stackexchange. It put no strain on TeX's memory allocations.

4.2 \doublebubblesort

\doublebubblesort[#1]{#2}{#3}{#4}{#5}: The \doublebubblesort (line 71), \@doublebubblesort@S (line 82), macros work much like \bubblesort, \@bubblesort@S. The code in the double'd version contains the identical code from \bubblesort, \@bubblesort@S except that A is added to the end of each of the internal macros. There is also a parallel copy of macros with B added and everything done with the A variables is mimicked with the B variables.

4.3 The remaining macros

The macro \@bubblesort@output#1#2 (line 67) is just to shorten the code since there are three places a stack or a list is output (lines 65, 129 and 130).

The macro \@bubblesort@EoS is the End-of-Stack indicator.

The remaining macros do the removing items from a list and inserting items at either end of a list. The shift macro is

\@bubblesort@shift#1#2\@bubblesort@EoS#3#4 (line 133)
which is used as
\expandafter\@bubblesort@shift{stack/list}\@bubblesort@EoS{\itemA}{\itemB}
which puts the leftmost item in the stack/list into \itemA and the rest of the stack/list into \itemB.

The macros \@bubblesort@rightappendItem (line 139) and \@bubblesort@lefttappendItem (line 141) are identical to Knuth's \rightappenditem and \leftappenditem except that there are no prepended \\'s.

Start of code: The rest of the file from the end of this paragraph onward is a copy of the file generated by TEXing the bubblesort.ins file except that the standard set of comments at the start of an ins generated file are omitted. And of course the line numbers on the left here are not present in the \bubblesort.sty file. The comments portion takes 15 lines. Most editors show line numbers and allow line number navigation so it has been arranged that the line numbers in the typeset bubblesort.dtx file below match the line numbers in the ins-generated bubblesort.sty file.

```
16 \NeedsTeXFormat{LaTeX2e}[2005/12/01]
17 \ProvidesPackage{bubblesort}
18       [2020/07/01 v1.0 implements a bubble sort]
19
20 \RequirePackage{etoolbox}
21
22 \makeatletter
23
24 \newcount\bubblesortflag
25 \newbool{did a flip}
26 \def\@bubblesort@EoS{}% End-of-Stack indicator
27
28 \def\realSort#1#2{%
29 \ifdimless{#2 pt}{#1 pt}{\bubblesortflag=1}{\bubblesortflag=-1}}
30 \def\alphSort#1#2{\ifnumequal{\pdfstrcmp{#1}{#2}}{1}%
31 {\bubblesortflag=1}{\bubblesortflag=-1}}
32 \newcommand{\bubblesort}[3][\realSort]{%
33 %% #1 comparator macro, #2 input stack/list, #3 answer list: #2=#3 OK
34 \def\@bubblesort@rightList{}%
35 \expandafter\@bubblesort@shiftOne#2{}{}{}\@bubblesort@EoS{%
36 \@bubblesort@testItem}{\@bubblesort@workingList}%
37 \expandafter\@bubblesort@S{#3}{#1}%
38 }
39 \def\@bubblesort@S#1#2{% #1 is name for answer --- #2 is comparator macro
40 \boolfalse{did a flip}\def\@bubblesort@leftList{}%
41 \expandafter\@bubblesort@shiftOne\@bubblesort@workingList{}{}\@bubblesort@EoS{%
42 \@bubblesort@nextItem}{\@bubblesort@workingList}%
43 \whileboolexpr{not test{\ifdefvoid{\@bubblesort@nextItem}}}{%
44 #2{\@bubblesort@testItem}{\@bubblesort@nextItem}\relax%
45 \ifnumequal{\bubblesortflag}{1}{% flip
46 \booltrue{did a flip}%
```

```
\expandafter\@bubblesort@rightappendItem\expandafter{\@bubblesort@nextItem}%
\to\@bubblesort@leftList%
}{%
\expandafter\@bubblesort@rightappendItem\expandafter{\@bubblesort@testItem}%
\to\@bubblesort@leftList%
\expandafter\def\expandafter\@bubblesort@testItem\expandafter{\@bubblesort@nextItem}%
}%
\expandafter\@bubblesort@shiftOne\@bubblesort@workingList\@bubblesort@EoS{%
\@bubblesort@nextItem}{\@bubblesort@workingList}%
}%
\expandafter\leftappendItem\expandafter{\@bubblesort@testItem}\to\@bubblesort@rightList%
\ifbool{did a flip}{%
\expandafter\def\expandafter\@bubblesort@workingList\expandafter%
{\@bubblesort@leftList{}{}{}}%
\expandafter\@bubblesort@shiftOne\@bubblesort@workingList\@bubblesort@EoS{%
\@bubblesort@testItem}{\@bubblesort@workingList}%
\def\@bubblesort@leftList{}%
\@bubblesort@S{#1}{#2}}%
{\@bubblesort@output{#1}{\@bubblesort@leftList\@bubblesort@rightList}%
}}
\def\@bubblesort@output#1#2{% #1 name of output list or empty ---  #2 sorted stack
\ifstrempty{#1}%
{#2}{\expandafter\edef\expandafter#1\expandafter{#2}}%
}
\newcommand{\doublebubblesort}[5][\realSort]{%
%% #1 comparator macro
%% #2 input stack/list  --- #3 output for #2 stack/list; #2=#3 OK
%% #4 second stack/list --- #5 answer list for #4; #4=#5 OK
\def\@bubblesort@rightListA{}\def\@bubblesort@rightListB{}%
\expandafter\@bubblesort@shiftOne#2{}{}{}\@bubblesort@EoS{%
\@bubblesort@testItemA}{\@bubblesort@workingListA}%
\expandafter\@bubblesort@shiftOne#4{}{}{}\@bubblesort@EoS{%
\@bubblesort@testItemB}{\@bubblesort@workingListB}%
\expandafter\@doublebubblesort@S{#3}{#5}{#1}%
}
\def\@doublebubblesort@S#1#2#3{%
%% #1 output for sorted stack/list --- #2 output for ``sorted'' stack/list
%% #3 comparator macro
\boolfalse{did a flip}\def\@bubblesort@leftListA{}\def\@bubblesort@leftListB{}%
\expandafter\@bubblesort@shiftOne\@bubblesort@workingListA{}{}%
\@bubblesort@EoS{\@bubblesort@nextItemA}{\@bubblesort@workingListA}%
\expandafter\@bubblesort@shiftOne\@bubblesort@workingListB{}{}%
\@bubblesort@EoS{\@bubblesort@nextItemB}{\@bubblesort@workingListB}%
\whileboolexpr{not test{\ifdefvoid{\@bubblesort@nextItemA}}}{%
#3{\@bubblesort@testItemA}{\@bubblesort@nextItemA}\relax%
\ifnumequal{\bubblesortflag}{1}{% flip
\booltrue{did a flip}%
\expandafter\@bubblesort@rightappendItem\expandafter{%
\@bubblesort@nextItemA}\to\@bubblesort@leftListA%
\expandafter\@bubblesort@rightappendItem\expandafter{%
```

```
\@bubblesort@nextItemB}\to\@bubblesort@leftListB%
}{%
\expandafter\@bubblesort@rightappendItem\expandafter{%
\@bubblesort@testItemA}\to\@bubblesort@leftListA%
\expandafter\@bubblesort@rightappendItem\expandafter{%
\@bubblesort@testItemB}\to\@bubblesort@leftListB%
\expandafter\def\expandafter\@bubblesort@testItemA\expandafter{%
\@bubblesort@nextItemA}%
\expandafter\def\expandafter\@bubblesort@testItemB\expandafter{%
\@bubblesort@nextItemB}%
}%
\expandafter\@bubblesort@shiftOne\@bubblesort@workingListA\@bubblesort@EoS{%
\@bubblesort@nextItemA}{\@bubblesort@workingListA}%
\expandafter\@bubblesort@shiftOne\@bubblesort@workingListB\@bubblesort@EoS{%
\@bubblesort@nextItemB}{\@bubblesort@workingListB}%
}%
\expandafter\leftappendItem\expandafter{\@bubblesort@testItemA}\to%
\@bubblesort@rightListA%
\expandafter\leftappendItem\expandafter{\@bubblesort@testItemB}\to%
\@bubblesort@rightListB%
\ifbool{did a flip}{%
\expandafter\def\expandafter\@bubblesort@workingListA\expandafter{%
\@bubblesort@leftListA{}{}{}}%
\expandafter\def\expandafter\@bubblesort@workingListB\expandafter{%
\@bubblesort@leftListB{}{}{}}%
\expandafter\@bubblesort@shiftOne\@bubblesort@workingListA\@bubblesort@EoS{%
\@bubblesort@testItemA}{\@bubblesort@workingListA}%
\expandafter\@bubblesort@shiftOne\@bubblesort@workingListB\@bubblesort@EoS{%
\@bubblesort@testItemB}{\@bubblesort@workingListB}%
\def\@bubblesort@leftListA{}%
\def\@bubblesort@leftListB{}%
\expandafter\@doublebubblesort@S{#1}{#2}{#3}}%
{\@bubblesort@output{#1}{\@bubblesort@leftListA\@bubblesort@rightListA}%
\@bubblesort@output{#2}{\@bubblesort@leftListB\@bubblesort@rightListB}}%
}

\def\@bubblesort@shiftOne#1#2\@bubblesort@EoS#3#4{%
\expandafter\gdef\expandafter#3\expandafter{#1}%
\expandafter\gdef\expandafter#4\expandafter{#2}%
}

\newtoks\ta\newtoks\tb
\long\def\@bubblesort@rightappendItem#1\to#2{\ta={{#1}}\tb=\expandafter{#2}%
\edef#2{\the\tb\the\ta}}
\long\def\leftappendItem#1\to#2{\ta={{#1}}\tb=\expandafter{#2}%
\edef#2{\the\ta\the\tb}}
\makeatother
\endinput
```