SortableArray Template, Lab Assignment 12

Re-submit Assignment

Due Nov 22 by 11:59pm **Points** 100 **Submitting** a file upload **File Types** cpp and h

Write a sortable array class template that implements a O(n log n) sorting algorithm. The resulting template can be used in any program in place of a C++ array whose data type supports operator-less-than.

Rewrite DynamicArray.h and MyDynamicArray.cpp from lab 3 as **SortableArray.h** and **MySortableArray.cpp**, adding a sort function. Name the new templated class **SortableArray**. Do *NOT* use "inheritance" -- do copy/paste/mark-up.

Use this prototype for a new, **public** sort setter function:

```
void sort(int);
```

...implementing either heapsort or quicksort or mergesort from this module's readings, *your choice*. The int parameter specifies how many elements to sort, starting from element zero to this "fill-line". That is, sort(10) should sort only elements [0] through [9]. Sorting should IGNORE **inUse** and assume that all elements below the fill line are in use.

Improve the app so that after all data entry is complete, it:

1. outputs the number of values entered by the user (as before),
2. outputs the unsorted list as index-value pairs (as before), but one a single line, space-separated,
3. **prompts the user to enter how many values to sort -- that is, specify the "fill-line",**
4. **sorts all the data entries made by the user, lo-to-hi up to the "fill-line", and outputs them in a format of your choosing,**
5. implements index lookups in a user-controlled loop, allowing multiple look-ups until the user elects to stop (as before).

For this to work, the user MUST enter values for all indexes up to the "fill-line". Do NOT try to validate this -- just expect the user to do this right. The user *may* enter values for indexes above the fill-line. You may explain all this in cout statements, if you want to.

EXAMPLE: I enter these pairs -- note the "hole" at index 4, and the zero value at index 1:

```
6 67
3 23
7 22
1 0
0 15
9 20
2 25
```

Output this as follows:

```
0=>15 1=>0 2=>25 3=>23 6=>67 7=>22 9=>20
```

When prompted *just once* for how many to include in the sorting (starting from index 0 -- *always*), I say `4` (or less, because the holes start at index 4). Output the list again -- note that the first 4 are reordered lo-to-hi:

```
0=>0 1=>15 2=>23 3=>25 6=>67 7=>22 9=>20
```

Then start the loop asking the user to enter an index -- if I enter `2`, the result should be `23`. If I enter `-1`, the result should be "not found" or something like that. If I enter `9`, the result should be `20`, even though it's past the "fill line". If I enter `10000`, the answer should be "not found" or something like that. *Remember that I can also enter indexes outside the sorted range -- less than zero and above the "fill line". It's possible that indexes above the fill line actually have values!*

Submit **SortableArray.h** and **MySortableArray.cpp**.

Lab Assignment Rubric

Criteria	Ratings							Pts
Fully accurate results, following all specifications view longer description	Works the first time. 70.0 pts	Works on the 2nd try 65.0 pts	Works on the 3rd try 60.0 pts	Works after 4 or more tries. 50.0 pts	Doesn't work after 2 weeks. Partial credit. 20.0 pts	Not submitted within two weeks of the due date. 0.0 pts	Work is not original -- appears to be a marked-up copy of the work of another or previous student. 0.0 pts	70.0 pts
Submits all work on time, fully complete if not fully correct. view longer description	Submitted on time 20.0 pts	Submitted on time, but one or more files are missing or not correctly named. 16.0 pts		Submitted on time, but with missing identification in one or more submitted CPP or H files. 15.0 pts		Submitted on time but not fully complete. 10.0 pts	Late or wholly incomplete! 0.0 pts	20.0 pts
Well-organized and professional quality code. view longer description	Fully meets expectations 10.0 pts	Mostly meets expectations, just needs to be a bit more careful. 8.0 pts		Many areas are well done, but there are a lot of areas that need work. 6.0 pts		Getting there, but needs to be a lot better. 3.0 pts	Needs a lot of work. See the instructor for guidance. 0.0 pts	10.0 pts
							Total Points: 100.0	