Alexander Miller

345 PROJECT 1 DEBUG DOCUMENTATION

Note on Comparable Implementation and Passing by Value:

All comparisons are done with *compareTo()* to allow for the sort algorithm to work over multiple different types of objects. However, in order to prevent passing by reference when you temporarily copy an element of the array to a buffer in order to facilitate a swap or slide, you have to make a deep copy of the object. Since not every class implements the *clone()* method in the Object Interface, there is no way to systematically make deep copies of objects of every class that implements the Comparable Interface. Therefore, this program only works as intended when a value is passed, rather than a reference, when another variable is assigned to the object (i.e. primitives or classes such as String, Integer, etc.)

BUBBLE SORT

- Question: Is it bubbling correctly at each iteration?
  Solution: Print the array after each complete bubble up
- Question: Are the right items in the right positions being identified for a swap?
  Solution: Prior to each swap, print which values are to be swapped and their positions
- Question: Does each swap yield the appropriate relative order of the elements, as position them in the correct places in the array?
  Solution: After each swap, print the resulting relative order of the elements, as well as the resulting array

SELECTION SORT

- Question: Did each iteration over the array perform the proper executions?
  Solution: Print the array being sorted each time prior to sort
- Question: Is the algorithm correctly identifying the minimum value and swapping it with the correct element in the array?
  Solution: Print the minimum value found each time, as well as the first unsorted value to swap with

INSERTION SORT

- Question: Did each iteration over the array perform the proper executions?
  Solution: Print the array being sorted each time prior to sort
- Question: Are the proper values being compared at the right time?
  Solution: Print which values are being compared at each comparison
- Question: When are slides occurring, and are they executing correctly?
  Solution: Any time a slide occurs, indicate as such, and print array

MERGESORT

- Question: Is the algorithm splitting the data correctly at each iteration?
  Solution: Print the starting and ending index of each section prior to a split

- Question: Is the base case being triggered as intended? What does the array look like that is being sorted by the n2 algorithm?
  Solution: If base case, print out the base case array length and the baseLen value, the starting and ending indices, as well as the array prior and following the n2 sort
- Question: Are merges occurring when intended? Are the values being compared correctly to facilitate the merge? Is the right value selected to paste to the buffer?
  Solution: Print when a merge is occurring, show which values are being compared as the merge is occurring, as well as which value is chosen from the 2
- Question: Does the merging sequence end because the "low array" and "high array" iterators intercepted, or because the "high array" iterator exceeded the upper bound?
  Solution: Indicate if the merging sequence ends because *i* hits the *j* range, or if the *j* exceeds the ending value
- Question: Is the temporary buffer properly sorted prior to pasting back into the main array? Does the operation of pasting back into the array complete successfully?
  Solution: Print buffer array only prior to pasting back, and the entire array, before and after merging

QUICKSORT

- Question: Is the proper range of the array being sorted with each iteration? What does the array look like at each iteration?
  Solution: Print array, the starting and ending index of each section being sorted prior to sort
- Question: Is a pivot being selected correctly, according to the specifications of "mode"?
  Solution: print the pivot index each time a pivot is selected, and the corresponding "mode" to remind the reader
- Question: Does the pivot swap to the beginning of the unsorted collection correctly?
  Solution: Print the array with the pivot moved to the beginning.
- Question: Are the array items being compared with the pivot in the appropriate sequence?
  Solution: For each comparison with the pivot, print the item being compared
- Question: When, if ever, do the "low array" and "high array" meet?
  Solution: Print if *i* overlaps with *j*
- Question: Does each successive partition execute correctly?
  Solution: Print array after a round of partitioning
- Question: Does the base case trigger as it is supposed to? What size array is being sorted by the n2 algorithm?
  Solution: If base case, print out the base case array length and the baseLen value
- Question: Are items being swapped when they're supposed to?

Solution: Any time a swap occurs, print items being swapped.