Sorting Algorithms – Insertion, Shell, Merge, and Quick
Cpt S 223 Homework Assignment
Washington State University

Complete this assignment by building on top of the previous assignment. You must support the commands listed below. All commands from the previous assignment are included. New commands that were not in the previous assignment are highlighted in red.

set(num1Value,num1Index;num2Value,num2Index; ... numXValue,numXIndex) : Sets the current array. The array consists of simple structures that have TWO integer values within them, a value and an index. Notice that one "item" is declared with two numbers that have a comma between them. Then the items are separated by semicolons. You will sort based on the value field alone but individual structures must remain intact as you sort. In other words, if one item is {57,2} then you should never be splitting those two values apart. You move the whole structure as one entity as you sort, even though '57' is the only value you look at when comparing items.

- All subsequent commands that you process will operate on this data. Note that this line may be VERY long in the input file. Set can be called at any time to change to a new data set. Do NOT display anything in response to this command, just update the current array.
- Note that there is not a semicolon after the very last parameter.
- All numbers within structures will be valid integer values.
- This is reiterated in the following command descriptions, but when you are going to sort this array you first make a copy of it, leaving the original data intact.
- You may want to use your MyString class from a previous homework assignment to assist with parsing of this command.

Below is an example of what the structure declaration for an item in your array might look like:

```
struct Item
{
   int Value;
   int Index;
};
```

insertion() : Sorts the current array using insertion sort and displays the results ON ONE LINE. Duplicate the array before sorting it, so that the original remains the same. In other words, duplicate current → newArr, and then sort newArr. See information in the notes section about how you need to display the sorted array contents.

shell(gap1,gap2,gap3,...) : Sorts the current array using shell sort and displays the results ON ONE LINE. Duplicate the array before sorting it, so that the original remains the same. In other words, duplicate current → newArr, and then sort newArr. See information in the notes section about how you need to display the sorted array contents.

- Run the shell sort first using the first gap value, then the second, then the third, and so on until you've sorted using all specified gap values.
- You may NOT assume that the last gap value will always be 1. While this IS required to do a proper sort, the grading process is going to test whether or not your interleaved list sorting logic is correct. So, in response to this command you may end up displaying certain lists that are not actually fully sorted.
- There will be at least 1 gap value, but no limit to how many gap values there will be total. The last gap value does not have a comma after it, just the closing ')' character.

merge_sort() : Sorts the current array using merge sort (either the classic or natural variant) and displays the results ON ONE LINE. Duplicate the array before sorting it, so that the original remains the same. In other words, duplicate current → newArr and sort newArr. See information in the notes section about how you need to display the sorted array contents.

- You only need to implement one of the two merge sort variants for this assignment, but need to know how both work for the exams. If you have the time (and motivation) feel free to implement both.
- Regardless of which variant you choose, you must have the function to merge two already sorted lists together. The implementation of this function is the same for the two variants.

partition(pivotIndex) : Runs a single partitioning step on the array contents and displays the results ON ONE LINE. Duplicate the array before partitioning it, so that the original remains the same. In other words, duplicate current → newArr and partion newArr. See information in the notes section about how you need to display the array contents.

- This is a single step of the partition function that is used by Quicksort.
- It will NOT sort the entire list. That is, it might end up sorting it by chance, but it is certainly not guaranteed to.
- You may assume that the pivot index parameter is always a valid index in the current array (but an extra check certainly wouldn't hurt if you want to add one).

Use the scheme we talked about in class: swap pivot with last element, move pointers/indices inward, swap when they point to elements that don't satisfy the condition, stop when pointers/indices are equal, and swap the pivot from the end back into the correct spot (2 possible cases for this).

quick_sort() : Sorts the current array using quick sort and displays the results ON ONE LINE. Duplicate the array before sorting it, so that the original remains the same. In other words, duplicate current → newArr and sort newArr. See information in the notes section about how you need to display the sorted array contents.

- Use your partition functionality that you implemented for the partition command. You will need it to return the final index of the pivot so that you know how to make the recursive calls.
- Choose the pivot as the median of the 3 values at 0, n-1, and n/2.

quit() : Breaks the command processing loop and terminates your program. Display the line "Done" as the very last thing before returning from main.

Final Notes:

- When displaying the array contents, print the items from the sorted array in a similar fashion as they were stored in the set command. Use a comma between the two values within a single item and semicolons after each item. Include a semicolon after the very last item on the line. Do not put any spaces in.
- See in input/output text files included in the zip file to get an idea of exactly what the commands look like before you start writing the code. Use I/O redirection for testing:
  ./a.out < in1.txt > myout1.txt
  diff out1.txt myout1.txt
- Put comments in your code! Give a brief explanation for each function at a minimum.