**Regis University CC&IS**

**CS324 Algorithms & Analysis**

**Alternate Homework 2 (includes Programming Assignment) V1**

**Assignment**Place your answers directly under each question below -- do ***not*** delete the questions.

**PART 1 – PROBLEMS**

NOTE: Although the selection, insertion, and bubble sorts are not part of this homework, they may still appear on the exam. So be sure to complete the online exercises for understanding.

***Faster Sorting Algorithms***

Use the algorithms located in the online Content for each of these sorts.

Remember that character values are ordered alphabetically (‘A’ < ‘B’ < ‘C’, etc.)

1) Shell Sort: Given the following unsorted array of characters to start: (*14 pts*)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [0] | [1] | [2] | [3] | [4] | [5] | [6] | [7] | [8] |
| K | J | C | E | B | P | F | G | A |

and using the **shell sort** with the halving method for the gap,

For each pass (label as pass 1, pass 2, etc.):

* + State the gap value
  + For each item being inserted within the pass:
    - **FIRST, state which value is being inserted.**
    - If any values were moved, show the array after placing the value, highlighting any values moved.
  + Show the array at the end of each pass

2) Merge Sort: Given the following unsorted array of characters to start: (*18 pts*)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [0] | [1] | [2] | [3] | [4] | [5] | [6] | [7] | [8] |
| K | J | C | E | B | P | F | G | A |

and using the **merge** sort:

* EITHER:
  + For each call, list the recursive calls would be made going forward
  + When returning, show the ordering of each sublist with more than one value in it, after each recursive call returns.
* OR
  + Show the call and return tree for the sort

3) QuickSort: Given the following unsorted array of characters to start: (*18 pts*)

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [0] | [1] | [2] | [3] | [4] | [5] | [6] | [7] | [8] | [9] |
| K | J | E | C | B | P | F | G | A | R |

and using the first **quick** sort algorithm from the online Content   
(with the first /leftmost value as the pivot value):

* For each call (label as call 1, call 2a, call 2b, etc.) going forward:
  + Identify the call indexes and the pivot value
  + State which values will be swapped during partitioning
  + Show the array after partitioning, with the pivot value highlighted,   
    and correct values on the left and on the right of the pivot
  + List the recursive calls that are made from the call

4) Complete problem **C8.3** on page 259 of your textbook (*10 pts*)

**PART 2 – PROGRAMMING ASSIGNMENT** (*40 pts*)

Two of the sorting algorithms that seem to be the hardest for students to understand are:

Shell sort

Quick sort

Last time you took this class, you chose a sort. For this homework problem, **choose the sort that you did NOT implement last time you took the course**.

Then write a Java program that will help someone understand the algorithm, by explaining each step, as it happens. Your Java project name should have your last name appended on the front of it. For example:

Smith-Hwk2program

To keep things simple, you can just implement your program using one Java class, containing all methods to sort and display the list.

The program will:

* + Read the number of items to sort from the user (and error check that the size entered is at least 12 and less than 100, looping until a good value is entered).
  + Create an array of that size and read the initial values into the list from input file of integers, **unsorted.txt** (located in an **input** directory within your NetBeans project).
    - The input file will contain 99 randomly ordered integers, separated by whitespace.
      * You should create a sample file yourself, for testing. But your instructor will test your program using their own data file.
  + Call the sort to sort the list. The parameters will be the list and two integer indexes, representing the first index, and the last index of the part of the list to be sorted.

For example, the calls a list that has 25 items in it would be:

**quickSort(list, 0, 24);**

or **shellSort(list, 0, 24);**

* + When the sort is complete, display the sorted list’s indexes and values, neatly to the screen.

WARNING: The purpose of the assignment is to provide a thorough walkthrough description of the sort. Simply implementing the sort, without the required displays will earn *only* **5 of the 30** points.

See the **appendices** of this assignment for the requirements, and to see EXACTLY what must be included in each of the displays, and the point allocations for displaying each.

Quicksort requirements: [Appendix A](#_Appendix_A:_)

Shellsort requirements: [Appendix B](#_Appendix_B:_)

**Submission**

This homework assignment is due by midnight of the date listed on the **Course Assignments by Week** page of the Content.

* For the problems, fill in all the answers within this Word doc. Before you submit your homework, append your last name to the front of the Word doc filename.   
  For example: **Smith-CS324-AltHwk2v1.docx**
* For the program, export your project from NetBeans:
  + Highlight the project name.
  + Click on **File** from the top menu, and select **Export Project**.
  + Select **To ZIP**
  + Name your export file in the following format:  
    **<lastname>-Hwk2Program.zip**

For example:

**Smith-Hwk2Program.zip**

NOTE: Save this zip file to some other directory, not your project directory.

Submit both your **.docx** file and your **.zip** file to the same **Wk 3 - Hwk Assn 2** Submission Folder (located under **Assignments** tab in online course).

***WARNING:****Any part of this homework submitted more than* ***3 days*** *past the due date will* ***not*** *be accepted,  
and will receive a grade of 0.*

# **Appendix A: Requirements for Quicksort**

Implement the Quicksort algorithm from the course Content, including the TWO improvements from Content section 5.4.2.3:

Improvement 1: Switch to another sort (selection or insertion sort)

when there are fewer than 7 items left in the list to sort

Improvement 2: Use median of 3 values to choose pivot (then move median value to leftmost index)

For **each call** to the Quicksort (original and recursive), have the program display: (4 *pts*)

* + The starting and ending indexes of the list from the call
  + Each index ***and*** its value, in part of the list to be sorted

Then if the list is large enough to be sorted using Quicksort, have the program display:

* + Details of call to median of 3: (*5 pts*)
  + Original indexes and values of the low, med, and high (before ordering)
  + Final values in the same three indexes, and which is the pivot (after ordering)
  + Details of partitioning: (*6 pts*)
  + Each pair of left/right values that are swapped
  + Final swap to put pivot in place
  + Results of partitioning: (*6 pts*)
  + Pivot value and its index
  + Indexes and values in the left half
  + Indexes and values in the right half

Or if the list is small enough to use another sort, have the program display: (*4 pts*)

* + A message stating which sort will be used
  + The part of the list that was sorted with an alternate sort, after sorting

***Sample Output***

Example output when the input file contains these 12 unsorted values:   
29 38 85 37 49 69 33 55 88 71 52 41

CALL Quicksort for indexes 0 to 11

List to sort is:

[0]29 [1]38 [2]85 [3]37 [4]49 [5]69 [6]33 [7]55 [8]88 [9]71 [10]52 [11]41

Median Of 3 (medianLeft):

Before sort: low is [0]29, mid is [5]69, high is [11]41

After sort: low is [0]29, mid is [5]41, high is [11]69

Swap low and mid to make pivot [0]41

During partitioning:

Swap [2]85 and [6]33

Swap [4]49 and [5]29

Swap [4]29 and pivot value [0]41

After partitioning:

Pivot in correct position

[4]41

Left values

[0]29 [1]38 [2]33 [3]37

Right values

[5]49 [6]85 [7]55 [8]88 [9]49 [10]85 [11]55 [12]88

CALL Quicksort for indexes 0 to 3

List to sort is:

[0]29 [1]38 [2]33 [3]37

Less than 7 items, so use InsertionSort

Result: [0]29 [1]33 [2]37 [3]38

CALL Quicksort for indexes 5 to 11

List to sort is:

[5]49 [6]85 [7]55 [8]88 [9]71 [10]52 [11]69

Median Of 3 (medianLeft):

Before sort: low is [5]49, mid is [8]88, high is [11]69

After sort: low is [5]49, mid is [8]69, high is [11]88

Swap low and mid to make pivot [5]69

During partitioning:

Swap [6]85 and [10]52

Swap [8]49 and pivot value [5]69

After partitioning:

Pivot in correct position

[8]69

Left values

[5]49 [6]52 [7]55

Right values

[9]71 [10]85 [11]88

CALL Quicksort for indexes 5 to 7

List to sort is:

[5]49 [6]52 [7]55

Less than 7 items, so use InsertionSort

Result: [5]49 [6]52 [7]55

CALL Quicksort for indexes 9 to 11

List to sort is:

[9]71 [10]85 [11]88

Less than 7 items, so use InsertionSort

Result: [9]71 [10]85 [11]88

Final Sorted List:

[0]29 [1]33 [2]37 [3]38 [4]41 [5]49 [6]52 [7]55 [8]69 [9]71   
[10]85 [11]88

# **Appendix B: Requirements for Shell sort**

Implement the shell sort algorithm from the course Content, using the ***halving method*** to determine the gap size.

When the shell sort is called have the program display: (*4 pts*)

* + The starting and ending indexes of the list from the call
  + The gap and list size initial values

For **each pass** of the Shell sort, have the program display: (*5 pts*)

* + The pass number
  + The gap size
  + The starting index (i.e. index of first item to insert)
  + BOTH the indexes ***and*** the values at each index, in list at the start of the pass

Then for **each item to be inserted** during the pass, display:

* + The value being inserted (*4 pts*)
  + Each item that the value is compared to, the comparison result, and whether the item is moved down to make room for the value being inserted (*8 pts*)
  + The final index where the item is inserted (or a note that it is already in the right place)   
    (*4 pts*)

***Sample Output***

Example output when the input file contains these 12 unsorted values:   
29 38 85 37 49 69 33 55 88 71 52 41

CALL Shell sort for indexes 0 to 11

INITIALIZE: gap = 12 listSize = 12

FOR PASS 1: gap = 6 start index = 6

LIST AT START OF PASS:

[0]29 [1]38 [2]85 [3]37 [4]49 [5]69 [6]33 [7]55 [8]88 [9]71 [10]52 [11]41

Insert Value: 33

COMPARE: [6]33 < [0]29 ? No

Already inserted in correct place

Insert Value: 55

COMPARE: [7]55 < [1]38 ? No

Already inserted in correct place

Insert Value: 88

COMPARE: [8]88 < [2]85 ? No

Already inserted in correct place

Insert Value: 71

COMPARE: [9]71 < [3]37 ? No

Already inserted in correct place

Insert Value: 52

COMPARE: [10]52 < [4]49 ? No

Already inserted in correct place

Insert Value: 41

COMPARE: [11]41 < [5]69 ? Yes - MOVE 69 to index [11]

INSERT 41 at index [5]

FOR PASS 2: gap = 3 start index = 3

LIST AT START OF PASS:

[0]29 [1]38 [2]85 [3]37 [4]49 [5]41 [6]33 [7]55 [8]88 [9]71 [10]52 [11]69

Insert Value: 37

COMPARE: [3]37 < [0]29 ? No

Already inserted in correct place

Insert Value: 49

COMPARE: [4]49 < [1]38 ? No

Already inserted in correct place

Insert Value: 41

COMPARE: [5]41 < [2]85 ? Yes - MOVE 85 to index [5]

INSERT 41 at index [2]

Insert Value: 33

COMPARE: [6]33 < [3]37 ? Yes - MOVE 37 to index [6]

33 < [0]29 ? No

INSERT 33 at index [3]

Insert Value: 55

COMPARE: [7]55 < [4]49 ? No

Already inserted in correct place

Insert Value: 88

COMPARE: [8]88 < [5]85 ? No

Already inserted in correct place

Insert Value: 71

COMPARE: [9]71 < [6]37 ? No

Already inserted in correct place

Insert Value: 52

COMPARE: [10]52 < [7]55 ? Yes - MOVE 55 to index [10]

52 < [4]49 ? No

INSERT 52 at index [7]

Insert Value: 69

COMPARE: [11]69 < [8]88 ? Yes - MOVE 88 to index [11]

69 < [5]85 ? Yes - MOVE 85 to index [8]

69 < [2]41 ? No

INSERT 69 to index [5]

FOR PASS 3: gap = 1 start index = 1

LIST AT START OF PASS:

[0]29 [1]38 [2]41 [3]33 [4]49 [5]69 [6]37 [7]52 [8]85 [9]71 [10]55 [11]88

Insert Value: 38

COMPARE: [1]38 < [0]29 ? No

Already inserted in correct place

Insert Value: 41

COMPARE: [2]41 < [1]38 ? No

Already inserted in correct place

Insert Value: 33

COMPARE: [3]33 < [2]41 ? Yes - MOVE 41 to index [3]

33 < [1]38 ? Yes - MOVE 38 to index [2]

33 < [0]29 ? No

INSERT 33 to index [1]

Insert Value: 49

COMPARE: [4]49 < [3]41 ? No

Already inserted in correct place

Insert Value: 69

COMPARE: [5]69 < [4]49 ? No

Already inserted in correct place

Insert Value: 37

COMPARE: [6]37 < [5]69 ? Yes - MOVE 69 to index [6]

37 < [4]49 ? Yes - MOVE 49 to index [5]

37 < [3]41 ? Yes - MOVE 41 to index [4]

37 < [2]38 ? Yes - MOVE 38 to index [3]

37 < [1]33 ? No

INSERT 37 to index [2]

Insert Value: 52

COMPARE: [7]52 < [6]69 ? Yes - MOVE 69 to index [7]

52 < [5]49 ? No

INSERT 52 to index [6]

Insert Value: 85

COMPARE: [8]85 < [7]69 ? No

Already inserted in correct place

Insert Value: 71

COMPARE: [9]71 < [8] 85 ? Yes - MOVE 85 to index [9]

71 < [7] 69 ? No

INSERT 71 to index [8]

Insert Value: 55

COMPARE: [10] 55 < [9] 85 ? Yes - MOVE 85 to index [10]

55 < [8] 71 ? Yes - MOVE 71 to index [9]

55 < [7] 69 ? Yes - MOVE 69 to index [8]

55 < [6] 52 ? No

INSERT 55 to index [7]

Insert Value: 88

COMPARE: [11]88 < [10]85 ? No

Already inserted in correct place

Final Sorted List:

[0]29 [1]33 [2]37 [3]38 [4]41 [5]49 [6]52 [7]55 [8]69 [9]71 [10]85 [11]88