Activity #2: Sorting
Recorder's Report

Manager: Reader:

Recorder: Driver:

Date: Score: Satisfactory / Not Satisfactory

Record your team's answers to the key questions (marked with ) below.

a) Model 1, Question #5

b) Model 1, Question #7

c) Model 2, Question #14

d) Model 2, Question #19

Activity #2: Sorting

In this course, you will work in teams of 3–4 students to learn new concepts. This activity will introduce you to sorting algorithms.

Content Learning Objectives

After completing this activity, students should be able to:

- Explain how arrays are sorted
- Explain the merge sort algorithm

Process Skill Goals

During the activity, students should make progress toward:

- Understand sorting algorithms
- Understand the merge sort algorithm

Model 1 Comparing

In computing, we often need to **sort** a set of items. As computer scientists, we study ways to sort very large sets, with thousands or millions of values, since searching and other operations are often easier when the set of values is sorted.
For example, the Harvard University Library has roughly 16,000,000 volumes, and the US Library of Congress has roughly 22,000,000 books, and over 100,000,000 total items. In 2010, a team at UC San Diego sorted one trillion (10^{12}) data records in 172 minutes. Simple $O(N^2)$ sorting algorithms work well for small lists, but are too slow for larger lists. Most software libraries (APIs) include excellent sorting algorithms, but exploring better sorting algorithms also demonstrates more general concepts in algorithm design and analysis.

```
1  void sort(int d[], int size);
2  void sort(string d[], int size);
3  void sort(City d[], int size);
4
5  void demo() {
6      int iu[] = {6, 3, 8, 2, 9};
7      int is[5];
8      sort(iu, 5);
9      string su[] = {"banana", "grape", "apple", "mango"};
10     string ss[4];
11     sort(su, 4);
12     City cu[] = {City::NYC, City::LAX, City::PHL, City::CHI};
13     City cs[4];
14     sort(cu, 4);
15 }
16
```

Refer to Model 1 above as your team develops consensus answers to the questions below.

Questions (15 min) Start time:

1. Each **sort()** function above takes an array input and sorts it. What is **different** about the first 3 functions?

2. What sequence of values should appear in each variable below:

a.	is	
b.	ss	

3. In the table below, specify tests for **sort()**:

	Input: array to sort	**Expected Result**
a.	[ ] (empty list)	empty list
b.	[5]	5
c.	[2, 4, 8]	
d.	[9, 7, 3, 5]	
e.	[27, 42, 35]	
f.	["a"]	
g.	["a", "b", "c"]	
h.	["c", "b", "a"]	
i.	["a3", "a1", "a2"]	
j.	["bar", "ball", "back"]	

4. It is easier to compare integers than strings. Explain why.

Suppose we had a list (e.g. a database or spreadsheet) of detailed information about cities:

Name	**Area (sq mi)**	**Population**	**Altitude**	**Latitude**	**Longitude**
Chicago	227.6	2,700,000	...	...	...
Los Angeles	468.7	3,800,000	...	...	...
New York	302.6	8,200,000	...	...	...
...	...	...	...	...	...

5. Explain why it could be harder to compare Cities than strings.

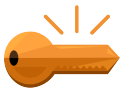
6. It can be useful to **abstract** the comparisons used in searching & sorting.

   a) In the **Sample Code** handout, what **function** is declared in the Comparable interface?

   b) The C++ `std::copysign()` function (or custom signum) returns 0 if its input is 0, +1 if its input is positive, and -1 if its input is negative. In the Sample Code handout, how does class City compare two cities?

   c) The C++ standard library class string provides comparison operators. Given this, what C++ expression would compare two strings: s1 and s2?

7. In class City, modify the return statement to compare Cities using:



a.	population	
b.	name (alphabetical)	
c.	name (reverse alphabetical)	
d.	population density	

8. Comparison can be abstracted in more than one way. In the Sample Code handout, what functions are declared in the Comparator interface?

9. For each class listed below, describe how it compares 2 objects.

a.	`class StringCompI`	
b.	`class StringCompL`	
c.	`class CityCompA`	
d.	`class CityCompP`	

10. Change the return statement in `CityCompA::compare()` to compare:

a.	name (alphabetical)	
b.	name (reverse alphabetical)	
c.	population density	

Model 2 Merge

Refer to Model 2 above as your team develops consensus answers to the questions below.

Questions (15 min) Start time:

Some sort algorithms (e.g. bubble sort and selection sort) are simple but inefficient. To understand a better sort algorithm, we'll start with something easier - **merge**.

11. Recall that $\theta()$, $O()$, and $\omega()$ **notation** describes how input size affects operation count and run time. If the input size **doubles**, what happens to the run time if an algorithm is O(1), O(N), etc? Which line at right shows this?

a. O(1)	b. O(N)	c. $O(N^2)$	d. O(log N)

12. Use the table below to specify **unit tests** for **merge(arrA, arrB)**, which **merges** two sorted arrays into one sorted array.

arrA	**arrB**	**Expected Result** (return value, exception, etc)
(empty)	(empty)	(empty)
"B"	(empty)	
(empty)	"A"	
"B"	"A"	
"B", "D"	"A", "C"	
"B", "C", "D"	"A", "C"	

13. Given 2 sorted arrays arrA & arrB that will be merged into one sorted array arrC.

arrA			
a^0	a^1	a^2	a^3

arrB			
b^0	b^1	b^2	b^3

⇒ **arrC**							
c^0	c^1	c^2	c^3	c^4	c^5	c^6	c^7

a) What is the length of `arrC`, in terms of `arrA.length` and `arrB.length`?

b) There are only 2 values (a_0 and b_0) that could go into c_0. Explain why.

c) Once c_0 is chosen, there are only 2 values that could go into c_1. Explain why.

d) Similarly, once c_0 to c_i are chosen, there are only 2 values that could go into c_{i+1}. Explain.

14. Given your answers above, how could we **merge()** 2 arrays efficiently?
In complete sentences or pseudocode, describe a general approach for **merge()**.

```
void merge(string arrA[], int sizeA, string arrB[], int sizeB, string arrC[])
```

15. The table below shows the contents of sorted arrays arrayA and arrayB. Complete the table to show how arrayC and the 3 index variables (ai, bi, ci) change over time.

arrayA					**arrayB**					**arrayC**									**vars**		
0	**1**	**2**	**3**		**0**	**1**	**2**	**3**		**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**		**ai**	**bi**	**ci**
2	6	12	14		3	5	15	17											0	0	0
2	6	12	14		3	5	15	17		2									1	0	1
2	6	12	14		3	5	15	17		2	3								1	1	2
2	6	12	14		3	5	15	17		2	3										
2	6	12	14		3	5	15	17		2	3										
2	6	12	14		3	5	15	17		2	3										
2	6	12	14		3	5	15	17		2	3										
2	6	12	14		3	5	15	17		2	3										
2	6	12	14		3	5	15	17		2	3										

16. During merge(), if there are N/2 values in each sorted array, and N values in the merged array (also sorted), explain the **O()** effort to find:

		O()	Explanation
a.	The **first value** in the merged array.		
b.	The **second value** in the merged array.		
c.	Each **successive** value in the merged array.		
d.	**All N** values in the merged array.		

17. In the following questions, a 2 item list is called a 2-list, a N item list is called an N-list, etc. Decide if each of the following is always sorted (Y or N):

a.	a 16-list of random numbers	
b.	the first half of a 32-list of random numbers	
c.	a 1-list	
d.	a 2-list made by merging 2 (sorted) 1-lists	
e.	a 16-list made by merging 2 (sorted) 8-lists	
f.	a N-list made by merging 2 (sorted) (N/2)-lists	

18. Decide how many new lists are made when pairs of lists are merged from:

a.	1024 1-lists into 2-lists	
b.	all of those 2-lists into 4-lists	
c.	N 1-lists (for any N) into 2-lists	
d.	all of those 2-lists into 4-lists	

19. How could we use merge to sort an unsorted list? (Hint: think recursively.)
In complete English sentences, describe a general approach for mergesort.