Project 4 - Sorting Algorithm Analysis

Justin Adams
Computer Science CS124

Compiled on Friday 15th November, 2019 at 16:01

1 Abstract

Bubble Sort, Insertion Sort, Selection Sort, Merge Sort, Heap Sort, and Quick Sort were analyzed using object counts varying from n = 1-10,000. Cross over analysis reveals that the compiler cost of running a divide and conquer paradigm is worth investing in at n = 46 objects. For objects greater than 47 Quick Sort consistently out performs the sorting algorithms tested. However, it is an unstable algorithm that consumes a lot of energy to sort objects. If stability or energy conservation is a priority when sorting non-primitive objects, then Merge Sort is the best sorting algorithm to use. Bubble Sort, Selection Sort, and Insertion Sort demonstrate a polynomial complexity of $O(n^2)$. Merge Sort, Quick Sort, and Heap Sort demonstrate a log-linear complexity of $O(n * log(n))$ 1. Quick Sort uses more read and write operations as a trade off for an exceptional run time improvement. When sorting 10,000 objects Quick Sort takes 0.068 seconds, Merge Sort takes 0.258 seconds, and Heap Sort takes 0.719 seconds. Trade offs between the algorithms include energy cost, sorting stability, and run-time.

Table 1: Sorting Algorithm Complexity, Memory, and Stability

Algorithm	Complexity	Memory	Stability
Bubble Sort	O(N^2)	N	Yes
Heap Sort	O($Nlog(N)$)	N	No
Insertion Sort	O(N^2)	N	Yes
Merge Sort	O($Nlog(N)$)	2N	Yes
Quick Sort	O($Nlog(N)$)	2N	Yes
Selection Sort	O(N^2)	N	No

2 Introduction

Project 4 requires students to analyze six sorting algorithms: Bubble Sort (Section 2.1), Selection Sort (Section 2.2), Insertion Sort (Section 2.3), Merge Sort (Section 2.4), Quick Sort (Section 2.4), Quick Sort (Section 2.5), Heap Sort (Section 2.6), and Two Sort (Section 2.7). Read, write, and run-time data were collected for small intervals of 100 objects from 100-1,000, large intervals of 1000 objects from 1,000-10,000, and a cross-over analysis from 0 objects to 70 objects for all of the sorting algorithms.

The base class applications.h (Appendix 8.1) used in these sorting algorithms for this lab experiment can be found in the appendix (Section 8). The Applications base class loads Google Play Store meta data from a csv file (Appendix 8.4) file available at https://www.kaggle.com[1]. This file contains n=10,843 meta data objects, while the total number of applications available on Google Play Store is currently 2.8 million. So, this file is a small fraction (0.004%) of the total applications meta data stored on Google's servers. For the purposes of this lab experiment a small fraction of the Google Play Store applications meta data is enough to discover emergent properties within the selected sorting algorithms.

2.1 Bubble Sort

Bubble sort is the simplest sorting algorithm. The algorithm iterates through the data structure from the front to the back either making no change or swapping the larger element toward the back of the structure. Once arriving at the end of the data structure, this process repeats until the list is in a sorted order. The complexity of this algorithm is $O(n^2)$.

2.2 Selection Sort

Selection Sort improves on Insertion Sort, but it also runs with $O(n^2)$ complexity. Selection Sort starts at the front of the data structure, upon discovering an item less than the first element in the array the index is stored. If an object is found that is less than the new index, then the index updates with the location of the lowest value. The lowest value in the data structure moves to the front of the data structure. The next lowest value moves to the second position. This procedure continues until all items are sorted.

2.3 Insertion Sort

Insertion Sort slightly improves the Bubble Sort algorithm, but it operates with the same computational complexity $O(n^2)$. Insertion sort sorts one item at a time bringing sorted items to their appropriate location in the sorted list. Beginning with the first element as the sorted list; the algorithm compares the neighboring element to the first element, the elements swap if the neighbor is smaller than the first element. Whether or not the element swap, the comparison continues between the next two elements in the list.

2.4 Merge Sort

Merge Sort greatly improves the process of sorting objects computationally. The divide and conquer paradigm is responsible for this computational improvement. Dividing a list of objects in to a sequence of pairs reduces the problem to $log(n)$ comparisons. Imagine a binary tree, which contains $log(n)$ nodes with respect to the number objects. Once the individual base pairs are ordered amongst themselves, the process of copying them back in order takes n computational time, which is equal to the number of objects to sort. Thus, the complexity of this sorting algorithm is $O(n * log(n))$.

2.5 Quick Sort

Quick Sort uses the same paradigm for a similar computational increase in performance. However, the sorting strategy is inverted due to a simple structural difference in the divide and conquer paradigm. Quick Sort establishes a partition element with which to organize all remaining elements in to categories. The categories are lesser, equal, and greater. The result is an n-tree where n = 3. By carefully structuring logic to take shortcuts during the sorting process and merging process, Quick Sort offers a computational performance improvement over Merge Sort. However, it operates with the same computational complexity as Merge Sort at $O(n * log(n))$.

As an aside, the Quick Sort algorithm has an interesting $2^{log(n)}$ sort property that occurs during each level of recursion. This is similar to Merge Sort, but more obvious graphically because the unsorted array appears unsorted until reaching the final two merge layers. Thus, $2^{log(n)-1}$ and $2^{log(n)}$ objects suddenly structure a pair of sorted lists and a completely sorted list respectively. Whereas Merge Sort invests time and energy into initially dividing the problem in to pairs of items; Quick Sort invests time and energy into sorting items as it divides the problem before assembling all of the items. This initial investment makes Quick Sort a faster sorting algorithm than Merge Sort.

2.6 Heap Sort

Heap Sort theoretically improves on Quick Sort. For large data sets it out performs Quick Sort and Merge Sort. Heap Sort merges the properties of Quick Sort and Merge Sort in to one algorithm. Heap Sort uses a Binary Tree for sorting data as it divides the problem. This reduces the n-Tree of

Quick Sort to Heap Sort's Binary Tree structure. Meanwhile, Merge Sort doesn't begin organizing information until dividing the problem in to its base case, which places Heap Sort at the top of the sorting algorithm list. This improvement in computational performance comes at a cost in two ways.

The first trade off is that Heap Sort is unstable. Second, the Binary Search tree structure leads to distributing information all over the computer's cache and memory, which slows down the process of sorting intermediate list sizes. These lists are large enough to gain an advantage using the divide and conquer paradigm, but small enough that information does not spill out of the processor's L1 cache, L2 cache, and RAM. Thus, in practice Heap Sort is rarely implemented because list sizes do not often leave the intermediate category. However, when object quantity and/or size is extremely large Heap Sort is the best method for sorting objects in $O(n * log(n))$ computational time.

2.7 Two Sort

Two Sort combines the computational speed of Heap Sort with the stability of Quick Sort to create a method for sorting application meta data from the Google Play Store. A function template allows the list of applications to be sorted quickly with Heap Sort, and a stable Quick Sort implementation sorts objects by a different field using a custom function template. In this case the function template selected uses the number of reviews to sort items with the same first letter. Stability requires copying items that have the same value in to a category based on their integer index, which maintains stability throughout the algorithm. The overall run time of the two sort algorithm is $O(n^2 * log(n)^2)$, but it manages memory in an unstable manner. Note: The implementation for this algorithm is unable to collect data beyond 1,000 objects due to segmentation faults, and the last 28 data points were collected in a piece wise fashion with the program crashing at 972 objects, 998 objects, and 1,000+ objects consistently. This is likely a memory management fault, and is resolvable using valgrind on a Unix based operating system.

3 Procedure

1. Store 1,000+ objects in a vector, initially unsorted.
2. Choose five sorting algorithms:
    1. Bubble Sort
    2. Selection Sort or Insertion Sort
    3. Merge Sort or Quick Sort
    4. Heap Sort
    5. Two-sort
3. For the two sort algorithm you may sort by any algorithm, then sort on a different field using a stable sorting algorithm.
4. Use each sorting algorithm to sort a vector of the first 100 objects from your dataset.
5. Record the number of reads (the number of times you retrieve and use an object from the vector) and writes (the number of times you assign an object into the vector).
6. Repeat steps 4, 5, and 6 for the following vector sizes: 200, 300, 400, 500, 600, 700, 800, 900, and 1,000.
7. Each of the five sorting algorithms should be given identical unsorted vectors to begin with.
8. Analyze the data.
9. Graph the number of reads and writes for each sorting algorithm and look at how the number of reads and writes grows when the size of the data set grows.
10. Compare and contrast the different sorting algorithms and draw conclusions about which sorting algorithms are more efficient.
11. Discuss complexities and their effects.
12. In analysis include answers to the following questions: If you need to sort a contacts list on a mobile app, which sorting algorithm(s) would you use and why? What about if you need to sort a database of 20 million client files that are stored in a datacenter in the cloud?-You must submit your source file(s), your data file(s), and your report.

4 Data

The following graphs collate cross over and large number run times for all of the sorting algorithms mentioned in the introduction.

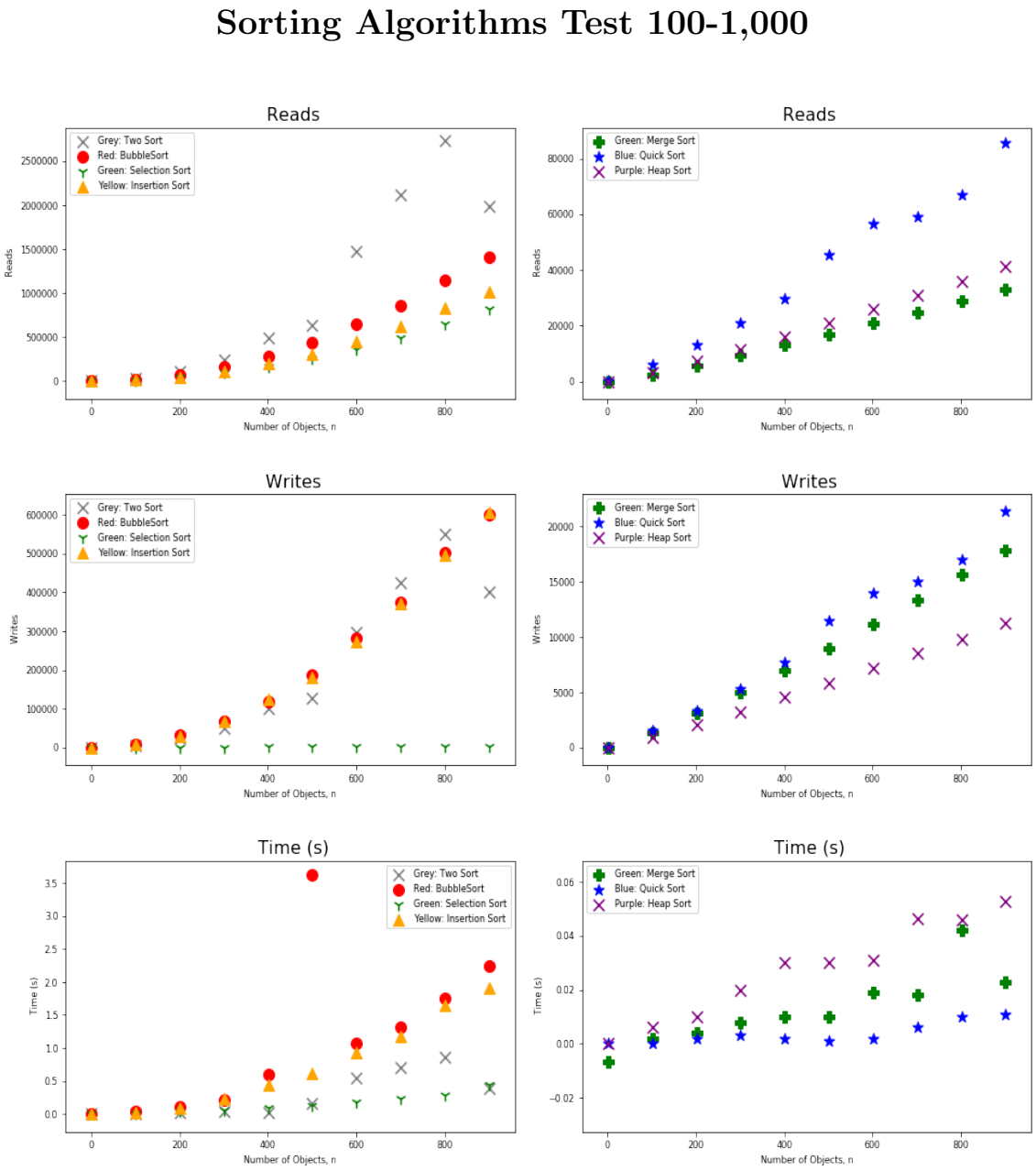

Figure 1: All of the sorting algorithms were tested with a random vector of objects ranging from 100-1,000 with a test interval of 100.

Sorting Algorithms Test 1,000-10,000

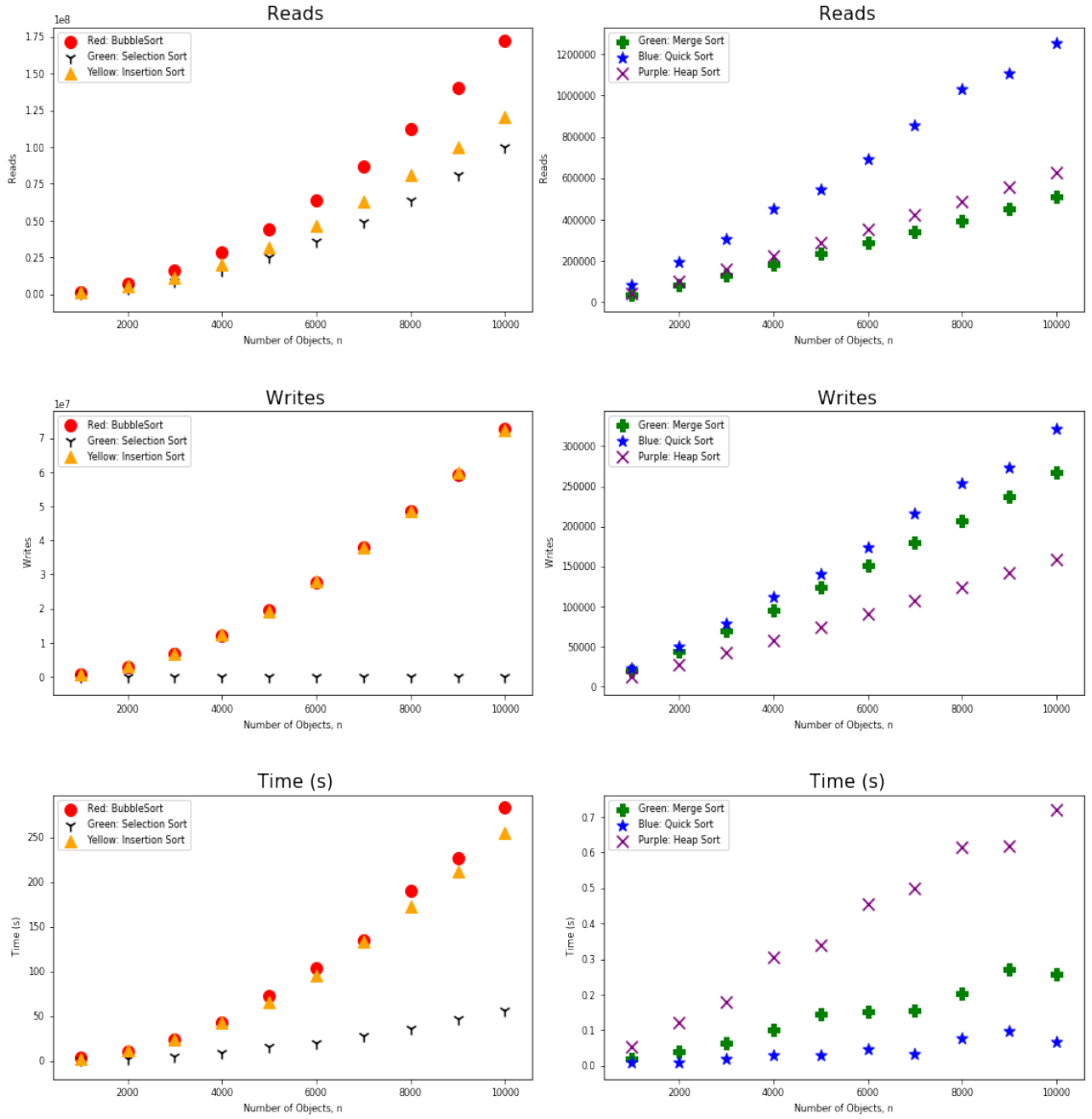

Figure 2: A second test applied large vector sizes to the same sorting algorithms to determine performance characteristics with large sets of data. In this case, random vectors contained 1,000-10,000 objects

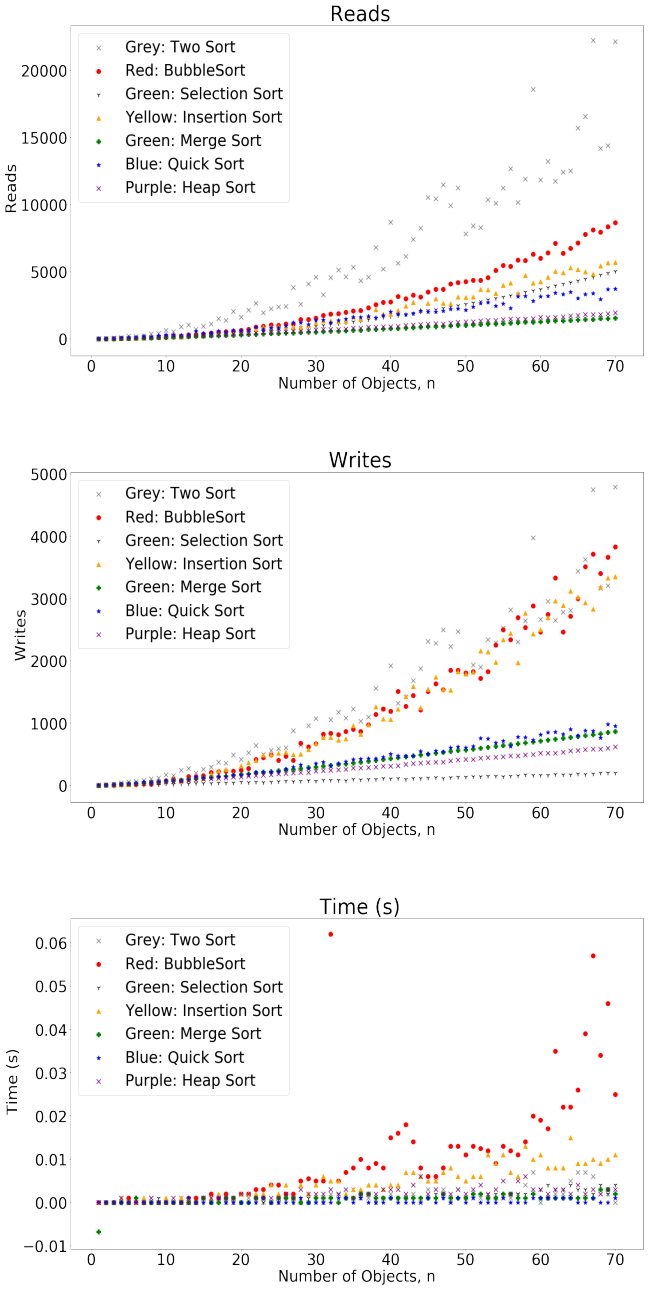

Figure 3: The cross over analysis for this graph demonstrates that Bubble Sort, Selection Sort, and Insertion Sort grow at polynomial rate, while Merge Sort, Quick Sort, and Heap Sort grow at a log-linear rate. At n = 46 objects Selection Sort is no longer a viable choice in comparison to Merge Sort, Quick Sort, and Heap Sort. This is due to the polynomial growth rate out weighing the constant time cost of running the divide and conquer programming paradigm.

5 Analysis

Sorting requires little time when objects are primitives like integers, floats, and doubles. Bucket sorting the objects is resolvable based on a constant, which is the length of their binary data. As long as the word size of the primitive is capable of entering the bus of a processor, sorting is a matter of constant time comparisons between bits of information. However, when the object is no longer a

primitive or the number is large enough that standard word lengths do not contain the object. The process of sorting these objects becomes more complex.

Testing the six sorting algorithms above reveals a significant cross over point, which is used in the Tim Sort implementation. It is the constant time cost of compiling a divide and conquer algorithm to sort objects. The cross over point observed in this lab experiment is n = 46 where Selection Sort appears just above Merge Sort and Quick Sort in the cross over analysis time graph(4.

Selection Sort took 2.000 milliseconds to complete its sort operation, whereas Merge Sort took 1 millisecond, and Quick Sort did not register its operation with the high resolution clock. This means that Quick Sort took less than 998 microseconds to sort 46 objects. When stability is not a factor Selection Sort coupled with Quick Sort are the two best sorting algorithms to use computationally.2

However, when stability is necessary Insertion Sort and Merge Sort are the best alternatives to performance. This is why Java's array sort operation uses Insertion Sort to maintain stability. When stability is not a priority Quick Sort is used. Finally, when stability is a priority Merge Sort is used as an alternative method.2

6 Conclusion

Implementing an effective sorting algorithm requires a careful analysis of performance requirements. When time is a priority unstable algorithms like Selection Sort and Quick Sort are the best choice. When energy or stability are a priority, then Merge Sort and Insertion Sort are the best alternatives to performance.

Sorting a small number of objects does not change these priorities, a contacts list requires stability because the user expects a number associated with a person to appear at the same location every time. Thus, a contacts list sorting fewer than 47 people should use Insertion Sort. However, a contacts list that grows larger than 47 should immediately switch to a Merge Sort implementation for stable sorting. None the less until sorting requires greater than 5 milliseconds a human has no expectation for the system to be responsive, and the average person has a response time of approximately 200 milliseconds4. This is based on gamer effective actions per minute, which may exceed 200+ user interface interrogations, and driver road test standards for the Department of Transportation.3

Sorting a large number of objects does not change these priorities, like 20 million client files in a cloud data center. Sorting these objects requires the initial cost of placing them in to buckets. This requires establishing acceptable parameters for the data, which creates a primitive data object for every field. This information is then hashed and indexed to reduce sorting to a constant time operation after creating the unique key associated with each field. This requires the same up front cost of indexing and hashing each field to provide a recurrent request optimization $O(n * b * log(k))$, where n is the number of objects, b is the number of symbols in the formal language, and k is the largest string size. Thus, when the number of objects is astronomically large the upfront cost of hashing and indexing all objects is an acceptable use of time.

7 Bibliography

References

[1] Kaggle: Your Home for Data Science. https://www.kaggle.com/. Accessed 24 Oct. 2019.

[2] Jon L. Bentley and M. Douglas McIlroy's "Engineering a Sort Function", Software-Practice and Experience, Vol. 23(11) P. 1249-1265 (November 1993)

[3] Blizzard Entertainment. 2012-02-21. https://starcraft2.com/en-us/ Patch 1.4.3 Now Live. Blizzard Entertainment. Accessed 2019-11-15.

[4] Koppa, Rodger. "Human Factors, Chapter 3" Federal Highway Administration, 2011 https://www.fhwa.dot.gov/publications/research/operations/tft/chap3.pdf Department of Transportation. Accessed 2019-11-15.

8 Appendix - GooglePlayApps Source Code - Project 4

8.1 Applications.h by Justin Adams

/**/

/ LICENSE**

Begin license text.

Copyright 24 October 2019 Justin C. Adams Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the * rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished * to do so, subject to the following conditions: The above copyright notice and this permission notice shall be included in all * copies or substantial portions of the Software. THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

End license text.

/**/

/ COPYRIGHT**

Begin copyright text.

Created by Justin C. Adams 18 October 2019 FILE: Algorithms.cpp DESCRIPTION: Contains all data structures and algorithms implemented by Justin C. Adams during the course of days working on computer programming. Implementation is modified to incorporate Open Source Multi-Processing from OpenMP Architecture Review Board.

End copyright text.

/**/

/***/

/ IMPLEMENTATION & STYLE**

Begin implementation text

IMPLEMENTATION

Code implementation uses OpenMP compiler contained in CygWin64 or MSys64/MinGW64

CygWin Compiler Installation Instructions:
https://www.math.ucla.edu/˜wotaoyin/windows_coding.html

MSys64/MinGW64 Installation Instructions:
https://www.math.ucla.edu/wotaoyin/windows_coding.html

STYLE

Code style based on Edgar Djikstra's Structured Programming and David Blaise's omp_hello.c FILE: omp_hello.c DESCRIPTION: OpenMP Example - Hello World - C/C++ Version In this simple example, the master thread forks a parallel region. All threads in the team obtain their unique thread number and print it. The master thread only prints the total number of threads. Two OpenMP library routines are used to obtain the number of threads and each thread's number. AUTHOR: Blaise Barney 5/99 LAST REVISED: 04/06/05

End implementation text

/***/
#ifndef GOOGLEPLAYAPPS_APPLICATIONS_H
#define GOOGLEPLAYAPPS_APPLICATIONS_H

#include ¡string¿
#include ¡vector¿
#include ¡iostream¿
#include ¡fstream¿
#include ¡iomanip¿
#include ¡sstream¿

```
using namespace std;

class Applications {

private:

    string name;
    string category;
    float rating {};
    string reviews;
    string size;
    int installs {};
```

```
    string type;
    float price {};
    string contentRating;
    string genre;
    string lastUpdated;
    string currentVer;
    string androidVer;

public:


// Default Constructor
Applications() {
    this->name = "Outdoor Adventure";
    this->category = "Offline";
    this->rating = 5.0;
    this->reviews = "0";
    this->size = "0M";
    this->installs = 0;
    this->type = "Free?";
    this->price = 0.00;
    this->contentRating = "Everyone";
    this->genre = "Pick your own adventure";
    this->lastUpdated = "Right";
    this->currentVer = "???";
    this->androidVer = "9.0";
}

// Copy constructor
Applications(const Applications &apps) {
    this->name = apps.name;
    this->category = apps.category;
    this->rating = apps.rating;
    this->reviews = apps.reviews;
    this->size = apps.size;
    this->installs = apps.installs;
    this->type = apps.type;
    this->price = apps.price;
    this->contentRating = apps.contentRating;
    this->genre = apps.genre;
    this->lastUpdated = apps.lastUpdated;
    this->currentVer = apps.currentVer;
    this->androidVer = apps.androidVer;
}

Applications(string &appName, string &appCategory, float &appRating,
string &appReviews, string &appSize, int &appInstalls,
string &appType, float &appPrice, string &appContentRating,
string &appGenres, string &appLastUpdated,
string &appCurrentVer, string &appAndroidVer) {
    this->name = appName;
    this->category = appCategory;
    this->rating = appRating;
    this->reviews = appReviews;
```

```
    this->size = appSize;
    this->installs = appInstalls;
    this->type = appType;
    this->price = appPrice;
    this->contentRating = appContentRating;
    this->genre = appGenres;
    this->lastUpdated = appLastUpdated;
    this->currentVer = appCurrentVer;
    this->androidVer = appAndroidVer;
}

/*
 * Requires: nothing
 * Modifies: nothing
 * Effects: Returns the application name
 */

string getName() const {
    return this->name;
}

/*
 * Requires: nothing
 * Modifies: nothing
 * Effects: Returns the category name
 */

string getCategory() const {
    return this->category;
}

/*
 * Requires: nothing
 * Modifies: nothing
 * Effects: Returns the application rating
 */

float getRating() const {
    return this->rating;
}

/*
 * Requires: nothing
 * Modifies: nothing
 * Effects: Returns the number of reviews
 */

string getReviews() const {
    return this->reviews;
}

/*
 * Requires: nothing
 * Modifies: nothing
```

```
 * Effects: Returns the file size
 */

string getSize() const {
    return this->size;
}

/*
 * Requires: nothing
 * Modifies: nothing
 * Effects: Returns the number of installations
 */

int getInstalls() const {
    return this->installs;
}

/*
 * Requires: nothing
 * Modifies: nothing
 * Effects: Returns the type of application license
 */

string getType() const {
    return this->type;
}

/*
 * Requires: nothing
 * Modifies: nothing
 * Effects: Returns the application price
 */

float getPrice() const {
    return this->price;
}

/*
 * Requires: nothing
 * Modifies: nothing
 * Effects: Returns the content rating for the application
 */

string getContentRating() {
    return this->contentRating;
}

/*
 * Requires: nothing
 * Modifies: nothing
 * Effects: Returns the genre of the application
 */

string getGenre() {
```

```
    return this->genre;
}

/*
 * Requires: nothing
 * Modifies: nothing
 * Effects: Returns the date the application was updated
 */

string getLastUpdated() {
    return this->lastUpdated;
}

/*
 * Requires: nothing
 * Modifies: nothing
 * Effects: Returns the current version of the application
 */

string getCurrentVersion() {
    return this->currentVer;
}

/*
 * Requires: nothing
 * Modifies: nothing
 * Effects: Returns the android version the application is designed for
 */

string getAndroidVersion() {
    return this->androidVer;
}

/*
 * Requires: @param appName - String containing the new name for the application
 * Modifies: @field app - String containing the old name of the application
 * Effects: Sets the application name
 */

void setName(string &appName){
    this->name = appName;
}

/*
 * Requires: @param appCategory - String containing the new category
 for the application
 * Modifies: @field category - String containing the old category of the application
 * Effects: Sets the application category
 */

void setCategory(string &appCategory){
    this->category = appCategory;
}
```

```
/*
 * Requires: @param appRating - String containing the new rating value
 for the application
 * Modifies: @field rating - String containing the old rating of the application
 * Effects: Sets the application rating
 */

void setRating(float &appRating) {
    this->rating = appRating;
}

/*
 * Requires: @param appReviews - String containing the new number
 for application reviews
 * Modifies: @field reviews - String containing the old number of application reviews
 * Effects: Sets the application reviews
 */

void setReviews(string &appReviews) {
    this->reviews = appReviews;
}

/*
 * Requires: @param appSize - String containing the new file size
 of the application
 * Modifies: @field size - String containing the old application size
 * Effects: Sets the application file size
 */

void setSize(string &appSize){
    this->size = appSize;
}

/*
 * Requires: @param appReviews - String containing the new number
 of application installs
 * Modifies: @field rating - String containing the old number of application installs
 * Effects: Sets the number of installations
 */

void setInstalls(int &appInstalls){
    this->installs = appInstalls;
}

/*
 * Requires: @param appReviews - String containing the license of the application
 * Modifies: @field rating - String containing the license of the application
 * Effects: Sets the application license
 */


void setType(string &appType){
    this->type = appType;
}
```

```
/*
 * Requires: @param appReviews - String containing the price of the application
 * Modifies: @field rating - String containing the price of the application
 * Effects: Sets the application price
 */

void setPrice(float &appPrice){
    this->price = appPrice;
}

/*
 * Requires: @param appContentRating - String containing the new content
 rating of the application
 * Modifies: @field rating - String containing the old content
 rating of the application
 * Effects: Sets the application rating
 */

void setContentRating(string &appContentRating) {
    this->contentRating = appContentRating;
}

/*
 * Requires: @param appGenre - String containing the new genre of the application
 * Modifies: @field appGenre - String containing the old genre of the application
 * Effects: Sets the application genre
 */

void setGenre(string &appGenre) {
    this->genre = appGenre;
}

/*
 * Requires: @param appUpdated - String containing the new update date
 * Modifies: @field lastUpdated - String containing the last time
 the application was updated
 * Effects: Sets the application update date
 */

void setLastUpdated(string &appUpdated) {
    this->lastUpdated = appUpdated;
}

/*
 * Requires: @param appUpdated - String containing the new application
 version number (int.int.int...etc format)
 * Modifies: @field lastUpdated - String containing the old application
version number
 * Effects: Sets the application version number
 */

void setCurrentVersion(string &appVersion) {
    this->currentVer = appVersion;
```

```
}

/*
 * Requires: @param appUpdated - String containing the new android
 version number (int.int.int...etc format)
 * Modifies: @field lastUpdated - String containing the old android version number
 * Effects: Sets the android version number
 */

void setAndroidVersion(string &appAndroidVer) {
    this->androidVer = appAndroidVer;
}

/*
 *  Requires:   @param &lhs - Appplication to compare in less than
 equality operator.
 *              @param &rhs - Application to compare in less than
 equality operator.
 *  Modifies:   nothing
 *  Effects:    Returns true if the left hand side application name appears
first alphabetically.
 */

bool friend operator < (const Applications &lhs, const Applications &rhs) {
    return (lhs.getName() < rhs.getName());
}

/*
 *  Requires:   @param &lhs - Appplication to compare in less than equality operator.
 *              @param &rhs - Application to compare in less than equality operator.
 *  Modifies:   nothing
 *  Effects:    Returns true if the right hand side application name appears
 first alphabetically.
 */

bool friend operator > (const Applications &lhs, const Applications &rhs) {
    return (lhs.getName() > rhs.getName());
}

/*
 *  Requires:   @param &lhs - Appplication to compare in less than equality operator.
 *              @param &rhs - Application to compare in less than equality operator.
 *  Modifies:   nothing
 *  Effects:    Returns true if the left hand side application name appears
 first alphabetically.
 */

bool friend operator == (const Applications &lhs, const Applications &rhs) {
    return (lhs.getName() == rhs.getName());
}

/*
 * Requires: @param appUpdated - String containing the new android version number
 (int.int.int...etc format)
```

```
 * Modifies: @field lastUpdated - String containing the old android version number
 * Effects: Sets the android version number
 */

friend ostream& operator << (ostream &travelingSalesman, Applications app) {
    travelingSalesman << setw(22) << app.getName().substr(0,20);
    travelingSalesman << setw(15) << app.getCategory().substr(0, 13);
    travelingSalesman << setw(8) << app.getRating();
    travelingSalesman << setw(8) << app.getReviews();
    travelingSalesman << setw(5) << app.getSize();
    travelingSalesman << setw(10) << app.getInstalls();
    travelingSalesman << setw(10) << app.getType();
    travelingSalesman << setw(6) << app.getPrice();
    travelingSalesman << setw(10) << app.getContentRating().substr(0, 8);
    travelingSalesman << setw(6) << app.getGenre().substr(0, 4);
    travelingSalesman << setw(9) << app.getLastUpdated();
    travelingSalesman << setw(5) << app.getCurrentVersion().substr(0, 3);
    travelingSalesman << setw(5) << app.getAndroidVersion().substr(0, 3);
    travelingSalesman << endl;
    return travelingSalesman;
}

   };

/*
* Requires: @param vector<Applications> applications - Vector passed by reference
from main function to encapsulate app data.
* Modifies: nothing
* Effects: Calculates the value of an application based on price, installs,
and rating.
*/

vector<float> applicationValueCalculator(vector<Applications> &applications) {
    vector<float> appValue;
    float price;
    float applicationValue;
    for(Applications app : applications) {
        price = app.getPrice();
        if(price <= 0) {
            applicationValue = app.getInstalls()*app.getRating();
        }
        else {
            applicationValue = app.getInstalls()*app.getRating()/price*100;
        }
        appValue.emplace_back(applicationValue);
        cout << "Application: " << setw(20) << app.getName().substr(0,20)
        << "Calculated Value (Installs*Rating/Price): " << applicationValue
        << endl;
    }
    return appValue;
}


/*
* Requires: @param &fileName - String containing filename to access
```

```
in the parent directory,
*           @param vector<Applications> applications - Vector passed by reference
from main function to encapsulate app data.
* Modifies: @field allfields - App, Category, Rating, Reviews, Size, Installs,
Type, Price, Content Rating, Genres, Last Updated, Current Ver, Android Ver
* Effects: Creates and stores application data
*/
```

void readApplicationsFromFile(string &fileName, vector &applications) { string applicationName; string categoryName; float applicationRating; string applicationReviews; string fileSize; int applicationInstalls; string applicationType; float applicationPrice; string applicationContentRating; string applicationGenre; string applicationLastUpdated; string currentVersion; string androidVersion; string header; char comma; string error;

```
    ifstream appStream;

    appStream.open("../" + fileName);


    getline(appStream, header);

    while(appStream && appStream.peek() != EOF) {
        getline(appStream, applicationName, ',');
        if(!appStream){
            appStream >> error;
            cout << error;
        }
        getline(appStream, categoryName, ',');
        if(!appStream){
            appStream >> error;
            cout << error;
        }
        appStream >> applicationRating;
        appStream >> comma;
        if(!appStream){
            appStream >> error;
            cout << error;
        }
        getline(appStream, applicationReviews, ',');
        getline(appStream, fileSize, ',');
        if(!appStream){
            appStream >> error;
            cout << error;
        }
        appStream >> applicationInstalls;
        appStream >> comma;
        if(!appStream){
            appStream >> error;
            cout << error;
        }
        getline(appStream, applicationType, ',');
        if(!appStream){
            appStream >> error;
            cout << error;
        }
```

```
        appStream >> applicationPrice;
        appStream >> comma;
        if(!appStream){
            appStream >> error;
            cout << error;
        }
        getline(appStream, applicationContentRating, ',');
        if(!appStream){
            appStream >> error;
            cout << error;
        }
        getline(appStream, applicationGenre, ',');
        if(!appStream){
            appStream >> error;
            cout << error;
        }
        getline(appStream, applicationLastUpdated, ',');
        if(!appStream){
            appStream >> error;
            cout << error;
        }
        getline(appStream, currentVersion, ',');
        if(!appStream){
            appStream >> error;
            cout << error;
        }
        getline(appStream, androidVersion);
        if(!appStream){
            appStream >> error;
            cout << error;
        }
        applications.emplace_back(Applications(applicationName, categoryName,
        applicationRating, applicationReviews, fileSize,
        applicationInstalls, applicationType, applicationPrice,
        applicationContentRating, applicationGenre, applicationLastUpdated,
        currentVersion, androidVersion));
    }
    appStream.close();
}

/*
* Requires: googleplaystore.csv in parent directory
* Modifies: nothing
* Effects: Tests all member functions and global functions of this class.
*/

void testApplicationsClass() {
    string fish = "herring";
    int weight = 20;
    float value = 1.0;
    string size = "20lb";
    string date = "9/15/2008";
    cout <<  "Testing the default constructor" << endl;
    Applications newApplication;
```

```
cout << "Testing the overloaded operator" << endl;
cout << left << newApplication;
cout << "Testing read file system." << endl;
vector<Applications> testVector;
string filename = "googleplaystore.csv";
readApplicationsFromFile(filename, testVector);
cout << testVector.size() << endl;
if(testVector.size() >= 10841) {
    cout << "Test vector success. All applications from Google Play Store
    loaded." << endl;
}
else {
    cout << "Test failed. Number of applications loaded is less than
    10842." << endl;
    int readInFailurPoint = testVector.size()-1;
    cout << "Loading object failure detected at line" <<
    readInFailurPoint << endl;
}
cout << "Application value calculator." << endl;
applicationValueCalculator(testVector);
cout << "Testing getters and setters on test application with a
20 lb fish." << endl;
newApplication.setName(fish);
cout << "Name the " << newApplication.getName() << endl;
newApplication.setCategory(fish);
cout << "Category please " << newApplication.getCategory() << endl;
newApplication.setRating(value);
cout << "Rate the smell " << newApplication.getRating() << endl;
newApplication.setReviews(fish);
cout << "The reviews for your rating are in: " << newApplication.getReviews()
<< endl;
newApplication.setSize(size);
cout << "If you don't believe me get the size: " << newApplication.getSize()
<< endl;
newApplication.setInstalls(weight);
cout << "It is hard to download a fish, but you can make " <<
newApplication.getInstalls() << endl;
newApplication.setType(size);
cout << "How much sushi could a " << newApplication.getType() << endl;
newApplication.setPrice(value);
cout << "Pound fish fetch at market? " << newApplication.getPrice() << endl;
newApplication.setContentRating(fish);
cout << newApplication.getContentRating();
newApplication.setGenre(fish);
cout << " " << newApplication.getGenre();
newApplication.setLastUpdated(date);
cout << "Date purchased: " << newApplication.getLastUpdated();
newApplication.setCurrentVersion(size);
cout << "Fish version: " << newApplication.getCurrentVersion();
newApplication.setAndroidVersion(size);
cout << "Market version: " << newApplication.getAndroidVersion();
// Output 1000+1 applications (for good measure)
cout << setw(22) << left << "Application Name" << setw(15) << "Category"
<< setw(8) << "Rating" << setw(8) << "Reviews" <<
```

```
    setw(5) << "Size"
    << setw(10) << "Downloads" << setw(10) << "License" << setw(6) << "Price"
    << setw(10) << "Content" << setw(6) << "Genre" << setw(9) << "vDate"
    << setw(5) << "vApp" << setw(5) << "vDroid" << endl;
    cout << setw(120) << setfill('-') << '-' << setfill(' ') << endl;
    for(int i = 0; i < 1001; i++) {
        cout << testVector[i] << endl;
    }
}

\#endif //GOOGLEPLAYAPPS\_APPLICATIONS\_H

\newpage
\hypertarget{algorithms.h-by-justin-adams}{%
\subsection{Algorithms.h by Justin
Adams}\label{algorithms.h-by-justin-adams}}

\paragraph{/*************************************************************************
\paragraph{/** LICENSE} \paragraph{Begin license text.} \paragraph{Copyright 24 October 2019 Ju
\paragraph{End license text.} \paragraph{/*************************************************
\paragraph{/** COPYRIGHT} \paragraph{Begin copyright text.} \paragraph{Created by Justin C. Ada
\paragraph{/*************************************************************************
\newpage
\paragraph{/*************************************************************************
\paragraph{/** IMPLEMENTATION \& STYLE}
\paragraph{Begin implementation text}
\paragraph{IMPLEMENTATION Code implementation uses OpenMP compiler contained in CygWin64 or MSy
\_cygwin.html https://www.math.ucla.edu/wotaoyin/windows\_coding.html}
\paragraph{STYLE Code style based on David Blaise omp\_hello.c FILE:
omp\_hello.c DESCRIPTION: OpenMP Example - Hello World - C/C++ Version In this simple example,
and each thread's number. AUTHOR: Blaise Barney 5/99 LAST REVISED: 04/06/05}
\paragraph{End implementation text}
\paragraph{/*************************************************************************

\begin{verbatim}

#include "BinarySearchTree.h"
#include "AVLTree.h"
#include "SplayTree.h"
#include "Applications.h"

#include <random>
#include <chrono>
#include <iostream>
#include <fstream>
using namespace std;

template class Algorithms {

private:

    int start = 0;
    int end = 255;
    int repeats = 2;
```

```
    string filename = "../numbers.txt";
    bool verbose = false;

public:

    // Default Constructor
    /* Default constructor. This does contain any parameters. */
    Algorithms<T>() {
        /* Initialize field start to 0. */
        this->start = 0;
        /* Initialize field end to 255 (INT8_MAX = 2^8 - 1). */
        this->end = 255;
        /* Initialize field repeats to 2. */
        this->repeats = 2;
        /* Initialize string filename to "../numbers.txt". */
        this->filename = "../numbers.txt";
    }

    /*
     *  Requires:   @param start - Integer start for the beginning of array data.
     *  Modifies:   @field start - Integer start for the beginning of array data.
     *  Effects:    Algorithms initializes required parameters for all functions.
    */

    /* Integer start initialization constructor. This contains the parameter int s,
    which initializes start. */
    explicit Algorithms<T>(int s) {
        /* Initialize field start to s. */
        this->start = s;
        /* Initialize field end to 255 + s (INT8_MAX = 2^8 - 1 + s). */
        this->end = 255 + s;
        /* Initialize field repeats to 2. */
        this->repeats = 2;
        /* Initialize string filename to "../numbers.txt". */
        this->filename = "../numbers.txt";
    }

    /*
     *  Requires:   @param filename - String filename for the end of array data.
     *  Modifies:   @field filename - String filename for the end of array data.
     *  Effects:    Algorithms initializes required parameters for all functions.
    */

    /* String filename initialization constructor. This contains the parameter
    string &f, which initializes filename. */
    explicit Algorithms<T>(string &f) {
        /* Initialize field start to 0. */
        this->start = 0;
        /* Initialize field end to 255 (INT8_MAX = 2^8 - 1). */
        this->end = 255;
        /* Initialize field repeats to 2. */
        this->repeats = 2;
        /* Initialize string filename to f. */
        this->filename = f;
```

```
}

/*
 *  Requires:   @param start - Integer start for the beginning of array data.
 *              @param end - Integer end for the end of array data.
 *  Modifies:   nothing
 *  Effects:    Algorithms initializes required parameters for all functions.
*/

/* Integer start and end initialization constructor.
*  This contains the parameter int s and int e, which initializes start
and end. */
Algorithms<T>(int s, int e) {
    /* Initialize field start to s. */
    this->start = s;
    /* Initialize field end to e. */
    this->end = e;
    /* Initialize field repeats to 2. */
    this->repeats = 2;
    /* Initialize string filename to "../numbers.txt". */
    this->filename = "../numbers.txt";
}

/*
 *  Requires:   @param start - Integer start for the beginning of array data.
 *              @param end - Integer end for the end of array data.
 *  Modifies:   nothing
 *  Effects:    Algorithms initializes required parameters for all functions.
*/

/* Integer start, end initialization constructor.
*  This contains the parameter int s, int e, string &f which initializes start,
end, and filename. */
Algorithms<T>(int s, int e, string &f) {
    /* Initialize field start to s. */
    this->start = s;
    /* Initialize field end to e. */
    this->end = e;
    /* Initialize field repeats to 2. */
    this->repeats = 2;
    /* Initialize string filename to f. */
    this->filename = f;
}

/*
 *  Requires:   @param start - Integer start for the beginning of array data.
 *              @param end - Integer end for the end of array data.
 *              @param repeats - Integer repeats sets the number of duplicates
 in array data.
 *  Modifies:   nothing
 *  Effects:    Algorithms initializes required parameters for all functions.
*/

/* Integer start, end repeats constructor.
```

```
*  This contains the parameter int s, int e, int r which initializes start, end,
and repeats. */
Algorithms<T>(int s, int e, int r) {
    /* Initialize field start to s. */
    this->start = s;
    /* Initialize field end to e. */
    this->end = e;
    /* Initialize field repeats to r */
    this->repeats = r;
    /* Initialize string filename to "../numbers.txt". */
    this->filename = "../numbers.txt";
}


/*
 *  Requires:   @param start - Integer start for the beginning of array data.
 *              @param end - Integer end for the end of array data.
 *              @param repeats - Integer repeats sets the number of duplicates
 *              in array data for the duplicatesIntegerArray function.
 *              @param filename - String filename sets the file name to write
 array data to.
 *  Modifies:   nothing
 *  Effects:    Algorithms initializes required parameters for all functions.
*/

/* Integer start, end, repeats, and filename constructor.
*  This contains the parameter int s, int e, int r, and string &f.
*  The constructor initializes start, end, repeats, and filename. */
Algorithms<T>(int s, int e, int r, string &f) {
    /* Initialize field start to s. */
    this->start = s;
    /* Initialize field end to e. */
    this->end = e;
    /* Initialize field repeats to r */
    this->repeats = r;
    /* Initialize string filename to f. */
    this->filename = f;
}

/*
 *  Default Deconstructor
 */

~Algorithms() = default;

/*
 *  Requires:   nothing
 *  Modifies:   nothing
 *  Effects:    Tests the Algorithms class.
*/

/*  Atkin, A. O. L., and D. J. Bernstein. Prime Sieves Using Binary Quadratic
Forms. Mathematics of Computation vol. 73, no. 246, Dec. 2003, pp. 102331.
Crossref, doi:10.1090/S0025-5718-03-01501-1.*/
```

```
bool testAlgorithms() {

    /** Initialize variables for the Algorithms class test suite. */

    /* Primitive data type instances. */
    /* Retrieve currentStart. */
    int currentStart = getStart();
    /* Retrieve currentEnd. */
    int currentEnd = getEnd();
    /* Retrieve currentRepeats. */
    int currentRepeats = getRepeats();
    /* Retrieve currentFilename. */
    string currentFilename = getFilename();

    /* Set testing boolean passed to true. */
    bool passed = true;
    /* Set testing integer newStart to -1. */
    int newStart = -1;
    /* Set testing integer newEnd to 256. */
    int newEnd = 256;
    /* Set testing integer newRepeats to 3. */
    int newRepeats = 3;
    /* Set testing bool newBool to true. */
    bool newBool = false;
    /* Reset verbose trigger. */
    bool resetVerbose = false;

    /* Derived data type instances. */
    /* Set newFilename to "../n.txt". */
    string newFilename = "../n.txt";
    /* Create a filename comparison string object. */
    string getFilename;
    /* Create a constant integer to initialize arrays. */
    const static int size = 257;
    /* Create an array to store an ordered sequence of integers. */
    vector<int> sequence;
    /* Create an array to store a random sequence of integers. */
    vector<int> random;
    /* Create an array to store a series of duplicate integers. */
    vector<int> duplicates;

    /** Verify getters and setters pass testing. */


    /* Check if verbose mode is on. */
    if(this->getVerbose()) {
        /* Declare to the user the current test. */
        cout << "Testing getVerbose." << endl;
        /* Set verbose mode to a value to verify testing. */
        this->setVerbose(newBool);
        /* Set reset verbose trigger. */
        resetVerbose = true;
    }
```

```
/* Check if getVerbose is successful. */
if(!this->getVerbose()) {
    /* Check if verbose mode is on. */
    if(resetVerbose) {
        /* Report getVerbose success. */
        cout << "getVerbose successful." << endl;
    }
}
    /* Report getVerbose failure. */
else {
    if(resetVerbose) {
        /* Declare to the user getVerbose failure. */
        cout << "getVerbose failure." << endl;
    }
    /* Update test status variable to false. */
    passed = false;
}


/* Check if verbose mode is on. */
if(this->getVerbose()) {
    /* Declare to the user the current test. */
    cout << "Testing setVerbose." << endl;
}
/* Change newBool to true. */
newBool = !newBool;
/* Set verbose to newBool = true. */
this->setVerbose(newBool);
/* Check if setVerbose is successful. */
if(this->getVerbose()) {
    /* Check if verbose mode is on. */
    if(resetVerbose) {
        /* Report setVerbose success. */
        cout << "setVerbose successful." << endl;
    }
    /* Check if resetVerbose was triggered. */
    if(!resetVerbose) {
        /* Set verbose back to original setting. */
        this->setVerbose(resetVerbose);
    }
}
    /* Report setVerbose failure. */
else {
    /* Declare to the user setVerbose failure. */
    cout << "setVerbose failure." << endl;
    /* Update test status variable to false. */
    passed = false;
}



/* Check if verbose mode is on. */
if(this->getVerbose()) {
    /* Declare to the user the current phase of testing. */
```

```
    cout << "Testing getters and setters." << endl;
}

/* Check if verbose mode is on. */
if(this->getVerbose()) {
    /* Declare to the user the current test. */
    cout << "Testing getStart." << endl;
}
/* Check if getStart is successful. */
if(this->getStart() == currentStart) {
    /* Check if verbose mode is on. */
    if(this->getVerbose()) {
        /* Report getStart success. */
        cout << "getStart successful." << endl;
    }
}
    /* Report getStart failure. */
else {
    /* Check if verbose mode is on. */
    if(this->getVerbose()) {
        /* Declare to the user getStart failure. */
        cout << "getStart failure." << endl;
    }
    /* Update test status variable to false. */
    passed = false;
}

/* Check if verbose mode is on. */
if(this->getVerbose()) {
    /* Declare to the user the current test. */
    cout << "Testing setStart." << endl;
}
/* Set start to newStart = -1. */
this->setStart(newStart);
/* Check if setStart is successful. */
if(this->getStart() == newStart) {
    /* Check if verbose mode is on. */
    if(this->getVerbose()) {
        /* Report setStart success. */
        cout << "setStart successful." << endl;
    }
}
    /* Report setStart failure. */
else {
    /* Check if verbose mode is on. */
    if(this->getVerbose()) {
        /* Declare to the user setStart failure. */
        cout << "setStart failure." << endl;
    }
    /* Update test status variable to false. */
    passed = false;
}

/* Check if verbose mode is on. */
```

```
if(this->getVerbose()) {
    /* Declare to the user the current test. */
    cout << "Testing getEnd." << endl;
}
/* Check if getEnd is successful. */
if(this->getEnd() == currentEnd) {
    /* Check if verbose mode is on. */
    if(this->getVerbose()) {
        /* Report getEnd success. */
        cout << "getEnd successful." << endl;
    }
}
    /* Report getEnd failure. */
else {
    /* Check if verbose mode is on. */
    if(this->getVerbose()) {
        /* Declare to the user getEnd failure. */
        cout << "getEnd failure." << endl;
    }
    /* Update test status variable to false. */
    passed = false;
}

/* Check if verbose mode is on. */
if(this->getVerbose()) {
    /* Declare to the user the current test. */
    cout << "Testing setEnd." << endl;
}
/* Set start to newEnd = 256. */
this->setEnd(newEnd);
/* Check if setEnd is successful. */
if(this->getEnd() == newEnd) {
    /* Check if verbose mode is on. */
    if(this->getVerbose()) {
        /* Report setEnd success. */
        cout << "setEnd successful." << endl;
    }
}
    /* Report setEnd failure. */
else {
    /* Check if verbose mode is on. */
    if(this->getVerbose()) {
        /* Declare to the user setEnd failure. */
        cout << "setEnd failure." << endl;
    }
    /* Update test status variable to false. */
    passed = false;
}

/* Check if verbose mode is on. */
if(this->getVerbose()) {
    /* Declare to the user the current test. */
    cout << "Testing getRepeats." << endl;
}
```

```
/* Check if getRepeats is successful. */
if(this->getRepeats() == currentRepeats) {
    /* Check if verbose mode is on. */
    if(this->getVerbose()) {
        /* Report getRepeats success. */
        cout << "getRepeats successful." << endl;
    }
}
    /* Report getRepeats failure. */
else {
    /* Check if verbose mode is on. */
    if(this->getVerbose()) {
        /* Declare to the user getRepeats failure. */
        cout << "getRepeats failure." << endl;
    }
    /* Update test status variable to false. */
    passed = false;
}

/* Check if verbose mode is on. */
if(this->getVerbose()) {
    /* Declare to the user the current test. */
    cout << "Testing setRepeats." << endl;
}
/* Set repeats to newRepeats = 3. */
this->setRepeats(newRepeats);
/* Check if setRepeats is successful. */
if(this->getRepeats() == newRepeats) {
    /* Check if verbose mode is on. */
    if(this->getVerbose()) {
        /* Report setRepeats success. */
        cout << "setRepeats successful." << endl;
    }
}
    /* Report setRepeats failure. */
else {
    /* Check if verbose mode is on. */
    if(this->getVerbose()) {
        /* Declare to the user setRepeats failure. */
        cout << "setRepeats failure." << endl;
    }
    /* Update test status variable to false. */
    passed = false;
}

/* Check if verbose mode is on. */
if(this->getVerbose()) {
    /* Declare to the user the current test. */
    cout << "Testing getFilename." << endl;
}
/* Set filename to newFilename = "../numbers.txt". */
this->getFilename();
/* Check if getFilename is successful. */
if(this->getFilename() == currentFilename) {
```

```
    /* Check if verbose mode is on. */
    if(this->getVerbose()) {
        /* Report getFilename success. */
        cout << "getFilename successful." << endl;
    }
}
    /* Report getFilename failure. */
else {
    /* Check if verbose mode is on. */
    if(this->getVerbose()) {
        /* Declare to the user getFilename failure. */
        cout << "getFilename failure." << endl;
    }
    /* Update test status variable to false. */
    passed = false;
}

/* Check if verbose mode is on. */
if(this->getVerbose()) {
    /* Declare to the user the current test. */
    cout << "Testing setFilename." << endl;
}
/* Set repeats to newFilename = "../n.txt". */
this->setFilename(newFilename);
/* Check if setFilename is successful. */
if(this->getFilename() == newFilename) {
    /* Check if verbose mode is on. */
    if(this->getVerbose()) {
        /* Report setFilename success. */
        cout << "setFilename successful." << endl;
    }
}
    /* Report setFilename failure. */
else {
    /* Check if verbose mode is on. */
    if(this->getVerbose()) {
        /* Declare to the user setFilename failure. */
        cout << "setFilename failure." << endl;
    }
    /* Update test status variable to false. */
    passed = false;
}



/** Verify class constructors pass testing. */

/* Check if verbose mode is on. */
if(this->getVerbose()) {
    /* Declare to the user the current phase of testing. */
    cout << "Testing constructors." << endl;
}
```

```
/* Check if verbose mode is on. */
if(this->getVerbose()) {
    /* Declare to the user the current test. */
    cout << "Testing default constructor." << endl;
}
/* Generate new Algorithms object using the default constructor. */
auto b = Algorithms();
/* Check if fields initialize successfully. */
if(!b.getVerbose()) {
    /* Check if verbose mode is on. */
    if(this->getVerbose()) {
        /* Declare to the user successful construction. */
        cout << "Default constructor success." << endl;
    }
}
    /* Fields did not initialize successfully. */
else {
    /* Check if verbose mode is on. */
    if(this->getVerbose()) {
        /* Declare to the user construction failure. */
        cout << "Default constructor failure." << endl;
    }
    /* Set test passed variable to false. */
    passed = false;
}


/* Check if verbose mode is on. */
if(this->getVerbose()) {
    /* Declare to the user the current test. */
    cout << "Testing constructor integer start." << endl;
}
/* Generate new Algorithms object using the start initialization
constructor. */
auto c = Algorithms(newStart);
/* Check if fields initialize successfully. */
if(c.getStart() == newStart) {
    /* Check if verbose mode is on. */
    if(this->getVerbose()) {
        /* Declare to the user successful construction. */
        cout << "Constructor integer start initialization success."
        << endl;
    }
}
    /* Fields did not initialize successfully. */
else {
    /* Check if verbose mode is on. */
    if(this->getVerbose()) {
        /* Declare to the user construction failure. */
        cout << "Constructor integer start initialization failure."
        << endl;
    }
    /* Set test passed variable to false. */
```

```
    passed = false;
}


/* Check if verbose mode is on. */
if(this->getVerbose()) {
    /* Declare to the user the current test. */
    cout << "Testing constructor string filename." << endl;
}
/* Generate new Algorithms object using the start
initialization constructor. */
auto d = Algorithms(newFilename);
/* Check if fields initialize successfully. */
if(d.getFilename() == newFilename) {
    /* Check if verbose mode is on. */
    if(this->getVerbose()) {
        /* Declare to the user successful construction. */
        cout << "Constructor string filename initialization success."
        << endl;
    }
}
    /* Fields fail to initialize. */
else {
    /* Set test passed variable to false. */
    passed = false;
    /* Check if verbose mode is on. */
    if(this->getVerbose()) {
        /* Declare to the user construction fail. */
        cout << "Constructor string filename initialization failure."
        << endl;
    }
}



/* Check if verbose mode is on. */
if(this->getVerbose()) {
    /* Declare to the user the current test. */
    cout << "Testing constructor integer field start and
    end initialization." << endl;
}
/* Generate new Algorithms object using the integer field start and end
initialization constructor. */
auto e = Algorithms(newStart, newEnd);
/* Check if fields initialize successfully. */
if(e.getStart() == -1 && e.getEnd() == newEnd) {
    if(this->getVerbose()) {
        /* Declare to the user successful construction. */
        cout << "Constructor integer field start and end initialization
        success." << endl;
    }
}
    /* Verbose mode is off. */
else {
```

```
    if(this->getVerbose()) {
        /* Declare to the user successful construction. */
        cout << "Constructor integer field start and end initialization
        failure." << endl;
    }
    /* Set test passed variable to false. */
    passed = false;
}


/* Check if verbose mode is on. */
if(this->getVerbose()) {
    /* Declare to the user the current test. */
    cout << "Testing integer field start, end, and string filename
    initialization constructor." << endl;
}
/* Generate new Algorithms object using the integer field start, end, and
string filename initialization constructor. */
auto f = Algorithms(newStart, newEnd, newFilename);
/* Check if fields initialize successfully. */
if(f.getStart() == newStart && f.getEnd() == newEnd && f.getFilename() ==
newFilename) {
    if(this->getVerbose()) {
        /* Declare to the user successful construction. */
        cout << "Constructor integer field start, end, and string
        filename initialization success." << endl;
    }
}
    /* Verbose mode is off. */
else {
    if(this->getVerbose()) {
        /* Declare to the user successful construction. */
        cout << "Constructor integer field start, end, and string filename
        initialization failure." << endl;
    }
    /* Set test passed variable to false. */
    passed = false;
}


/* Check if verbose mode is on. */
if(this->getVerbose()) {
    /* Declare to the user the current test. */
    cout << "Testing integer field start, end, and repeats initialization
    constructor." << endl;
}
/* Generate new Algorithms object using the integer field start, end,
and repeats initialization constructor. */
auto g = Algorithms(newStart, newEnd, newRepeats);
/* Check if fields initialize successfully. */
if(g.getStart() == newStart && g.getEnd() == newEnd && g.getRepeats() ==
newRepeats) {
    /* Check if verbose mode is on. */
    if(this->getVerbose()) {
```

```
        /* Declare to the user successful construction. */
        cout << "Constructor integer field start, end, and
        repeats initialization success." << endl;
    }
}
    /* Set passed variable to false */
else {
    /* Check if verbose mode is on. */
    if(this->getVerbose()) {
        /* Declare to the user failed construction. */
        cout << "Constructor integer field start, end, and repeats
        initialization failure." << endl;
    }
    /* Set test passed variable to false. */
    passed = false;
}


/* Check if verbose mode is on. */
if(this->getVerbose()) {
    /* Declare to the user the current test. */
    cout << "Testing integer field start, end, and repeats initialization
    constructor." << endl;
}
/* Generate new Algorithms object using the integer field start, end, and
repeats initialization constructor. */
auto h = Algorithms(newStart, newEnd, newRepeats, newFilename);
/* Check if fields initialize successfully. */
if(h.getStart() == newStart && h.getEnd() == newEnd && h.getRepeats() ==
newRepeats && h.getFilename() == newFilename) {
    /* Check if verbose mode is on. */
    if(this->getVerbose()) {
        /* Declare to the user successful construction. */
        cout << "Constructor integer field start, end, repeats, and string
        filename initialization success." << endl;
    }
}
    /* Set passed variable to false */
else {
    /* Check if verbose mode is on. */
    if(this->getVerbose()) {
        /* Declare to the user failed construction. */
        cout << "Constructor integer field start, end, repeats and string
        filename initialization  fail." << endl;
    }
    /* Set test passed variable to false. */
    passed = false;
}



/** Verify algorithms pass testing criteria. */

/* Fill sequence with an ordered set of integers from start to end. */
```

```
    sequence = integerVector(start, end);
    /* Fill random with an unordered set of integers without duplicates
    selected from start to end. */
    random = randomIntegerVector(start, end);
    /* Fill duplicates with an ordered set of integers with repeats number of
    duplicates from start to end. */
    duplicates = duplicatesIntegerVector(start, end, repeats);
    /* Fill primes with an ordered set of prime integers. */
    vector<int> primes = sieveOfAtkin(2000);
    /* Print out the three integer arrays to terminal. */
    outputTerminal(sequence, random, duplicates, primes, size);
    /* Write the information to a file. */
    writeArray(filename, sequence, random, duplicates, primes, size);

    /** Check if the algorithms class passed all
    test cases. */
    if(passed && this->getVerbose()) {
        /* Declare to the user that the algorithms class passed all test
        cases. */
        cout << "Algorithms class passed all test cases." << endl;
    }
    return passed;
}


/*
 *  Requires:   nothing
 *  Modifies:   nothing
 *  Effects:    @return start - Returns the integer start for the
 algorithms object.
*/

/* function int getStart() returns a copy of the integer that stores the
start integer value for this class. */
int getStart() {
    /* Integer primitive copy constructor of the field start for the
    integer function "int Algorithms::getStart()". */
    return this->start;
}

/*
 *  Requires:   @param start - Integer containing the new start value.
 *  Modifies:   @field start - Integer containing the old start value.
 *  Effects:    setStart - Sets the integer start for the algorithms functions.
*/


/* function setStart() deletes the start integer value for this class and
replaces it with the new value s. */
void setStart(int s) {
    /* Integer primitive copy constructor for field start. */
    this->start = s;
}
```

```
/*
 *  Requires:   nothing
 *  Modifies:   nothing
 *  Effects:    @return end - Returns the integer start for the
 algorithms object.
*/

/* function int getEnd() returns a copy of the integer that stores the
end integer value for this class. */
int getEnd() {
    /* Integer primitive copy constructor of the field start for the
    integer function
    "int Algorithms::getEnd()". */
    return this->end;
}

/*
 *  Requires:   @param end - Integer containing the new end value.
 *  Modifies:   @field end - Integer containing the old end value.
 *  Effects:    setEnd - Sets the integer end for the algorithms functions.
*/

/* function setEnd() deletes the end integer value for this class and
replaces it with the new value e. */
void setEnd(int e) {
    /* Integer primitive copy constructor for field end. */
    this->end = e;
}

/*
 *  Requires:   nothing
 *  Modifies:   nothing
 *  Effects:    @return repeats - Returns the integer repeats for the
 algorithms object.
*/

/* function int getRepeats() returns a copy of the integer that stores
the repeats integer value for this class. */
int getRepeats() {
    /* Integer primitive copy constructor of the field start for the integer
    function "int Algorithms::getRepeats()". */
    return this->repeats;
}

/*
 *  Requires:   @param repeats - Integer containing the new repeats value.
 *  Modifies:   @field repeats - Integer containing the old repeats value.
 *  Effects:    setRepeats - Sets the integer repeats for the algorithms
 functions.
*/

/* function setRepeats() deletes the repeats integer value for this class
and replaces it with the new value r. */
void setRepeats(int r) {
```

```
    /* Integer primitive copy constructor for field repeats. */
    this->repeats = r;
}

/*
 *  Requires:   nothing
 *  Modifies:   nothing
 *  Effects:    @return filename - Returns the string filename for
 the algorithms object.
*/

/* function string getFilename() returns a copy of the string that
stores the filename string value for this class. */
string getFilename() {
    /* Immutable character array class string copy constructor of the
    field filename for string function
     * "string Algorithms::getFilename()". */
    return this->filename;
}

/*
 *  Requires:   @param filename - String containing the new filename value.
 *  Modifies:   @field filename - String containing the old filename value.
 *  Effects:    setFilename - Sets the string filename for the algorithms
 function writeArray.
*/

/* function setFilename() deletes the end integer value for this class and
replaces it with the new value r. */
void setFilename(string &f) {
    /* Immutable character array class string copy constructor for the
    field filename. */
    this->filename = f;
}

/*
 *  Requires:   nothing
 *  Modifies:   nothing
 *  Effects:    @return verbose - Returns the bool verbose setting for the
 algorithms object.
*/

/* function bool getVerbose() returns a copy of the boolean that stores the
verbose bool value for this class. */
bool getVerbose() {
    /* Boolean primitive copy constructor of the field verbose for
    bool function "bool Algorithms::getVerbose()". */
    return this->verbose;
}


/*
 *  Requires:   @param filename - Bool containing the new verbose setting.
 *  Modifies:   @field repeats - Bool containing the old verbose setting.
```

```
 *  Effects:    setVerbose - Sets the bool verbose setting for the
 algorithms function testAlgorithms.
 *              testAlgorithms will print all status reports for the
 algorithms object.
*/

/* function setVerbose() deletes the verbose boolean value for this class
and replaces it with the new value m. */
void setVerbose(bool m) {
    /* Boolean primitive copy constructor for the field verbose. */
    this->verbose = m;
}

/*
 *  Requires:   @param start - Start integer for array data.
 *              @param end - End integer for array data.
 *              @param vector<int> data - Vector of integers.
 *  Modifies:   @param vector<unsigend int> data - Vector of integers.
 *  Effects:    integerVector generates the numbers from start-end in parallel.
*/

/** function integerVector(int s, int e, int *d)
* Creates a well-ordered array of integers from s-e stored in array d. */
vector<int> integerVector(int s, int e) {
    /* Create return vector. */
    vector<int> d;
    /* Fork a team of threads to initialize data. */
    #pragma omp parallel for
    /* For every location in data... */
    for(int i = s; i <= e; ++i) {
        /* Store the next number in the series 0-n. */
        d.push_back(i);
    }
    return d;
}

/*
 *  Requires:   @param start - Start integer for array data.
 *              @param end - End integer for array data.
 *              @param vector<int> data - Vector of integers.
 *  Modifies:   @param vector<int> data - Vector of integers.
 *  Effects:    randomIntgerVector generates the numbers from start-end
 in a random order in parallel.
*/

/** function randomIntgerVector(int s, int e, int *d)
* Stores a randomly ordered array data of length e-s with values singly
selected from s-e . */
vector<int> randomIntegerVector(int s, int e) {
    /* Create return vector. */
    vector<int> d;
    /* Create a vector to check for duplicates. */
    vector<int> wellOrdered;
    /* Fork a team of threads to initialize data. */
```

```
    #pragma omp parallel for
    /* For every location in data... */
    for(int i = 0; i < (e-s); ++i) {
        /* Ensure data[i] only contains a value less than start. */
        d.push_back(s-2);
        /* Ensure wellOrdered[i] only contains a value less than start. */
        wellOrdered.push_back(s-2);
    }
    /* Obtain a random seed value. */
    auto seed = chrono::high_resolution_clock::now().time_since_epoch().count();
    /* Seed mt19937 with the random source. */
    mt19937 mt_rand(seed);
    /* Create an integer to store newly generated numbers. */
    int num = 0;
    /* Fork a team of threads to generate the integers from 1-n in a
    random order. */
    #pragma omp parallel for
    for(int i = 0; i < (e-s); ++i) {
        /* Generate a new random number. */
        num = mt_rand()%(e-s) + s;
        /* Check to see if there is a repeat. */
        while(wellOrdered[num - s] != s-2) {
            /* If there is a repeat, then generate a new number. */
            num = mt_rand()%(e-s) + s;
        }
        /* Add the new number to the return array. */
        d.at(i) = num;
        /* Add the new number to the well ordered array. */
        wellOrdered.at(i) = num;
    }
    return d;
}

/*
 *  Requires:   @param start - Integer start is the first number in array data.
 *                             Integer start is the minuend of the divisor for
 calculating the end integer for array data.
 *              @param end - Integer end is the subtrahend of the divisor for
 calculating the end integer for array data.
 *              @param repeats - Integer repeats sets the number of duplicate
 integers in the array data.
 *                               Integer repeats sets the divisor for calculating
 the end integer for array data.
 *              @param vector<int> data - Vector of integers.
 *  Modifies:   @param vector<int> data - Vector of integers.
 *  Effects:    repeatsIntegerArray generates integers start - (start-end)/repeats
 with repeats duplicates in parallel.
*/


/* function duplicatesIntegerArray(int s, int e, int r, int *d)
 * generates an array in d that contains r duplicates beginning at s and
 ending at (e-s)/r */
vector<int> duplicatesIntegerVector(int s, int e, int r) {
```

```
    /* Create a return vector. */
    vector<int> d;
    /* Fork a team of threads to initialize data. */
    #pragma omp parallel for
    /* For every location in data... */
    for(int i = 0; i < (e-s); ++i) {
        /* ensure data[i] receives the floor division value i/a. */
        d.push_back(i/r + s);
    }
    return d;
}

/*
 *  Requires:   @param n - integer to use as the maximum value for a prime
 integer sieve.
 *              @param *data - Integer array that will contain all prime numbers
 up to n.
 *  Modifies:   @param *data - Integer array that will contain all prime numbers
 up to n.
 *  Effects:    Creates a prime integer array.
 */

/* function sieveOfAtkin(int n, int *data)
* returns an array that contains only prime numbers ranging from 2-n. */
vector<int> sieveOfAtkin(int n) {
    /* Create a prime number count variable. */
    int count = 0;
    /* Create a constant integer to initialize the array with. */
    const int size = n;
    /* Create a temporary storage array. */
    vector<bool> numbers;
    /* Calculate the largest potential prime number required to sieve the
    range 1-n. */
    int max = static_cast<int>(sqrt(n)) + 1;
    /* Fork a team of threads to initialize the array. */
    #pragma omp parallel for
    /* For every value in temp... */
    for(int i = 0; i < n; ++i) {
        /* Initialize all numbers to composite. */
        numbers.push_back(false);
    }
    /* Begin sieve using binary quadratic forms i = x in the parametric
    function. */
    for(int i = 1; i < max; i++) {
        /* Begin sieve using binary quadratic forms j = y in the parametric
        function. */
        for(int j = 1; j < max; j++) {
            /*  Prime sieve using the binary quadratic form from Theorem 6.1
            Page 1028. */
            int k = 4 * i * i + j * j;
            /*  The following modulo 60 remainders are prime if there are an
            odd number of solutions 4*i*i+j*j. */
            if((k <= n) && ((((k % 60) == 1) || ((k%60) == 13) || ((k%60) == 17) ||
            ((k%60) == 29) || ((k%60) == 37) || ((k%60) == 41) || ((k%60) == 49) ||
```

```
        ((k%60) == 53))) {
            /* Flip the value. */
            numbers[k] = !numbers[k];
        }
        /*  Prime sieve using the binary quadratic form from Theorem 6.2
        Page 1028. */
        k = 3 * i * i + j * j;
        /*  The following modulo 60 remainders are prime if there are an
        odd number of solutions to 3*i*i+j*j. */
        if((k <= n) && (((k % 60) == 7) || ((k % 60) == 19) ||
        ((k % 60) == 31) || ((k % 60) == 43))) {
            /* Flip the value. */
            numbers[k] = !numbers[k];
        }
        /*  Prime sieve using the binary quadratic form from Theorem 6.3
        Page 1028-1029. */
        if(i > j) {
            /* If i is more than j and it satisfies the polynomial an
            odd number of times for the following values. */
            k = 3 * i * i - j * j;
            /* The following modulo 60 remainders are prime if there are an
            odd number of solutions to 3*i*i-j*j. */
            if((k < n) && (((k % 60) == 11) || ((k % 60) == 23) ||
            ((k % 60) == 47) || ((k % 60) == 59))) {
                /* Flip the value. */
                numbers[k] = !numbers[k];
            }
        }
    }
}
/* Mark 2 as prime. */
numbers[2] = true;
/* Mark 3 as prime */
numbers[3] = true;
/* Mark 5 as prime */
numbers[5] = true;
/* Create a comparison integer. */
int comp = 0;
/* For all values within max... */
for(int i = 5; i <= max; i++) {
    /* Initialize the comparison integer. */
    comp = numbers[i];
    /* If the number is prime... */
    if(comp == 1) {
        /* Calculate its square value. */
        int square = i * i;
        /* Remove all factors of the square from the prime list. */
        for(int j = 1; square * j < max; j++) {
            /* Mark composites with false. */
            numbers[square*j] = false;
        }
    }
}
/* Create a storage vector for the prime numbers. */
```

```
    vector<int> primes;
    /* Fork a team of threads to count the number of prime numbers
    in the sieve. */
    #pragma omp parallel for
    /* For every integer from 0-n... */
    for(int i = 0; i < n; i++) {
        /* If the number is prime... */
        if (numbers[i] == true) {
            /* Store the number in the returnArray object. */
            primes.push_back(i);
        }
    }
    return primes;
}

/*
 *  Requires:   @param vector<int> d1 - Integer vector (size is > n).
 *              @param vector<int> d2 - Integer vector (size is > n).
 *              @param vector<int> d3 - Integer vector (size is > n).
 *              @param vector<int> d4 - Integer vector (size is > n).
 *              @param n - Integer size to traverse through the vectors.
 *  Modifies:   nothing
 *  Effects:    repeatsIntegerArray generates integers start - (start-end)/repeats
 with repeats duplicates in parallel.
*/

void outputTerminal(vector<int> d1, vector<int> d2, vector<int> d3,
vector<int> d4, int n) {
    /* Fork a team of threads to output data. */
    #pragma omp parallel for
    /* For every value in every array of length n... */
    for(int i = 0; i < n; ++i) {
        /* Print the values to the terminal */
        printf("sequence[%d] = %d , random[%d] = %d, duplicates[%d] = %d\n,
        primes[%d] = %d\n", i, d1[i], i, d2[i], i, d3[i], i, d4[i]);
    }
}

/*
 *  Requires:   @param filename - String containing the name of the
 file to write to.
 *              @param vector<int> d1 - Integer vector (size is > n).
 *              @param vector<int> d2 - Integer vector (size is > n).
 *              @param vector<int> d3 - Integer vector (size is > n).
 *              @param vector<int> d4 - Integer vector (size is > n).
 *              @param n - Integer value to traverse through the vectors.
 *  Modifies:   @param vector<int> d1 - Integer vector (size is > n).
 *              @param vector<int> d2 - Integer vector (size is > n).
 *              @param vector<int> d3 - Integer vector (size is > n).
 *              @param vector<int> d4 - Integer vector (size is > n).
 *  Effects:    writeArray - Writes or overwrites the file named filename
 with the three arrays using a csv format.
*/
```

```
    static void writeArray(string &f, vector<int> d1, vector<int> d2,
    vector<int> d3, vector<int> d4, int n) {
    /* Create a file output stream. */
    ofstream file(f);
    /* Open a file to write to. */
    if(file.is_open()) {
        /* Fork a team of threads to output data. */
        #pragma omp parallel for
        /* For every value in every array of length n... */
        for(int i = 0; i < n; ++i) {
            /* Output file string. */
            file << d1[i] << ',' << d2[i] << ',' << d3[i] << d4[i] << endl;
        }
    }
};
```

8.2 Sorting-Analysis 1-1000.csv by Justin Adams

	Sorting Algorithm	Objects	Reads	Writes	Time (s)
0	Bubble Sort	1	0	0	0.000000
1	Bubble Sort	2	5	3	0.000000
2	Bubble Sort	3	6	0	0.000000
3	Bubble Sort	4	21	9	0.000000
4	Bubble Sort	5	35	15	0.000999
...	...	...	...	...	...
6995	Two Sort	996	2415378	487950	0.451696
6996	Two Sort	997	2407067	486221	0.480622
6997	Two Sort	998	2408816	486598	0.426022
6998	Two Sort	999	2511250	506928	0.651030
6999	Two Sort	1000	2418422	488516	0.534763

[7000 rows x 5 columns]

8.3 Sorting-Analysis 1,000-10,000.csv by Justin Adams

	Sorting Algorithm	Objects	Reads	Writes	Time
0	Bubble Sort	1000	1758912	759912	3.007830
1	Bubble Sort	2000	7020491	3022491	10.668100
2	Bubble Sort	3000	15854901	6857901	24.179900
3	Bubble Sort	4000	28238457	12242457	43.313500
4	Bubble Sort	5000	44526206	19531206	73.091300
..	...	...	...	...	...
57	Heap Sort	8000	489905	124437	0.613784
58	Heap Sort	9000	559045	141569	0.617950
59	Heap Sort	10000	629866	159134	0.718989
60	Heap Sort	10000	629246	158942	0.068096
61	Two Sort	1000	3725680	750290	0.020137

[62 rows x 5 columns]

8.4 googleplaystore.csv

App, Category, Rating, Reviews, Size, Installs, Type, Price, Content Rating, Genres, Last Updated,

Current Ver, Android Ver

Photo Editor Candy Camera Grid ScrapBook, ART_AND_DESIGN, 4.1, 159, 19M, 10000, Free, 0, Everyone, Art Design, 1/7/18, 1.0.0, 4.0.3 and up

Coloring book moana, ART_AND_DESIGN, 3.9, 967, 14M, 500000,Free, 0, Everyone, Art Design; Pretend Play, 1/15/18, 2.0.0, 4.0.3 and up

U Launcher Lite – FREE Live Cool Themes Hide Apps,ART_AND_DESIGN,4.7,87510,8.7M, 5000000,Free,0,Everyone,Art Design,8/1/18,1.2.4,4.0.3 and up

...

Parkinson Exercices FR,MEDICAL,0,3,9.5M,1000,Free,0,Everyone,Medical,1/20/17,1,2.2 and up

The SCP Foundation DB fr nn5n,BOOKS_AND_REFERENCE,4.5,114,var,1000,Free,0,Mature 17+,Books Reference,1/19/15,Varies with device,Varies with device

iHoroscope - 2018 Daily Horoscope Astrology,LIFESTYLE,4.5,398307,19M,10000000,Free,0, Everyone,Lifestyle,7/25/18,Varies with device,Varies with device

8.5 Project 4 written by Professor Lisa Dion

CS 124 Project 4

Due on Gradescope by Wednesday, November 13th

For this project, you will sort the 1,000 objects from your data set. You will modify each sorting algorithm to collect data. You will analyze the results from the different sorting algorithms.

Implement

- You should have your 1,000+ objects stored in a vector, initially unsorted.
- Choose five sorting algorithms:
    1. Bubble Sort
    2. Selection Sort or Insertion Sort
    3. Merge Sort or Quick Sort
    4. Heap Sort
    5. Two-sort: Sort by any algorithm (one of the above or a different one), then sort on a different field using a stable sorting algorithm (again, one of the above or a different one).
- Use each sorting algorithm to sort a vector of the first 100 objects from your dataset. Record the number of reads (the number of times you retrieve and use an object from the vector) and writes (the number of times you assign an object into the vector). Then do the same for a vector of size 200, 300, 400, 500, 600, 700, 800, 900, and 1,000. Each of the five sorting algorithms should be given identical unsorted vectors to begin with.
- Analyze the data. Graph the number of reads and writes for each sorting algorithm and look at how the number of reads and writes grows when the size of the data set grows. Compare and contrast the different sorting algorithms and draw conclusions about which sorting algorithms are more efficient. Discuss complexities and their effects. All of this will go in your report. - In your report, also include answers to the following questions: If you need to sort a contacts list on a mobile app, which sorting algorithm(s) would you use and why? What about if you need to sort a database of 20 million client files that are stored in a datacenter in the cloud? - You must submit your source file(s), your data file(s), and your report. Please submit your report in PDF format.

Extra Credit

To earn up to 10 extra credit points (at the grader's discretion), you can get more thorough results. This can include:
- Setting timers to record how long it takes you to sort the objects with each algorithm.
- Performing the same experiment, except double the size of the data set each time (instead of having it grow linearly).
- Using more sorting algorithms.

If you implement any of these extra credit items, make sure to explicitly state it in your report with the results and analysis.

Grading

The project is out of 90 points.
5 pts Program compiles and runs.
5 pts Code style. Readable, naming style is consistent, comments where appropriate.
5 pts Use five sorting algorithms according to the directions above.
15 pts Sort the 100, 200, ... 1,000 objects according to the directions above.
40 pts Record the correct number of reads and writes for each sort.
15 pts Report: analysis of results.
5 pts Report: professional, grammatically correct, cites sources.