

PROGRAM 7 / CSC1310

SEARCHING & SORTING ALGORITHM EFFICIENCY



IMPORTANT DATES

Assignment Date: Monday, March 19, 2018

Due Date: Monday, April 2, 2018

PROGRAM ASSIGNMENT DESCRIPTION

Take the Movie program that you did for **program 6** and modify it.

Modify the add movie functions so that they no longer sort or search (which means duplicate movie titles will be allowed).

Add a function for **all** of the search & sort algorithms that we have covered: linear search, binary search (should already have this), bubble sort, insertion sort (should already have this), insertion sort descending, selection sort, merge sort, and quick sort.

Create a function called **algorithmAnalysis** that will use a timer to time how long it takes to run the algorithm on a linked list of movies and print out the times. Several movie text files will be provided for you in order for you to test your program.

FILES THAT SHOULD BE INCLUDED IN YOUR SUBMISSION

- crockett_movie_data_24.txt
- crockett_movie_data_112.txt
- crockett_movie_data_1112.txt
- crockett_movie_data_2112.txt
- crockett_movie_data_5024.txt
- crockett_movie_data_10112.txt [note – only test with this file if you have loads of time to spare]
- crockett_movie_data_100112.txt [note – only test with this file if you have loads of time to spare]
- Driver.cpp
- LinkedList.h
- Makefile
- Movie.cpp
- Movie.h
- Movies.cpp
- Movies.h
- randomGenerator.cpp (this file not used by your program – included it just in case you were curious how I generated the crockett_movie_data.txt files)
- runProgram.bat
- TEST_CASE_112.txt
- TEST_CASE_1112.txt
- TEST_CASE_2112.txt
- TEST_CASE_5024.txt
- Text.cpp
- Text.h
- Timer.cpp

- **Timer.h**

PROGRAM SPECIFICATIONS (DIRECTIONS ON HOW TO WRITE THE PROGRAM)

TIMER CLASS

Whenever you see “start timer” and “stop timer” and “print out total time” in the directions in this document, that means you will be using the Timer class provided for you.

You will need to `#include "Timer.h"` to use this class.

When you want to use a timer in your program, you will first need to create variables of data type `time_t`.

```
time_t start;
time_t end;
```

The `getTime()` function will get the current time and return it as a `time_t` data type. You will use this function to get the start time and the end time.

The `totalTime()` function accepts two `time_t` variables (start & end) and then returns the difference as a `double` (which is the number of seconds between the start & end time).

DRIVER.CPP

Modify the menu so that there is an addition option. The option should be inserted after “Print all movies” and before “Delete All movies...” and will be your new #7. Then the user will choose between 1 and 8.

If the user chooses #7, then you should call the `algorithmAnalysis` function, which is a member function of the `Movies` class.

MOVIES CLASS

- You should already have an `insertionSort()` and `binarySearch()` function in this program (written for program 6)
- Modify the add movie functions & edit movie function so that they no longer sort or search (which means duplicate movie titles will be allowed). If you do not do this then your program will take a long, long, long time to do testing.
- You will create 9 new functions. All of the following functions except for `algorithmEfficiency()` should be made private.
 - **linearSearch** – This function should search for a particular movie title to see if it is in the list. It should return -1 if the Movie title could not be found. Remember that Movie titles are of data type `Text*`. Use the linear search algorithm to implement this function.
 - **bubbleSort** – This function should sort the `LinkedList` of `Movies` in ascending order by Movie title. This function will call a function called `swap` in the `LinkedList` class to swap values in the linked list when necessary. Use the bubble sort algorithm to implement this function.
 - **insertionSortDescending** – This function should be the same as `insertionSort` except it will sort the `LinkedList` of `Movies` in descending order by Movie title instead of ascending order.
 - **selectionSort** - This function should sort the `LinkedList` of `Movies` in ascending order by Movie title. This function will call a function called `swap` in the `LinkedList` class to swap values in the linked list when necessary. Use the selection sort algorithm to implement this function.
 - **mergeSort & merge** – These two functions work together to implement the merge sort algorithm, which should sort the `LinkedList` of `Movies` in ascending order by Movie title. The `mergeSort` function is a recursive function which calls the `merge` function. The `merge` function dynamically allocates a new linked list of `Movie` pointers to use as the merged

linked list. At some point in the merge function you will need to replace a node....you will do this by deleting the node (deleteNode) at a particular position and then inserting a new node (insertNode function) at a particular position.

- **quickSort & partition** – These two functions work together to implement the quick sort algorithm, which should sort the LinkedList of Movies in ascending order by Movie title. The quicksort function is a recursive function which calls the partition function. The partition function will use a pivot string (c-string). The partition function will call a function called swap in the linkedList class to swap values in the linked list when necessary.
- **algorithmAnalysis**- This is the “driver” function that will call all of these other functions to test the efficiency of each one. Here is the pseudocode for this function below:

1. Start timer
Call linearSearch sending a temporary Text* with a c-string named “Llama” to it. (I don’t care if It finds this string or not)
Stop timer
Print out total time for this algorithm
2. Start timer
Call binarySearch() sending a temporary Text* with a c-string named “Llama” to it.
Stop timer
Print out total time for this algorithm
3. Call insertionSortDescending() to put linked list in opposite order.
Start timer
Call bubbleSort()
Stop timer
Print out total time for this algorithm
4. Call insertionSortDescending() to put linked list in opposite order.
Start timer
Call selectionSort()
Stop timer
Print out total time for this algorithm
5. Call insertionSortDescending() to put linked list in opposite order.
Start timer
Call insertionSort()
Stop timer
Print out total time for this algorithm
6. Start timer
Call insertionSortDescending()
Stop timer
Print out total time for this algorithm
7. Start timer
Call mergeSort() sending 0 and the length of the linked list (# of nodes) minus 1 to the function
Stop timer
Print out total time for this algorithm
8. Start timer
Call quickSort() sending 0 and the length of the linked list (# of nodes) minus 1 to the function
Stop timer
Print out total time for this algorithm

READABILITY OF OUTPUT & CODE DOCUMENTATION

- Make sure that your output looks similar to my sample output (below). When I run your program, it shouldn't make me want to scream. It should be extremely readable and user-friendly.
- **For this program, don't worry about comments except put your NAME at the top of all files that you modify!!!**

WHAT TO TURN IN

Zip ALL the files required to compile & run the program, in a single zipped file named whatever you want.

Then, upload this zip file to the assignment folder in ilearn. I will **remove one point** if you turn in unzipped files. Programs that do not include all the files listed in the "FILES" section above **will not be graded**.



SAMPLE OUTPUT

5024 movies in linked list:

```
C:\Windows\System32\cmd.exe - Movies
g++ -o Movies.exe Driver.o Movie.o Movies.o Text.o Timer.o

C:\Users\acrockett\Desktop\CSC1310 Spring 2018\PROGRAMS\PROGRAM 7>Movies

What would you like to do?
1. Read movies from file.
2. Save movies to a file.
3. Add a movie.
4. Delete a movie.
5. Edit a movie.
6. Print all movies.
7. Run algorithm analysis.
8. Delete ALL movies and end the program.
CHOOSE 1-8: 1

What is the name of the file? (example.txt): crockett_movie_data_5024.txt

5024 movies were read from the file.

What would you like to do?
1. Read movies from file.
2. Save movies to a file.
3. Add a movie.
4. Delete a movie.
5. Edit a movie.
```

```
C:\Windows\System32\cmd.exe
5. Edit a movie.
6. Print all movies.
7. Run algorithm analysis.
8. Delete ALL movies and end the program.
CHOOSE 1-8: 7
      Linear Search:      0.00
      Binary Search:     0.00
      Bubble Sort:       593.00
      Selection Sort:    475.00
      Insertion Sort:    590.00
      Insertion Sort Descending: 590.00
      Merge Sort:        2.00
      Quick Sort:        1.00

what would you like to do?
1. Read movies from file.
2. Save movies to a file.
3. Add a movie.
4. Delete a movie.
5. Edit a movie.
6. Print all movies.
7. Run algorithm analysis.
8. Delete ALL movies and end the program.
CHOOSE 1-8: 8

GOODBYE!
```

24 movies in linked list:

```
what is the name of the file? (example.txt): crockett_movie_data_24.txt

24 movies were read from the file.

what would you like to do?
1. Read movies from file.
2. Save movies to a file.
3. Add a movie.
4. Delete a movie.
5. Edit a movie.
6. Print all movies.
7. Run algorithm analysis.
8. Delete ALL movies and end the program.
CHOOSE 1-8: 7
      Linear Search:      0.00
      Binary Search:     0.00
      Bubble Sort:       0.00
      Selection Sort:    0.00
      Insertion Sort:    0.00
      Insertion Sort Descending: 0.00
      Merge Sort:        0.00
      Quick Sort:        0.00
```

112 movies in linked list:

```
what is the name of the file? (example.txt): crockett_movie_data_112.txt

112 movies were read from the file.

what would you like to do?
1. Read movies from file.
2. Save movies to a file.
3. Add a movie.
4. Delete a movie.
5. Edit a movie.
6. Print all movies.
7. Run algorithm analysis.
8. Delete ALL movies and end the program.
CHOOSE 1-8: 7
      Linear Search:      0.00
      Binary Search:     0.00
      Bubble Sort:       0.00
      Selection Sort:    0.00
      Insertion Sort:    0.00
      Insertion Sort Descending: 0.00
      Merge Sort:        0.00
      Quick Sort:        0.00
```

1112 movies in linked list:

What is the name of the file? (example.txt): crockett_movie_data_1112.txt

1112 movies were read from the file.

What would you like to do?

1. Read movies from file.
2. Save movies to a file.
3. Add a movie.
4. Delete a movie.
5. Edit a movie.
6. Print all movies.
7. Run algorithm analysis.
8. Delete ALL movies and end the program.

CHOOSE 1-8: 7

Linear Search:	0.00
Binary Search:	0.00
Bubble Sort:	5.00
Selection Sort:	5.00
Insertion Sort:	4.00
Insertion Sort Descending:	5.00
Merge Sort:	0.00
Quick Sort:	0.00

2112 movies in the linked list:

What is the name of the file? (example.txt): crockett_movie_data_2112.txt

2112 movies were read from the file.

What would you like to do?

1. Read movies from file.
2. Save movies to a file.
3. Add a movie.
4. Delete a movie.
5. Edit a movie.
6. Print all movies.
7. Run algorithm analysis.
8. Delete ALL movies and end the program.

CHOOSE 1-8: 7

Linear Search:	0.00
Binary Search:	0.00
Bubble Sort:	43.00
Selection Sort:	32.00
Insertion Sort:	39.00
Insertion Sort Descending:	39.00
Merge Sort:	0.00
Quick Sort:	0.00

5112 movies in the linked list:

5112 movies were read from the file.

What would you like to do?

1. Read movies from file.
2. Save movies to a file.
3. Add a movie.
4. Delete a movie.
5. Edit a movie.
6. Print all movies.
7. Run algorithm analysis.
8. Delete ALL movies and end the program.

CHOOSE 1-8: 7

Linear Search:	0.00
Binary Search:	0.00
Bubble Sort:	763.00
Selection Sort:	554.00
Insertion Sort:	724.00
Insertion Sort Descending:	714.00
Merge Sort:	4.00
Quick Sort:	1.00

10112 movies in the linked list:

What is the name of the file? (example.txt): crockett_movie_data_10112.txt

10112 movies were read from the file.

What would you like to do?

1. Read movies from file.
2. Save movies to a file.
3. Add a movie.
4. Delete a movie.
5. Edit a movie.
6. Print all movies.
7. Run algorithm analysis.
8. Delete ALL movies and end the program.

CHOOSE 1-8: 7

Linear Search:	0.00
Binary Search:	0.00
Bubble Sort:	7096.00
Selection Sort:	5445.00
Insertion Sort:	6477.00
Insertion Sort Descending:	6558.00
Merge Sort:	13.00
Quick Sort:	4.00