Part 2 – Searching

Group 1: Peter Lombardo, Gregory Hoar, MD Bhuiyan, Giovanni Braun

Searches

Beatles

```
searching for: Beatles
Binary search comparisons: 13
Loop comparisons: 337
Total comparisons: 350
Theoretical complexity (log2 N): 13.36
```

Santana

```
searching for: Santana
Binary search comparisons: 3
Loop comparisons: 114
Total comparisons: 117
Theoretical complexity (log2 N): 13.36
```

Arlo

```
searching for: Arlo
Binary search comparisons: 14
Loop comparisons: 6
Total comparisons: 20
Theoretical complexity (log2 N): 13.36
```

A

```
searching for: A
Binary search comparisons: 13
Loop comparisons: 582
Total comparisons: 595
Theoretical complexity (log2 N): 13.36
```

Z

```
searching for: Z
Binary search comparisons: 14
Loop comparisons: 141
Total comparisons: 155
Theoretical complexity (log2 N): 13.36
```

X

```
searching for: X
Binary search comparisons: 14
Loop comparisons: 2
Total comparisons: 16
Theoretical complexity (log2 N): 13.36
```

Task 2 Unit test output from SongCollection

```
Total songs
10514, first songs:
Aerosmith, "Adam's Apple"
Aerosmith, "Ain't Enough"
Aerosmith, "Ain't that a Bitch"
Aerosmith, "All Your Love"
```

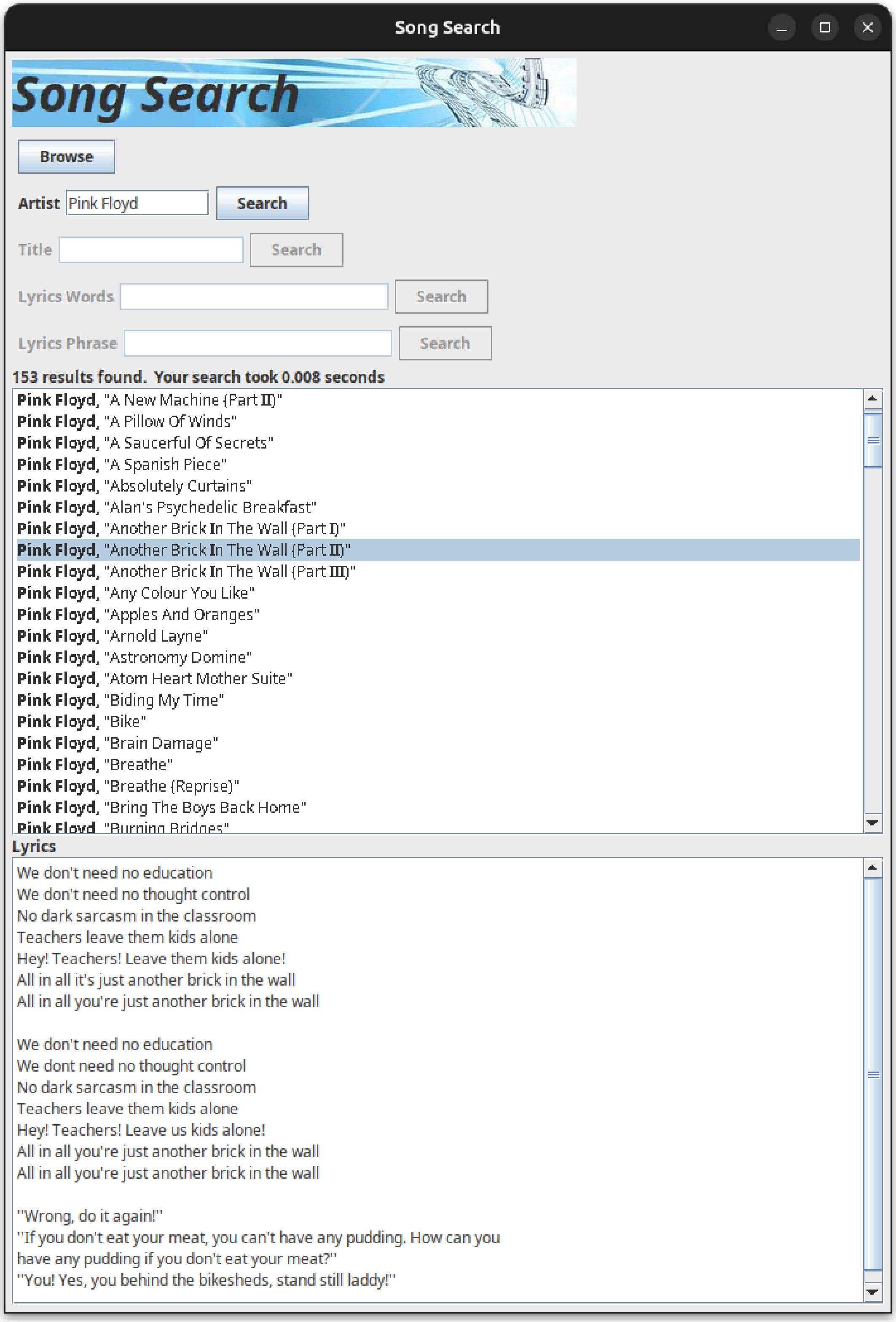
```
Aerosmith, "Amazing"

Aerosmith, "Angel"

Aerosmith, "Angel's Eye"

Aerosmith, "Animal Crackers"

Aerosmith, "Attitude Adjustment"

Aerosmith, "Avant Garden"

BUILD SUCCESSFUL (total time: 0 seconds)
```

3. The *O(K + log2N)* Goal

This search method does meet that goal since binary search as implemented in the Java Arrays.binarySearch method has a worst case big O of log2N in the worst case scenario. When you combine that with while loops for adjacent elements now that you’re in the neighborhood, you find that you will always conduct K number of operations if K is the number of songs that have that prefix. This gives us a big O notation of O(K + log2N)

.

Task 3 output GUI Test



Incomplete parts: None

Known bugs: None