*CSCI-290* *Project Part 2*

*Artist Search*

**Objectives:** Searching for artist prefix matches. Writing a Comparator class. Using Arrays.binarySearch(). Instrumenting code for performance analysis.

You will be writing a search method to search the array for all songs matching a user specified artist prefix. For instance searching for "Be" will return all songs by Beach Boys, Beatles, and Bee Gees.

Starting files are on BrightSpace:

- SearchByArtistPrefix.java - starting code for searching by artist
- CmpCnt.java - a super class used to instrument your Comparator

**Task1:** Create a nested Comparator class within your Song class that compares songs by artist

1. This is a nested static class. Declare it within your Song class as:

```
public static class CmpArtist implements Comparator<Song> {
   public int compare(Song s1, Song s2) {
       ... your code goes here ...
   }
}
```

2. It should compare the two songs by their artist fields. This comparison should be case insensitive. (this is ONE line of code)
3. Add testing code to the main() method. Check that your comparator is consistent. If s1 <s2 then it should also return s2 >s1. Mimic the tests for compareTo(). Create a Comparator object and then have it call compare with the same pairs of songs that are used in the compareTo() tests.

**Task2:** Complete the class SearchByArtistPrefix.

1. In these assignments we are going to be building a variety of searches and supporting data structures. This one is easy. You are going to use binary search to search the sorted array of songs for all songs matching a specified artist or artist prefix.
2. The constructor is already finished. It just stores a reference to the song array in the private variable songs.
3. Write the method public Song[] search(String artistPrefix). This takes in either a complete artist name such as "Beach Boys", or a prefix of an artist name such as "Be". After searching, it should return an array containing all the matching songs. Preserve their sorted order.
4. Suggestion: Use the built-in searching method Arrays.binarySearch(), but think carefully about how the java binary search works:

- The java binary search returns the first match it finds, which is not necessarily the earliest match in the list.
- A failed binary search returns a negative value that can be used to calculate the location the missing item would be found at. This will be useful! Look at the documentation in the Java API.
- Make a temporary song with the search term as the artist.
- Use your artist Comparator from task 1. Do not modify your comparator to look for a prefix! Just work with binary searches return value. Finally make a loop (or loops) to scoop up all matching songs.

5. The final returned result needs to be just a simple array of songs, however one way to build this is to first collect the songs in an ArrayList and then convert it using .toArray(Song[] a).
6. Testing: The main() is mostly already written. It takes 2 command line arguments: a song file name and the artist to search for. It creates an instance of this search class and performs the specified search. Finish the testing method by printing the total number of matches as well as the Artist and Title of the first 10 matches. Reuse your code from part 1 to do this.
7. You will need to the run configuration to specify the arguments.
8. For full credit, your search should take time $O(K + log_2 N)$, where K is the number of matches found and N is the size of the song array.

**Task 3.** Test that your code works with the graphical user interface SongSearch.jar. If you have followed the specifications above (correct class names, method names, and return types) then your code will plug into the GUI without any changes on your part.

**Task 4.** You are now going to instrument your code to count the number of comparisons it takes to perform a search.

1. The hard part is finding out how many comparisons Arrays.binarySearch() takes. You are going to do this by putting a counter inside your comparator.
2. I have supplied a base class CmpCnt.java to use when declaring your comparator.
3. Change your artist comparator declaration to:

   ```
   public static class CmpArtist extends CmpCnt implements Comparator <Song>{
   ```

   This base class contains a variable cmpCnt that will be used to count each call to your comparator.
4. Add `cmpCnt++;` as the first line of your compare() method.
5. Now in your search method after you call binary search and it does all its work, you can then call getCmpCnt() on your comparator to get the call count. E.g.: `((CmpCnt)cmp).getCmpCnt()`
6. You also probably have 1 or 2 loops to find and scoop up all of the matches. Add counters to count all of the artist comparisons done in these loops.

7. Print your counter values in an understandable fashion. Check that these statistics match the time goal of $O(K + log_2 N)$. For example:

```
Searching for be
index from binary search is : -599
Binary search comparisons: 14
Front found at 598
Comparisons to build the list: 504
Actual complexity is: 518

k is 503
log_2(n) = 13
Theoretical complexity k+log n is: 516
```

N for this data is 10514 so what is $log_2(10514)$ ? That is easily calculated. $10,514_10 == 0010100100010010_2$ normalize to get the exponent $1.0100100010011X2^{13}$ The exponent is the log base 2.

**What to turn in:** Written part: (named Proj2Writeup.pdf)

1. Your searching results showing the total matches and the first 10 matching songs for these searches: "Beatles", "Santana", "Arlo", "A", "Z" and "X". (The total matches should be 335, 71, 4, 581, 140 and 0 respectively.) Also include your statistics on the number of comparisons for each search.
2. Does your search meet the $O(K + log_2 N)$ time goal? Explain.
3. Using the GUI interface, find the Pink Floyd song "Another Brick in The Wall (Part II)" and click on it to show its lyrics. Save a screen capture of the GUI into your document.
4. It is your responsibility to test your code. Explain any incomplete parts, or any known bugs. I take off fewer points if you are aware of a bug rather than oblivious to it. Electronic submission:
   - I will create groups and put you into them. Anyone in the group can upload files.
   - I will run your unit testing code, so each class must implement the specified testing and include a main method.
   - Print your written part as a pdf named Proj2Writeup.pdf
   - Upload the written part, the SearchByArtistPrefix.java, and Song.java files only!
   - Make sure that all group members are listed at the top on the writeup!!

**Grading:**

| | |
|---|---|
| Written part, testing output and statistics. | 10% |
| Artist comparator | 20% |
| SearchByArtistPrefix | 40% |
| Works with the GUI | 10% |
| Meets search time goals | 20% |

I expect your programs to be properly commented and use good style. Points may be deducted for egregious disregard of these matters. Every method in every class must have complete documentation including authorship. A full an complete revision history should be at the top of every class file! You should NEVER delete any revision comments. Just add to them when necessary.