ACTIVITY **3.1.8**

Search Algorithms

INTRODUCTION

People access digital information every day: on their devices, on local networks, and on the Internet. All of these actions require a computer program to locate information as quickly and efficiently as possible. Over the years, several search algorithms have been developed by various programmers and mathematicians. In this activity, you will explore two of these search algorithms.

Materials

- Playing cards
- Computer with Android™ Studio
- Android™ tablet and USB cable, or a device emulator
- Free Backendless account per student

RESOURCES

Lesson 3.1 Reference Card for Backendless
Resources available online

Procedure

Part I: Sequential Search

Have you ever played the "Guess My Number" game? One person thinks of a number in a range of numbers (such as 1–100), and the other person tries to guess the number by asking a series of Yes/No questions.

1. Take a few minutes to play the game with a partner.
   a. How many questions did you have to ask before guessing the correct number?
   b. What was your strategy?

2. For an explanation of **Sequential Search**, review the slideshow:

   a. Sketch the flowchart for this algorithm. You may refer to the flowchart examples provided in *3.1.7 Sort Algorithms Supplemental* of the previous activity.

   b. Write the pseudocode for this algorithm. You may refer to the pseudocode examples provided in *3.1.7 Sort Algorithms Supplemental* of the previous activity.

sequential search

A search algorithm that scans through a list of items, one at a time, until it finds the target or until the list is exhausted. Also known as "linear search".

Search Algorithms

Search algorithms are essential to many applications used today

- Two common algorithms are
  - Sequential Search
  - Binary Search

Computer Science A

Sequential Search

- The sequential search (also known as linear search) scans through a list of items, one at a time, until it finds the target or until the list is exhausted.
- In sequential search, the list of items does **not** need to be sorted.

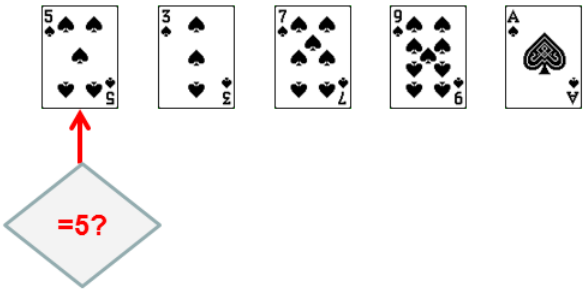
Computer Science A

Sequential Search

Best-case scenario:
Target is 5

Computer Science A

Sequential Search



Best-case scenario:
Target is 5

Computer Science A

Part II: Binary Search

3 For an explanation of **Binary Search**, review the slideshow, and then do **one** of the following:

a. Sketch the flowchart for this algorithm. You may refer to the flowchart examples provided in *3.1.7 Sort Algorithms* supplemental document of the previous activity.

b. Write the pseudocode for this algorithm. You may refer to the pseudocode examples provided in *3.1.7 Sort Algorithms* supplemental document of the previous activity.

binary search

A search algorithm that compares the item at the midpoint of asortedlist to the target.

- If the items are equal, then the target is found.
- If the midpoint item is greater than the target, then the algorithm repeats the search in the bottom half of the list.
- If it is less than the target, then it repeats the search in the top half of the list.
- The process repeats until the target is found or the list is exhausted.

Sequential Search

Best-case scenario:
Target is 5

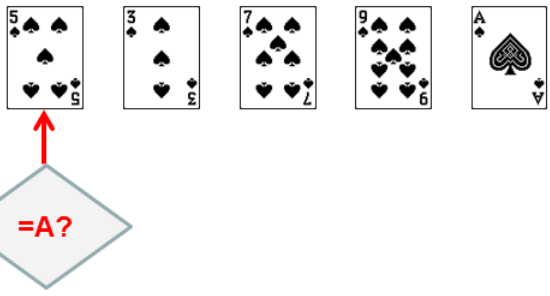
Done!

Worst-case scenario:
Target is Ace

Sequential Search

Best-case scenario:
Target is 5

Done!

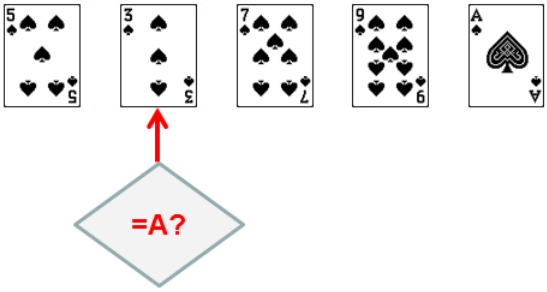


Worst-case scenario:
Target is Ace

Sequential Search

Best-case scenario:
Target is 5

Done!

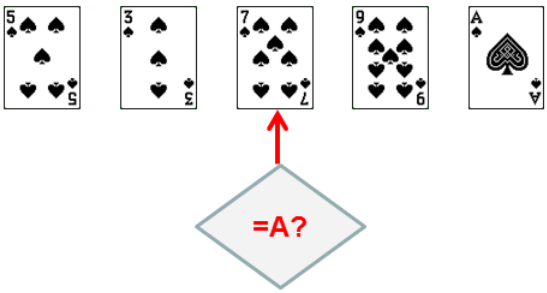


Worst-case scenario:
Target is Ace

Sequential Search

Best-case scenario:
Target is 5

Done!



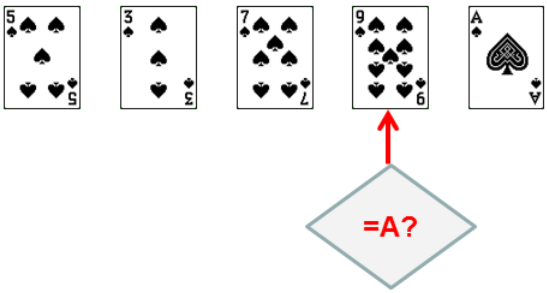
Worst-case scenario:
Target is Ace

Computer Science A
© 2016 Project Lead The Way, Inc.

Sequential Search

Best-case scenario:
Target is 5

Done!

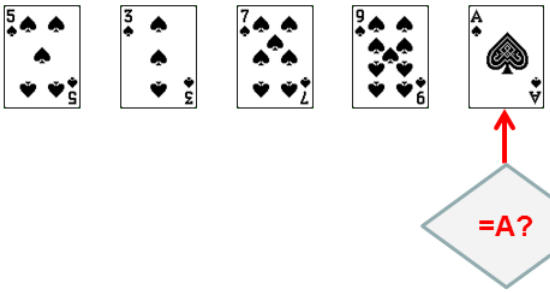


Worst-case scenario:
Target is Ace

Sequential Search

Best-case scenario:
Target is 5

Done!



Worst-case scenario:
Target is Ace

Computer Science A
© 2016 Project Lead The Way, Inc.

Sequential Search

Best-case scenario:
Target is 5

Done!

Worst-case scenario:
Target is Ace

Done!

Computer Science A

Conclusion

- Sequential search is a simple, but not so efficient search algorithm.
- It does **not** require the list to be sorted, which simplifies the overall system needs.
- The best-case scenario is when the target is in the first position in the list. In this case the number of iterations is 1.
- The worst-case scenario for a list with N items is when the target is in the last position in the list or not in the list at all. In this case, the search will have to execute N iterations.
- On average, the sequential search executes N/2 iterations.

Computer Science A

Part III: Find a Trip

4. Open your TripTracker app in Android Studio. If you were unable to complete *Activity 3.1.7 Sort Algorithms*, obtain and import *`3.1.7TripTracker_Solution`* as directed by your teacher.
5. Review the following:
   - How listener callback methods are created
   - How to use `findViewById()` to get data from device widgets
   - Your `ArrayListSorter` implementation from the last activity

6. Add a search section to the TripList screen that will allow users to search for a trip by name. Make sure that you write your own search method. To keep things a bit simpler, execute your search algorithm on the already loaded `mTrips` (do not access Backendless for this search). The results can be: the first trip that matches the search keyword, all trips that match the search (a bit more work for this one!), or an empty list if no matches exist.

   Note that ideally, a program would apply search criteria directly on a database query. To practice writing a search algorithm, you will ***not*** *provide a search query; rather you will write the algorithm yourself, using the list of trips already set in* `mTrips`.

   a. For the UI, consider using a search icon in an `ImageButton` such as

   ```
   1 android:src="@android:drawable/ic_menu_search" />
   ```

   b. In `refreshTripList`, update the `mTrips` list to contain the search results. Because `TripListFragment` extends a Fragment and not an Activity class, you need to register the listener for the Search button in `onCreateView()` instead of `onCreate()`.

   c. When you write the search algorithm, model it after the `ArrayListSorter` you wrote in the previous activity. Also, you will need to cast the items in the list as a Trip object.

   d. Call `refreshTripList()` from your search listener to see the new search results.

   e. To clear the results, select **REFRESH** from the options bar.

   f. When you test your app, you may notice the soft keyboard automatically shows because your edit view has the focus. If you have wondered why there is an empty text view in your *`fragment_trip_list`* file, it is so that it can grab the focus away from the edit view, thereby hiding the soft keyboard. Add the `<requestFocus/>` property to the `TextView` with "No data".

7. Test your app and its new search feature.

CONCLUSION

1. If you are designing a search solution that should work for search criteria of various data types, how would you implement that in Java?
2. If the list of items of a sequential search is sorted, how would that affect the algorithm? Explain.