# CSC220 Lab05
# Searching and Recursion

The goal of this week's lab is:

1. Practice searching
2. Continue learning the significance of special cases!
3. Learning how to write test to check your implementation

**<u>Things you must do:</u>**

1. There are many details in this lab. Make sure you read the whole thing carefully before writing any code and closely follow this instruction.
2. Always remember Java is case sensitive.
3. Your file names, class names, and package name must match exactly as they are specified here.

**<u>Things you must not do:</u>**

1. You must not change the file names, class names, package names.
2. You must not change the signature of any of these methods (name, parameters, …). Just fill in the missing code inside them. However, you are more than welcome to create any helper methods you need as necessary.

## Part 0

- Create a new Java project and call it **<u>Lab05</u>.**
- Create a package inside your project and call it **<u>lab05</u>**.
- Download the Lab05-Assignment05 ZIP folder from Google Drive (link from blackboard assignment). **Copy and paste** the following files into your new lab05 package:

- **SortedBinarySet.java.** This file will contain the representation of the class you are about to implement.
- **Tester.java**. This file will contain the main function and tests for SortedBinarySet.

## Part 1 - SortedBinarySet **Class Description**

For this lab (and assignment), you are asked to construct a class called SortedBinarySet that handles a list(s) of doubles. We have provided the skeleton of this class for you. This class requires:

- All of the usual operations for a list, such as the ability to add items, remove items, search for items, grow itself before running out of space.
- You must make your search routine(s) as efficient as possible. **You are not allowed to use sort algorithms**. Instead, you need to consider the fact that any list will begin as an empty list (which is already in sorted order), and you can simply insert items in the correct order to ensure that the list remains sorted **at all times**.
- Furthermore, the list must not contain any **duplicates**. (Because it's a "set")
- The data in SortedBinarySet must be backed by a basic array (**do not use a Java List, ArrayList, or any other Java class**).
- It is **not acceptable** for the array to run out of space for new items, nor is it acceptable to create a gigantic array. We will start with a modestly-sized array, say **size 11**, and **increase the capacity as needed** (see the **grow()** function description).
- Unlike previous assignments you are not given any tests as a starting point. You must create your own tests and submit them with your program. (Note that the previous assignment will be very useful in helping you accomplish this).

## **Part 2 –** SortedBinarySet **Class Implementation**

We start by implementing the easier methods. Remember that the list must be sorted at all times. Please pay close attention to the following notes:

- **public SortedBinarySet()** // default constructor
    - This constructor must initialize an array of **size 11** (hint: use the final variable INITIAL_STORAGE_CAPACITY)
    - and set the rest of the field members accordingly. Pay attention to the member variables of the class! Each should be set to some (appropriate) initial value.

- **public boolean empty()**
    - returns **true** only if "theData" in the SortedBinarySet contains no elements

- **public int size()**
    - returns the number of elements in this SortedBinarySet.

- **public void grow()**
    - this function doubles the size of theData and modifies member variables accordingly.
    - consult this week's slides

- **public String toString()**
    - this method should return the elements of the list, its capacity, and its current size

    information as a string.
  - This method is purely to help you test your implementation.
  - We will NOT test your toString() method when grading.

- **private int sequentialFind(double target)**

  - this method will return the index where the element target occurs. This method must make use of the Sequential search we learned in class. If target is not present in the list, then it should return the index where it should be added (-index - 1). Does this formula make sense?
  - There are three cases to consider. What should be returned in each?
    - target is equal to theData[index]
    - target becomes less than theData[index]
    - the loop terminates without success

- **public boolean insert(double newVal)**

  - If the list does not contain newVal, add it to the correct position of the list and return true. If the list already includes the value, return false.
  - The method should first make sure there is enough room in the list to handle the operation. If not, what should you do?
  - It should then make a call to findIndex(), which then calls the private sequentialFind you just implemented, to see if newVal already occurs in the list (how do we know?). If it isn't in the list, the index returned by findIndex specifies the position where newVal is to be inserted (but it is negative! what should you do? :-)
  - Be careful about which member variables should be updated before returning true.

- **public void clear()**

  - Removes all the elements from the list. A call to empty() should return true after this method is called.
  - Be careful about which member variables should be updated.

## Part 3 – Testing

Unlike previous labs/assignments you are not given any tests as a starting point. You must create your own tests to examine each method you implemented. These tests can be written inside **Tester.java**.

**Here is one potential testing strategy:** create an empty SortedBinarySet. Then, check its size, check whether it is empty. Then, start adding a few numbers to your SortedBinarySet and make sure they are added properly (that is, they are in the sorted order, the other member variables reflect the changes, etc). Then, continue adding values to your SortedBinarySet till you go beyond its original capacity. Make sure you are not having any errors in this case and that the SortedBinarySet does indeed grow. Finally, test your clear method to see if all elements are removed from the SortedBinarySet.

## Part 1 – SortedBinarySet **Class Implementation**

After making sure the methods you implemented during lab work properly, we will extend the functionality of our class by adding new methods. **Please implement the methods in the order given** and pay close attention to the notes:

- public boolean remove(double value)
    - This method should first make a call to findIndex() to find the location of value, which for now still calls the private sequentialFind (this will change in the next step). If the element is not present in the list (how do we know?), return **false**. Otherwise, remove the element and return **true** because the list contained the given value.
    - Be careful about which **member variables** should be updated before returning true.
    - Be sure to test your remove method before moving forward.

- public boolean binaryFind(double target)
    - this method will return the index where the element target occurs. This method **must** make use of binary search. If target is not present in the list, it should return the index where it should be added. Use the same formula we learned during the lab.
    - Change the boolean `usesBinarySearch` to be **false** so that findIndex now calls this method instead. insert() and remove() now make use of binary search :-)

- public boolean contains(double value)
    - this function returns true if this list contains the value
    - otherwise, it returns false
    - this method must take advantage of the list being sorted and use a binary search to determine if an item is in the array. (Hint: try reusing your other methods.)

- public boolean containsAll(double[] elements)
    - this function returns **true** if all the input values (stored in elements) are in the list
    - otherwise, return **false**
    - this method must take advantage of the list being sorted and use a binary search to determine if an item is in the array. (**Hint**: try reusing your other methods.)

- public SortedBinarySet(double[] input)
    - This is another constructor for your class that will accept an array and create an object of SortedBinarySet that includes the values in the input array
    - Be careful of the following cases
        - the input might have duplicates (remember your list is not allowed to contain duplicates)
        - the input might not be sorted. **Hint:** think whether you can reuse any of the methods you have implemented to make your job easier.

## Part 2 – Testing

Unlike previous assignments you are not given any tests as a starting point. You **must** create your own tests to examine each method you implemented in the previous part.

## Part 3 – sequentialFind **or** binaryFind? 🤯

As we have seen this week during lecture, some algorithms are more efficient than others. For Lab05 we implemented a *search* routine two ways: sequentialFind(), and binaryFind().

- For the last part of this assignment, we would like you to think about which one is more efficient. Remember that findIndex() will use either method based on the boolean variable usesBinarySearch.
- When you think you have an answer, set the variable usesBinarySearch to the appropriate value (either **true** or **false**) so findIndex() makes use of that method.
    - For example, setting usesBinarySearch to false would indicate that sequentialFind() is faster than binaryFind().

To test this out in your Tester, you may consider inserting 100,000 elements (using a loop) into the sorted set and comparing the time it takes (using System.nanoTime()) to complete the operation when using sequentialFind or binaryFind.