Nicolas Grajales

**Introduction:**

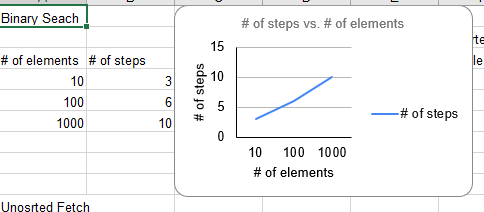
In this lab we study the Big O notation of fetch, insert and binary search operations on unsorted arrays in Java. Our motivation is to understand the differences in time complexity and efficiency between the operations on these types of arrays as well as to compare the experimental results.

**Procedure:**

1. Create multiple unsorted arrays of different sizes which are one small one of 10 elements, a medium one of 100 elements and a large one of 1000 elements.
2. Implement the fetch, insert and binary search algorithms in java
3. Execute the algorithms on each array and record them using the amount of inputs and the amount of steps that were performed.
4. Created a plot diagram with each one representing the amount of steps that were taking by the algorithm by using the set given number of elements.

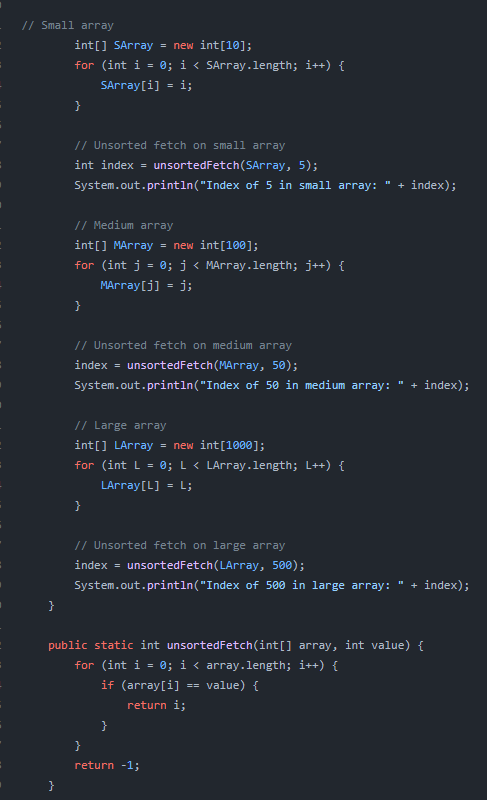
**Results:**

As you can see the binary search algorithm used the least amount of steps in order to finish. Going from the medium array to the large array it shows that the number of steps starts to increase since at first it only started with 3 steps. This made us think that it would be going in steps of 3 but this was not the case since from the medium array to the large array it incremented by 4 steps.



**Discussion:**

Unsorted fetch



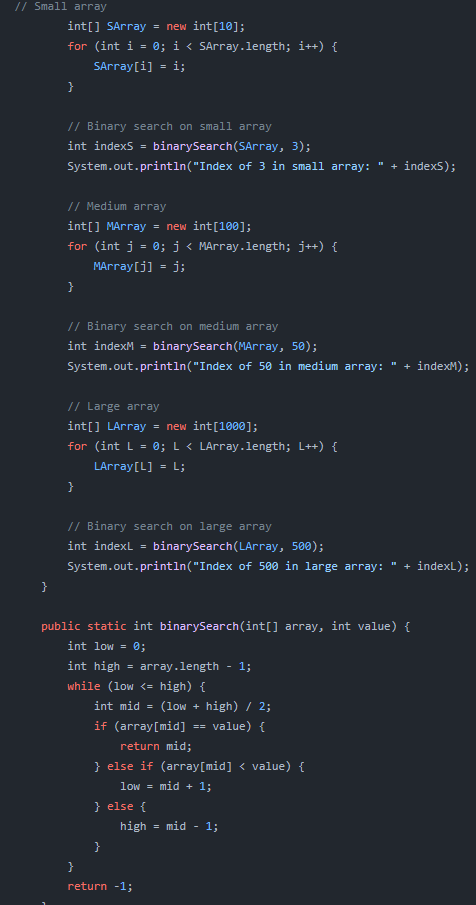
This Java program demonstrates how to find the index of a specific value in an array using an unsorted fetch algorithm. The program creates a small, medium and large array and initializes them with sequential integers and then performs an unsorted fetch of a value in each array using the unsorted fetch algorithm. The unsorted fetch method takes an array and a value as input and returns the index of the value in the array if it is present or -1 if it is not present. The algorithm simply loops through the array and checks each element for the specified value using an if statement. If the value is found the algorithm returns the index of that element. The program then prints out the index of the specified value in each array to the console using the System.out.println method. This unsorted fetch method has a time complexity of O(n) where n is the length of the input array since it requires looping through the entire array to find the value. This algorithm is inefficient for large arrays especially if multiple fetches are needed since each fetch requires looping through the entire array. In such cases a different search algorithm such as a binary search would be more efficient.

Unsorted Insert



This Java program demonstrates how to insert an element at a specific index in an array using an unsorted insert. The program creates three arrays of small, medium, and large. The unsorted insert algo takes an array, an index and a value as input and returns a new array with the specified value inserted at the specified index. First a new array is created with a length one greater than the input array then loops through the input array and copies its values into the new array but shifts the values up by one index if they are after the insert point. Finally the method sets the new value at the insertion point and returns the new array. The program then prints out the original arrays and the new arrays with the inserted values to the console using the Arrays.toString method. This unsorted insert algorithm has a time complexity of O(n) where n is the length of the input array since it requires copying all elements of the input array to the new array. This method can be inefficient for large arrays especially if multiple insertions are needed since each insertion requires copying the entire array.

Binary Search



The number of steps that the binary search algorithm takes to find an element in a sorted array of size n is given by O(log n). The number of steps required by the algorithm grows much slower than the size of the array. For example, in an array of size 1000, the binary search algorithm will take at most log2(1000) = 10 steps to find an element. In an array of size 1 million, the algorithm will take at most log2(100) = 6 steps. And in an array size of log2(100)=6 This is why binary search is considered a very efficient algorithm for searching through large sorted arrays.

**Conclusion:**

By comparing each algorithm we gained insight on their complexities written in java code and their efficiencies. This showed proof that some others are more efficient than other algorithms.This knowledge that me and my partner have gained can now be used in real world applications and can be used for future research on algorithm performance. We can possibly examine other data structures and their corresponding algorithms as well as learning how to further optimize those algorithms.