Experiment 5. Data Structs

Ⅰ. Experiment purpose

For this lab, you will practice the quick sort, merge sort, sequential search and quick sort. You will implement the sorting and searching algorithms in a Java program and show the process step-by-step.

Ⅱ. Use instruments and materials

Eclipse2020-06 (4.16.0), ProcessOn

Ⅲ. Experiment content

Given the following list of string objects,

{“Jujube”, “Orange”, “Logan”, “Pomegranate”, “Raspberry”, “Cantaloupe”, “Carambola”, “Date palm”, “Coconut”}

• Sort the list by the quick sort algorithm. You may choose any element as the pivot.

• Sort the list again by the merge sort algorithm.

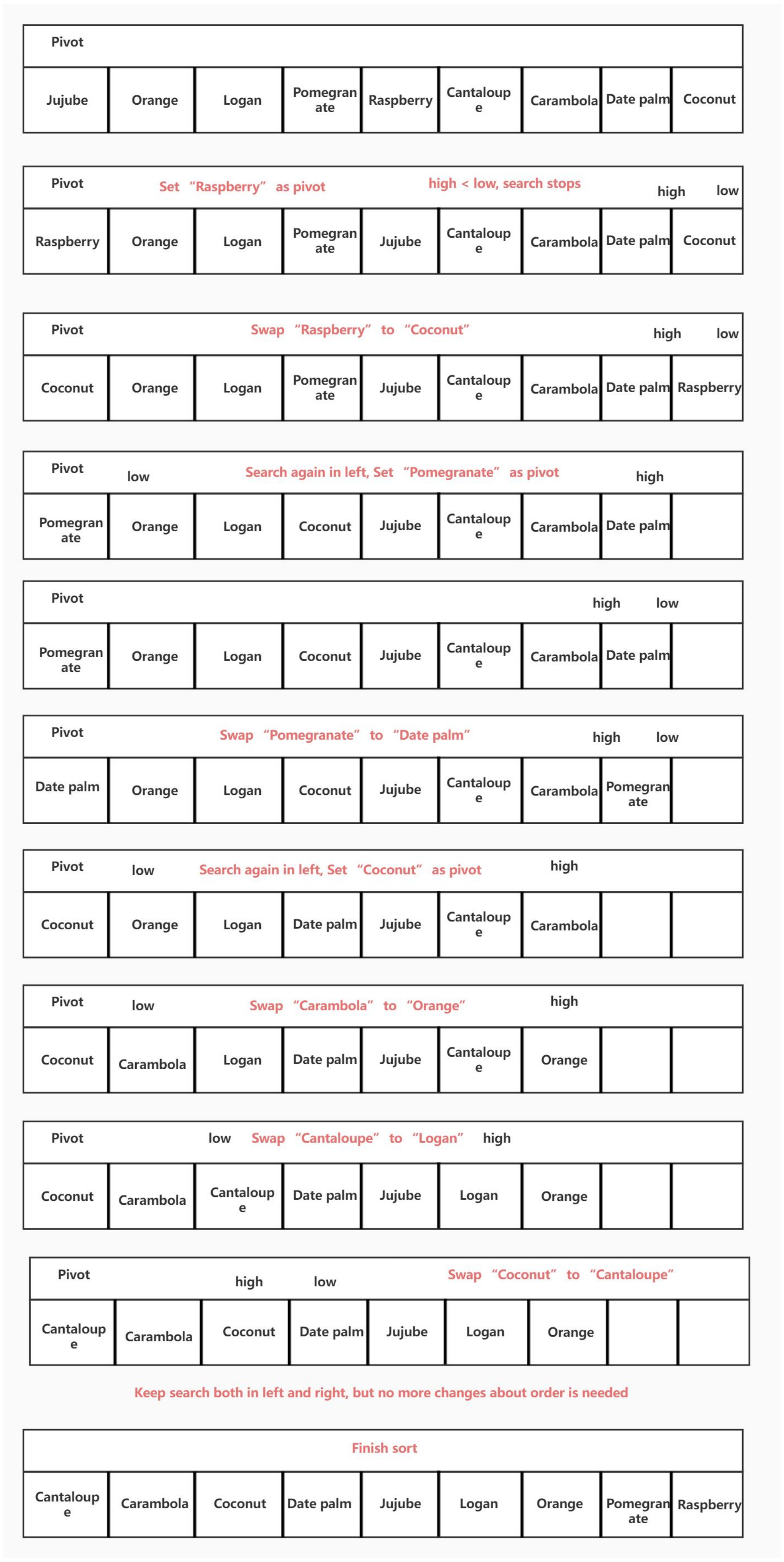
• In the sorted list, search “Logan” using the binary search algorithm.

• In the sorted list, search ”Apple” using the sequential search algorithm.

Ⅳ. Test steps and process records

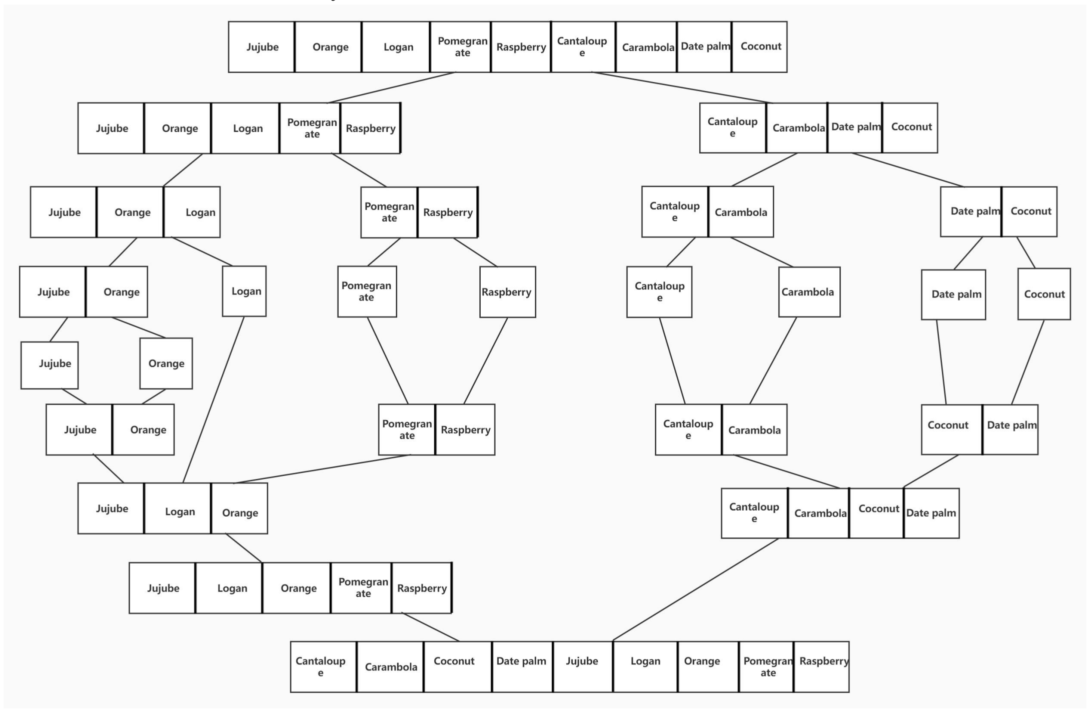
Step1:

Write QuickSorter class to complete the quick sort algorithm. In this class, finishing quickly sort according to the steps explained in the class. Briefly explain, it will partition an array into items that are less than the pivot and those that are greater than or equal to the pivot. Then search for the first element from left forward in the array that is greater than the pivot, then search for the first element from right backward in the array that is less than or equal to the pivot. Swap these two elements.Repeat the same search and swap operations until all the elements are searched in a while loop.


Pivot
Jujube | Orange | Logan | Pomegranate | Raspberry | Cantaloupe | Carambola | Date palm | Coconut
Pivot
Set “Raspberry” as pivot
high < low, search stops
high
low
Raspberry | Orange | Logan | Pomegranate | Jujube | Cantaloupe | Carambola | Date palm | Coconut
Pivot
Swap “Raspberry” to “Coconut”
high
low
Coconut | Orange | Logan | Pomegranate | Jujube | Cantaloupe | Carambola | Date palm | Raspberry
Pivot
low
Search again in left, Set “Pomegranate” as pivot
high
Pomegranate | Orange | Logan | Coconut | Jujube | Cantaloupe | Carambola | Date palm
Pivot
high
low
Pomegranate | Orange | Logan | Coconut | Jujube | Cantaloupe | Carambola | Date palm
Pivot
Swap “Pomegranate” to “Date palm”
high
low
Date palm | Orange | Logan | Coconut | Jujube | Cantaloupe | Carambola | Pomegranate
Pivot
low
Search again in left, Set “Coconut” as pivot
high
Coconut | Orange | Logan | Date palm | Jujube | Cantaloupe | Carambola
Pivot
low
Swap “Carambola” to “Orange”
high
Coconut | Carambola | Logan | Date palm | Jujube | Cantaloupe | Orange
Pivot
low
Swap “Cantaloupe” to “Logan”
high
Coconut | Carambola | Cantaloupe | Date palm | Jujube | Logan | Orange
Pivot
high
low
Swap “Coconut” to “Cantaloupe”
Cantaloupe | Carambola | Coconut | Date palm | Jujube | Logan | Orange
Keep search both in left and right, but no more changes about order is needed
Finish sort
Cantaloupe | Carambola | Coconut | Date palm | Jujube | Logan | Orange | Pomegranate | Raspberry

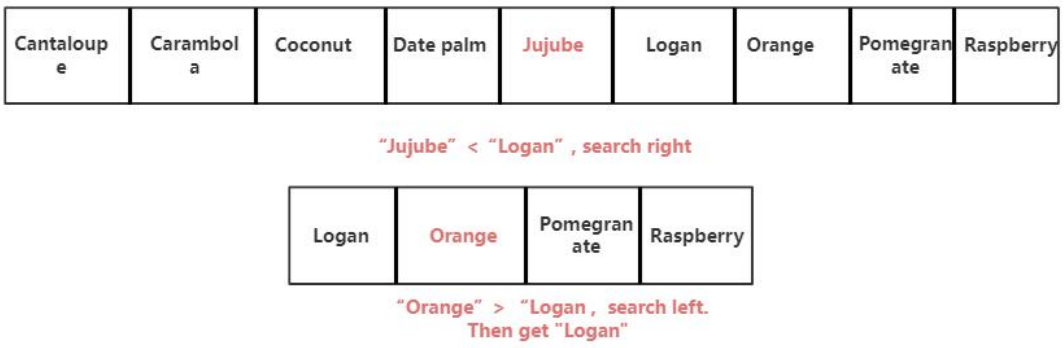
Graph 1 QuickSort Process

Step2: Write MergeSorter class to complete the merge sort algorithm. It's strategy is dividing an array into halves, sorting each half then merge the sorted halves into one sorted array.



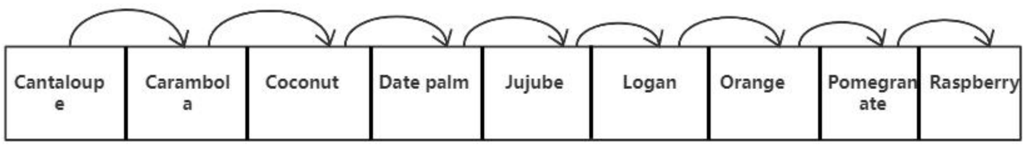
Graph 2 MergeSort Process

Step3: Write BinarySearcher class to complete the binary search algorithm. It compares the target with the middle element of the array. If they are equal, the search ends. If the target value is greater than the middle element, search the right half of the array. If the target value is less than the middle element, search the left half of the array. If the search ends with the remaining half being empty, the target is not in the array.



Graph 3 BinarySearch Process

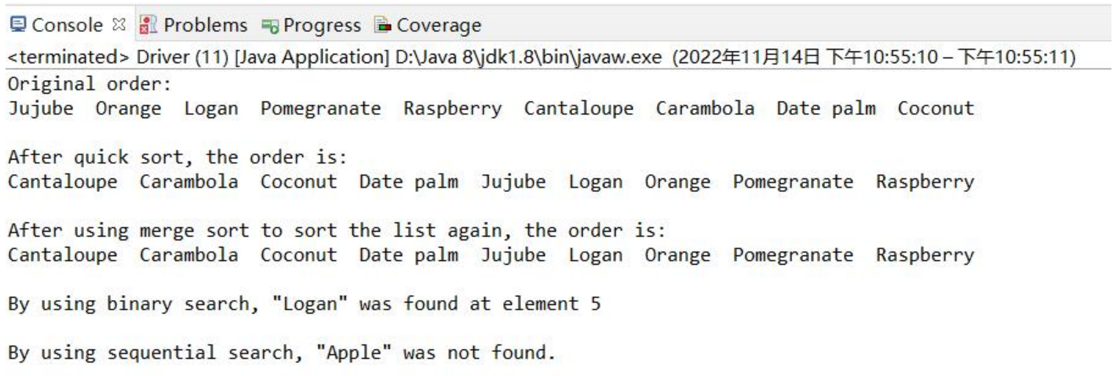
Step4: Write SequentialSearcher class to complete the sequential search algorithm. It compare the elements in array by order until find the element that equals to the value to search.



Graph 4 SequentialSearch Process

Step5: Write a driver class to finish task by invoking 4 classes created before. Using QuickSorter to sort the list. Using MergeSorter to sort the list again. Using BinarySearcher to search “Logan”. Using SequentialSearcher to search “Apple”.

All codes will be showed at the end.



```
<terminated> Driver (11) [Java Application] D:\Java 8\jdk1.8\bin\javaw.exe (2022年11月14日 下午10:55:10 – 下午10:55:11)
Original order:
Jujube  Orange  Logan  Pomegranate  Raspberry  Cantaloupe  Carambola  Date palm  Coconut

After quick sort, the order is:
Cantaloupe  Carambola  Coconut  Date palm  Jujube  Logan  Orange  Pomegranate  Raspberry

After using merge sort to sort the list again, the order is:
Cantaloupe  Carambola  Coconut  Date palm  Jujube  Logan  Orange  Pomegranate  Raspberry

By using binary search, "Logan" was found at element 5

By using sequential search, "Apple" was not found.
```

Graph 5 Running results

V. Experimental gains and experiences

1. Learn about the qucik sort, merge sort, quick search and sequential search.
2. Learn about the inner steps and how to complete by coding about sort and search algorithm.
3. Learn about the pros and cons about each algorithm.

VI. Appendix:Codes

QuickSorter

```
package Lab5;

/**
 * Quick sort on a a list of strings
 * @author Letao Han
 * @version 1.0
 */
public class QuickSorter{
/**
 * Invokes the doQuickSort method to sort an array.
 * @param array The array to sort.
    */
    public static void quickSort(String array[]){
      doQuickSort(array, 0, array.length - 1);
    }

    /**
     * Uses the QuickSort algorithm to sort an array.
     * @param array The array to sort
     * @param start The starting subscript of the list to sort
     * @param end The ending subscript of the list to sort
     */
    private static void doQuickSort(String array[], int start, int end){
      int pivotPoint;

      if (start < end){
         // Get the pivot point.
         pivotPoint = partition(array, start, end);

         // Sort the first sub list.
         doQuickSort(array, start, pivotPoint - 1);

         // Sort the second sub list.
         doQuickSort(array, pivotPoint + 1, end);
      }
    }

   /**
    * Selects a pivot value in an array and arranges the array into two sub
lists.
    * @param array The array to partition
    * @param start The starting subscript of the area to partition
    * @param end The ending subscript of the area to partition
    * @return The subscript of the pivot value
    */
   private static int partition(String array[], int start, int end){
  String pivotValue;    // pivot value
      int endOfLeftList; // last element in the left sub list.
      int mid;           // mid-point subscript

      // Find the subscript of the middle element
      mid = (start + end) / 2;

      // Swap the middle element with the first element.
      swap(array, start, mid);
      // Save the pivot value for comparisons.
      pivotValue = array[start];

      // For now, the end of the left sub list is the first element.
      endOfLeftList = start;

      // Scan the entire list and move any values that
      // are less than the pivot value to the left sub list.
      for (int scan = start + 1; scan <= end; scan++){
         if (array[scan].compareTo(pivotValue)<0)
         {
            endOfLeftList++;
            swap(array, endOfLeftList, scan);
         }
      }

      // Move the pivot value to end of the left sub list.
      swap(array, start, endOfLeftList);

      // Return the subscript of the pivot value.
      return endOfLeftList;
   }

   /**
    * Swaps the contents of two elements in an array.
    * @param The array containing the two elements
    * @param a The subscript of the first element
    * @param b The subscript of the second element
    */
   private static void swap(String[] array, int a, int b){
   String temp;

      temp = array[a];
      array[a] = array[b];
      array[b] = temp;
   }


}
```

MergeSorter:

```
package Lab5;

/**
```

```
 * Merge sort a string array
 * @author Letao Han
 * @version 1.0
 */
public class MergeSorter {
/**
 * Sorts an array and creates a temporary array used for merge.
 * @param array The array of items to be sorted
 */
public static void mergesort(String array[])
{
    String[] tempArray = new String[array.length];
    mergesort(array, tempArray, 0, array.length - 1 );
}

/**
 * Merges two sorted array segments first..mid and mid+1..last into one
sorted array.
 * @param array The array of items to be sorted
 * @param tempArray An array to contain sorted items
 * @param first The first index of the array
 * @param mid The mid index of the array
 * @param last The last index of the array
 */
private static void merge(String[] array, String[] tempArray, int first,
int mid, int last) {
    // initialize the local indexes to indicate the subarrays
    int first1 = first;
    int last1  = mid;
    int first2 = mid + 1;
    int last2  = last;

    // next available location in tempArray
    int index = first1;

    // Sort two array segments and store them into tempArray
    while ((first1 <= last1) && (first2 <= last2)) {
         if (array[first1].compareTo(array[first2])<0) {
           tempArray[index] = array[first1];
           first1++;
         }
         else {
           tempArray[index] = array[first2];
           first2++;
```

```
            }
            index++;
        }

        // finish off the nonempty subarray
        while (first1 <= last1) {
            tempArray[index] = array[first1];
            first1++;
            index++;
        }
        while (first2 <= last2) {
            tempArray[index] = array[first2];
            first2++;
            index++;
        }

        // copy the result back into the original array
        for (index = first; index <= last; ++index) {
           array[index] = tempArray[index];
        }
    }

    /**
     * Sorts the items in an array into ascending order.
     * @param array The array of items to be sorted
     * @param tempArray An array to contained sorted items
     * @param first The first index of the array
     * @param last The last index of the array
     */
    public static void mergesort(String[] array, String[] tempArray,  int first,
int last) {
        if (first < last)
        {
         int mid = (first + last)/2;
         // sort left half
         mergesort(array, tempArray, first, mid);
         // sort right half
         mergesort(array, tempArray, mid+1, last);
         // merge the two halves
         merge(array, tempArray, first, mid, last);
        }
    }
    }
```

BinarySearcher:

```
package Lab5;

/**
 * Binary search on a a list of Strings.
 * @author Letao Han
 * @version 1.0
 */
public class BinarySearcher{
   /**
    * Searches an array for the string passed to value. If the number is found,
its array subscript is
    * returned. Otherwise, -1 is returned indicating the value was not found
in the array.
    * @param array The array to search.
    * @param value The string to search for.
    */
   public static int search(String[] array, String value) {
      int first;      // First array element
      int last;       // Last array element
      int middle;     // Mid point of search
      int position;   // Position of search value
      boolean found;  // Found or not

      first = 0;
      last = array.length - 1;
      position = -1;
      found = false;

      // Search for the string.
      while (!found && first <= last) {
         middle = (first + last) / 2;

         // If value is found at midpoint
         if (array[middle] == value) {
            found = true;
            position = middle;
         }else if (array[middle].compareTo(value) > 0){
        // else if value is in lower half
            last = middle - 1;
         }else{
             // else if value is in upper half
            first = middle + 1;
         }
      }

      // Return the position of the string, or -1 if it was not found.
      return position;
   }
}
```

SequentialSearcher:

```
package Lab5;
/**
 * Sequential search on a a list of Strings
 * @author Letao Han
 * @version 1.0
 */
public class SequentialSearcher{
   /**
    * Searches an array for a value.
    * @param array The array to search
    * @param value The value to search for
    * @return The subscript of the value if found in the array, otherwise
-1
    */
   public static int search(String[] array, String value){
      int index;       // Loop control variable
      int position;    // Position the value is found at
      boolean found;   // Flag indicating search results

      // Element 0 is the starting point of the search.
      index = 0;

      // Store the default values position and found.
      position = -1;
      found = false;

      // Search the array.
      while (!found && index < array.length) {
         if (array[index] == value) {
            found = true;
            position = index;
         }
         index++;
      }

      // Return the found element's position,
         // or -1 if not found.
         return position;
      }
   }
   Driver:
   package Lab5;

   /**
    * Driver class to complement the tragets of lab5
    * @author Letao Han
    * @version 1.0
    */
   public class Driver {
   /**
    * Complements the tragets of lab5
    * @param args A reference to a string array containing command-line
arguments
    */
       public static void main(String[] args){
         String[] values = { "Jujube", "Orange", "Logan", "Pomegranate",
"Raspberry", "Cantaloupe", "Carambola",
              "Date palm", "Coconut"};// An array with the given strings
         int binaryResult; // Store the value of binary search
         int sequentialResult;// Store the value of sequential search

         // Display the array's contents.
         System.out.println("Original order: ");
         for (String element : values){
            System.out.print(element + "  ");
         }
         System.out.println();

         // Sort the array by quick sort
         QuickSorter.quickSort(values);

         // Display the array sorted by quick sort
         System.out.println("\nAfter quick sort, the order is: ");
         for (String element : values){
            System.out.print(element + "  ");
         }
         System.out.println();

         // Sort the array by merge sort
         MergeSorter.mergesort(values);
```

```
          // Display the array sorted by merge sort
          System.out.println("\nAfter using merge sort to sort the list again, the order is: ");
          for (String element : values){
             System.out.print(element + "  ");
          }
          System.out.println();

          // Use binary search to search "Logan"
          binaryResult = BinarySearcher.search(values, "Logan");
          // Display the results.
          if (binaryResult == -1){
             System.out.println("\nBy using binary search, \"Logan\"" + " was not found.");
          }else{
              System.out.println("\nBy using binary search, \"Logan\" was found at element " + binaryResult);
          }

          // Use sequential search to search "Apple"
          sequentialResult = SequentialSearcher.search(values, "Apple");
          // Display the results.
          if (sequentialResult == -1){
             System.out.println("\nBy using sequential search, \"Apple\"" + " was not found.");
          }else{
              System.out.println("\nBy using sequential search, \"Apple\" was found at element " + binaryResult);
          }
       }
    }
```