L2-Project Assignment

TopGear
Engage | Be Future Ready

Project Title – Java Search And Sort Algorithms -L2-Assignment

Proejct Type	Project Name/ID	Hosted By TG/Acccount	Duration	Proejct Owner /Account
L2 Assignment	Java Search And Sort Algorithms -L2-Assignment	TopGear	5 PDs	NA

Table of Contents

About This Project

This is L2 level assignment for experienced Java programmers on algorithms based search and sort programming.

Who Is Eligible

All users who are experienced in OOPS concepts, basic knowledge on search and sort functionalities using Java programming.

Pre-Requisites

1. User need to have experience in Object Oriented Programming Concepts (OOPS) and Java Collection API.
2. Should have completed Core Java L1 level training / need to have hands on experience in Java programming.
3. User is recommended to consult a TopGear mentor and clarify doubts/queries (if any) before joining the project.
4. Project to be submitted to TopGear project repository using GIT with appropriate documentation and packaged as guided by TopGear mentor.
5. Irrelevant submission will be barred from review and will be considered not completed, which will be re-opened with appropriate comments.

Scope of Work & Deliverables

1. Coding & Testing
2. Source files packaged as guided.

Technologies

This assignment uses Java language to explore and learn Core Java Design Patterns.

Training Tasks

1. Task to implement binary search using Recursive Algorithm :

A binary search or half-interval search algorithm finds the position of a specified value (the input "key") within a sorted array. In each step, the algorithm compares the input key value with the key value of the middle element of the array. If the keys match, then a matching element has been found so its index, or position, is returned. Otherwise, if the sought key is less than the middle element's key, then the algorithm repeats its action on the sub-array to the left of the middle element or, if the input key is greater, on the sub-array to the right. If the remaining array to be searched is reduced to zero, then the key cannot be found in the array and a special "Not found" indication is returned.
Every iteration eliminates half of the remaining possibilities. This makes binary searches very efficient - even for large collections. Binary search requires a sorted collection. Also, binary searching can only be applied to a collection that allows random access (indexing).
Worst case performance: O(log n)
Best case performance: O(1)
Recursion is used in this algorithm because with each pass a new array is created by cutting the old one in half. The binary search procedure is then called recursively, this time on the new array. Typically the array's size is adjusted by

manipulating a beginning and ending index. The algorithm exhibits a logarithmic order of growth because it essentially divides the problem domain in half with each pass.

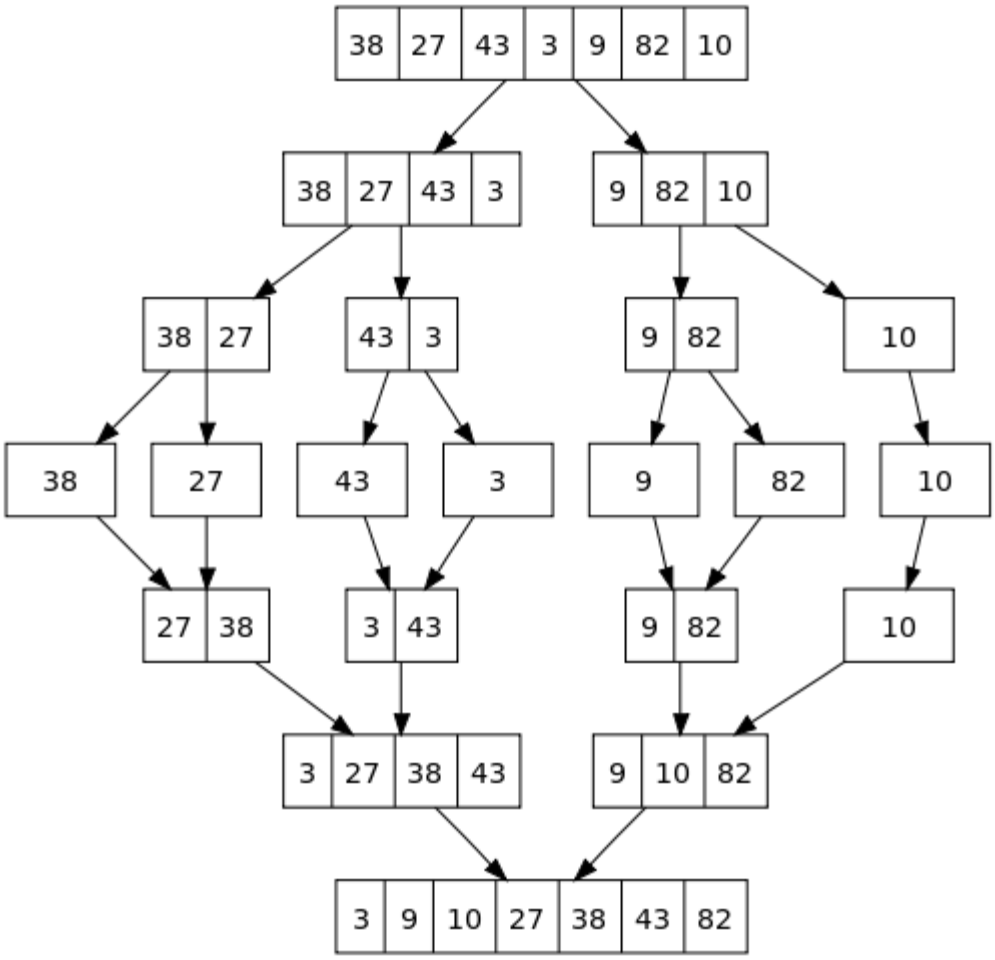
```
public class MyRecursiveBinarySearch {


    public static int recursiveBinarySearch(int[]
sortedArray, int start, int end, int key) {

//write your logic
    }


    public static void main(String[] args) {

        int[] arr1 = {2,45,234,567,876,900,976,999};
        int index = recursiveBinarySearch(arr1,0,arr1.length,45);
        System.out.println("Found 45 at "+index+" index");
        index = recursiveBinarySearch(arr1,0,arr1.length,999);
        System.out.println("Found 999 at "+index+" index");
        index = recursiveBinarySearch(arr1,0,arr1.length,876);
        System.out.println("Found 876 at "+index+" index");
    }
}
```

Output:
Found 45 at 1 index
Found 999 at 7 index
Found 876 at 4 index

2. Task to implement merge sort in Java:

Merge sort is a divide and conquer algorithm.
Steps to implement Merge Sort:

1) Divide the unsorted array into n partitions, each partition contains 1 element. Here the one element is considered as sorted.
2) Repeatedly merge partitioned units to produce new sub-lists until there is only one sub list remaining. This will be the sorted list at the end.



Merge sort is a fast, stable sorting routine with guaranteed O(n*log(n)) efficiency. When sorting arrays, merge sort requires additional scratch space proportional to the size of the input array. Merge sort is relatively simple to code and offers performance typically only slightly below that of quicksort.

```
public class MyMergeSort {

    private int[] array;
    private int[] tempMergArr;
    private int length;

    public static void main(String a[]){

        int[] inputArr = {45,23,11,89,77,98,4,28,65,43};
        MyMergeSort mms = new MyMergeSort();
        mms.sort(inputArr);
        for(int i:inputArr){
            System.out.print(i);
            System.out.print(" ");
        }
    }
```

```
    public void sort(int inputArr[]) {
        this.array = inputArr;
        this.length = inputArr.length;
        this.tempMergArr = new int[length];
        doMergeSort(0, length - 1);
    }


    private void doMergeSort(int lowerIndex, int higherIndex) {


        if (lowerIndex < higherIndex) {
            int middle = lowerIndex + (higherIndex - lowerIndex) / 2;
            // Below step sorts the left side of the array
            doMergeSort(lowerIndex, middle);
            // Below step sorts the right side of the array
            doMergeSort(middle + 1, higherIndex);
            // Now merge both sides
            mergeParts(lowerIndex, middle, higherIndex);
        }
    }


    private void mergeParts(int lowerIndex, int middle, int higherIndex) {


        for (int i = lowerIndex; i <= higherIndex; i++) {
            tempMergArr[i] = array[i];
        }
        int i = lowerIndex;
        int j = middle + 1;
        int k = lowerIndex;
        while (i <= middle && j <= higherIndex) {
            if (tempMergArr[i] <= tempMergArr[j]) {
                array[k] = tempMergArr[i];
                i++;
            } else {
                array[k] = tempMergArr[j];
                j++;
            }
            k++;
        }
        while (i <= middle) {
            array[k] = tempMergArr[i];
            k++;
            i++;
        }


    }
}
```

Output:
4 11 23 28 43 45 65 77 89 98

3. Task to implement bubble sort in java :

Bubble sort, also referred to as sinking sort, is a simple sorting algorithm that works by repeatedly stepping through the list to be sorted, comparing each pair of adjacent items and swapping them if they are in the wrong order. The pass through the list is repeated until no swaps are needed, which indicates that the list is sorted. The algorithm gets its name from the way smaller elements "bubble" to the top of the list. Because it only uses comparisons to operate on elements, it is a comparison sort. Although the algorithm is simple, most of the other sorting algorithms are more efficient for large lists.

Bubble sort has worst-case and average complexity both O(n2), where n is the number of items being sorted. There exist many sorting algorithms with substantially better worst-case or average complexity of O(n log n). Even other O(n2) sorting algorithms, such as insertion sort, tend to have better performance than bubble sort. Therefore, bubble sort is not a practical sorting algorithm when n is large. Performance of bubble sort over an already-sorted list (best-case) is O(n).

```
public class MyBubbleSort {



    // logic to sort the elements
    public static void bubble_srt(int array[]) {
        int n = array.length;
        int k;
        for (int m = n; m >= 0; m--) {
            for (int i = 0; i < n - 1; i++) {
                k = i + 1;
                if (array[i] > array[k]) {
                    swapNumbers(i, k, array);
                }
            }
            printNumbers(array);
        }
    }

    private static void swapNumbers(int i, int j, int[] array) {

        int temp;
        temp = array[i];
        array[i] = array[j];
        array[j] = temp;
    }

    private static void printNumbers(int[] input) {

        for (int i = 0; i < input.length; i++) {
            System.out.print(input[i] + ", ");
        }
        System.out.println("\n");
    }

    public static void main(String[] args) {
        int[] input = { 4, 2, 9, 6, 23, 12, 34, 0, 1 };
        bubble_srt(input);

    }
}
```

Output:

2, 4, 6, 9, 12, 23, 0, 1, 34,

2, 4, 6, 9, 12, 0, 1, 23, 34,

2, 4, 6, 9, 0, 1, 12, 23, 34,

2, 4, 6, 0, 1, 9, 12, 23, 34,

2, 4, 0, 1, 6, 9, 12, 23, 34,

2, 0, 1, 4, 6, 9, 12, 23, 34,

0, 1, 2, 4, 6, 9, 12, 23, 34,

0, 1, 2, 4, 6, 9, 12, 23, 34,

0, 1, 2, 4, 6, 9, 12, 23, 34,

0, 1, 2, 4, 6, 9, 12, 23, 34,

Environment

To execute this training project user can use own desktop/laptop set with essential Java development environment (need GitBash tool) or can use TopGear provided VDI (Spring SOA Hibernate) which is configured for Java and Java EE development

Completion Criteria

- Employee should complete the project within timeline provided.
- In cased user has not follows appropriate steps or job is partly finished, then project can be reopened/reverted back.
- Employee must follow all rules and task as instructed to avoid rejection/reopening of projects.
- Once the project is closed successfully after submission and review, user will earn # reward points and the user will visible to appropriates practices and accounts.
- If the user has completed a Use case, project or Solution project, he/she must update his/her Wipro profile add the project details he worked.
- User should update efforts against TopGear projects in time sheet

Assumptions

Users joining this program have thorough knowledge of OOPS concepts, Search and Sort functionalities using Java programming.

Project Submission Guidelines

- Login https://topgear.wipro.com
- Go to Java-J2EE → Java→Training Tab
- Join Java Search And Sort Algorithms -L2-Assignment project.
- Practice on assignment as per project specification.
- Once complete the project work, submit/check-in the same as instructed to TopGear GitLab.

- User can refer following document to understand execution and submission procedure of TopGear training and case study projects

https://topgear-app.wipro.com/sites/default/files/steps_to_execute_topgear_training_case_study_projects_v-1-2-1.pdf

References

- http://www.vogella.com/tutorials/JavaAlgorithmsSearch/article.html
- http://www.vogella.com/tutorials/JavaAlgorithmsMergesort/article.html

http://www.javatpoint.com/bubble-sort-in-java