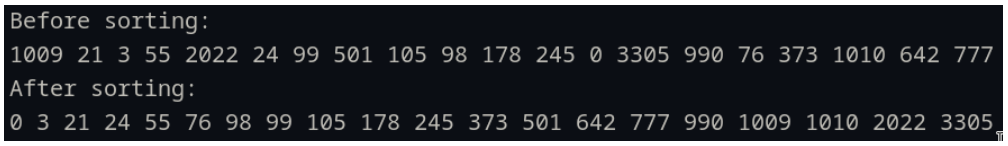
Assignment 5: Big-O Sorting
Part 3: Merge Sort
CS3305/W01 Data Structures

Casey Hampson

September 18, 2024

Program Output



```
Before sorting:
1009 21 3 55 2022 24 99 501 105 98 178 245 0 3305 990 76 373 1010 642 777
After sorting:
0 3 21 24 55 76 98 99 105 178 245 373 501 642 777 990 1009 1010 2022 3305
```

Source Code

```java
// Name: Casey Hampson
// Class: CS 3305/W01
// Term: Fall 2024
// Instructor: Sharon Perry
// Assignment: 05-Part-3-Sorting


import java.io.File;
import java.io.FileNotFoundException;
import java.util.LinkedList;
import java.util.List;
import java.util.ListIterator;
import java.util.Scanner;


public class P3 {

    // grabs all the integers from the text file
    public static List<Integer> GetListFromFile(String file_path) {
        // create the linked list as well as the file handle
        List<Integer> list = new LinkedList<>();
        File file = new File(file_path);

        // try-with-resources for the scanner
        try(
            Scanner file_scanner = new Scanner(file);
        ) {
            while (file_scanner.hasNextLine()) {
                list.add(file_scanner.nextInt());
            }
        }
        // creating a scanner from a file path throws a FileNotFoundException, which must be caught.
        catch (FileNotFoundException e) {
            System.out.printf("File \"%s\" not found!\n", file_path);
        }

        return list;
    }

    // just a helper function to print the list
    public static void ListPrint(List<Integer> list) {
        ListIterator<Integer> it = list.listIterator();
        while (it.hasNext()) System.out.printf("%d ", it.next());
        System.out.printf("\n");
    }

    // returns a copy of a subsection of a list (inclusive on both ends)
    public static List<Integer> ListSplit(List<Integer> list, int from, int to) {
        // perform bounds checking to catch errors first
        int size = to - from + 1; //inclusive range on both ends, so we need the +1
        if ((to > list.size()) || (from < 0)) {
            System.out.printf("Invalid bounds!\n");
        }

        // create our new list and an iterator in the input list
        // that starts at the 'from' index
        List<Integer> new_list = new LinkedList<>();
        ListIterator<Integer> it = list.listIterator(from);

        // then add `size` elements from `list` to `new_list`
        for ( ; size>0; size--) new_list.add(it.next());

        return new_list;
    }


    // merges list1 and list2 into the `dest` list with the elements sorted
    // in increasing order
    public static void ListMerge(
        List<Integer> list1,
        List<Integer> list2,
        List<Integer> dest
    ) {
        // clear the `dest` list since we are overwriting it, basically
        dest.clear();
        // keep track of the indices for the other two lists
        int
            list1_idx = 0,
            list2_idx = 0;

        // our first loop will take as a condition that BOTH of the indices (list 1 and 2)
        // are less than their respective lengths, so that we are, obviously,
        // considering both lists during the loop
        // and merging them in increasing order
        while ((list1_idx < list1.size()) && (list2_idx < list2.size())) {
            // we just compare and place the smaller number into the `dest` list
            if (list1.get(list1_idx) > list2.get(list2_idx)) {
                dest.add(list2.get(list2_idx++));
            } else {
                dest.add(list1.get(list1_idx++));
            }
        }

        // after the above loop exits, then only one of the two lists contains elements still,
        // so we can just consider each case separately
        // where we just add all the remaining elements since they should be sorted already
        while (list1_idx < list1.size()) {
            dest.add(list1.get(list1_idx++));
        }
        while (list2_idx < list2.size()) {
            dest.add(list2.get(list2_idx++));
        }
        // at last, then, `dest` contains the sorted list.
    }
```

```
    public static void MergeSort(List<Integer> list) {
        // we first have the base case of ensuring the list has more than one element
        // otherwise, it's just a number, and is trivially sorted, so we can stop
        if (list.size() > 1) {

            // first, we need to split the linked list into halves,
            // we want them to be copies so the original is not modified;
            // list.subList() does not do a copy, so when conquering and dividing,
            // we get ConcurrentModificationExceptions
            List<Integer>
                first_half  = ListSplit(list, 0, list.size()/2-1),
                second_half = ListSplit(list, list.size()/2, list.size()-1);

            // then we call MergeSort on each half
            MergeSort(first_half);
            MergeSort(second_half);

            // lastly, we merge the two together
            ListMerge(first_half, second_half, list);
        }
    }

    public static void main(String[] args) {
        List<Integer> list = GetListFromFile("./P3/res/mergetext.txt");

        System.out.printf("Before sorting:\n");
        ListPrint(list);

        MergeSort(list);
        System.out.printf("After sorting:\n");
        ListPrint(list);
    }
}
```