Timothy Allec

Name, SID, Date ....

In Class Assignment 9: Merge Sort

Benjamin Sanders, MS November 25, 2020

1 Introduction

You will need to work individually to complete this assignment. Write your name at the top of all pages for this assignment. Turn in all work to Blackboard on or before the deadline to receive credit.

You may use additional libraries and online resources, if you get them approved in writing, over email, from the instructor first. If you have received approval from the instructor, write the approved libraries and any references in the space below.

import java.util.Vector; import java.LinkedList;

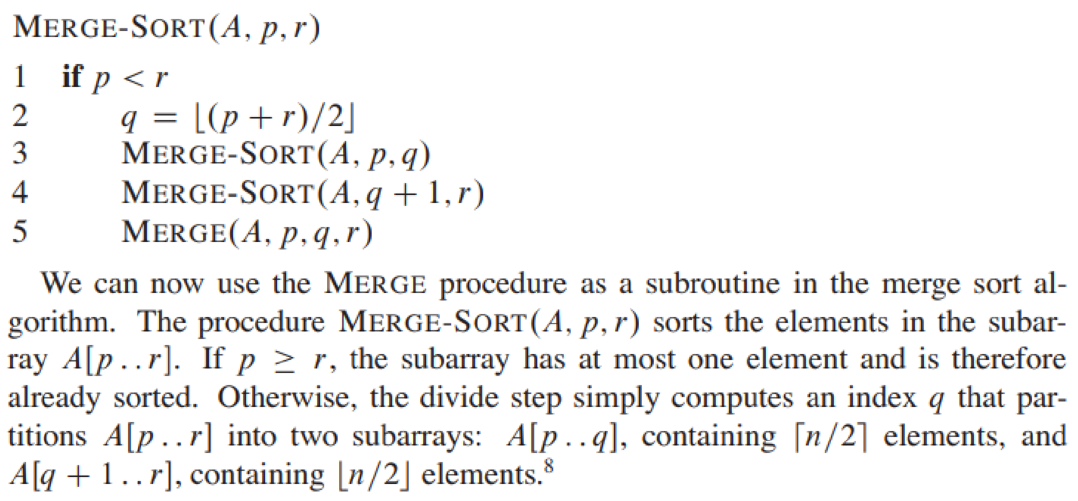
2 Assignment Description

2.1 Big Picture

This algorithm is the most common sorting method that professional programmers implement when they quickly need to create a sorting method for a non-standard object, such as something which is not an Integer, Double or String.

2.2 Algorithm Implementation

Implement the following algorithm in Java, using the Vector data structure for any 1-D array, 2-D array, or linear algebra purposes.



```
MERGE-SORT($A, p, r$)
1  if $p < r$
2      $q = \lfloor (p + r)/2 \rfloor$
3      MERGE-SORT($A, p, q$)
4      MERGE-SORT($A, q + 1, r$)
5      MERGE($A, p, q, r$)
```

We can now use the MERGE procedure as a subroutine in the merge sort algorithm. The procedure MERGE-SORT(A, p, r) sorts the elements in the subarray $A[p \,..\, r]$. If $p \geq r$, the subarray has at most one element and is therefore already sorted. Otherwise, the divide step simply computes an index q that partitions $A[p \,..\, r]$ into two subarrays: $A[p \,..\, q]$, containing $\lceil n/2 \rceil$ elements, and $A[q + 1 \,..\, r]$, containing $\lfloor n/2 \rfloor$ elements.[8]

Name, SID, Date ....

```
MERGE(A, p, q, r)
n_1 = q - p + 1
n_2 = r - q
let L[1 .. n_1 + 1] and R[1 .. n_2 + 1] be new arrays
for i = 1 to n_1
        L[i] = A[p + i - 1]
for j = 1 to n_2
        R[j] = A[q + j]
L[n_1 + 1] = ∞
R[n_2 + 1] = ∞
i = 1
j = 1
for k = p to r
        if L[i] ≤ R[j]
                A[k] = L[i]
                i = i + 1
        else A[k] = R[j]
                j = j + 1
```

```
import java.util.Vector;
public class MergeSort{
    public static void merge_sort(Vector<Integer> A, int p, int r){
        if(p < r){
            int q =(p+r)/2;
            merge_sort(A, p, q);
            merge_sort(A, q+1, r);
            merge(A, p, q, r);
        }
    }

        public static void merge(Vector<Integer> A, int p, int q, int r){
            int n1 = q-p+1;
            int n2 = r-q;
            Vector<Integer> L = new Vector<Integer>();
            Vector<Integer> R = new Vector<Integer>();
            for(int i = 0; i < n1; i++){
                L.add(A.get(p+i));
            }
            for(int j = 0; j < n2; j++){
                R.add(A.get(q+j+1));
            }
            L.add(Integer.MAX_VALUE);
            R.add(Integer.MAX_VALUE);
            int i = 0;
            int j = 0;
            for(int k = p; k <= r; k++){
                if(L.get(i) <= R.get(j)){
                    A.set(k, L.get(i));
                    i++;
                }
                else{
                    A.set(k, R.get(j));
                    j++;
                }
            }

    public static void main(String[] args) {
            Vector<Integer> A = new Vector<Integer>();
            A.add(3);
            A.add(41);
            A.add(52);
            A.add(26);
            A.add(38);
            A.add(57);
            A.add(9);
            A.add(49);
            System.out.println("Before sorting: " + A);
            merge_sort(A, 0, A.size()-1);
            System.out.println("After sorting: " + A);

            }
        }
```

2.3 Time Complexity Analysis

Where n is the number of data points in A, analyze the time complexity of the given algorithm with respect to n. Write the result of your analysis in big-O notation, i.e. $O(n^2)$ in the space below.

O(n * log(n))
....

2.4 Space Complexity Analysis

Where n is the number of data points in A, analyze the space complexity of the given algorithm with respect to n. Write the result of your analysis in big-O notation, i.e. $O(n \cdot log(n))$ in the space below.

O(n * log(n))
....

Name, SID, Date ....

2.5 Optimize the Algorithm for a Purpose

Choose an optimization, ~~either in time or~~ in space, for the given algorithm. Write your intended big-O notation, i.e. $O(1)$, $O(n)$, etc., in the space below, and write N/A in the other space.

- ~~Time Complexity:~~ ....N/A...
- Space Complexity: ....O(n)..

What application would benefit from the purpose of the above optimization? Why? Write two sentences to answer these questions in the space below.

This would be useful at a company like the one where Professor Sander's brother works at (Google) where they hold annual Merge Sort optimization competitions

2.6 New Algorithm Design and Implementation

In the space below, design an algorithm that achieves the same purpose of the given algorithm, but includes the optimization you have specified above. Use pseudocode written in a style similar to the given algorithm, and implement it in Java. You may use as many additional pages as necessary for this purpose.

```
// this optimization introduces an "in line merge sort" that is more efficient than the original merge sort
import java.util.LinkedList;
public class MergeSortOptimization{
    public static void inLineMergeSort(LinkedList<Integer> A, int p, int r){
        if(p < r){
            int q = (p + r) / 2;
            inLineMergeSort(A, p, q);
            inLineMergeSort(A, q + 1, r);
            inLineMerge(A, p, q, r);
        }
    }

    public static void inLineMerge(LinkedList<Integer> A, int p, int q, int r){
        LinkedList<Integer> L = new LinkedList<Integer>(A.subList(p, q + 1));
        LinkedList<Integer> R = new LinkedList<Integer>(A.subList(q + 1, r + 1));
        L.add(Integer.MAX_VALUE);
        R.add(Integer.MAX_VALUE);
        int i = 0;
        int j = 0;
        for(int k = p; k <= r; k++){
            if(L.get(i) <= R.get(j)){
                A.set(k, L.get(i));
                i++;
            } else {
                A.set(k, R.get(j));
                j++;
            }
        }
    }
```

```
    public static void main(String[] args){
        LinkedList<Integer> A = new
LinkedList<Integer>();
        A.add(3);
        A.add(41);
        A.add(52);
        A.add(26);
        A.add(38);
        A.add(57);
        A.add(9);
        A.add(49);
        System.out.println("Before
sorting: " + A);
        inLineMergeSort(A, 0, A.size() - 1);
        System.out.println("After sorting: "
+ A);
    }
}
```

3 What to Turn In

Turn in one PDF or Word document on Blackboard, containing the following items.

1. All pages scanned or photographed of the In Class Assignment completed document.
2. Any additional pages you used to complete the assignment.
3. All code created for the assignment, along with test cases.
4. One statement indicating which parts of your implementation(s) are working, and which parts are not.
5. Screenshots demonstrating the code working, if it is working.