Noah Buchanan

Problem Set 2
Algorithms

August 27, 2020

Modifications

The modifications made were extremely simple, I simply included a new arguement, i, into the mergeSort() alogrithm to change the base case to be i < h, h of course being the height that we wanted to stop merge sorting and begin insertion sorting, and i being an index to keep track of the current height. Once the height is reached we sort the remaining arrays with InsertionSort() and merge them together.

Arguement for $\theta(g(n))$

For any well sized h, the algorithm's average case is $\theta(nlgn)$ it is logarithmic time to reach h and mergeSort() touches all values for each iteration, then once it reaches h InsertionSort() touches each array that must be sorted, this depends on the value of h that we choose. InsertionSort() runes in $O(n^2)$ at its worse case but in the event of it being implemented into mergeSort() InsertionSort() has a best case run time of $\omega(n)$ this is because the inner for loop never executes if the collection is already sorted, the way this works for UASortAndMerge is that each InsertionSort() runs very close to linear time as it splits up each collection into smaller collections, which in turn means less inversions therefore the time runs closer to linear.

UASortAndMerge Class source code

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;

public class UASortAndMerge {
```

```
public static void main(String[] args) throws IOException{

        String inputFile = args[0];
        String outputFile = args[1];
        int h = Integer.parseInt(args[2]);

        BufferedReader br = new BufferedReader(new FileReader(inputFile));

        String line = "";
        String hold = "";

        while((line = br.readLine()) != null) {
                hold += line;
        }

        String[] A = hold.split(" ");

        mergeSort(A,0,A.length-1,h,0);

        BufferedWriter bw = new BufferedWriter(new FileWriter(outputFile));

        int i = 0;
        while(i < A.length) {
                bw.write(A[i++] + " ");
        }


        br.close();
        bw.close();
}

public static void mergeSort(String[] A, int p, int r, int h, int i) {

        if (i < h) {
                int q = (p + r) / 2;
                mergeSort(A, p, q,h,++i);
                mergeSort(A, q + 1, r,h,++i);
                InsertionSort(A);
                i = 0;
                merge(A, p, q, r);
        }
}

public static void merge(String[] A, int p, int q, int r) {

        int n1 = q - p + 1;
```

```
                int n2 = r - q;

                String[] L = new String[n1 + 1];
                String[] R = new String[n2 + 1];

                for (int i = 0; i < n1; i++) {
                        L[i] = A[p + i];
                }
                for (int j = 0; j < n2; j++) {
                        R[j] = A[q + j + 1];
                }

                L[n1] = "zzzzz";
                R[n2] = "zzzzz";

                int i = 0, j = 0;

                for (int k = p; k <= r; k++) {

                        if (L[i].compareTo(R[j]) < 0) {
                                A[k] = L[i];
                                i++;
                        } else {
                                A[k] = R[j];
                                j++;
                        }
                }
        }

        public static void InsertionSort(String[] A) {

                for (int j = 1; j < A.length; j++) {

                        String key = A[j];
                        int i = j - 1;
                        while (i >= 0 && A[i].compareTo(key) >= 0) {

                                A[i + 1] = A[i];
                                i--;
                        }
                        A[i + 1] = key;
                }
        }

}
```