Scott Sullivan

INFSCI 2591

Pengfei Zhou

17 April 2025

Homework 10

Chapter 8 Exercise 2

If we assume that the input arrays (S and T) are sorted in ascending (or descending) order, then we can achieve a runtime of Θ(m+n).

public static void main (String [] args){

int [] S = {2, 5, 7, 14, 19, 20};

int [] T = {4, 5, 6, 7, 13, 20};

//int [] T = {3, 6, 8, 15, 18, 21}; //worst case no matching elements

int [] U = new int[S.length + T.length];

int i, j, k;

i = 0;

j = 0;

k = 0;

int comparisons = 0;

while(i < S.length && j < T.length){

comparisons++;

if(S[i] == T[j]){

U[k] = S[i];

i++;

j++;

k++;

}else if(S[i] < T[j]){

i++;

}else{

j++;

}

}

for(i = 0; i < k; i++){

System.out.print(U[i] + ", ");

}

System.out.println("\n" + comparisons);

}

Chapter 8 Exercise 13:

Files:

* TreeTest.java
* ThreeTwoTree.java
* BTreeNode.java

Analyze your algorithm and show results using order notation.

My implementation in the worst case will have at certain points will have n nodes each containing one key. This worst case is when inserting keys in either ascending or descending order. In this worst-case structure there will be levels. This is also the number of nodes we must traverse for the insert algorithm which must start its insertion at a leaf node and propagate back up the tree when there is not space. At each node we do a maximum of 2 comparisons for the existing keys, however this coefficient would be dropped in order notation.

The time complexity in order notation of my implementation is O(.

Chapter 8 Exercise 19:

public static void find\_both2(int n, int [] S){

int small;

int large;

int i;

if(S[0] < S[1]){ // one comparison

small = S[0];

large = S[1];

}else{

small = S[1];

large = S[0];

}

for(i = 2; i < n-1; i = i +2){ //when even this runs (n-2)/2 times. when odd its runs (n-3)/2 times. multiply by 3

if(S[i] < S[i+1]){ //one comparison

if(S[i] < small){ //two comparison

small = S[i];

}

if(S[i+1] > large){ //three comparison

large = S[i+1];

}

}else{

if(S[i+1] < small){

small = S[i+1];

}

if(S[i] > large){

large = S[i];

}

}

}

if(n%2 == 1){

if(S[n-1] < small){ //one comparison

small = S[n-1];

}else if(S[n-1] > large){ //two comparison

large = S[n-1];

}

}

System.out.println("Small: " + small + "\nLarge: " + large);

}

Show that its time complexity is given by

In my comments you can see me break down the comparisons present in different sections of the code. First at the start we have 1 comparison to establish the initial small and large. Then we have a for loop contains 3 comparisons on each iteration. This loop starts at index 2, which means it runs times when n is even. When n is odd it runs for iterations, excluding the first two elements and the last element. Finally, we have the last section which handles the last element when n is odd using 2 comparisons. Adding up these different sections we get the follow:

Which match up with the given time complexities