Philippe WS CSC 349

Lab 3

7.) **Algorithm**

//a is an array of sorted distinct integers.

//left is the lesser index of the array used to determine the mid point

//right is the greater index of the array used to determine the mid point

findEquivIndex(a, left, right){

mid = floor((left+right)/2);

if(left == right && a[mid] != mid) //Base case, where there is only one element left to compare.

return false;

else if(a[mid] == mid)

return true;

else if(a[mid] > mid)

return *findEquivIndex*(a, left, mid);

else

return *findEquivIndex*(a, mid+1, right);

}

**Recurrence Relation**:

The function cuts the list in half every time, then does a constant number of comparisons on a single element.

**Complexity by Back Substitution**

…

20.) Algorithm Complexity Derivation (by back substitution)

…

22.a) Analysis of brute force algorithm.

1. Problem size: **n**, the number of total elements amongst, **k**, the number of lists.

2. Basic operation: Comparison

3. Worst case analysis: every element must be compared; the final operation involves a single element on one of the two lists.

4.

|  |  |
| --- | --- |
| **kth list** | **Number of Operations** |
| 2 |  |
| 3 |  |
| … | … |
|  |  |

5.

22.b) Pseudo code and analysis of k-way merge.

**Algorithm**

mergeKLists(inputArray[][], left, right){

if(left+1 == right)

return *combine*(inputArray[left], inputArray[right]);

else if(left=right)

return inputArray[left];

else

array1 = *mergeKLists*(inputArray, left, (left+right)/2)

array2 = *mergeKLists*(inputArray, (left+right)/2 +1, right)

return *combine*(array1, array2);

}

combine(array1[], array2[]){

tempArray = []; j=0;

for i less than array1 length

if(array1[i] < array2[j])

tempArray[i+j] = array[i];

else

tempArray[i+j] = array[j];

j++;

return tempArray;

}

**Analysis**: By recursive tree

Level 0: Work Done:

Level 1: Work Done:

…

Level t: Work Done:

Thus, sum of work done

23.) Local min algorithm

**Algorithm**

a is a 1 dimensional array of integers

findLocalMin(a[], left, right){

mid = (left+right)/2;

if(left == right)

return left;

else if(a[mid-1] > a[mid] && a[mid] < a[mid+1])

return mid;

else if(a[mid-1] > a[mid] && a[mid] > a[mid+1])

lowerMid = (a[mid-1] < a[mid+1]) ? mid-1 : mid+1 //ternary operator

if(lowerMid == mid-1)

*findLocalMin*(a[], left, lowerMid);

else if(lowerMid == mid+1)

*findLocalMin*(a[], lowerMid, right);

else if(a[mid-1] < a[mid])

*findLocalMid*(a[], left, mid)

else

*findLocalMid*(a[], mid+1, right);

}

**Analysis**: By back substitution

…