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: $\mathrm{T}(n) = \mathrm{T}\left(\frac{n}{2}\right) + \mathrm{O}(1)$

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

Complexity by Back Substitution

$$\mathrm{T}(n) = \mathrm{T}\left(\frac{n}{2}\right) + \mathrm{O}(1), \qquad \mathrm{T}(1) = 1$$

$$\mathrm{T}(n) = \mathrm{T}\left(\frac{n}{4}\right) + 2 * \mathrm{O}(1)$$

$$\dots$$

$$\mathrm{T}(n) = \mathrm{T}\left(\frac{n}{2^k}\right) + k * \mathrm{O}(1)$$

$$\mathrm{T}(1) = 1, \qquad \frac{n}{2^k} = 1 \rightarrow n = 2^k \rightarrow k = \log(n)$$

$$T(n) = T(1) + \log(n) * O(1)$$

Complexity: $\theta(\log(n))$

20.) Algorithm Complexity Derivation (by back substitution)

$$T(n) = T(n-1) + 2^n, \qquad T(0) = 0$$

$$T(n) = (T(n-2) + 2^{n-1}) + 2^n$$

$$T(n) = T(n-2) + 2^n + 2^{n-1}$$

…

$$T(n) = T(n-k) + 2^n + 2^{n-1} + \cdots + 2^{n-(k-1)}$$

$$T(0) = 0, \qquad n - k = 0 \rightarrow n = k$$

$$T(n) = T(0) + 2^n + 2^{n-1} + \cdots + 2^1$$

$$T(n) = \sum_{i=1}^{n} 2^i = 2^{n+1} - 1 - 1$$

Complextity: $\theta(2^n)$

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.

k^{th} list	**Number of Operations**
2	$2 * \frac{n}{k}$
3	$3 * \frac{n}{k}$
…	…
k	$\frac{n}{k}\sum_{i=2}^{k} i$

5.

$$\frac{n}{k}\sum_{i=2}^{k} i = \frac{n}{k} * (\frac{i(i+1)}{2} - 3)$$

$$\frac{n}{k} * (\frac{k^2 + k}{2} - 3)$$

Complexity: $\theta(k * n)$

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: $T(2^s)$ Work Done: 2^s

Level 1: $T(2^{s-1})$ & $T(2^{s-1})$ Work Done: $2^{s-1} + 2^{s-1} = 2^s$

…

Level t: $T(2^{s-t})$ & ... & $T(2^{s-t})$ Work Done: $2^{s-t} + \cdots + 2^{s-t} = 2^s$

Thus, sum of work done

$$\sum_{i=0}^{t} 2^s \rightarrow 2^s \sum_{i=0}^{t} 1 \rightarrow 2^s(t+1)$$

$$k = 2^t, n = 2^s \rightarrow \log(k) = t, \log(n) = s$$

$$n(\log(k) + 1)$$

$$\text{Complexity: } \theta(\, n \log(k)\,)$$

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

$$\mathrm{T}(n) = \mathrm{T}\left(\frac{n}{2}\right) + \mathrm{O}(1), \qquad \mathrm{T}(1) = 1$$

$$\mathrm{T}(n) = (\mathrm{T}\left(\frac{n}{2^2}\right) + \mathrm{O}(1)) + \mathrm{O}(1)$$

$$\mathrm{T}(n) = \mathrm{T}\left(\frac{n}{2^2}\right) + 2 * \mathrm{O}(1)$$

$$\dots$$

$$\mathrm{T}(n) = \mathrm{T}\left(\frac{n}{2^k}\right) + \mathrm{k} * \mathrm{O}(1)$$

$$\frac{n}{2^k} = 1 \rightarrow n = 2^k \rightarrow \log(n) = k$$

$$\mathrm{T}(n) = \mathrm{T}(1) + \log\,(\mathrm{n}) * \mathrm{O}(1)$$

$$\text{Complexity: } \theta(\log(n))$$