Darren DeVera
4/26/20
CSC401-01

Assignment 3

Problem 11

```
NewBinarySearch(int[] A, int key, int low, int high)
A = array
key = target value
low = start index
high = end index
Need to split array
Use two more indexes
int x = low + ((high - low)/3) // top of lower array
int y = high - ((high - low)/3) // bottom of higher array
low can't be larger than high
If low is greater than high
   return -1, so program gives null
compare key to values in A[x] and A[y]
If key is equal to value in A[x]
    return x
If key is equal to value in A[y]
    return y
If key is less than value in A[x]
   return NewBinarySearch(A, key, low, x-1)
   // search lower array
If key is greater than value in A[y]
   return NewBinarySearch(A, key, y+1, high)
   //search higher array
Else key is none of the above
   return NewBinarySearch(A, key, x+1, y-1)
  //search middle array
```

$\frac{low + high}{2}$ used to split array in half

$\frac{high - low}{3}$ divides array into thirds

Binary Search Tree has a time complexity of $O(\log_2 n)$, because it searches two halves of an array. Searching a third array would make the time complexity $O(\log_3 n)$.

Problem 2)

FindLargest(int[] A, int low, int high)

A = array

low = start index

high = end index

Array needs to be split using $\frac{low+high}{2}$

Need two indexes to hold largest integers

int x = left side int z = $\frac{low+high}{2}$

int y = right side

Case where start index and end index are the same

If low is equal to high

    return A[low]

    // This value would already be the largest

Use recursion to check both sides of array

X = FindLargest(A, low, Z) // search left side

Y = FindLargest(A, Z+1, high) // search right side

// Z+1 so Z is not checked again

Compare values of x and y // return the largest

If x is greater than y of the two.

    return x

If y is greater than x

    return y

Similar to a Binary Search Tree, which searches two subtrees, with a time complexity of O(n). FindLargest searches two sides of an array, which should make its time complexity O(n) as well.

Problem 31

```
FindMajority(int[] A, int low, int high)
A=array
low=start index
high=end index
Need to split array using (low+high)/2
Need two indexes
X=left side   let z=(low+high)/2
Y=right side
Need case if start index and end index are the same
If low is equal to high
    return A[low]
    //array length is 1 size of 1
Set X and Y indexes and search using recursion
X=FindMajority(A,low,z)
Y=FindMajority(A,z+1,high)
Case if X and Y are the same
If X is equal to Y
    return X // Majority is already X and Y
Need to count values to find majority
int leftCount=0;
int rightCount=0
Case if leftCount and rightCount are the same
If leftCount is equal to rightCount
    return leftCount // both would be majority
For loop // for counting values // while loop could also work too
int i=0; i>=z; i++
  If A[i] is equal to X
     leftCount=leftCount+1
For loop
int i=0; i<z+1; i++
  If A[i] is equal to Y
     rightCount=rightCount+1
```

Majority needs to be greater than $\frac{\text{array size}}{2} = W$

```
Compare left side and right side
If X is greater than W
    return X
If Y is greater than W
    return Y
```

Problem 3 | cont'

Search both sides of the array, similar to Binary Search, makes time complexity $O(n)$. Comparing values and merging them is similar to a Merge Sort, which has a time complexity of $O(\log n)$. Since the algorithm FindMajority searches both sides of the array. and compares and counts, or merges, then the time complexity would be $O(n \log n)$