C-3.1 Suppose you are given a sorted array, A, of n distinct integers in the range from 1 to n+1, so there is exactly one integer in this range missing from A. Describe an O(log n)-time algorithm for finding the integer in this range that is not in A.

```
def ModifiedBinarySearch(arr, l, r, x):
    while l <= r:
        mid = (l + r)/2;

        if arr[mid] == mid + 1:
            return mid
        elif arr[mid] > mid:
            r = mid - 1
        else:
            l = mid + 1
```

C-3.2 Let S and T be two ordered arrays, each with n items. Describe an O(log n)-time algorithm for finding the kth smallest key in the union of the keys from S and T (assuming no duplicates).

```
def kthlargest(arr1, arr2, k):

     if len(arr1) == 0:

          return arr2[k]

      elif len(arr2) == 0:

          return arr1[k]

     m1 = len(arr1)/2

     m2 = len(arr2)/2

     if m1 + m2 < k:

          if arr1[m1] > arr2[m2]:

               return kthlargest(arr1, arr2[m2+1:], k-m2-1)

          else:

               return kthlargest(arr1[m1+1:], arr2, k-m1-1)

     else:

           if arr1[m1]>arr2[m2]:

               return kthlargest(arr1[:m1], arr2, k)
```

```
        else:
            return kthlargest(arr1, arr2[:m2], k)
```

C-3.3 Describe how to perform the operation findAllElements(k), which returns every element with a key equal to k (allowing for duplicates) in an ordered set of n keyvalue pairs stored in an ordered array, and show that it runs in time O(log n+s), where s is the number of elements returned.

```
def findAllElements(arr, l, r, k):
    while l <= r:
        mid = (l + r)/2;

        if arr[mid].key <= k:
            r = mid - 1
        else:
            l = mid + 1
    allElements = []
    while arr[mid].key == k:
        allElements.append(arr[mid])
        mid += 1
    return allElements
```

C-3.4 Describe how to perform the operation findAllElements(k), as defined in the previous exercise, in an ordered set of key-value pairs implemented with a binary search tree T, and show that it runs in time O(h + s), where h is the height of T and s is the number of items returned.

```
def findAllElements(k, v, c):
    if v is an external node then
        return c
    if k = key(v) then
        c.addLast(v)
        return findAllElements(k,T.right(v), c)
    else if k < key(v) then
        return findAllElements(k,T.left(v), c)
    else // {we know k > key(v)}
        return findAllElements(k,T.right(v), c)
```

C-3.7 Let S be an ordered set of n items stored in a binary search tree, T, of height h. Show how to perform the following method for S in O(h) time:
countAllInRange(k1, k2): Compute and return the number of items in S with key k such that k1 ≤ k ≤ k2.

```
def getCount(root, low, high):
    if root.data == high and root.data == low:
      return 1

    # If current node is in range, then include it in count and
    # recurse for left and right children of it

    if root.data <= high and root.data >= low:
        return (1 + getCount(root.left, low, high) +
                    getCount(root.right, low, high))

    # If current node is smaller than low,
    # then recurse for right child
    elif root.data < low:
        return getCount(root.right, low, high)

    # Else recur for left child
    else:
        return getCount(root.left, low, high)
```

C 3.12 Without using calculus (as in the previous exercise), show that, if n is a power of 2 greater than 1, then, for Hn, the nth harmonic number,
Hn ≤ 1 + Hn/2.
Use this fact to conclude that Hn ≤ 1 + log n, for any n ≥ 1.

$H_n = 1 + ½ + ⅓ + ¼ + … + 1 /(n/2) + 1/(n/2 + 1) + … + 1/n$

$H_{n/2} = 1 + ½ + ⅓ + ¼ + … + 1 / n/2$

$H_n - H_{n/2} = 1/(n/2 + 1) + 1/(n/2 + 2) + … + 1/n$

$H_n - H_{n/2} <= 1/(n/2) + 1/(n/2) + … + 1/(n/2)$

$H_n - H_{n/2} <= (n/2) * 1/(n/2)$ // 1/(n/2) is being added (n/2) times

$H_n - H_{n/2} <= 1$
$H_n <= 1 + H_{n/2}$

$H_n \leq 1 + \log n$

$H_n <= 1 + H_{n/2}$

$H_{n/2} <= 1 + H_{n/4} = 1 + H_{n/(2^2)}$

$H_n <= 1 + 1 + H_{n/4} = 2 + H_{n/(2^2)}$

$H_n <= 1 + 1 + 1 + H_{n/8} = 3 + H_{n/(2^3)}$

...

...

...

$H_n <= 1 + 1 + 1 \ldots 1 + H_{n/(2^k)} = k(1) + H_{n/(2^k)}$

when 2^k = n ⇒ k = log(n)

$H_n <= \log n + H_1$

$H_n <= \log n + 1$