CSCI 3104 Assignment 4

10:00 - 10:50 Wanshan

Jackson Chen

22 February 2016

1. (a) Have two index trackers (i and j) for each array. If the ith element in `a` is larger than the jth element in `b`, then increment j. Otherwise increment i. If the two elements are equal, then return `true`. Whenever i or j become larger than the size of their respective array, then return `false`.

$$T(n) = O(n)$$

```
i = 0
j = 0
while i < len(a) && j < len(b):
  if a[i] > b[j]:
    j++
  else if a[i] < b[j]:
    i++
  else
    return true
return false
```

   (b) Loop through the array `b` and do a binary search for those elements in array `a`. The time complexity is $O(n \log_2 n)$ since each binary search requires a time complexity of $O(log_2 n)$.

```
for (i in range(len(b))):
  if binarySearch(b[i], a):
    return true
return false
```

   (c) Loop through array `a` and check each element with each element in array `b`. The time complexity is $O(n^2)$.

```
for (i in range(len(a))):
  for (j in range(len(b))):
    if (i == j):
      return true
return false
```

(d) Loop through the array `b` once and keep track of a counter of the number of elements in `b` that are smaller than a. Since the algorithm only loops through the array once, its time complexity is $\Theta(n)$.

```
counter = 0
for (i in range(len(b))):
  if (b[i] < minimum(a)):
    counter += 1
return counter
```

(e) Take the smaller heap of the two (whichever number `m` or `n` is smaller) and insert elements of that heap into the larger heap. The time complexity is $\Theta(m \log_2 n)$ if m is smaller than n.

```
for (i in range(len(a))):
  heapinsert(a[i], b)
```

(f) Find the largest element in a, and the smallest element in b. Check if the largest element is smaller than the smallest element, if so print YES, else print NO.

```
smallest = findSmallest(b)
largest = findLargest(a)
if (largest < smallest):
  print("YES")
else:
  print("NO")
```

2. (a) Min-heap: Add the new quote as a leaf to the tree. Then bubble the element up until it satisfies the requirements of a min-heap.

$$T(n) = O(\log_2 n)$$

(b) BST: Start at the root, and check if the new quote is larger or smaller than that node. Then navigate down the tree in this fashion until there is an open leaf that the quote can be added into.

$$T(n) = O(\log_2 n)$$

(c) Sorted Array: Loop through the array, once the algorithm reaches an element in the array that is larger than the added quote then insert the quote into the spot before that larger element. This may require instantiating a new array with a larger size and then copying all of the elements over.

$$T(n) = O(n)$$