cs3821 Assignment 2

Jennifer z5207930

April 2020

1 Question 1

1.1 Table

Array	Rotated	Rotation Value m
$[3, 5, 1, 2]$	Yes	2
$[5, 3, 1, 4, 6]$	No	-
$[5, 6, 1, 2, 3, 4]$	Yes	2
$[5, 8, 10, 14, 4]$	Yes	4
$[5, 6, 7, 8]$	Yes	0
$[1, 3, 2, 4]$	No	-
$[1, 10, 6, 4]$	No	-
$[2]$	Yes	0

1.2 $O(logn)$ Algorithm

```
# Find the peak in a array 'a' in the given range of head and tail
def find_peak(a, head, tail):

    if (tail - head) == 2:
        return max(a[head], a[tail])

    middle = ceiling((tail - head) / 2)

    A = a[head]
    B = a[middle]
    C = a[tail]
    # If B is the peak
    if A < B and B > C: return middle

    # If peak is on the left
    if A > B and B > C: return find_peak(a, head, middle)

    # If peak is on the right
    if A < B and B < C: return find_peak(a, middle, tail)

def hasX(a, x):

    if len(a) == 2:
        if (a[0] == x or a[1] == x): return True
        return False

    peak = find_peak(a, 0, len(a))

    # Search left side of peak, an increasing array
    search_left = binary_search(a[:peak], x)

    # Search right side of peak, a decreasing array
    search_right = decreasing_binary_search(a[peak:], x)

    # Returns true if either are true
    return max(search_left, search_right)
```

1.3 Worst-Case Complexity

The algorithm, once the peak is found, splits the array in two and performs two separate binary searches to see if x exists. Here, worst and best case, we would be doing $2 * logn$ searches.

The peak is found using a binary recursive call, either the peak is on the left or on the right and each time we narrow down the possibilities by half. Hence, the worst case scenario is that there is no peak and we would have done $logn$ searches to determine that.

Altogether, the algorithm does $3 * logn$ searches in the worst case. Therefore, it has an $O(logn)$ time complexity.

1.4 Proof of correctness

The algorithm ensures that we can determine whether x exists in a since we are performing two binary searches - one from the start to index i and the second from i to the end, where i is the index of where peak is.

The algorithm is guaranteed to terminate as we are ensured that our list contains a peak and binary search is also ensured to return true or false.

Hence this algorithm terminates and can successfully find whether x exists within the array a.