HW2 DC1

Chengqi Huang chengqihuang@gatech.edu

Problem 1

Given:

An infinite array A[.] in which the first n entries contain different integers in sorted order and the rest are filled with ∞

Input:

Integer x

Find:

A position in the array containing x, if such position exists

Solution:

Since it is sorted array, we can use a binary search. The issue is we need to find the bounds and apply the algorithm.

Steps:

1. Let low be the 1st element of the array, high be the 2nd element of array, compare x to high
2. If x is greater than A[high], copy index of high to low, and double the high index
3. If x if smaller than A[high], use binary search between A[low] and A[high]
4. Binary search: compare the mid element of A[low] and A[high] (A[mid]) with x
5. If A[mid] > x, ignore the upper half of the array. Search among A[low] to A[mid]
6. If A[mid] < x, ignore the lower half of the array. Only search A[mid] to A[high]

Runtime:

The size of the binary search problem is an array of size n, because worst case: index high < 2n, index low < n.

T(n) = T(n/2) +O(1), using the master theorem: a = 1, b =2, f(n) = 1. n^logb(a) = n^0 = 1 = f(n).

T(n) = O(n^logb(a) * log n) = log n

Why correct:

Since the array is sorted, The idea of the algorithm is to find a interval such that A[low] < x < A[high]. Then apply binary search between A[Low] and A[high]. Then if x exists it has to be inside this interval.

Problem 2

Given:

A sorted array of n distinct integers A = {a(1), a(2), … , a(n)}

Find:

If exists index i such that a(i) = i

Solution:

To find if exists index I such that a(i) = I, using an algorithm that is similar to binary search:

Steps:

1. Compare the mid element of the whole array: a(n/2) to n/2
2. If a(n/2) == n/2, then we got the solution, return yes
3. If a(n/2) < n/2, we narrow the search to the upper half of the array: {a(n/2 + 1), … ,a(n)}.
4. If a(n/2) > n/2, only search to the lower half of the array: {a(1), …, a(n/2-1)}
5. Repeatedly check using the recursive call, until we find the value or the interval is empty.

Runtime:

Since the recurrence is T(n) = T(n/2) + O(1), using the master theorem: a = 1, b =2, f(n) = 1. n^logb(a) = n^0 = 1 = f(n). T(n) = O(n^logb(a) * log n) = log n

Why correct:

Since a(1), a(2), … , a(n) are sorted distinct integers, so for i = 1 ->n, j = i -> n, a(i) < a(j). if we compare i with a(i), if i < a(i), then we can get rid of the upper half, as: i + n < a(i) + n < a(i + n). for i > a(i), we have similar statement so we can get rid of the lower half. By doing this we narrow down the search interval.