CS515 - Algorithms & Data Structures

Practice Assignment 1

Vy Bui - 934370552

Instructor: Professor Glencora Borradaile

The School of Electrical Engineering and Computer Science
Oregon State University

Problem 1
A fixed point of an array A[1..n] is an index i such that A[i] = i. Given a sorted array of distinct integers A[1..n] as input, give a divide-and-conquer algorithm to determine if A has a fixed point that runs in time $O(logn)$.

Description Let $FP(i, j)$ be the function that checks if there exists a fixed point in $A[i, j]$. Imagine $A[k]$ as a discrete function of index k with $i \geq k \geq j$, then the fixed point of A is the intersection between $A[k]$ and the identity function $I[k]$. Because A is a distinct increasing array of integer, the rate of change of A at each index k must be greater than or equal to $I[k]$, which is equal 1 for all k. Therefore, $A[k]$ can only intersect $I[k]$ if $A[i] \leq i$ and $A[j] \geq j$.

Recurrence Let $m = \frac{i+j}{2}$, the recursive formulation can be described as follows

$$FP(i,j) = \begin{cases} False & i > j \\ \begin{cases} True & A[m] = m \\ FP(i,m) & A[m] > m \\ FP(m+1,j) & A[m] < m \end{cases} & otherwise \end{cases}$$

Pseudocode

Algorithm 1 $FP(i, j)$

```
if i > j then
    return False
end if
m ← (i + j)/2
if A[m] == m then
    return True
else if A[m] > m then
    return FP(i, m)
else
    return FP(m + 1, j)
end if
```

Proof of Correctness

Base Case: first, if A only has one element, if there exists a fixed point, then it has to be that element. Second, the left index i greater than the right index j indicates an empty array, which implies that no fixed point exists.

Inductive Hypothesis: $FP(i, m)$ and $FP(m + 1, j)$ correctly determine if $[i, m]$ and $A[m + 1, j]$ contain a fixed point, respectively.

Inductive Step: It is trivial that if at least one of $FP(i, m)$ and $FP(m+1, j)$ is true, then $FP(i, j)$ is true because they use the same indices and values of A.

Runing Time Analysis

On each recursive call, the algorithm splits the problem into two roughly equal halves and only solves one of them. It takes constant time to check if the midpoint is the fixed point. Therefore, the total running time of this algorithm is $T(n) = T(\frac{n}{2}) + O(1) = O(logn)$.

Problem 2
For a sequence of n numbers $a_1, .., a_n$, a *significant inversion* is a pair (a_i, a_j) such that $i < j$ and $a_i > 2a_j$. Assuming each of the numbers a_i is distinct, give an $O(nlogn)$ time algorithm to count the number of significant inversions in a sequence. (Hint: modify merge sort.)

Description

Recurrence

$$FP(i,j) = \begin{cases} \begin{cases} True & A[i] = i \\ False & otherwise \end{cases} & i = j \\ \begin{cases} FP(i, \frac{i+j}{2}) \vee FP(\frac{i+j}{2} + 1, j) & otherwise \end{cases} \end{cases}$$

Pseudocode

Algorithm 2 $FP(i, j)$

```
if i == j then
    if A[i] == i then
      return True
    else
      return False
    end if
end if
m ← (i+j)/2
      return FP(i, m) ∨ FP(m + 1, j)
```

Proof of Correctness

Base Case Inductive Hypothesis Inductive Step

Runing Time Analysis

Problem 3
You are given two sorted arrays of size m and n. Give an $O(logm+logn)$ time algorithm for computing the k-th smallest element in the union of the two arrays.

Description Let $FP(i, j)$ be the function that checks if there exists a fixed point in A[i,j]. Observe that if there exists a fixed point k A[i,j], then k must be either in the left half or the right half of A[i,j]. Instead of checking the entire array, we can check its two halves and then combine the results.

Recurrence

$$FP(i,j) = \begin{cases} \begin{cases} True & A[i] = i \\ False & otherwise \end{cases} & i = j \\ \left\{ FP(i, \frac{i+j}{2}) \vee FP(\frac{i+j}{2} + 1, j) \right. & otherwise \end{cases}$$

Pseudocode

Algorithm 3 $FP(i, j)$

```
if i == j then
    if A[i] == i then
      return True
    else
      return False
    end if
end if
m ← (i+j)/2
      return FP(i, m) ∨ FP(m + 1, j)
```

Proof of Correctness

Base Case Inductive Hypothesis Inductive Step

Runing Time Analysis

Problem 4
You are given an $n \times n$ matrix A[1..n, 1..n] where all elements are distinct. We say that an element A[x] is a *local minimum* if it is less than its (at most) four neighbors, i.e. its up, down, left and right neighbors. Give an $O(n)$ time algorithm to find a local minimum of A.

Description Let $FP(i, j)$ be the function that checks if there exists a fixed point in A[i,j]. Observe that if there exists a fixed point k A[i,j], then k must be either in the left half or the right half of A[i,j]. Instead of checking the entire array, we can check its two halves and then combine the results.

Recurrence

$$FP(i,j) = \begin{cases} \begin{cases} True & A[i] = i \\ False & otherwise \end{cases} & i = j \\ \left\{ FP(i, \frac{i+j}{2}) \vee FP(\frac{i+j}{2} + 1, j) \right. & otherwise \end{cases}$$

Pseudocode

Algorithm 4 $FP(i, j)$

```
if i == j then
    if A[i] == i then
      return True
    else
      return False
    end if
end if
m ← (i+j)/2
      return FP(i, m) ∨ FP(m + 1, j)
```

Proof of Correctness

Base Case Inductive Hypothesis Inductive Step

Runing Time Analysis