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]. 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

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

Pseudocode

Algorithm 1 $FP(i, j)$

```
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: when A has only one element, it is obvious that

Inductive Hypothesis: assume that we know the results of $FP(i, m)$ and $FP(m + 1, j)$, with $m = \frac{i+j}{2}$.

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

The algorithm splits the problem into two roughly equal halves and merge the results in constant time. Therefore, the running time of this algorithm is $T(n) = 2T(\frac{n}{2}) + O(1) = O(n)$

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 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

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

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

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

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

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

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