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 Let $SI(A)$ denotes the function that counts the number of significant inversions in A, and let $BS(i, j, t)$ denote the function that can find the minimum number that is greater than or equal to target t. Assume that A is sorted, for each a_i, we can use BS to search for $2a_i + 1$, and then count the numbers of the following numbers in A. The final result is the sum of all counts.

Recurrence

Let $m = \frac{i+j}{2}$. The reccurence can be described as follows

$$BS(i,j,t) = \begin{cases} i+1 & i > j \\ \begin{cases} i & A[i] \geq t \\ i+1 & otherwise \end{cases} & i = j \\ \begin{cases} m & A[m] = t \\ \begin{cases} min(BS(i, m-1, t), m) & t < A[m] \\ BS(m+1, j, t) & t > A[m] \end{cases} & otherwise \end{cases} & otherwise \end{cases}$$

Pseudocode

Algorithm 2 $SI(A[1, n])$

```
SortA
count ← 0
for i : [1, n − 1] do
    k = BS(i + 1, n, 2A[i] + 1)
    if k ≠ −1 then
        count ← count + max(0, n − k + 1)
    end if
end for
    return count
```

Proof of Correctness

First, we assume that $BS(i, j, t)$ correctly determines the minimum number that is greater than or equal target t and we prove that $SI(A[1, n])$ can count the number of significant inversions. For each a_i, BS is use to find the minimum number that is greater or equal to $2a_i + 1$ in subarray $A[i + 1, n]$. Because A is sorted, we do not need to search in the subarray before a_i. If BS cannot find the minimum number that is greater or equal to $2a_i + 1$ and return -1, then there are no significant inversions that contain a_i. In case that BS can find such a number a_k, then all of the numbers following and including a_k in A can make a significant inversion with a_i. Applying this for every number in A and summing up the counts will produce the final result to the problem.

Algorithm 3 $BS(i, j, t)$

```
if i > j then return −1
else if i = j then
    if A[i] > t then return i
    elsereturn i + 1
    end if
else
    m ← (i+j)/2
    if t = A[m] then
     return m
    else if t < A[m] then
     return BS(i, m − 1, t)
    else
     return BS(m + 1, j, t)
    end if
end if
```

Second, we prove that $BS(i, j, t)$ correctly determines the minimum number that is greater than or equal target t in a sorted array A. Similar to binary search, the algorithm also searchs for target t in $A[i, j]$. If it can find t, then t is also the number of interest. It it cannot find t, then it returns the minimum number that is greater than t.

Base Case: when $i = j$, there is only one element to consider. If it is equal to the target then it is the number of interest. If it is not equal to the target, *Inductive Hypothesis Inductive Step*

Runing Time Analysis

We can use merge sort or quick sort to sort A in $O(nlogn)$. BS takes $O(logn)$ to search. Because it is called $O(n)$ times, the entire algorithm will take $O(nlogn)$.

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

Recurrence

Pseudocode

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

Pseudocode

Proof of Correctness

Base Case Inductive Hypothesis Inductive Step

Runing Time Analysis