CSCI 570 - Fall 2015 - HW 6 solutions

October 5, 2015

1.
```
Function(A,n){
          i=floor(n/2);
          if A[i]==i
                    return TRUE;
          if  (n==1)&&(A[i]!=i)
                    return FALSE;
          if A[i]<i
                    return  Function(A[i+1:n],  n-i);
          if A[i>i]
                    return  Function(A[1:i],  i);
}
```

   Proof: The algorithm is based on Divide and Conquer. Every time we break the array into two halves. If the middle element i satisfy A[i]¡i, we can see that for all $j < i, A[j] < j$. This is because A is a sorted array of DISTINCT integers. To see this we note that $A[j+1] - A[j] >= 1$ for all j. Thus in the next round of search we only need to focus on $A[i+1:n]$. Likewise, if $A[i] > i$ we only need to search $A[1:i]$ in the next round. For complexity $T(n) = T(n/2) + O(1)$. Thus $T(n) = O(\log n)$.

2. $T(n) = 4T(\frac{n}{3})+\lg n + cn$ by master theorem, $T(n)$ is $O(n^{log_3 4})$.

3.
   - Divide: divide array up in 2 equal pieces
   - Conquer: solve the two subproblems recursively and return the average
   - Combine: compute the average for the original problem by: Average = (number of items in sub problem 1 * average for subproblem 1 + number of items in sub problem 2 * average for subproblem 2)/ (size of the original problem)

   The complexity of divide and combine is $O(1)$. $T(n) = 2T(n/2) + O(1)$. Master theorem gives you a complexity of $O(n)$ for the solution.

4. The idea is to first find out p and then break A into two separated sorted arrays, then use binary search on these two arrays to check if x is belong to A. Let FindPeak() be the function of finding the peak p in A . Then FindPeak($A[1:n]$) works as follows: Look at $A[n/2]$, there are 3 cases: (1) If $A[n/2-1] < A[n/2] < A[n/2+1]$, then the peak must come strictly after $n/2$. Run FindPeak($A[n/2:n]$). (2) If $A[n/2-1] > A[n/2] > A[n/2+1]$, then the peak must come strictly before $n/2$. Run FindPeak($A[1:n/2]$). (3) If $A[n/2-1] < A[n/2] > A[n/2+1]$, then the peak is $A[n/2]$, return $n/2$. Now we know the peak index(p value). Then we can run binary search on $A[1:p]$ and $A[p+1:n]$ to see if x belong to them because they are both sorted. In the procedure of finding p, we halve the size of the array in each recurrence. The running time $T(n)$ satisfies $T(n) = T(n/2) + O(1)$. Thus $T(n) = O(\log\ n)$. Also both binary search has running time at most $O(\log\ n)$, so total running time is $O(\log\ n)$.

   Pitfalls: Note that trying to get x in one shot (in one divide and conquer recursion) usually does not work. Binary search certainly cannot work because the array is not sorted. Modified binary search can hardly work either. The problem is that in each round you need to abandon half the array. However, this decision is hard to made. For example, the array is [1 2 3 4 5 -1], $x = -1$. When divide we have $mid = 3$. We find $A[mid-1] < A[mid] < A[mid+1]$. A common but wrong practice here is to continue search only in the $A[1:mid]$. We can see clearly $x = -1$ is in the second half of the array. On the other hand you cannot throw away the first half of the array either. The counter example is $[12345-1]$ where $x = 1$. One other mistake is to search p in a linear fashion. This take $O(n)$ in the worst case.