Homework Assignment # 1

Worth: 11% **Due:** Wednesday Januray 27 (at tutorial)

For each question, please write up detailed answers carefully. Make sure that you use notation and terminology correctly, and that you explain and justify what you are doing. Marks may be deducted for incorrect/ambiguous use of notation and terminology, and for making incorrect, unjustified or vague claims in your solutions.

1. [10 marks]
Consider the following two algorithms that find the two largest elements in an array $A[1..n]$, where $n >= 2$.

```
Algorithm 1
if A[1] >= A[2]
then L ← A[1]
     S ← A[2]
else L ← A[2]
     S ← A[1]
for i ← 3 to n
     if A[i] > L
     then S ← L
          L ← A[i]
     else if A[i] > S
          then S ← A[i]
end for
return(L,S)
```

```
Algorithm 2
if A[1] >= A[2]
then L ← A[1]
     S ← A[2]
else L ← A[2]
     S ← A[1]
for i ← 3 to n
     if A[i] > S then
          if A[i] > L
          then S ← L
               L ← A[i]
          else S ← A[i]
end for
return(L,S)
```

(a) Formally define an appropriate sample space and probability distribution with which to analyze the average case complexity of these algorithms.

(b) Derive the expected number of element comparisons performed by Algorithm 1.

(c) Derive the expected number of element comparisons performed by Algorithm 2.

2. [10 marks]
Recall that the deterministic QuickSort algorithm we described in class chooses the first element of the input as pivot element. We have seen that this algorithm has worst-case running time of $\Omega(n^2)$ and an input showing this is one in which the input sequence is already sorted (increasing or decreasing). In this question we examine what happens when the input sequence is *almost* sorted.

We say that a sequence S is k-almost increasing if there are k elements, the exclusion of which leaves the sequence ordered in an increasing fashion. For example, the sequence 1,2,3,8,6,4,5,7,9,10 is 2-almost increasing as it is increasing once 8 and 6 are excluded. 4,3,2,1 is 3-almost increasing but not 2-almost increasing (check).

(a) Show that the best case running time of QuickSort on 1-almost increasing sequences is still $\Omega(n^2)$. In other words, there is a constant $c > 0$ so that for every sequence of size n that is 1-almost increasing, the number of comparisons that QuickSort performs on the sequence is at least cn^2.

(b) (**bonus**) Show for a general k that the best case running time of QuickSort on k-almost increasing sequences is still $\Omega(n^2)$. Hint: Let $T(n,k)$ be the best case running time for k-almost increasing sequences. Show inductively that $T(n,k) \geqslant \frac{n^2}{2 \cdot 2^k}$.

3. [10 marks]
Consider a binary tree T. Let $|T|$ be the number of nodes in T. Let x be a node in T, let L_x be the left subtree of x adn let R_x be the right subtree of x. We say that x has the "approximately balanced property", $APB(x)$, if $|R_x| <= 2|L_x|$ and $|L_x| <= 2|R_x|$.

(a) What is the maximum height of a binary tree T on n nodes where $ABP(root)$ holds?

(b) We calll T an ABP-tree if $ABP(x)$ holds for every node x in T. Prove that if T is an ABP-tree, then the height of T is $O(\log n)$. More precisely, show that height(T) $\leqslant \log_2 n / \log_2 \frac{3}{2}$.

4. [10 marks]
Consider a binary search tree with *seven distinct* elements; the tree is perfectly balanced (that is, of height 2), and rooted at *root*. In this question, we will consider two slightly different methods for searching the tree for a key k. For each method, we will be interested in the *expected* time to search for k, when k is chosen at random from amongst the keys in the tree.

(a) Consider the following search method SEARCH1($root, k$):

```
SEARCH1($r$,$k$) /*Return a pointer to a node with key $k$ in subtree rooted at at $r$.*/
    if $k$ = key($r$) then
        return $r$
    elseif $k >$ key($r$) then
        return SEARCH1(rightchild($r$),$k$)
    else
        return SEARCH1(leftchild($r$),$k$)
    end if
end SEARCH1
```

Counting each "=" or ">" test as a comparison, what is the *expected* number of comparisons to do a search, where the expectation is over the random choice of k from amongst the keys in the tree, with all of them being equally likely? Briefly explain your reasoning and show your work.

(b) Next consider the search method SEARCH2($root, k$):

```
SEARCH2($r$,$k$)
/*Return a pointer to a node with key $k$ in subtree rooted at at $r$.*/
  if $k$ > key($r$) then
    return SEARCH2(rightchild($r$),$k$)
  elseif $k$ = key($r$) then
    return $r$
  else
    return SEARCH2(leftchild($r$),$k$)
  end if
end SEARCH2
```

Again, suppose that each time we do a search, the element to be sought is chosen at random from amongst the seven elements in the tree. Compute the *expected* number of comparisons using SEARCH2.

(c) What if, instead of a perfectly balanced tree of height 2, we started with a perfectly balanced tree of much larger height, say 10, and considered the same two experiments above. Which of the two procedures, SEARCH1 or SEARCH2, would yield the smaller expected number of comparisons? Justify your answer.

Hint: Both algoithms have two possible comparisons (not counting the ones from the recursive calls). Consier the events that will lead to one or two comparisons in each of them and and their probabilities. Consider using recurrance relation for the average number of comparisions in the above algorithms.