Note. *MT* = The Master Theorem

Problem 1.

(a) PG PROBLEM 341.

In the worst case, the search will continue in the $\frac{2}{3}$'s interval. So the recurrence is $T(n) \le T(\frac{2}{3}n)+1$. So $a=1$, $b=3/2$, $d=0$, and thus, $0=d=\log_{3/2}1=0$. Hence, by **MT**, $T(n)\in O(n^d \log n)=O(\log n)$.

(b) PG PROBLEM 343. Since A is sorted and consists of distinct integers, it follows that $A[i+1]-A[i]\ge 1$ for $1\le i<n$. Hence we have that $A[1]-1\le A[2]-2\le A[3]-3\le\cdots\le A[n]-n$. So the function $f(i)=A[i]-i$ is nondecreasing and finding a zero of this function is equivalent to finding an i with $i=A[i]$. Hence, a modified binary search suffices. That is, `search(A,1,n)` will such an i if it exists.

```
function search(A,low,hi)
  if low=hi then return low
  m ← low + ⌊(hi−low)/2⌋
  if (A[m]-m)< 0
    then return search(A,m+1,hi)
    else return search(A,low,m)
```

The recurrence for this is $T(n)=T(\frac{n}{2})+O(1)$.
So by **MT**, $T(n)=O(\log n)$.

Problem 2.

BACKGROUND. The procedure merge takes $\Theta(m+n)$ time, where m and n are the lengths of the respective lists. So let c be a constant such merge(l_1,l_2) takes $\le c\cdot(\mathsf{len}(l_1)+\mathsf{len}(l_2)+1)$ time.

(a) DPV Problem 2.19(a) The suggested algorithm is:

```
ans ← lst[1]
for i ← 2 to k do ans ← merge(ans,lst[i])
return ans
```

There are $(k-1)$-many merges. In the i-th merge $(i=2,\ldots,k)$, the size of ans is $(i-1)\cdot n$, the size of $lst[i]$ is n, and so the cost of this merge is $\le c((i-1)n+n+1)=c(i\cdot n+1)$. Thus the total cost of these merges is $\le c\cdot(n\cdot(\frac{1}{2}\cdot k\cdot(k+1)-1)+1)\le c\cdot(n\cdot(2+3+\cdots+k)+1)=c\cdot(n\cdot\frac{1}{2}\cdot k\cdot(k+1)-1)+1)$ which is $O(k^2\cdot n)$.

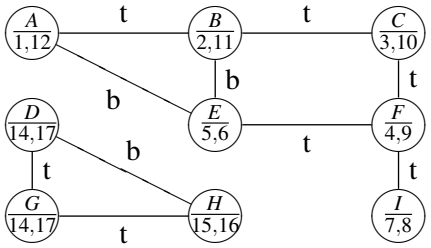
(b) DPV Problem 2.19(b) Here is a divide and conquer algorithm that is similar to mergesort:

```
function multiMerge(A[i..j])
  if i = j then return A[i]
  else if i+1 = j then return merge(A[i],A[j])
  else m ← i + ⌊(j−i)/2⌋
    return merge(multiMerge(A[i..m]),
                 multiMerge(A[(m+1)...j]))
```

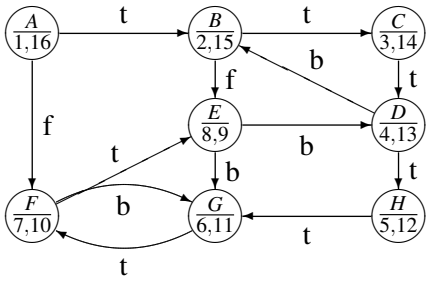
Problem 3.

Edge labels: t = tree, f = forward, b = back, c = cross.

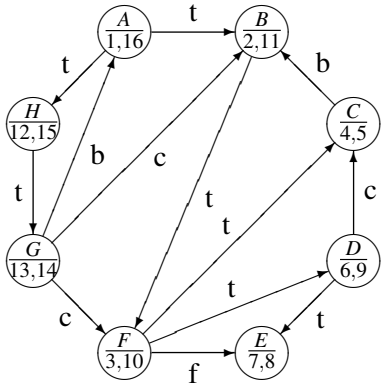
(a) DPV Problem 3.1.



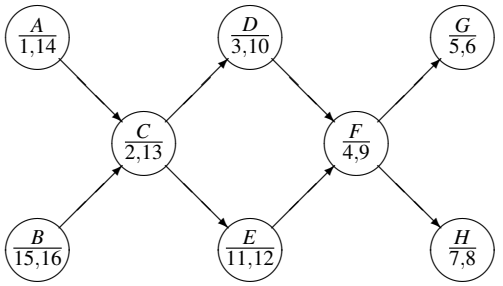
(b) DPV Problem 3.2(a)



(c) DPV Problem 3.2(b)



Problem 4. DPV Problem 3.3



(a) See above.

(b) A and B are sources and G and H are sinks

(c) $B<A<C<E<D<F<H<G$

(d) 8 = (2 choices for ordering A and B) × (2 choices for ordering D and E) × (2 choices for ordering G and H)

Problem 5. DPV Problem 3.11

Let u and v be the endpoints of e. Here is the algorithm:

1. Remove the edge e from the graph.
2. In the altered graph, do an explore from u.
3. If v was marked as visited,
   then return True else return `False`.

Since this is just DFS with a little constant time bookkeeping, the time is $O(|V|+|E|)$.