(i) DPV Problem 2.5.

(a) $T(n) = 2T(n/3) + 1$. So $a = 2$, $b = 3$, and $d = 0$. Thus $0 = d < \log_b a = \log_3 2$ and by the Master Theorem, $T(n) \in O(n^{\log_3 2})$.

(b) $T(n) = 5T(n/4) + n$. So $a = 5$, $b = 4$, and $d = 1$. Thus $1 = d < \log_b a = \log_4 5 \approx 1.161$ and by the Master Theorem, $T(n) \in O(n^{\log_4 5})$.

(c) $T(n) = 7T(n/7) + n$. So $a = 7$, $b = 7$, and $d = 1$. Thus $1 = d = \log_7 7 = 1$ and by the Master Theorem, $T(n) \in O(n \log n)$.

(d) $T(n) = 9T(n/3) + n^2$. So $a = 9$, $b = 3$, and $d = 2$. Thus $2 = d = \log_3 9 = 2$ and by the Master Theorem, $T(n) \in O(n^2 \log_n)$.

(e) $T(n) = 8T(n/2) + n^3$. So $a = 8$, $b = 3$, and $d = 2$. Thus $2 = d > \log_b a \approx 2.893$ and by the Master Theorem, $T(n) \in O(n^2)$.

(g) $T(n) = n^2 - n - 1 + 2$. Suppose $T(0) = c$. Then $T(1) = T(0) + 2 = c + 2$, $T(2) = T(1) + 2 = (c + 2) + 2 = c + 4$, $T(3) = T(2) + 2 = c + 6$. . . . So we guess: $T(n) = c + 2 \cdot n$.

Proof by induction: BASE CASE: $T(0) = c = c + 2 \cdot 0$.

INDUCTION CASE: Suppose $T(n) = c + 2 \cdot n$. Then $T(n + 1) = T(n) + 2 = (c + 2 \cdot n) + 2 = c + 2 \cdot (n + 1)$.

(ii) DPV Problem 2.12. Let $T(n)$ be the number of lines the program prints. When $n = 0$, it does not print at all, hence $T(1) = 0$. When $n > 1$, it prints $1 + 2T(n/2)$ lines, hence $T(n) = 2T(n/2) + 1$. So $a = 2$, $b = 2$, $d = 0$, and hence, $0 = d < \log_b a = \log_2 2 = 1$.

Hence, by the Master Theorem, $T(n) = O(n^{\log_2 2}) = O(n)$.

(iii) PG Problem 337.

The program is buggy. Consider $A[0] = 1$, $A[1] = 2$, $x = 2$, $\ell = 0$, and $r = 1$. Then $m = \lceil (0 + 1)/2 \rceil = 0$ and $2 = x < A[m] = 1$.

So the recursive call is search($A, x, m, r$). But since $m = \ell$, this is an infinite recursion.

(v) DPV Problem 2.16.

Here is the algorithm.

```
function findx(A, x)
    i ← 1
    do i ← 2 ⋅ i until x ≤ A[i]
    return binarySearch(A, x, 0, i)
```

Notice that in each step of the do-until loop, $i$ is one-bit longer than in the step before. So is $n$ is a $k$-bit number, the loop will terminate after $k + 1$ steps. Since $k = 1 + \lfloor \log_2 n \rfloor$ and since the binary search will take $O(\log i) \subseteq O(\log n)$ time, the entire algorithm is $O(\log n)$.

(vi) PG Problem 341.

In the worst case, the search will continue in the $2^m$'s interval. So the recurrence is $T(n) = T(n/2) + 1 = T(n/(2^m)) + 1$. So $a = 1$, $b = 3/2$, $d = 0$, and thus, $0 = d = \log_2 3/2 = 1$. Hence, by the Master Theorem $T(n) \in O(n^d \log n) = O(\log n)$.

(vii) DPV Problem 2.17 (and PG 343).

Since $A[1] < A[2] < \cdots A[n]$ and each $A[i]$ is an integer, it follows that each $A[i + 1]$ is at least 1 greater than $A[i]$. Hence, we have that $A[1] - 1 \leq A[2] - 2 \leq \cdots \leq A[n] - n$ and that the $i$ we are looking for has $A[i] - i = 0$. So an initial call of findi($A, 1, n$) to the following variant of binary search does the job.

```
function findi(A, $l$, $r$)
    if $l = r$ then return $l$
    $m ← \lfloor(l + r)/2\rfloor$
    if $A[m] - m \leq 0$ then return findi($A, \ell, m$)
    else return findi($A, m + 1, r$)
```