1)

a) $T(1) = 1; T(n) = 3T(n-1) + 2 = 3(3T(n-2) + 2 + 2) = 9T(n-2) + 4 = 9(3T(n-3) + 2) + 4 = 27T(n-3) + 6 = 3^iT(n-i) + 2i; i = n-1; 3^nT(1) + 2n = 3^n + 2n$

b) $T(1) = 1; T(n) = 2T\left(\frac{n}{2}\right) + 6n - 1 = 2\left(2T\left(\frac{n}{4}\right) + \frac{6n}{2} - 1\right) + 6n - 1 = 4T\left(\frac{n}{4}\right) + 12n - 3 = 4\left(2T\left(\frac{n}{8}\right) + \frac{6n}{4} - 1\right) + 12n - 3 = 8T\left(\frac{n}{8}\right) + 18n - 7 = 2^iT\left(\frac{n}{2^i}\right) + 6ni - \left(2^i - 1\right); i = log2(n); 2^{log2(n)}T(1) + 6nlog2(n) - \left(2^{log2(n)} - 1\right) = 2^{log2(n)} + 6nlog2(n) - 2^{log2(n)} + 1 = 6nlog2(n) + 1$

c) $T(1) = 1; T(n) = 3T\left(\frac{n}{2}\right) + n^2 - n = 3\left(3T\left(\frac{n}{4}\right) + n^2 - n\right) + n^2 - n = 9T\left(\frac{n}{4}\right) + 4n^2 - 4n = 9\left(3T\left(\frac{n}{8}\right) + n^2 - n\right) + 4n^2 - 4n = 27T\left(\frac{n}{8}\right) + 13n^2 - 13n = 27\left(3T\left(\frac{n}{16}\right) + n^2 + n\right) + 13n^2 + 13n = 81T\left(\frac{n}{16}\right) + 40n^2 + 40n = 3^iT\left(\frac{n}{2^i}\right) + \frac{(3^i-1)n^2}{2} + \frac{(3^i-1)n}{2}; i = log2(n); 3^{\log2(n)}T\left(\frac{n}{2^{log2(n)}}\right) + \frac{(3^{log2(n)}-1)n^2}{2} + \frac{(3^{log2(n)}-1)n}{2} = 3^{log2(n)} + \frac{(3^{log2(n)}-1)(n^2+n)}{2}$

2)

a) LinSearch(A, v, n):

```
for i from 0 to n-1:
    if A[i] == v:
        return i
return Nil
```

b) Loop invariant:

for all j < i, A[j] != v

Initialization:

i=0, no j < 0, statement holds.

Maintenance:

For each loop, either A[i] = v and it returns, or A[i] != v in which case i incriments and the invariant holds.

Termination:

Either A[i] = v, in which case i has been found, or the loop ends and the algorithm returns Nil, in which case v is not in A.

3)

a)

```
bool search(A, x, l, r):
    if r-l == 0:
        if A[l] == x:
            return true
        return false
    if l-r < 2: //0 indexed, so [0,1,2]. 2-0 is 2, in which case it can be repartitioned.
        if A[l] == x: return true
        if A[r] == x: return true
        return false
    leftSplit = l + floor((r-l)/3)
    rightSplit = leftSplit + ceiling((r-l)/3)
    if (x < A[leftSplit]): return search(A, x, l, leftSplit)
    if (x > A[rightSplit]): return search(A, x, rightSplit+1, r)
    else: return search(A, leftSplit+1, rightSplit)
```

b)

T(1) = 1

T(2) = 1

T(n) = T(n/3) + O(n^0)

By masters therom: d=0, a=1, b=3, 1 = 3^0, T(n) = O(logn)

4)

Height:

a)

```
int binTreeHeight(T):
    if empty(T): return -1
    if child(T) == Nil: return 1
    leftChild = left(T)
    rightChild = right(T)
    return max(binTreeHeight(leftChild), binTreeHeight(rightChild)) + 1
```

b)

T(0) = 1

T(1) = c

$T(n) = 2T\left(\frac{n}{2}\right) + c_2 = 2^i T\left(\frac{n}{2^i}\right) + ic_2 = 2^{log2(n)}T(1) + log2(n)c_2 = 2^{log2(n)}c_1 + log2(n)c_2$

Leaf:

a)

```
int binTreeLeafs(T):
    if empty(T): return -1
    if child(T) == Nil: return 1
    leftChild = left(T)
    rightChild = right(T)
    return binTreeLeafs(leftChild) + binTreeLeafs(rightChild)
```

5)

a)

int[][] createSchedule(k): //each subarray int[n] will be the matchups for player n+1. int[n][m] will be n+1's opponent on day m+1

```
    if k==1: return [[2][1]]
    halfProblem = createSchedule(k-1) // should be a 2**(k-1)x(2**(k-1) - 1) 2d array.
    offsetHelper = 2**k-1 + 1
    for i from 0 to 2**(k-1) – 1:
        for j from 2**(k-1) -1 to 2**k - 1:
            halfProblem[i][j] = offsetHelper
            offsetHelper++
            if (offsetHelper > 2**k):
                offsetHelper = 2**(k-1) + 1
    //players 1 through 2**(k-1) now have their matchups
    for i from 0 to 2**(k-1)-2: // copy original halfProblem, but add 2**(k-1). Runs through days. Offset for zero index
        for j from 0 to 2**(k-1) – 1:
            halfProblem[2**(k-1) + j][i] = halfProblem[j][i] + 2**(k-1)
    // players 1 through 2**k now have days 1 through 2**(k-1) – 1
    for i from 2**(k-1) -1 to 2**k - 2: //days
        for j from 0 to 2**(k-1)-1: //players
            halfProblem[2**(k-1) + j][i] = halfProblem[j][i] – 2**(k-1)
    // all players now have all assignments.
    return halfProblem
```

PROBLEM SCREENSHOTS ON NEXT PAGE

```
Joel:Assignment2 $ ./a.out
Please input k: 1
        1
1       2
2       1
```

```
Joel:Assignment2 $ ./a.out
Please input k: 2
        1       2       3
1       2       3       4
2       1       4       3
3       4       1       2
4       3       2       1
```

```
Joel:Assignment2 $ ./a.out
Please input k: 3
        1       2       3       4       5       6       7
1       2       3       4       5       6       7       8
2       1       4       3       6       7       8       5
3       4       1       2       7       8       5       6
4       3       2       1       8       5       6       7
5       6       7       8       1       2       3       4
6       5       8       7       2       3       4       1
7       8       5       6       3       4       1       2
8       7       6       5       4       1       2       3
```

```
Joel:Assignment2 $ ./a.out
Please input k: 4
        1       2       3       4       5       6       7       8       9       10      11      12      13      14      15
1       2       3       4       5       6       7       8       9       10      11      12      13      14      15      16
2       1       4       3       6       7       8       5       10      11      12      13      14      15      16      9
3       4       1       2       7       8       5       6       11      12      13      14      15      16      9       10
4       3       2       1       8       5       6       7       12      13      14      15      16      9       10      11
5       6       7       8       1       2       3       4       13      14      15      16      9       10      11      12
6       5       8       7       2       3       4       1       14      15      16      9       10      11      12      13
7       8       5       6       3       4       1       2       15      16      9       10      11      12      13      14
8       7       6       5       4       1       2       3       16      9       10      11      12      13      14      15
9       10      11      12      13      14      15      16      1       2       3       4       5       6       7       8
10      9       12      11      14      15      16      13      2       3       4       5       6       7       8       1
11      12      9       10      15      16      13      14      3       4       5       6       7       8       1       2
12      11      10      9       16      13      14      15      4       5       6       7       8       1       2       3
13      14      15      16      9       10      11      12      5       6       7       8       1       2       3       4
14      13      16      15      10      11      12      9       6       7       8       1       2       3       4       5
15      16      13      14      11      12      9       10      7       8       1       2       3       4       5       6
16      15      14      13      12      9       10      11      8       1       2       3       4       5       6       7
```

6)

a) Code included

b) Worst case is it hits line 19 every time but the leafs, leading to 2 isHeap checks every time. So T(0) = c1; T(n) = 2T(n/2) + c2 = 2(2T(n/4)+c2) + c2 = 2^i T(n/2^i) + ic2 = 2^log2(n)c1 + log2(n)c2 = nc1 + log2(n)c2 = O(n)

7)

a)

```
void delElem(h, n, v): // n is the size of h
    for i from 0 to n-1:
        if (h[i] == v):
            swap(h, i, n-1)
            delete(h[n-1])
            max-heapify(h[i]) // max-heapify is logn
            break
```

b) Worst case is O(nlogn). n because worst case is it loops all the way to the last element before finding v, logn because it must call max-heapify after swaping and deleting.