Question 1

1. True

   $\sqrt{3n^2 + 4n - 5} \leq \sqrt{3n^2 + 4n}$ Overestimated -5 to 0

   $\sqrt{3n^2 + 4n - 5} \leq \sqrt{3n^2 + 4n^2}$ Overestimated 4n to $4n^2$($4n^2 \geq 4n$ for all $n \geq 1$)

   $\sqrt{3n^2 + 4n - 5} \leq \sqrt{7n^2}$ Simplified

   $\sqrt{3n^2 + 4n - 5} \leq \sqrt{7} * n$ Simplified

   $\sqrt{3n^2} \leq \sqrt{3n^2 + 4n - 5}$ Underestimated $4n - 5$ to 0($0 \leq 4n - 5$ for all $n \geq 2$)

   $\sqrt{3} * n \leq \sqrt{3n^2 + 4n - 5}$ Simplified

   $\sqrt{3} * n \leq \sqrt{3n^2 + 4n - 5} \leq \sqrt{7} * n$ Combined both inequalities

   $c_1 = \sqrt{3},\ c_2 = \sqrt{7},\ n_0 = 2$ Proof is complete

   Important note: n_0 can't be 1 for the solution shown here, because we underestimated $4n - 5$ to 0 in our analysis, and 0 is only an underestimation of $4n - 5$ for $n \geq 2$

2. True

   $\log_2(n^2) = 2\log_2(n)$ Simplified

   $c = 2,\ n_0 = 1$ Proof is complete

3. False

   There is no constant c such that $log_2(n) \geq c * \sqrt{n}$. This is because $\sqrt{n}$ grows much faster than $log_2(n)$. You'd have to apply a log to $\sqrt{n}$ first in order for it to be within a constant factor of $log_2(n)$:

   $log_2(\sqrt{n}) = log_2(n^{0.5}) = 0.5 * log_2(n)$

Question 2

```
lst1 = [10, [2, 30], [4, 5], 6]
lst2 = [10, [2, 30], [4, 5], 6]
lst3 = [1, [2, 30], [4, 5], 6]
lst4 = [1, [2, 3], [4, 5], 6]
```

Explanation:

- lst2 refers to the exact same object as lst1, so all changes to lst1 appear in lst2
- lst3 is a shallow copy, so changes to primitive values in lst1, like changing the integer 1 to the integer 10, don't appear, but changes to objects in lst1, like changing the sublist [2,3] to [2,30], do appear
- lst4 is a deep copy, so no changes to lst1 appear in lst4

Question 3

```
def positive_prefix_sum(lst):
  return [i for i in range(len(lst)) if sum(lst[:i+1]) > 0]
```

[1,-1,4,-2]

Explanation:

For each index in the lst, find the prefix sum ending at that index and check if it's positive to decide whether to include it in the final output of the list comprehension. The prefix sum ending at an index i is `sum(lst[:i+1])`.

Question 4

1. $\Theta(n^2)$

   The function inserts into the 0th index of rev_lst, which costs a number of basic operations roughly equal to the current length of rev_lst. Here's the number of basic operations used for each iteration of the while loop:

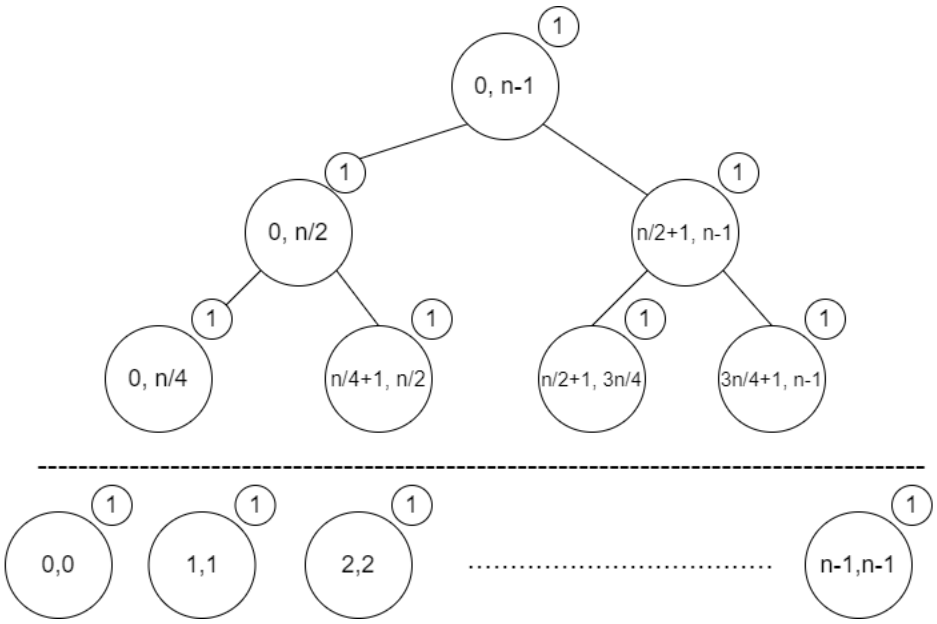
   i = 0, len(rev_lst) = 0, basic operations ≈ 0
   i = 1, len(rev_lst) = 1, basic operations ≈ 1
   i = 2, len(rev_lst) = 2, basic operations ≈ 2
   i = 3, len(rev_lst) = 3, basic operations ≈ 3
   …
   i = n-1, len(rev_lst) = n-1, basic operations ≈ n-1

   So the sum of the basic operations over every iteration of the while loop is roughly 0+1+2+3+...+n-1, which is equal to $\frac{n(n-1)}{2}$, which is $\Theta(n^2)$

2. $\Theta(n)$

   The function appends n times, and append is amortized constant time, meaning a series of n appends on a list costs roughly n basic operations.

Question 5

1. 

2. $\Theta(n)$

   The number of basic operations in each function call is constant because it only does things like computing a middle index and checking if min1 is less than min2.

   Based on the recursion tree, the number of calls roughly doubles from level to level, so the sum of the number of basic operations per level is roughly equal to 1+2+4+8+...+n, which is equal to 2n-1, which is $\Theta(n)$

Question 6

```
def remove_all_evens(lst):
  first_even_index = 0

  for i in range(len(lst)):
    if lst[i]%2 == 1:
      lst[first_even_index], lst[i] = lst[i], lst[first_even_index]
      first_even_index += 1

  while first_even_index < len(lst):
    lst.pop()

  return lst
```

Explanation:
If we keep track of the index of the first even element in lst, we can swap each new odd element we find with the first even element to keep all the odds before all the evens.

After all the swaps are done, the end of the list will have all the even elements if there were any. We can remove these by using pop from the end (which is amortized constant time) until there are no evens left. first_even_index < len(lst) is a valid way to check if there are any evens left since an index being < len(lst) means it's an existing index in lst.

Question 7

```
def is_sorted(lst, low, high):
  if low == high:
    return True

  return is_sorted(lst, low, high-1) and lst[high-1] <= lst[high]
```

[1 2,Infinity, -3]

Explanation:

A list is sorted for the indexes [low, high] if everything before the last element, [low, high-1], is sorted and the last element lst[high] is greater than or equal to the previous element lst[high-1]