Q1. The following incomplete recursive function sorts a list of numbers. Write the missing lines of code (including any recursive call).

```python
def sort(xs):
    """ Sorts a list of numbers.
    e.g. sort([3,7,2,4,3]) -> [7,4,3,3,2] """
    if not xs:
        return ...
    a = min(xs)
    i = xs.find(a)
    return ...        # Hint: take slices of xs using i
```

Q2. In lectures, we wrote a program which gives instructions to solve the Tower of Hanoi problem. The solution described in lectures is to move the first $n-1$ disks from A to B, then move the largest disk from A to C, then the $n-1$ disks from B to C.

A variant of the problem is to have 3 towers A, B and C as before, but you cannot move a piece from A to C or C to A. Find the solution to this problem, then write a function that takes a number n and prints the instructions to move the tower from A to C, without moving any pieces directly $A \to C$ or $C \to A$.

Run your function on $n = 3$ and verify the output by following the instructions.

Q3. Consider the following `sum1` function, which attempts to compute the sum of a list of numbers. Give an example of a list `xs` where this function will not work, and explain why it doesn't work.

```python
def sum1(xs):
    n = len(xs)
    if n == 0:
        return 0
    else:
        m = n / 2  # m = ⌊n/2⌋
        return sum1(xs[:m]) + sum1(xs[m:])
```

Q4. Next, consider the following `sum2` function, which is a modification of the above. Compute by hand the sum of the list `[3,4,5,6,7]`. In your working, indicate the recursive calls that are made.

Then, use induction to prove that this function will always return the correct sum. (Hint: `xs[:m]` is a list of length m, and `xs[m:]` is a list of length $n - m$.)

Given that we know `sum1` doesn't work, explain why we could not use the same technique to prove `sum1` is correct. (i.e. what part of the proof for `sum2` requires that $n = 1$ is a base case?)

```python
def sum2(xs):
    n = len(xs)
    if n == 0:
        return 0
    elif n == 1:
        return xs[0]
    else:
        m = n / 2
        return sum2(xs[:m]) + sum2(xs[m:])
```

Solutions

Q1. A sample `sort` function is as follows.

```python
def sort(xs):
    """ Sorts a list of numbers.
    e.g. sort([3,7,2,4,3]) -> [7,4,3,3,2] """
    if not xs:
        return xs
    a = min(xs)
    i = xs.find(a)
    return sort(xs[:i] + xs[i+1:]) + [a]
```

Q2. Solution: move the first $n-1$ disks to C, then move the n^{th} to B, then move the first $n-1$ disks back to A, then move the n^{th} to C, then move the first $n-1$ back to C.

```python
def hanoi(n):
    _hanoi_helper(n, 'A', 'C')

def _hanoi_helper(n, A, C):
    if n == 0:
        return
    _hanoi_helper(n-1, A, C)
    print A, "to B"
    _hanoi_helper(n-1, C, A)
    print "B to", C
    _hanoi_helper(n-1, A, C)
```

Q3. Consider the input `xs = [1]`. This will set $n = 1$, go into the `else` statement, and set $m = 1/2 = 0$. Then, the function will return `sum([]) + sum([1])`. The second of these calls is the same as the original call, so we will have an infinite recursion.

Q4.

`sum2([3,4,5,6,7]) = sum2([3,4]) + sum2([5,6,7])` $= \ldots = 25$
  `sum2([3,4]) = sum2([3]) + sum2([4])` $= \ldots = 7$
    `sum2([3]) = 3`
    `sum2([4]) = 4`
  `sum2([5,6,7]) = sum2([5]) + sum2([6,7])` $= \ldots = 18$
    `sum2([5]) = 5`
    `sum2([6,7]) = sum2([6]) + sum2([7])` $= \ldots = 13$
      `sum2([6]) = 6`
      `sum2([7]) = 7`

Claim: The `sum2` function returns the sum of the list `xs`.

Base cases: if $n = 0$, then the list is empty, so the sum is 0 (by definition of the empty sum). If $n = 1$, then the sum of the list is the value of the single element.

Inductive hypothesis: For some $n \geq 2$, suppose that the `sum2` function returns the sum of any list of length k, for any $0 \leq k < n$.

Inductive step: If `xs` has length $n \geq 2$, then $m = \lfloor \frac{n}{2} \rfloor$ satisfies $1 < m < n$. Then, the sublists of length $m < n$ and $n-m < n$ will return $\sum_{i=0}^{m-1} \texttt{xs}[\texttt{i}]$ and $\sum_{i=m}^{n-1} \texttt{xs}[\texttt{i}]$ respectively, then the function will return $\sum_{i=0}^{n-1} \texttt{xs}[\texttt{i}]$, the correct sum. □

If we tried the same proof for `sum1`, it would require the inductive step to hold for $n \geq 1$ (instead of $n \geq 2$). However, if $n \geq 1$, it is not guaranteed that $n - m < n$ (since it is possible that $m = 0$), hence we cannot apply the inductive hypothesis to the list `xs[m:]`.