CSC236 tutorial exercises, Week #6
sample solutions

1. Recall the function subset_sum which we saw in lecture on week 4:

```python
def subset_sum(A, target):
  if len(A) == 0:
    return target == 0
  return subset_sum(A[1:], target) or subset_sum(A[1:], target-A[0])
```

Informally, subset_sum is supposed to return a boolean corresponding to whether or not there exists a subset of A, a list of integers, which adds up to the given integer $target$. For example, `subset_sum([4, 2, -1], 3)` should return True, and `subset_sum([4, 2, -1], 0)` should return False.*

> ***Note:** As several people noticed, `subset_sum([4, 2, -1], 0)` actually returns True! This example was an oversight on my part. It is intended behaviour that we return True whenever target is 0, because we consider the empty sum to be equal to zero. The answers below are written with this interpretation in mind.
>
> If we (not unreasonably) used an interpretation where the expected return value of `subset_sum([4, 2, -1], 0)` is indeed False, we would need to make the following modifications to the answers below:
>
> - Our postcondition would be almost identical, but would require the stipulation that the set I must be non-empty.
> - The same precondition would work. Adding the constraint that A be non-empty would be acceptable in this case, but not necessary.
> - For part (d), we might just throw up our hands and say that no proof is possible, since the given code doesn't meet our specifications. Alternatively, we could consider a 'fixed' version of the code, where our recursion bottoms out at length 1, and line 3 returns `target == A[0]`. The correctness proof for this function would be almost identical to the sample solution, just with a different base case.

(a) Devise a precondition for subset_sum. (It should be no more restrictive than necessary.)

Solution:
Precondition: A is a list containing only integers. target is an integer.

Note that we are *not* requiring A to be non-empty. We are saying that if it *does* contain any elements, they should all be ints.

(b) Consider the following proposed postcondition for subset_sum:

Postcondition: `subset_sum(A, target)` returns True if there exists a set of indices, I, such that $\sum_{i \in I} A[i] =$ target.

Explain why this postcondition does *not* adequately describe the intended behaviour of subset_sum. Hint: think about other algorithms that would also satisfy this postcondition.

Solution:

The postcondition above doesn't say anything about our algorithm's behaviour if there isn't such a set I. Technically, we could easily write a function that satisfies this postcondition, by just always returning True.

(c) Devise a better postcondition for subset_sum. Your postcondition should be precise and unambiguous. (The failed postcondition from part b may be useful as a starting point.)

Solution:

Postcondition: `subset_sum(A, target)` returns True if there exists a set of indices, I, such that $\sum_{i \in I} A[i] =$ target, and returns False otherwise.

There are other ways this idea could be phrased, for example by replacing the "if" with "if and only if". (This leaves it implicit that, if the function doesn't return True, it returns False.)

(d) Use simple induction to prove that subset_sum is correct with respect to the precondition and postcondition you defined above.

Solution:

Define $P(n)$: "for any list A having length n and satisfying the precondition, and for any target $\in \mathbb{Z}$, `subset_sum(A, target)` terminates and satisfies the postcondition."

<u>**Base Case**</u>:Let A be a list of length 0 which satisfies the precondition, and let `target` be an arbitrary integer. A contains only one subset of elements, the empty set, which sums to 0. So the postcondition says that we should return True if and only if `target` is equal to 0, which is what our function does. So $P(0)$.

<u>**Inductive Step**</u>:Let $n \in \mathbb{N}$ and assume $P(n)$. Let A be a list of length $n+1$ satisfying the precondition, and let target be some integer.

To begin, note that $n + 1 > 0$, so our code reaches line 4. Also, `A[1:]` is a list of length n satisfying the precondition, so by our IH, the recursive calls on line 4 return 'correct' values (i.e. values satisfying our postcondition).

We'll consider two cases:

<u>Case 1:</u> there exists a set of indices, I, such that $\sum_{i \in I} A[i] =$ target. In this case, the postcondition requires that we return True.

<u>Case 1a:</u> $0 \in I$. Then $\sum_{i \in I - \{0\}} A[i] = target - A[0]$. By our IH, this means that `subset_sum(A[1:], target-A[0])` will return True, and we thus return True on line 4.

<u>Case 1a:</u> $0 \notin I$. Then $\sum_{i \in I} A[i] = target$. By our IH, this means that `subset_sum(A[1:], target)` will return True, and we thus return True on line 4.

In either sub-case, we return True, as required by the postcondition.

<u>Case 2:</u> there does not exist a set of indices, I, such that $\sum_{i \in I} A[i] =$ target. In this case, the postcondition requires that we return False.

Then by our IH, `subset_sum(A[1:], target)` will return False, since this case implies that there is also no subset of `A[1:]` that sums to target. Similarly, `subset_sum(A[1:], target-A[0])` will return False by the IH, since the existence of a subset of `A[1:]` summing to that value would contradict the condition of our case.

So on line 4, we return False, as required.

In both cases, the behaviour of our function on A matches the postcondition. So for all A of length n+1 which meet the precondition, our algorithm terminates and meets the postcondition. i.e. $P(n+1)$ holds. ■

2. Prove that the following function terminates on all valid inputs.

```
1  def mystery(n):
2    """Pre: n is a positive integer
3    Post: ???
4    """
5    if n == 1:
6      return 1
7    elif n % 2 == 1:
8      return 1 + mystery(n+1)
9    else:
10     return 1 + mystery(n//2)
```

Solution:
Assume, for the sake of deriving a contradiction, that mystery does not terminate on all inputs. Consider the set

$$S = \{n \in \mathbb{N} \mid \text{mystery}(n) \text{ does not terminate}\}$$

By our assumption, S is non-empty, so by the Principle of Well-Ordering, it must have a minimum element, call it j.

Case 1: $j = 1$. By lines 5-6, mystery(1) terminates, so this is a contradiction.

Case 2: $j > 1 \wedge j$ is even. By lines 9-10, a call to mystery(j) recursively calls mystery($j \,//\, 2$). Since $j > 1$, $j \,//\, 2 < j$. If this recursive call doesn't terminate, then $j \,//\, 2$ should belong to S, but this would contradict the minimality of j. Contradiction.

Case 2: $j > 1 \wedge j$ is odd. By lines 7-8, mystery will call mystery($j+1$) in this case. Since j is odd, $j+1$ must be even, meaning that the recursive call which reach line 10 and call mystery($(j+1) \,//\, 2$). j is at least 3, meaning that $(j+1) \,//\, 2 < j$. Again, if mystery(j) fails to terminate, then mystery($(j+1) \,//\, 2$) must not terminate, but this contradicts the minimality of j.

In each case, we are led to a contradiction, so our initial assumption was faulty. The function mystery must terminate on all inputs.

Remark: As usual with PWO proofs, this could be reframed as a complete induction argument without too much modification. Which one you find more natural is just a matter of taste.