Correctness

Establishing correctness is a key element of designing algorithms.

- that an algorithm is correct is often not as obvious as it is for data structures, especially for optimization problems or when cleverness is needed to improve running times

Reasoning about correctness needs more than just “it looks reasonable”.

- proof involves a chain of reasoning from assumptions to end result

Correctness

Ingredients.

- a clear problem specification is essential
  - defines the set of allowed input instances
  - defines the required properties for the output

Impediments.

- having too broad a class of input instances
  - may need a more restricted problem in order to find an efficient algorithm
- a poorly defined question
  - e.g. looking for the “best” solution without defining what that is
- compound goals that can't be achieved simultaneously or which become difficult to reason about
- wrong level of detail in the algorithm
  - too much detail obscures the idea
  - not enough detail means you don't have enough to work with

Proof Techniques – Induction

- applies to both loops and recursion
  - with loops, establish a *loop invariant* P(*i*) which is a statement about correctness at the beginning of iteration *i*
  - with recursion, P(*n*) is that a correct solution for size *n*
- strategy – to prove P(*n*) for $n \geq n_0$
  - *base case* – show that P(n_0) is true
  - *inductive case* – show P(*k*) → P(*k*+1), that is, if P(*k*) is true, then the code does the right thing to produce P(*k*+1)

Proof Techniques – Contradiction

- assume that what you want to prove is false
- develop logical consequences from this assumption, until you get to one that is demonstrably false
- since there were no flaws in the deduction, the assumption that what you want to prove is false must have been faulty and thus what you want to prove is true

Proof Techniques – Incorrectness

One counterexample is all that is needed to prove an algorithm incorrect.

Properties of a good counterexample.

- simple, which often means small
- verifiable – need to be able to compute the algorithm's output and give a better answer

Strategies.

- think exhaustively – can often enumerate all possible inputs of a small size
- hunt for weakness – look for a case where the algorithm's choice is the wrong thing to do
- try inputs with duplicates or ties, as that neutralizes the algorithm's choice
- seek extremes rather than uniformity

Counterexamples

Find a counterexample to prove the following statement false:

```
a + b >= min(a,b)
```

- both numbers negative e.g. -5, -2
  - -5 + -2 = -7 ≥ min(-5,-2) = -5 → false

Counterexamples

1-2. *[3]* Show that $a \times b$ can be less than $\min(a, b)$.

1-3. *[5]* Design/draw a road network with two points a and b such that the fastest route between a and b is not the shortest route.

1-4. *[5]* Design/draw a road network with two points a and b such that the shortest route between a and b is not the route with the fewest turns.

1-5. *[4]* The subset sum problem is as follows: given a set of integers $S = \{s_1, s_2, \ldots, s_n\}$, and a target number T, find a subset of S that adds up exactly to T. For example, there exists a subset within $S = \{1, 2, 5, 9, 10\}$ that adds up to $T = 22$ but not $T = 23$.

Find counterexamples to each of the following algorithms for the subset sum problem That is, give an S and T where the algorithm does not find a solution , even though a solution exists.

(a) Pick elements of S in left to right order if they fit,

(b) Pick elements of S from smallest to largest, that is,

(c) Pick elements of S from largest to smallest.