Proofs and Dynamic Programming Section
Problems

September 2015

1 Proofs of Correctness

Good things to consider when writing proofs:

• First state what you are going to prove. What does it mean for your algorithm to be correct? For every sentence in your proof, make sure it can relate back to this. Then state how you are going to prove this. Give the reader something to expect.

• Consider an arbitrary input, case, etc. Use variables for any arbitrary values of the input (ex: an arbitrary list of size $n$). Unless we ask for it, a proof by example is not adequate.

• Employ the appropriate proof technique (induction, contradiction, cases, etc.)

• Don’t repeat yourself if possible. Focus on the crucial parts of the proof. Short and concise proofs are more powerful than long, wordy proofs.

• When using pseudocode, reference specific lines in your proof.

• Don’t forget a conclusion! It should match what you said you would prove.

1.1 Selection Sort

Prove the correctness of the Selection Sort algorithm described below.

```plaintext
Selection Sort(L)
1  n = length(L)
2  if $n \leq 1$
3      halt
4  $x = \text{min}(L)$
5  swap $x$ and $L[0]$
6  Selection Sort(L[1 : n])
```
2 Dynamic Programming

Dynamic programming is a way to efficiently solve complex problems by cleverly building on solutions to smaller sub-problems. Not every problem you encounter can be solved with dynamic programming, but you will find that the problems that can be solved with dynamic programming have essentially the same structure. This makes it easy to develop a consistent strategy for solving dynamic programming problems - and more importantly, proving that your strategy is correct.

2.1 Dynamic Programming Proofs

Dynamic programming proofs are an extension off of our basic proof framework.

- **First state what you are going to prove and how you are going to prove it.** With dynamic programming this is usually induction over filling out each element of your table. Specify what “correct” means and also the order in which you will fill out the table.

- **Employ the appropriate proof technique.** With dynamic programming this is usually induction.
  - **Base case:** Explain correctness of the initialization of the table.
  - **Consider an arbitrary input.** For dynamic programming this is an entry into the table. “Consider an arbitrary entry into table, $T(i,j,...)$.”
    - **Inductive Hypothesis:** Assume all entries filled out before $T(i,j,...)$ are correct.
    - **Consider the optimal solution and the last decision made to get to this solution.**
    - **Break down the possible values of $T(i,j,...)$ into cases.** Then explain why in each case your algorithm can do at least as well as the optimal solution.
    - **Verify that any previous entries used to compute $T(i,j,k..)$ are computed before $T(i,j,k...)$ is.**

- **Don’t repeat yourself if possible.**
- **Don’t forget a conclusion!**

2.2 General Doubleday

You are a decorated general in charge of $n$ soldiers. You can deploy any number of these soldiers across $m$ possible battlefields occupied by various enemy troops. Having scouted ahead (you are a decorated general, after all), you have a formula $f(i,j)$ that tells you how many of your soldiers will survive deployment if you deploy $j$ soldiers to the $i^{th}$ battlefield. Assuming that you must deploy all of
your troops, devise an algorithm for determining the maximum possible number of survivors. Prove the runtime of your algorithm. Prove that your algorithm is correct.