CPSC 320: Steps in Algorithm Design and Analysis *

In this worksheet, you'll practice five useful steps for designing and analyzing algorithms, starting from a possibly vague problem statement. These steps will be useful throughout the class. They could also be useful when you find yourself thinking on your feet in an interview situation. And hopefully they will serve you well in your work post-graduation too!

We'll use the **Stable Matching Problem (SMP)** as our working example. Following the historical literature, the text formulates the problem in terms of marriages between men and women. We'll avoid the gender binaries inherent in that literature and use employers and job applicants instead. Imagine for example the task faced by UBC's co-op office each semester, which seeks to match hundreds of student applicants to employer internships. To keep the problem as simple as possible for now, assume that every applicant has a full ranking of employers and vice versa (no ties).

Step 1: Build intuition through examples.

1. **Write down small and trivial instances of the problem.**
   We'll use the words "instance" and "inputs" interchangeably.

2. **Write down potential solutions for your instances.**
   Are some solutions better than others? How so?

*Copyright Notice: UBC retains the rights to this document. You may not distribute this document without permission.

Step 2: Develop a formal problem specification

1. **Develop notation for describing a problem instance.** What quantities or data (such as numbers, sets, lists, etc.) matter? Give them short, usable names. Think of these as input parameters to the algorithm code. Use your earlier examples to illustrate your notation.

2. **Develop notation for describing a potential solution.**
   Use your earlier examples to illustrate your notation.

3. **Describe what you think makes a solution *good*.**

Step 3: Identify similar problems. What are the similarities?

Step 4: Evaluate simple algorithmic approaches, such as brute force.

1. **Design a brute force algorithm for the SMP problem.**
   A "brute force" algorithm has the form given in the following pseudocode. **Flesh out the details: what is the problem input (i.e., input) and potential solution in this case. How would you test if a potential solution is a good solution?**

```
function SMP-Brute-Force(⟨problem instance⟩)

    for each potential solution $M$ do

        if $M$ is a good solution then
            return $M$
    return "no good solution"
```

2. **Analyze the worst case running time of brute force.**

   - **Count the number of iterations of the for loop.** We need to count the number of potential solutions, i.e., perfect matchings. Imagine that the n applicants are arranged in some order. How many different ways can we arrange (permute) the n employers next to them?

   - **Bound the running time of your procedure to test if a potential solution is a good solution.**

   - **Put the previous two parts together to bound the overall running time of brute force.**

Step 5: Design a better algorithm.

1. **Brainstorm some ideas, then sketch out an algorithm.**
   Try out your algorithm on some examples.