Fundamentals of Computer Algorithms
Homework 1

Patrick Canny

Due: 2018-08-28

1. Book Question 1.1
   The presented claim is false, because it is a fact that in any stable matching, for some pair (m, w), m is the "worst" valid partner for w. This fact is illustrated in more detail by conclusion 1.8 in the book. In any instance of the G-S algorithm, the side that does the proposing ends up with the best possible stable matching from their perspective, while the side being proposed to ends up with the worst possible stable matching.

   Consider m_1, m_2, w_1, w_2.
   m_1 prefers w_1 to w_2
   m_2 prefers w_2 to w_1
   w_1 prefers m_2 to m_1
   w_2 prefers m_1 to m_2

   In this situation, it's impossible to create a stable matching with pair (m, w) such that m is the ranked highest for w and vice versa. The pair (m_1, w_1) is preferential for m_1, but not w_1, since m_2 is preferred to m_1 by w_1.

2. Book Question 1.2
   The presented claim is true. We will display this using a proof by contradiction.

   Assume that there is an instance of the stable matching problem where $\exists(m, w'), (m', w) \in S$ such that m is ranked highest for w, and w is ranked highest for m.
   Since m is ranked higher on w's priority list, and w is ranked higher on m's priority list, this specific instance is unstable, a contradiction. Therefore, the pair (m, w) must belong to S.

3. Book Question 1.4
   This algorithm is fairly similar to the original Gale-Shapely algorithm, but with a few minor changes specific for this case

```
While a hospital has at least one opening and hasn't offered every student S:
    Choose some hospital H with at least one opening
    H offers S (S is the highest ranked student that hasn't been offered by H)
    if S is available:
        S accepts the offer, and H has one less opening
    else S has already accepted an offer from H':
        if S prefers H':
            S will reject the offer
            The number of openings at H stays the same
        else S prefers H:
            S accepts H's offer
            H will have one less opening
```

H' will have one more opening
Return 1. the set of matches between students and hospital openings
2. the set of all students not matched

Now, we must show that the algorithm will always generate a stable matching. We can do this using a proof by contradiction.

Assume that the algorithm generates an instability, i.e.
1. s is assigned to h, s' is unassigned, and h prefers s'.
2. $\exists$ pairs $(s,h),(s',h')$ but h prefers s' and s' prefers h.

In the first case, we use the fact that h will make offers in order of preference. Since s' is unassigned and prefered to s by h, this means that h must've not made an offer to s', a contradiction.

In the second case, if h preferred s' to s, an offer would've been made to s' first. Down the line, s' would have had the option to "trade up" for a different hospital. If s' truly prefers h, then this "trade" would never occur, leading to a contradiction of this case.
Since neither instability occurs, the assignment generated by the algorithm will always be stable.

4. Prove that $\sum_{k=0}^{n} k = (1/2)n(n+1)$.

1. First, we will define our base case (i.e. the case where n = 0)
$$\Rightarrow (1/2)0(0+1) = \sum_{k=0}^{0} k = (1/2)0(0+1) = 0$$
2. Next, we hypothesize that this claim holds true for n terms, and prove that it holds for n+1 terms
$$\text{i.e.: } \sum_{k=0}^{n+1} k = (1/2)n + 1(n+2)$$
$$\Rightarrow 0+1+2+...+n+(n+1) = (1/2)n+1(n+2)$$
$$\Rightarrow (1/2)n(n+1)+(n+1) = (1/2)n+1(n+2)$$
$$\Rightarrow (n^2+n/2)+(n+1) = \tfrac{n^2+3n+2}{2}$$
$$\Rightarrow (n^2/2)+(3n/2)+1 = (n^2/2)+(3n/2)+1$$
$\therefore$ The claim is proven via induction.

5. Prove that $\sum_{k=0}^{n} a^k = \dfrac{a^{n+1}-1}{a-1}$.

1. First, we will define our base case (i.e. the case where n = 0)
$$\Rightarrow \tfrac{0^{0+1}-1}{0-1} = \sum_{k=0}^{0} n^k = \frac{0^{0+1}-1}{0-1}$$
2. Next, we hypothesize that this claim holds true for n terms, and prove that it holds for n+1 terms
$$\text{i.e.: } \sum_{k=0}^{n+1} a^k = \frac{a^{n+1+1}-1}{a-1}$$
$$\Rightarrow 1+a+a^2+...+a^n+a^{n+1} = \tfrac{a^{n+1+1}-1}{a-1}$$
$$\Rightarrow \tfrac{a^{n+1}-1}{a-1}+(n+1) = \tfrac{a^{n+1+1}-1}{a-1}$$
$$\Rightarrow a^{n+1}-1+a^{n+1}(a-1) = a^{n+2}-1$$
$$\Rightarrow a^{n+1}-1+a^{n+2}+a^{n+1} = a^{n+2}-1$$
$$\Rightarrow a^{n+2}-1 = a^{n+2}-1$$
$\therefore$ The claim is proven via induction.

6. For each of the sums below, (1) write a python function to evaluate it and (2) evaluate it by hand, showing your work (you need not simplify).

(i) $\sum_{k=0}^{100} k$.

1)

```
def foo():
    sum = 0
    for k in range(0,101):
        sum += k
    return sum
```

2) In general, $\sum_{k=0}^{n} k = \frac{n(n+1)}{2}$.

By this rule, the summation can be expressed as: $\sum_{k=0}^{100} k = \frac{100(100+1)}{2}$

(ii) $\sum_{k=0}^{101} k^2$.

1)

```
def bar():
    sum = 0
    for k in range(0,101):
        sum += k**2
    return sum
```

2) In general, $\sum_{k=0}^{n} k^2 = \frac{n(n+1)(2n+1)}{6}$.

By this rule, the summation can be expressed as: $\sum_{k=0}^{100} k^2 = \frac{100(100+1)(2(100)+1)}{6}$

(iii) $\sum_{k=12}^{123} k^2 + k + 1$.

1)

```
def foobar():
    sum = 0
    for k in range(12, 124):
        sum += k**2 + k + 1
    return sum
```

2) In general, a summation that contains a list of summed terms can be expressed as list of summations of those terms.

$$\text{i.e. } \sum_{k=12}^{123} k^2 + k + 1 = \sum_{k=12}^{123} k^2 + \sum_{k=12}^{123} k + \sum_{k=12}^{123} 1.$$

These summations are fairly similar to the rules above, but we must implement an offset for each one to account for starting at a number larger than 0 (12 in this case).

$$\text{i.e: } (\sum_{k=0}^{123} k^2 - \sum_{k=0}^{11} k^2) + (\sum_{k=0}^{123} k - \sum_{k=0}^{11} k) + (\sum_{k=0}^{123} 1 - \sum_{k=0}^{11} 1)$$

Finally, we can simplify the summations above per the summation rules mentioned in the previous problems, as well as the fact that $\sum_{k=0}^{n} 1 = n$.

$$\text{i.e: } (\tfrac{123(123+1)(2(123)+1)}{6} - \tfrac{11(11+1)(2(11)+1)}{6}) + (\tfrac{123(123+1)}{2} - \tfrac{11(11+1)}{2}) + (123 - 12).$$

7. Evaluate $\sum_{0 \le k < n^2} \sum_{j=1}^{\lfloor \sqrt{k} \rfloor} 1$.

First, establish a table of values to get a feeling for how the floor function will look at a variety of k

k	$\lfloor \sqrt{k} \rfloor (l)$	Block Size
0	0	1
1..3	1	3
4..8	2	5
9..15	3	7
16..24	4	9

Next, we will denote the discrete set of values of $\lfloor \sqrt{k} \rfloor$ as l, and refactor the summation in terms of l .

$$\sum_{0 \le k < n^2} \sum_{j=1}^{\lfloor \sqrt{k} \rfloor} 1 = \sum_{0 \le k \le n^2 - 1} \lfloor \sqrt{k} \rfloor$$

At this point, we notice that there are a total of $n-1$ blocks that we are trying to sum up. This is because the original sum is being run from 0 to $n^2 - 1$, and when $\lfloor \sqrt{k} \rfloor$ is evaluated for $k = n^2 - 1$, the result will be $n - 1$. Each of these blocks has a length that can be defined in terms of l.

$$\therefore \text{ we can rewrite the summation explicitly in terms of } l\text{: } \sum_{0 \le l \le n-1} l * (blocksize).$$

The "block size" corresponding to a block of ls can be defined as $(l+1)^2 - l^2$.

$$\therefore \text{ the summation can be written as: } \sum_{0 \le l \le n-1} l * ((l+1)^2 - l^2)$$

$$= \sum_{0 \le l \le n-1} l(2l+1) = \sum_{0 \le l \le n-1} 2l^2 + l = \sum_{0 \le l \le n-1} 2l^2 + \sum_{0 \le l \le n-1} l$$

$$= \tfrac{n(n-1)(2n-1)}{3} + \tfrac{n(n-1)}{2}$$

8. Consider the algorithm below.

```
def S(N):
  A = []
  for n in range(2,N+1):
    do_add_n = True
```

```
    for a in A:
      if n % a == 0:
        do_add_n = False
        break
    if do_add_n:
      A.append(n)
  return A
```

(i) Implement this code in python.

```
def S(N):
    A = []
    for n in range(2,N+1):
        do_add_n = True
        for a in A:
            if n % a == 0:
                do_add_n = False
                break
        if do_add_n:
            A.append(n)
    return A
```

(ii) What is $S(10)$?
$S(10) = [2, 3, 5, 7]$

(iii) What is $S(100)$?
$S(100) = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97]$

(iv) What does S do?
S calculates the list of primes from 2 to N

(v) It is a famous fact that, in the outer loop, `len(A)` $= O(n/\ln(n))$. Use this to estimate the O-complexity of $S(N)$. It may help in your analysis to ignore the `break` statement.

We will use the "dominating step" approach to solving this problem, where we will primarily consider the most time-intensive steps when solving the problem.
We know immediately that the contents of the outer `for` loop will run N many times, because the length of this loop strictly depends on the value of N.
By the famous fact, we can determine that the inner loop will, at worst case, run in $O(N/\ln(N))$ time.
All other operations in the problem can be ignored, since they do not depend on the value of N. We wil treat them as constants in this case.
$\therefore$ the final complexity of the algorithm can be given by: $N * N/\ln(N) * 1 = N^2/\ln(N)$