CS 180 Summer 25 – Homework 1

Due Saturday, July 5, 11:59pm

- Please write your student ID and the names of anyone you collaborated with on the first page of your submission. Do not include your name on your assignment as the homework will be blind graded.
- You may use any theorem proven in class or in the textbook without proof. When referring to these theorems, write "as was proven in class" or "as was proven in the textbook" accordingly.
- **If you are asked to write an algorithm, you must prove its correctness.** If you are asked for a time complexity analysis, then you should give a reasonable big-O notation upper bound.
- The starred problems (Problems 4, 5, and 6) will be graded by correctness.
- All other problems (Problems 1, 2, 3, and 7) will be graded by completeness. You will receive full credit on these problems as long as a good-faith effort was made to solve them. This includes showing your work and providing proofs of your statements.

1. **Proof Techniques**

   (a) Prove by induction on n that for all integers $n \geq 1$,

   $$1 + 4 + 7 + \ldots + (3n-2) = \frac{n(3n-1)}{2}.$$

   Next Page.

   (b) Let x and y be positive real numbers. Prove by contradiction: If $x^2 - y^2 = 1$, then x or y (or both) are not integers.

2. **Order Notation**

   Indicate the relationships between each pair of functions (A, B) listed below by writing a "yes" or a "no" in each empty box of the table below. For example, if $A \in o(B)$, then you should write a "yes" in the corresponding row of the first empty column. If the base of a logarithm is not specified, you should assume it is base 2. For this problem, you do not have to show your work or justify your answer.

A	B	A ∈ o(B)	A ∈ O(B)	A ∈ θ(B)	A ∈ Ω(B)	A ∈ ω(B)
$\log_2(n)$ $>$	$\log_5(n)$	no	no	no	yes	yes
$\log\log(n)$ $<$	$\sqrt{\log(n)}$	yes	yes	no	no	no
$n^{4/3}\log(n)$ $<$	$n^{\log(n)}$	yes	yes	no	no	no
$2^{\log^6(n)}$ $>$	n^6	no	no	no	yes	yes
$n^3 2^n$ $<$	3^n	yes	yes	no	no	no

①

ⓐ For any integer $n \geq 1$, the following formula holds: $1+4+7+\dots+(3n-2) = \frac{n(3n-1)}{2}$

<u>Base case $(n=1)$:</u>
We test the formula for the first case, $n=1$.

$$1 = \frac{1(3-1)}{2} = \frac{2}{2} = 1.$$

The base case is true.

<u>Inductive step:</u>

1. <u>Inductive Hypothesis</u>: Assume the formula is true for some arbitrary integer $k \geq 1$.

$$1+4+7\dots+(3k-2) = \frac{k(3k-1)}{2}$$

2. <u>Goal</u>: We will show that the formula holds for $k+1$.

$$1+4+7+\dots+(3k-2)+(3(k+1)-2) = \frac{(k+1)(3(k+1)-1)}{2} = \frac{(k+1)(3k+2)}{2}$$

3. <u>Derivation</u>

$$\begin{aligned}(1+4+7+\dots+(3k-2))+(3(k+1)-2) &= \frac{k(3k-1)}{2}+(3(k+1)-2)\\ &= \frac{k(3k-1)}{2}+3k+1\\ &= \frac{k(3k-1)}{2}+\frac{6k+2}{2}\\ &= \frac{3k^2+5k+2}{2}\\ &= \frac{(3k+2)(k+1)}{2}\end{aligned}$$

4. <u>Conclusion:</u>
The result matches our hypothesis. We have successfullu shown that if the formula holds for k, it must hold for $k+1$. Thus, by induction, the formula holds for all integers $n \geq 1$.

ⓑ Let $x^2-y^2=1$. Assume both numbers are integers Then;

If $x^2-y^2=1$, then:

$$(x-y)(x+y)=1$$

Only two pairs of integers multiply to 1: $(1,1)$ & $(-1,-1)$.

So, either

$$\begin{aligned}x-y&=1\\x+y&=1\end{aligned} \quad \text{or} \quad \begin{aligned}x-y&=-1\\x+y&=-1\end{aligned}$$

If $(1,1)$:

$$\underbrace{\begin{aligned}(x-y)+(x+y)&=2x\\1+1&=2x\\2&=2x\\1&=x\end{aligned} \qquad \begin{aligned}(x+y)-(x-y)&=2y\\1-1&=2y\\0&=2y\\0&=y\end{aligned}}_{\text{This contradicts our statement that } y>0.}$$

If $(-1,-1)$:

$$\underbrace{\begin{aligned}(x-y)+(x+y)&=2x\\-1+(-1)&=2x\\-2&=2x\\-1&=x\end{aligned}}_{\text{This contradicts } x>0.}$$

<u>Conclusion</u>: In both cases, assuming that x and y are integers leads to a contradiction that $x,y>0$. Therefore, it is impossible for both x and y to be integers and satisfy $x^2-y^2=1$, with $x,y>0$. So, at least one of x or y must not be an integer.

3. **Runtime Analysis**

For each pseudocode snippet below, give the asymptotic running time in Θ notation and explain how you derived it. Assume that basic arithmetic operations $(+, -, \times, /)$ are constant time.

(a)

```
for $i = 1$ to $n$ do          (n)
    $j = 0$;
    while $j \leq i$ do
        $j = j + 2$;
    end
end
```

$\left(\frac{i}{2}\right)+1 \rightarrow \frac{n}{2}+1 = \frac{1}{2}n+1 \approx n$

$n\left(\frac{1}{2}n+1\right) = \frac{1}{2} \cdot n(n+2)$

$\frac{n(n+2)}{2} = \frac{n^2+2n}{2} = \frac{1}{2}n^2+2n$

Outer: n operations

Inner: n operations

$f(n) = \Theta(n) \cdot \Theta(n)$

$\boxed{f(n) = \Theta(n^2)}$

(b)

```
O(1)  $i = 2$;
O(?)  while $i \leq n$ do
O(1)      $i = i^2$;
      end
```

1 $i = 2^2$

2 $i = (2^2)^2 = 2^4$

3 $i = (2^{2^2})^2 = 2^8$

$2^{2^k} > n$

$2^k = \log(n)$, $k = \log_2(\log_2(n)) \rightarrow$ max number of iterations in while loop

$\boxed{f(n) = \Theta(\log(\log(n)))}$

4. ***Smallest Element with Minimum Frequency**
(Graded by Correctness)

Let A be an array of size n which contains only positive integers. Give an algorithm which takes as input such an array A and finds the SMALLEST element with MINIMUM frequency. Additionally, provide a time complexity analysis of your algorithm in terms of n.

Example

- **Input**: $n = 5, A = [2, 2, 7, 50, 4]$
- **Output**: 4
  (All values have frequency 1 except the value 2. Thus, 4 is the smallest element with minimum frequency.)

$\boxed{\text{Next Page.}}$

Algorithm:

```
1. Initialize hashmap "map"
2. Iterate through array A once:
    a. For each element a
        i. map[a]++
3. Initialize variable min_freq = n+1
    a. We start with n+1 so that this value is greater than the max frequency
4. Initialize variable result = -1
    a. Arbitrary value of -1
5. Iterate through "map" once
    a. For each pair (a, f)
        i. If f<min_freq
            1. min_freq=f; result=a;
        ii. Else if f == min_freq
            1. result = minimum(a, result)
6. Return result
```

Correctness Analysis:

Claim:

The algorithm always terminates.

Proof:

There are two loops in the algorithm. The first loop iterates through the array A, of size n, exactly once. The second loop iterates through the hashmap ‘map’, of size n, exactly once. Thus, the algorithm terminates in at most n+n=2n iterations

Claim:

The algorithm always correctly outputs the smallest element with minimum frequency from an array A.

Proof:

Suppose for the sake of contradiction the algorithm outputs an incorrect element ‘a’/ This means that either:

1. ‘a’ was incorrectly counted from A
   or
2. The map was sorted incorrectly and output ‘a’
3. or
4. Both (1) and (2)

By the algorithm, each element in A is mapped to its frequency, which is only ever incremented when said element is encountered during a one-time traversal of A. By observation, 'a' could only be miscounted if it was not incremented or was incremented extraneously. However, this contradicts the algorithm, so case (1) is impossible.

Now consider case (2). Let the truly correct element be 'c'. If the algorithm incorrectly returned 'a' instead of 'c', it must have wrongly selected 'a' during the second loop. This implies one of two conditions:

Condition A: freq(c) < freq(a)

Condition B: freq(c) == freq(a) and c < a

In Condition A, c has a lower frequency than a. As the algorithm iterates through the map, it would eventually process c. At that point, the condition f < minimum_freq would be met, and the algorithm would set result = c. Any subsequent processing of a would fail this check because its frequency is higher. Thus, the algorithm would have returned c. This is a contradiction.

In Condition B, c and a have the same minimum frequency, but c is the smaller value. When processing either a or c, the condition f == minimum_freq would be true. The algorithm then sets the result = minimum(a, result) (or minimum(c, result)). Because the logic correctly chooses the minimum of the current key and the current result, the smaller element, c, is guaranteed to be the final result. This contradicts the assumption that the algorithm returned a.

Since both possible cases for an incorrect output lead to a contradiction with the algorithm's own logic, the initial assumption must be false. Thus, the algorithm must always correctly output the smallest element with the minimum frequency from an array A.

Time Complexity Analysis:

1. Initializing the hash map is O(n) time, since the hash map will contain at most n elements. (from piazza)
2. Iterating through the array A of size n takes O(n) iterations/time.
    a. Lookup in hashmap is O(1)
    b. Update hashmap is O(1)
        i. So this would be something like:
            1. if(map[a]!=0){map[a]++;}else{map[a]=1;}
3. Initialize an int is O(1)
4. Initialize an int is O(1)
5. We loop through each pair in the map. Since there are n elements, there are n pairs. O(n)
    a. Int comparison is O(1)
        i. Int update is O(1)
        ii. Int update is O(1)
    b. Int comparison is O(1)
        i. Int update is O(1)
6. Return

So we have: O(n)+[O(n)*O(1)]+O(1)+[O(n)*O(1)] which is O(n)+O(n)+O(1)+O(n)=3O(n)+O(1), which is bounded by O(n). Thus, f(n)=O(n)

5. ***Stable Matching with Unequal Numbers and Multiple Partners (Problem 1.4 in the Textbook)**
(Graded by Correctness)

Gale and Shapley published their paper on the Stable Matching Problem in 1962; but a version of their algorithm had already been in use for ten years by the National Resident Matching Program, for the problem of assigning medical residents to hospitals.

Basically, the situation was the following. There were m hospitals, each with a certain number of available positions for hiring residents. There were n medical students graduating in a given year, each interested in joining one of the hospitals. Each hospital had a ranking of the students in order of preference, and each student had a ranking of the hospitals in order of preference. We will assume that there were more students graduating than there were slots available in the m hospitals.

The interest, naturally, was in finding a way of assigning each student to at most one hospital, in such a way that all available positions in all hospitals were filled. (Since we are assuming a surplus of students, there would be some students who do not get assigned to any hospital.)

We say that an assignment of students to hospitals is stable if neither of the following situations arises.

- First type of instability: There are students s and s' and a hospital h such that
  - s is assigned to h, and
  - s' is assigned to no hospital, and
  - h prefers s' to s.
- Second type of instability: There are students s and s', and hospitals h and h' such that
  - s is assigned to h, and
  - s' is assigned to h', and
  - h prefers s' to s, and
  - s' prefers h to h'.

So we basically have the Stable Matching Problem, except that (i) hospitals generally want more than one resident, and (ii) there is a surplus of medical students.

Show that there is always a stable assignment of students to hospitals, and give an algorithm to find one. Additionally, provide a time complexity analysis of your algorithm.

Algorithm:

```
1. M=hashmap of key (int, hospital index) value (list of students assigned to h) pairs.
2. While some hospital h has at least one open position and has not made an offer to every
   student
     a. Let s be the 1st student on h's list where h has not yet made an offer
     b. If s is unmatched
          i. Add s to M[h]
     c. If s is matched, let h' be its current match
          i. If s prefers h to h'
               1. Add s to M[h]
               2. Remove s from M[h']
          ii. Otherwise: s rejects h.
3. Return M
```

Analysis:

Claim:

The algorithm always terminates.

Proof:

Each hospital makes at most n offers, one to each student. Thus, the total number of offers is bounded by m*n. Therefore, the algorithm terminates in at most m*n iterations.

Stable-Matching Proof

We will now prove that the algorithm always outputs a stable matching. To do this, we will use two smaller proofs:

Claim:

The first case of instability never occurs.

Proof:

Suppose for the sake of contradiction there are students s and s', and hospitals h and h' such that s and h are matched, s' is unmatched, and h prefers s' to s.

By the algorithm, hospitals make offers in decreasing preferential order. By the algorithm, unmatched students always accept their first offer, and once matched, students never become unmatched, and their matching can only improve. However, there is a contradiction. If h prefers s' to s, it must have made an offer to s'. If s' rejected this offer, it must have already been matched. If s' was unmatched, it must have accepted this offer. Since h and s' are now unmatched, if they were matched, s' could only have been rematched to another hospital, not become unmatched again.

Claim:
The second case of instability never occurs.

Proof:
Suppose for the sake of contradiction there are students s and s’, and hospitals h and h’ such that s is matched with h, s’ is matched with h’, and h prefers s’ to s, and s’ prefers h to h’.

There are two cases:

1. h made an offer to s’, Since s’ is no longer matched to h, s’ must have rejected h in favor of some other hospital h’’. Since s’ partners only improve, s’ must prefer h’’>h. Since s is now matched with h’, it must then prefer h’>h’’, and thus h’>h.
2. h never made an offer to s’. Since s is matched to h, h made an offer to s. But h makes offers in decreasing preferential order, so h must prefer s to s’.

Thus, there is a contradiction, and the algorithm never outputs such a case of instability.

Conclusion: From the above proofs, we can conclude that the algorithm always terminates, never outputs the first case of instability, and never outputs the second case of instability. Thus, the algorithm always outputs a stable matching.

Time Complexity Analysis

1. Construct an array, studentpref[s][h] which contains the ranking of h for s for all students/hospitals. This takes O(m * n) time.
2. Initialize the hashmap M. The maximum number of total open positions is bounded by n, so this is bounded by O(n) time (from piazza).
3. Initialize array student of size n, where student[s] = matched hospital index. If unmatched, -1. This takes O(n) time.
4. Each hospital (m) makes at most n offers. This is O(m * n)
    a. Each offer takes:
        i. Next student on list O(1) from stack of preferences
        ii. Checking if s is matched takes O(1) time from student[s]
            1. Hashmap insertion is O(1)
        iii. Comparing student preferences from studentpref[s][h] and studentpref[s][h’] takes O(1) time.
            1. Hashmap deletion is O(1)
                a. This could be done by:
                    i. m = M[h]
                    ii. m[m.size-1] = -1
                b. Or something similar.
            2. Hashmap insertion is O(1)
    b. All operations take O(1) time, so each iteration is O(1) time.

Thus, the total time complexity is O(m*n) + O(n) + O(n) + O(m*n)*O(1), which is bounded by O(m*n). Thus, f(n) = O(m*n)

6. ***Peripatetic Shipping Lines (Problem 1.6 in the Textbook)**
(Graded by Correctness)

Peripatetic Shipping Lines, Inc., is a shipping company that owns n ships and provides service to n ports. Each of its ships has a schedule that says, for each day of the month, which of the ports it's currently visiting, or whether it's out at sea. (You can assume the "month" here has m days, for some m > n.) Each ship visits each port for exactly one day during the month. For safety reasons, PSL Inc. has the following strict requirement:

$(*)$ No two ships can be in the same port on the same day.

The company wants to perform maintenance on all the ships this month, via the following scheme. They want to truncate each ship's schedule: for each ship S_i, there will be some day when it arrives in its scheduled port and simply remains there for the rest of the month (for maintenance). This means that S_i will not visit the remaining ports on its schedule (if any) that month, but this is okay. So the truncation of S_i's schedule will simply consist of its original schedule up to a certain specified day on which it is in a port P; the remainder of the truncated schedule simply has it remain in port P.

Now the company's question to you is the following: Given the schedule for each ship, find a truncation of each so that condition $(*)$ continues to hold: no two ships are ever in the same port on the same day.

Show that such a set of truncations can always be found, and give an algorithm to find them. Additionally, provide a time complexity analysis of your algorithm.

Example. Suppose we have two ships and two ports, and the "month" has four days. Suppose the first ship's schedule is

port P_1; at sea; port P_2; at sea

and the second ship's schedule is

at sea; port P_1; at sea; port P_2

Then the (only) way to choose truncations would be to have the first ship remain in port P_2 starting on day 3, and have the second ship remain in port P_1 starting on day 2.

Hint. Try to relate this problem to the stable matching problem.

Claim:
This problem can be solved using the G-S Algorithm, where each ship has a 'preference list' which contains the order of the ports it visits. Each port has a 'preference list' of its visiting ships in reverse order (ships are ranked from latest arrival to earliest arrival).

Suppose the schedules of 2 ships which visit 2 ports in a 4 day month is as follows:

	Day 1	Day 2	Day 3	Day 4
S1	P1	S	P2	S
S2	S	P1	S	P2

Then, the preference lists would be:

S1	P1	P2
S2	P1	P2

P1	S2	S1
P2	S2	S1

A stable matching with these lists would be: (S1, P2), and (S2, P1). This schedule would be valid, as no two ships are in the same port on the same day. In fact, we see that if two ships are in the same port on the same day, there is an instability. For instance:

	Day 1	Day 2	Day 3	Day 4
S1	P1	P1	P1	P1
S2	S	P1	S	P2

This is an invalid schedule, wherein S1 and S2 are both in P1 on day 2. This comes with the pairings (S1, P1) and (S2, P2). However, this is unstable, as P1 prefers S2 to S1, and S2 prefers P1 to S2.

To generalize, we can say that every ship S has a preference list which contains the order of the ports they visit. Each port has a preference list which is the inverse order in which ships arrive. Thus, any pairing which breaks the condition that no two ships cannot be in the same port on the same day must also be unstable, since such a pairing implies that S is matched with P, and S' is matched with P', such that S prefers P' to P and P' prefers S to S'. This correctness is proved in depth below.

Algorithm:

```
1. M=empty
2. While some ship s has not scheduled maintenance and has not made an offer to each
   port
3. Let p be the first port of s' preference list
     a. If p is not matched
          i.   Match p and s
     b. Otherwise, let s' be p's current match
          i.   If p prefers s over s'
                 1. Remove (s', p)
                 2. Add (s, p)
          ii.  Otherwise, p rejects s
4. Return M
```

Claim:
No two ships are ever in the same port on the same day

Proof:
Suppose for the sake of contradiction there are ships s and s', and a port p. Suppose s' arrives at p, but s is already there for maintenance. For this to happen, s must be matched with p, and s' must be matched with some other port, p'. The conflict means that s''s visit to p must occur before its own scheduled maintenance at p'. This implies that s' prefers p over p'. This also means that s' must have arrived at p on or after the day s arrived. Since ports have a preference list in reverse order of arrival (latest is best), port p prefers s' over s.

Thus, s' prefers p over its own match p', and p prefers s' over its own match s. This is an instability, which is impossible since the Gale-Shapley algorithm guarantees a stable match. Therefore, the initial assumption must be false, and the resulting schedule is valid.

Claim:
A valid set of truncations can always be found.

Proof:
Because the problem can be reduced to finding a stable matching between two equal-sized groups (n ships and n ports). The Gale-Shapley algorithm is guaranteed to produce a stable, perfect matching in such cases, which corresponds to a valid set of truncations.

Claim:
The algorithm always terminates

Proof:
Each ship makes at most n offers, one to each port. Thus, the total number of offers is bounded by n*n. Therefore, the algorithm terminates in at most n*n iterations.

<u>Time Complexity Analysis</u>

<u>Setup:</u>

Input: Schedules of all the ships in stacks

1. Create ports preference list:
    a. To do this, we loop through each ship's schedule O(n)
        i. For each ship, we store the schedule in reverse order O(n)

This takes O(n^2) time.

1. Construct an array, portprefs[p][s] which contains the ranking of p for s for all ports/students. This takes O(n * n) time.
2. Initialize array ships of size n, where ships[s] = matched port index. O(n)
3. Initialize array ports of size n, where ports[p] = matched ship index O(n)
4. Initialize free_ships queue. O(n)
5. Each ship makes at most one offer to every port. So, at most n*n offers. O(n^2)
    a. Each offer takes:
        i. Next port on list O(1) from stack of preferences
        ii. Checking if p is matched takes O(1) time from ports[p]
            1. Adding a pair to M takes O(1) time
        iii. Comparing port preferences from portprefs[p][s] and portprefs[p][s'] takes O(1) time.
            1. Pair deletion is O(1)
            2. Pair insertion is O(1)
    b. All operations take O(1) time, so each iteration is O(1) time.

Thus, the total time complexity is O(n^2)+ O(n^2)+O(n)+O(n)+O(n)+O(n^2)*O(1) which is 3O(n^2)+2O(n), which is bounded by O(n^2). Thus, f(n) = O(n^2).

7. **Truthfulness in the Stable Matching Problem (Problem 1.8 in the Textbook)**

   For this problem, we will explore the issue of truthfulness in the Stable Matching Problem and specifically in the Gale-Shapley algorithm. The basic question is: Can a man or a woman end up better off by lying about his or her preferences? More concretely, we suppose each participant has a true preference order. Now consider a woman w. Suppose w prefers man m to m', but both m and m' are low on her list of preferences. Can it be the case that by switching the order of m and m' on her list of preferences (i.e., by falsely claiming that she prefers m' to m) and running the algorithm with this false preference list, w will end up with a man m" that she truly prefers to both m and m'? (We can ask the same question for men, but will focus on the case of women for purposes of this question.)

   Resolve this question by doing one of the following two things:

   (i.) Give a proof that, for any set of preference lists, switching the order of a pair on the list cannot improve a woman's partner in the Gale-Shapley algorithm; or

   (ii.) Give an example of a set of preference lists for which there is a switch that would improve the partner of a woman who switched preferences.

Preference Lists:

Men

1	B>A>C
2	A>B>C
3	A>B>C

Women

A	1>2>3
B	2>1>3
C	1>2>3

Let us presume these are all truthful preferences and run the G-S algorithm.

1. Round 1:
    a. Man 1 proposes to B → (1,B) are engaged.
    b. Man 2 proposes to A → (2,A) are engaged.
    c. Man 3 proposes to A. A checks her true list (1 > 2 > 3) and rejects 3.
2. Round 2:
    a. Man 3 (having been rejected) proposes to B. B checks her list (2 > 1 > 3) and rejects 3.
3. Round 3:
    a. Man 3 proposes to C → (3,C) are engaged.

The final stable matching is: (1, B), (2, A), and (3, C).

Now, let's say Woman A lies, and instead of her truthful list (1>2>3), she gives 1>3>2

1. Round 1:
    a. Man 1 proposes to B → (1,B) are engaged.
    b. Man 2 proposes to A → (2,A) are engaged.
    c. Man 3 proposes to A. A checks her false list (1 > 3 > 2), rejects 2 → (3,A) are engaged.
2. Round 2:
    a. Man 2 (having been rejected) proposes to B. B checks her list (2 > 1 > 3), rejects 1 → (2,B) are engaged.
3. Round 3:
    a. Man 1 (having been rejected) proposes to A. A checks her false list (1 > 3 > 2), rejects 3 → (1,A) are engaged.
4. Round 4:
    a. Man 3 (having been rejected) proposes to B. B checks her list (2 > 1 > 3) and rejects 3.
5. Round 5:
    a. Man 3 proposes to C → (3,C) are engaged.

The final stable matching is: (1, A), (2, B), and (3, C).

By submitting the false list, Woman A was matched with Man 1. This is a better outcome for her, as her true preference was 1 > 2 > 3, and telling the truth would have resulted in her being matched with Man 2.