INFO 6205 – Program Structure and Algorithms Assignment 1

Student Name: Vidhi Bharat Patel
Professor: Nik Bear Brown

Q1 (5 Points) Arrange the following functions in increasing order of growth:

- log(55n)
- $55^n + 11^n$
- $0.99^n + 1$
- n / log(n)
- $\log(n) + n^{0.5}$
- log(log(n))
- $\log(n^3)$
- $n!\ e^n$
- √n
- 5^n

Solution:

1. $0.99^n + 1$
2. log(log(n))
3. log(55n)
4. $\log(n^3)$
5. √n
6. $\log(n) + n^{0.5}$
7. n/ log(n)
8. 5^n
9. $55^n + 11^n$
10. $n!\ e^n$

Q2 (5 Points) Give a brief definition for the following:

i. Algorithm
ii. Computational tractability
iii. Stable Matching
iv. Big-Oh notation
v. Asymptotic order of growth

Solution:

i. Algorithm

An algorithm is a detailed process for completing a given task or problem. It is a collection of guidelines that, when followed, will carry out a certain activity or address a particular issue. Depending on the purpose they are intended to do, algorithms can be simple or complicated and can be described in any language. Numerous computer science disciplines, such as data structures, programming, artificial intelligence, and cryptography, require algorithms. They may also be applied in other disciplines including logic, operations research, and mathematics.

ii. Computational tractability

A problem is said to be computationally tractable when it can be solved by computer algorithms that execute in polynomial time, i.e., when the time or number of steps required to reach the answer is a polynomial function of n.

iii. Stable Matching

The stable marriage problem asks to pair (match) the men and women in such a way that no two people prefer each other over their matched partners. It is basically the problem of finding a stable matching between two equally sized sets of elements given an ordering of preferences for each element.

iv. Big-Oh notation

Big O Notation is a tool used to describe the time complexity of algorithms. It calculates the time taken to run an algorithm as the input grows. In other words, it calculates the worst-case time complexity of an algorithm.

v. Asymptotic order of growth

Asymptotic order of growth, also known as "big-O" notation, is a way to describe the performance or complexity of an algorithm as the input size (typically denoted as "n") becomes arbitrarily large. It gives an upper bound on the growth of the algorithm's running time or space usage. The Asymptotic order of growth is represented by a mathematical function that describes how fast the running time of an algorithm increases as the input size increases. The most common examples are O(1) (constant time), O(log n) (logarithmic time), O(n) (linear time), O(n log n) (linear logarithmic time), O(n^2) (quadratic time), and O(2^n) (exponential time).

Q3 (10 Points)

This exercise shows that stable matchings need not exist if there are not "two sides." Consider the following "roommate" problem. There are four people, Pat, Chris, Dana, and Leslie. They must pair off (each pair will share a two-bed suite). Each has preferences over which of the others they would prefer to have as a roommate. The preferences are:
Leslie: Pat > Chris >_ Dana
Chris: Leslie _> Pat _> Dana
Pat: Chris > Leslie> _ Dana
Dana: Chris> _ Leslie> _ Pat
Show that no stable matching exists. (That is, no matter who you put together, they will always be two potential roommates who are not matched but prefer each other to their current roommate.) give examples to justify your solution.

Solution:
According to the Stable matching or the Gale-Shapley algorithm, we can carry out the roommate problem as followed: -

Leslie	**Chris**	**Pat**	**Dana**
Pat	Leslie	Chris	Chris
Chris	Pat	Leslie	Leslie
Dana	Dana	Dana	Pat

According to the table above, there will be no stable matching since either Leslie, Chris, or Pat are to be paired with Dana but when paired with Dana, one of the other two paired will think that Dana's partner is the better roommate. For example, if Dana and Chris are paired together and Leslie and Pat are paired together then Pat will prefer to stay with Chris since it's the first choice for Pat and second for Chris. This would be same for anybody who is paired with Dana.

Q4 (10 Points)
Prove the following:
In the Gale-Shapley algorithm, run with n men and n women, what is the maximum number of times any woman can be proposed to?
Solution: n2 – n + 2

Solution: If Gale-Shapley algorithm is run with n men and n women, then it will be terminated with n^2 – n + 2 rounds.
This is because the algorithm has the input pair of n x n matrices which is the input size of theta(n^2), it basically represents the linear-time algorithm. Here, a man will get rejected at the most n-1 times, every stage there is at least one rejection, every woman will receive a proposal before it ends. It can be proved as followed: -

1) Initial proposal -> 1 stage
2) If every man gets rejected exactly n – 1 time, then it will be n(n-1) stages
3) Final proposal -> 1 stage
4) Therefore, total number of stages (worst case scenario) will be 1 + n(n-1) + 1 = n^2 – n + 2

Q5 (10 Points)
Consider there are x number of gaming companies, each with a certain number of available positions for hiring students. There were n Computer Science and Game Design students graduating in a given year at Northeastern, each interested in joining one of the companies. Each company has a ranking of students based on their portfolio and their GPA, and each student also has a ranking of company based on the work and the pay, benefits and location. We will assume that there are more students graduating than there are available positions in m companies.

Our goal is to find a way of assigning each student to at most one company, in a way that all the available positions in a particular company are filled. (There will be some students who did not get a company as we have assumed the number of students is greater than the number of companies).
The assignment of student to a particular company is stable if neither of the situations arises:
1. There are students s1 and s2 and company c, so that
- s1 is assigned to c, and

- s2 is assigned to no company, and
- c prefers s2 to s1 based on his academics an projects.

2. There are students s1 and s2 and companies c1 and c2, so that

- s1 is assigned to c1, and
- s2 is assigned to c2, and
- c1 prefers s2 to s1, and
- s2 prefers c1 to c2

So, we basically have a stable matching problem, except that
(a) Companies generally want more than one hiring student, and
(b) There is a surplus of Computer Science and Game Design graduates.
Either:

a. Show that there is always a stable assignment of students to companies, and devise an algorithm to find one. or

b. Show that an algorithm that always generates a stable assignment cannot exist.

Solution:
The algorithm is quite similar to the fundamental Gale-Shapley algorithm from the textbook. A student is either employed by the company at any given time or is not. Either a company is hiring or it is already full. This is how the algorithm works: While some companies (c1) have openings, (c1) offers the following student (s1) a position based on their academic performance (GPA) and projects.
If s1 is available, then s1 accepts the offer; else (s1 is already committed to the company, say c(k))
If s1 prefers c(k) to c1 then s1 remains committed to c(k);
otherwise s1 becomes committed to c1.
As a result, c(k) has one more open position while c1 has one less.
As each company only offers a position to a student once, and as some companies offer positions to certain students during each iteration, the algorithm ends in O(mn) steps.

Q6 (10 Points)

A (5 points) Decide whether you think the following statement is true or false. If it is true, give a short explanation. If it is false, give a counterexample.
True or false? In every instance of the Stable Matching Problem, there is a stable matching containing a pair (m, w) such that m is ranked first on the preference list of w and w is ranked first on the preference list of m.

Solution: The given statement is False. This can be proved with taking an example of 2 men and 2 women. The preferences of the men and women is given as followed:
m1: w1 > w2
m2: w2 > w1
w1: m2 > m1
w2: m1 > m2
This proves that there are no pairs at all where each ranks the other first, so clearly no such pair (m, w) can show up in a stable matching.

B (5 points) Suppose we are given an instance of the stable matching problem for which there is a man m who is the first choice of all women. Prove or give a counterexample: In any stable matching, m must be paired with his first choice.

Solution:
This statement is false. There can be a stable matching in which m is not paired with his first choice. A counterexample to this statement can be constructed as follows:
Let there be three men m1, m2, m3 and three women w1, w2, w3.
Suppose, m1 is the first choice of all women, w1 is the first choice of m1 and m2, w2 is the first choice of m2 and m3, and w3 is the first choice of m1 and m3.
Then, the following matching is stable: (m1, w2), (m2, w1), (m3, w3)
In this case, m1 is not paired with his first choice w1, but the matching is still stable as there is no blocking pair (m, w') such that m prefers w' to his current match and w' prefers m to her current match.

Q7 (10 Points)
The basic Gale-Shapley algorithm assumes all men and women have fully ranked and complete preference lists. Consider a version of the Gale-Shapley algorithm both the men and the women could be *indifferent* to some choices. That is, there could be ties and multiple members of a preference list could have the same rank. The size of the preference lists is the same, but we allow for ties in the ranking. We say that w *prefers* m to m' if m is ranked higher on the preference list (i.e. m is not tied m'). The converse holds for men.
With indifference in the rankings, we could consider at least two forms of instability:
A. A *strong instability* in a perfect matching S consists of a man *m* and a woman *w*, such that each of m and w *prefers* the other to their partner in S. Does there always exist a perfect matching with no *strong instability?* Give an algorithm guaranteed to find a perfect matching with no *strong instability* or a counterexample.
B. A *weak instability* in a perfect matching S consists of a man *m* and a woman *w*, such that their partners in S are w' and m' respectively and one of the following holds:
  i. -m prefers w to w', and w either prefers m to m' or is indifferent
  ii. -w prefers m to m', and m either prefers w to w' or is indifferent
Give an algorithm guaranteed to find a perfect matching with no *weak instability* or a counter-example.

Solution:
A. In the Stable matching problem with indifferences, there always exists a solution with no strong instability. In order to find such solution, then first we have to assign an arbitrary ordering to each indifference in the given preference lists. For example, if a woman w is indifferent between m and m', then we could randomly choose the ordering that w prefers to m and m'. After that, use the resulting free of indifference preferences and run the Gale-Shapley algorithm on the problem. If there is any strong instability in the result with respect to the original preferences would have to correspond to an instability in the result of the Gale-Shapley algorithm. Because it is impossible, the solution from the Gale-Shapley algorithm must constitute a strong instability-free solution to the problem with indifferences.
B. There could be a case where m and m' both men prefer w to w' and both the women w and w' are indifferent to m and m'. The two possible perfect matchings will be (m, w), (m', w'), and (m, w'), (m',

w). In the first case, (m', w) there constitutes a weak instability; in the second case, (m, w) constitutes a weak instability.

Q8 (10 Points)

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:

(a) 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

(b) 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.

Solution:

If we take 3 men and 3 women. Then the preferences are given as followed: -

m1: w1 > w2 > w3

m2: w3 > w1 > w2

m3: w1 > w2 > w3

w1: m3 > m2 > m1

w2: m1 > m3 > m2

w3: m2 > m3 > m1 (in reality m2 > m1 > m3)

With the above information, the men will propose to their first preference, m1 and m3 both will propose to w1 but w1 will chose m3 because she has that as her first preference. Also, m2 will propose to w3, which she will accept because it is her first preference. And, w2 will not be engaged because she had no proposals. In the next stage, m1 will propose to w2 which is his second preference. All the pairings will then be completed. It will be as followed: -

(m1, w2)

(m2, w3)

(m3, w1)

Even if w3 has a preference of m3 over m1, but in reality, she preferred m1 over m3, it did not affect her pairing with her best preference m2 who was always her first preference. Hence, this is the switch that improved the partner of a woman who switched preferences.

Q9 (5 Points)

One algorithm requires n $\log_2(n)$ seconds and another algorithm requires √n seconds. Which one is asymptotically faster? What is the cross-over value of n? (The value at which the curves intersect?)

Solution:

The algorithm that requires n log2(n) seconds is asymptotically faster. The cross-over value of n can be calculated by equating the two expressions and solving for n:

n log2(n) = √n
log2(n) = ½ log2(√n) [taking log on both sides and using the change of base formula]
it simplifies to 2 log2(n) = log2(n^(1/2)),
Then dividing both sides by log2(n) and simplifying it gives 2 = n^(1/2), which gives the answer n = 4.
Hence, the cross-over value is n = 4, where the two algorithms have the same performance.

Q10 (25 Points): Coding Problem

The World Cup is changing its playoff format using the Gale-Shapley matching algorithm. The eight best teams from groups A, B & C, called Super Group 1, will be matched against the eight best teams from groups D, E & F, called Super Group 2, using the Gale-Shapley matching algorithm. Further social media will be used to ask fans, media, players and coaches to create a ranking of which teams they would most like to see play their favorite team.
Solutions: https://colab.research.google.com/drive/1kHY19JKMbFrX2UwUBg8yILoll8iC522z?usp=sharing

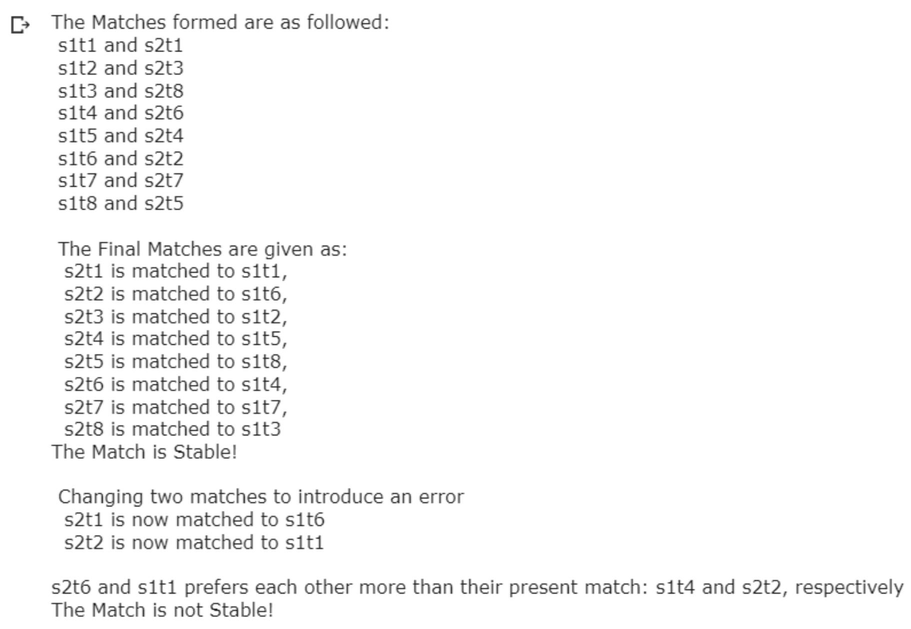
```
import random
import time
import copy
def check(matched, supergrp1, supergrp2):
  matchinverse = dict((v,k) for k,v in matched.items())

  for grp2, grp1 in matched.items():
    grp2likes = supergrp2prefers[grp2]
    grp2likesbetter = grp2likes[:grp2likes.index(grp1)]
    grp1likes = supergrp1prefers[grp1]
    grp1likesbetter = grp1likes[:grp1likes.index(grp2)]

    for g1 in grp2likesbetter:
      grp12 = matchinverse[g1]
      grp1likes = supergrp1prefers[g1]

      if grp1likes.index(grp12) > grp1likes.index(grp2):
        print("%s and %s prefers each other more than their present match: %s and %s, respectively"
           % (grp2, g1, grp1, grp12))
        return False

    for g2 in grp1likesbetter:
      grp21 = matched[g2]
      grp2likes = supergrp2prefers[g2]
      if grp2likes.index(grp21) > grp2likes.index(grp1):
        print("%s and %s prefers each other more than their present match: %s and %s, respectively"
           % (grp1, g2, grp2, grp21))
        return False
  return True
def teamfix(supergrp1, supergrp2):
  grp1free = supergrp1[:]
```

```
    matched  = {}
    supergrp1prefers2 = copy.deepcopy(supergrp1prefers)
    supergrp2prefers2 = copy.deepcopy(supergrp2prefers)

    while grp1free:
        g1 = grp1free.pop(0)
        supergrp1list = supergrp1prefers2[g1]
        g2 = supergrp1list.pop(0)
        matchs = matched.get(g2)

        if not matchs:
            # grp2's free
            matched[g2] = g1
            print(" %s and %s" % (g1, g2))
        else:
            # The boy proposes to a matched girl!
            supergrp2list = supergrp2prefers2[g2]

            if supergrp2list.index(matchs) > supergrp2list.index(g1):
                # grp2 prefers new g1
                matched[g2] = g1
                print("  %s dumped %s for %s" % (g2, matchs, g1))

                if supergrp1prefers2[matchs]:
                    # Eg has more to try
                    grp1free.append(matchs)
            else:
                # grp2 is faithful to old match
                if supergrp1list:
                    grp1free.append(g1)
    return matched
supergrp1prefers = {
 's1t1': ['s2t1', 's2t5', 's2t3', 's2t4', 's2t6', 's2t2', 's2t8', 's2t7'],
 's1t2': ['s2t3', 's2t8', 's2t1', 's2t4', 's2t5', 's2t6', 's2t2', 's2t7'],
 's1t3': ['s2t8', 's2t5', 's2t1', 's2t4', 's2t2', 's2t6', 's2t7', 's2t3'],
 's1t4': ['s2t6', 's2t4', 's2t7', 's2t8', 's2t5', 's2t2', 's2t3', 's2t1'],
 's1t5': ['s2t4', 's2t2', 's2t3', 's2t6', 's2t5', 's2t1', 's2t8', 's2t7'],
 's1t6': ['s2t2', 's2t1', 's2t4', 's2t7', 's2t5', 's2t3', 's2t8', 's2t6'],
 's1t7': ['s2t7', 's2t5', 's2t2', 's2t3', 's2t1', 's2t4', 's2t8', 's2t6'],
 's1t8': ['s2t1', 's2t5', 's2t8', 's2t6', 's2t3', 's2t2', 's2t7', 's2t4']}

supergrp2prefers = {
 's2t1': ['s1t2', 's1t6', 's1t7', 's1t1', 's1t4', 's1t5', 's1t3', 's1t8'],
 's2t2': ['s1t2', 's1t1', 's1t3', 's1t6', 's1t7', 's1t4', 's1t5', 's1t8'],
 's2t3': ['s1t6', 's1t2', 's1t5', 's1t7', 's1t8', 's1t3', 's1t1', 's1t4'],
 's2t4': ['s1t6', 's1t3', 's1t1', 's1t8', 's1t7', 's1t4', 's1t2', 's1t5'],
 's2t5': ['s1t8', 's1t6', 's1t4', 's1t1', 's1t7', 's1t3', 's1t5', 's1t2'],
 's2t6': ['s1t2', 's1t1', 's1t5', 's1t4', 's1t6', 's1t7', 's1t3', 's1t8'],
 's2t7': ['s1t7', 's1t8', 's1t6', 's1t2', 's1t1', 's1t3', 's1t5', 's1t4'],
 's2t8': ['s1t7', 's1t2', 's1t1', 's1t4', 's1t8', 's1t5', 's1t3', 's1t6']}
```

A. Find a Gale-Shapley implementation in python on Github and modify it so that the eight Super Group 1 teams will be matched against the eight Super Group 2 teams. You can make up the preference lists for each team. Make sure you cite any code you use or state that you wrote it from scratch if you did.

Solution:

```
 supergrp1 = sorted(supergrp1prefers.keys())
supergrp2 = sorted(supergrp2prefers.keys())

print('The Matches formed are as followed:')
matched = teamfix(supergrp1, supergrp2)

print('\n The Final Matches are given as:')
print('  ' + ',\n  '.join('%s is matched to %s' % matches
                          for matches in sorted(matched.items())))

print('The Match is Stable!'
      if check(matched, supergrp1, supergrp2)
      else 'The Match is not Stable!')

print('\n Changing two matches to introduce an error')
matched[supergrp2[0]], matched[supergrp2[1]] = matched[supergrp2[1]], matched[supergrp2[0]]
for g2 in supergrp2[:2]:
    print('  %s is now matched to %s' % (g2, matched[g2]))
print()
print('The Match is Stable!'
      if check(matched, supergrp1, supergrp2)
      else 'The Match is not Stable!')
```

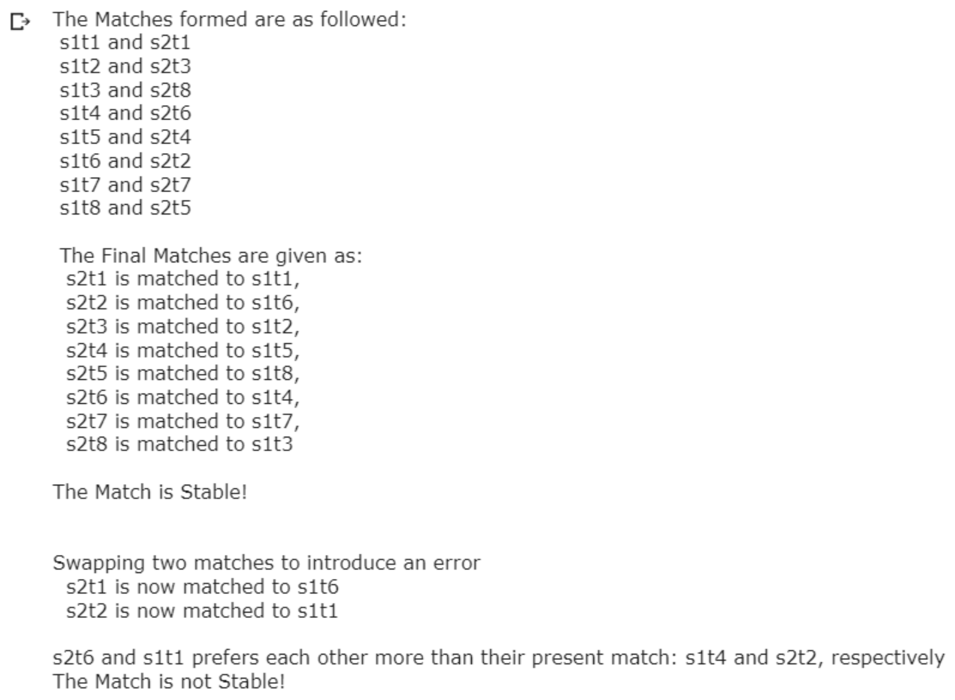


```
The Matches formed are as followed:
 s1t1 and s2t1
 s1t2 and s2t3
 s1t3 and s2t8
 s1t4 and s2t6
 s1t5 and s2t4
 s1t6 and s2t2
 s1t7 and s2t7
 s1t8 and s2t5

 The Final Matches are given as:
  s2t1 is matched to s1t1,
  s2t2 is matched to s1t6,
  s2t3 is matched to s1t2,
  s2t4 is matched to s1t5,
  s2t5 is matched to s1t8,
  s2t6 is matched to s1t4,
  s2t7 is matched to s1t7,
  s2t8 is matched to s1t3
The Match is Stable!

 Changing two matches to introduce an error
  s2t1 is now matched to s1t6
  s2t2 is now matched to s1t1

s2t6 and s1t1 prefers each other more than their present match: s1t4 and s2t2, respectively
The Match is not Stable!
```

B. Use a loop to shuffle the preference lists for each team 1000 times. Calculate the percentage of stable playoff matches. See the function random.shuffle(x[, random]) https://docs.python.org/2/library/random.html

Solution:

```
sprgrp1tmplist = list(supergrp1prefers.values())
sprgrp2tmplist = list(supergrp2prefers.values())

for i in range(1000):
   random.shuffle(sprgrp1tmplist)
   random.shuffle(sprgrp2tmplist)

supergrp1newprefers = dict(zip(supergrp1prefers, sprgrp1tmplist))
supergrp2newprefers = dict(zip(supergrp2prefers, sprgrp2tmplist))

supergrp1 = sorted(supergrp1newprefers.keys())
supergrp2 = sorted(supergrp2newprefers.keys())

print('The Matches formed are as followed:')
matched = teamfix(supergrp1, supergrp2)

print('\n The Final Matches are given as:')
print('  ' + ',\n  '.join('%s is matched to %s' % matches
                   for matches in sorted(matched.items())))

print()
print('The Match is Stable!'
     if check(matched, supergrp1, supergrp2)
     else 'The Match is not Stable!')

print('\n\nSwapping two matches to introduce an error')
matched[supergrp2[0]], matched[supergrp2[1]] = matched[supergrp2[1]], matched[supergrp2[0]]
for g2 in supergrp2[:2]:
   print('  %s is now matched to %s' % (g2, matched[g2]))
print()
print('The Match is Stable!'
     if check(matched, supergrp1, supergrp2)
     else 'The Match is not Stable!')
```



```
The Matches formed are as followed:
 s1t1 and s2t1
 s1t2 and s2t3
 s1t3 and s2t8
 s1t4 and s2t6
 s1t5 and s2t4
 s1t6 and s2t2
 s1t7 and s2t7
 s1t8 and s2t5

 The Final Matches are given as:
  s2t1 is matched to s1t1,
  s2t2 is matched to s1t6,
  s2t3 is matched to s1t2,
  s2t4 is matched to s1t5,
  s2t5 is matched to s1t8,
  s2t6 is matched to s1t4,
  s2t7 is matched to s1t7,
  s2t8 is matched to s1t3

The Match is Stable!


Swapping two matches to introduce an error
  s2t1 is now matched to s1t6
  s2t2 is now matched to s1t1

s2t6 and s1t1 prefers each other more than their present match: s1t4 and s2t2, respectively
The Match is not Stable!
```

C. Randomly assume certain teams win and lose each round and eliminate the losers from the preference lists for each team. Can the Gale-Shapley matching algorithm be applied over and over in each round (16 teams, 8 teams, 4 teams, 2 teams) to create stable matches? You can answer this with code or rhetoric.

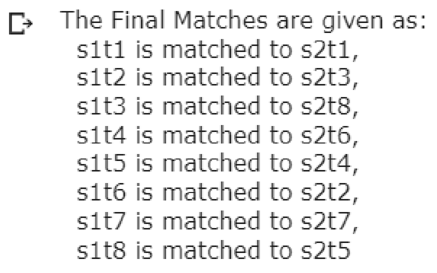
Solution:

Yes, the Gale-Shapley algorithm can be applied over and over in each round to create stable matches. The algorithm can be applied to the remaining teams after each round, using the updated preferences lists of the teams that are still in the competition. The process can be repeated until a stable matching is reached for the final two teams. However, there must be some inclusion of the risk of asymmetric team elimination. In other words, if we take away 3 teams from the east, we must also take away 3 teams from the west, or else the lists will have different numbers. Even while some teams won't have assignments and we'd need to modify the algorithms to utilize it in a bracket-style playoff system, eliminating teams asymmetrically would still produce consistent matching.

D. Now combine the lists so that any team can be matched against any other irrespective of conference. Can the Gale-Shapley matching algorithm still create stable matches? (With just one list matching against itself?) You can answer this with code or rhetoric.

Solution:

```
def team_matches(supergrp1, supergrp2):
  supergrp1 = list(supergrp1prefers.keys())
  supergrp2 = list(supergrp2prefers.keys())

  sg1_matches = {}
  sg2_matches = {}
  free_sg1 = supergrp1[:]

  while free_sg1:
    sg1 = free_sg1.pop(0)
    sg1_preferences = supergrp1prefers[sg1]

    for sg2 in sg1_preferences:
      if sg2 not in sg2_matches:
        sg1_matches[sg1] = sg2
        sg2_matches[sg2] = sg1

        break

      else:
        current_match = sg2_matches[sg2]
        sg2_preferences = supergrp2prefers[sg2]

        if sg2_preferences.index(sg1) < sg2_preferences.index(current_match):
          sg1_matches[sg1] = sg2
          sg2_matches[sg2] = sg1
          free_sg1.append(current_match)
          break
  return sg1_matches
supergrp1 = sorted(supergrp1prefers.keys())
supergrp2 = sorted(supergrp2prefers.keys())

supergrp1 = supergrp1 + supergrp2
supergrp1 = supergrp2

matched = team_matches(supergrp1, supergrp2)

print('The Final Matches are given as:')
print('  ' + ',\n  '.join('%s is matched to %s' % matches
            for matches in sorted(matched.items()))))
```



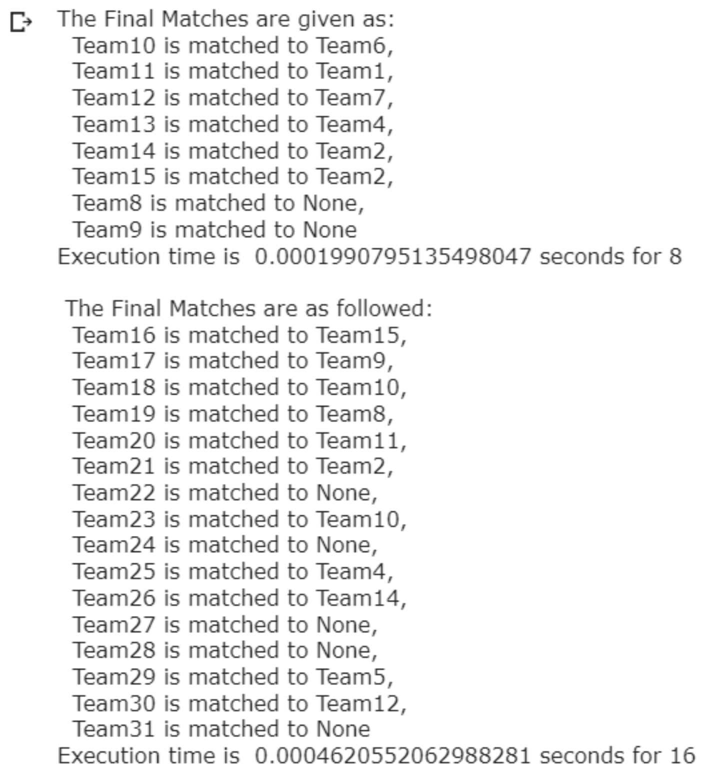
```
The Final Matches are given as:
 s1t1 is matched to s2t1,
 s1t2 is matched to s2t3,
 s1t3 is matched to s2t8,
 s1t4 is matched to s2t6,
 s1t5 is matched to s2t4,
 s1t6 is matched to s2t2,
 s1t7 is matched to s2t7,
 s1t8 is matched to s2t5
```

E. Double the size of the lists in *problem A* several times (you can make up team names like team1, team2, etc.) and measure the amount of time it takes to create stable matches. How fast does the execution time grow in relation to the size of the lists?

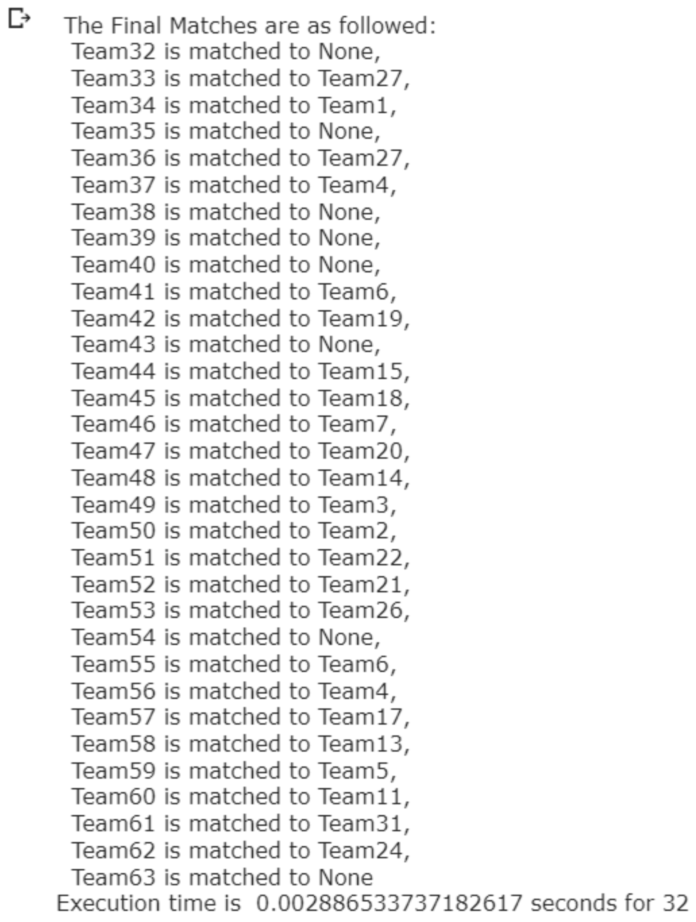
Solution:

```
def match_teams(team_group_1, team_group_2):
  team_group_1_preferences = {team: random.sample(team_group_2, len(team_group_2)) for team in
team_group_1}
  team_group_2_preferences = {team: random.sample(team_group_1, len(team_group_1)) for team in
team_group_2}
  team_group_1_proposals = {team: 0 for team in team_group_1}
  team_group_2_matches = {team: None for team in team_group_2}
  free_team_group_1 = team_group_1.copy()

  while free_team_group_1:
    group_1_team = free_team_group_1.pop()
    group_2_team =
team_group_1_preferences[group_1_team][team_group_1_proposals[group_1_team]]
    team_group_1_proposals[group_1_team] += 1
    if team_group_2_matches[group_2_team] is None:
      team_group_2_matches[group_2_team] = group_1_team
    else:
      group_2_preferred_team =
team_group_2_preferences[group_2_team].index(team_group_2_matches[group_2_team])
      group_1_preferred_team = team_group_2_preferences[group_2_team].index(group_1_team)
      if group_1_preferred_team < group_2_preferred_team:
        team_group_2_matches[group_2_team] = group_1_team
        free_team_group_1.append(team_group_2_matches[group_2_team])
  return team_group_2_matches
team_group_1 = ["Team" + str(i) for i in range(8)]
team_group_2 = ["Team" + str(i) for i in range(8, 16)]

start = time.time()
matchings = match_teams(team_group_1, team_group_2)
end = time.time()
```

```
print('The Final Matches are given as:')
print('  ' + ',\n  '.join('%s is matched to %s' % matches
                    for matches in sorted(matchings.items())))
print("Execution time is ", end - start, "seconds for 8")

team_group_1 = ["Team" + str(i) for i in range(16)]
team_group_2 = ["Team" + str(i) for i in range(16, 32)]

start = time.time()
matchings = match_teams(team_group_1, team_group_2)
end = time.time()
print('\n The Final Matches are as followed:')
print('  ' + ',\n  '.join('%s is matched to %s' % matches
                    for matches in sorted(matchings.items())))
print("Execution time is ", end - start, "seconds for 16")

team_group_1 = ["Team" + str(i) for i in range(32)]
team_group_2 = ["Team" + str(i) for i in range(32, 64)]

start = time.time()
matchings = match_teams(team_group_1, team_group_2)
end = time.time()
print('\n The Final Matches are as followed:')
print('  ' + ',\n  '.join('%s is matched to %s' % matches
                    for matches in sorted(matchings.items())))
print("Execution time is ", end - start, "seconds for 32")
```



```
The Final Matches are given as:
  Team10 is matched to Team6,
  Team11 is matched to Team1,
  Team12 is matched to Team7,
  Team13 is matched to Team4,
  Team14 is matched to Team2,
  Team15 is matched to Team2,
  Team8 is matched to None,
  Team9 is matched to None
Execution time is  0.0001990795135498047 seconds for 8

 The Final Matches are as followed:
  Team16 is matched to Team15,
  Team17 is matched to Team9,
  Team18 is matched to Team10,
  Team19 is matched to Team8,
  Team20 is matched to Team11,
  Team21 is matched to Team2,
  Team22 is matched to None,
  Team23 is matched to Team10,
  Team24 is matched to None,
  Team25 is matched to Team4,
  Team26 is matched to Team14,
  Team27 is matched to None,
  Team28 is matched to None,
  Team29 is matched to Team5,
  Team30 is matched to Team12,
  Team31 is matched to None
Execution time is  0.0004620552062988281 seconds for 16
```



```
 The Final Matches are as followed:
  Team32 is matched to None,
  Team33 is matched to Team27,
  Team34 is matched to Team1,
  Team35 is matched to None,
  Team36 is matched to Team27,
  Team37 is matched to Team4,
  Team38 is matched to None,
  Team39 is matched to None,
  Team40 is matched to None,
  Team41 is matched to Team6,
  Team42 is matched to Team19,
  Team43 is matched to None,
  Team44 is matched to Team15,
  Team45 is matched to Team18,
  Team46 is matched to Team7,
  Team47 is matched to Team20,
  Team48 is matched to Team14,
  Team49 is matched to Team3,
  Team50 is matched to Team2,
  Team51 is matched to Team22,
  Team52 is matched to Team21,
  Team53 is matched to Team26,
  Team54 is matched to None,
  Team55 is matched to Team6,
  Team56 is matched to Team4,
  Team57 is matched to Team17,
  Team58 is matched to Team13,
  Team59 is matched to Team5,
  Team60 is matched to Team11,
  Team61 is matched to Team31,
  Team62 is matched to Team24,
  Team63 is matched to None
Execution time is  0.002886533737182617 seconds for 32
```