INFO 6205 – Program Structure and Algorithms
Assignment 1

Student Name: Shabina Singh
Professor: Nik Bear
Brown

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

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

Answer -

- 0.99n + 1 - Θ(n)
- log(log n) - Θ(log(log(n)))
- log(55n) - Θ(log(n))
- log(n^3) - Θ(log(n))
- √n - Θ(√n)
- log(n) + n^0.5 - Θ(√n)
- n / log(n) - Θ(n / log(n))
- 5^n - Θ(n)
- 55n + 11n - Θ(n)
- n! e^n - Θ(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*

Answer -

Algorithm: An algorithm is a step-by-step procedure for solving a problem or achieving a desired output. It is a set of rules that are followed in a specific order to perform a specific task.

Computational Tractability: Computational tractability refers to the feasibility of solving a problem in a reasonable amount of time. A problem is considered computationally tractable if it can be solved in a time proportional to some polynomial function of the size of the input. This means it is possible to obtain the largest possible running time for a given input of size N.

Stable Matching: Stable matching refers to a pairing of elements from two equally sized sets of elements with an order of preference for each element, in such a way that there is no pair that would both prefer each other over their current partner.

Gale Shapley's Algorithm guarantees a stable match in O(n^2) time.

Big-Oh Notation: Big-Oh notation is a mathematical notation to describe the limiting behavior or upper bound of the growth rate of an asymptotic function.

For example,

Binary Search can be represented by O(log n)

Asymptotic Order of Growth: Asymptotic order of growth refers to the growth of a function that is tightly bound by both upper and lower bound functions on the running time with the size of its input increases.

For example,

T(n) = 32n2 + 17n + 32
T(n) is O(n2), O(n3), Ω(n2), Ω(n), and Θ(n2)
T(n) is not O(n), Ω(n3), Θ(n), or Θ(n3)

Reference -
https://www.cs.princeton.edu/~wayne/kleinberg-tardos/pearson/02AlgorithmAnalysis-2x2.pdf

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.

Answer -

A stable roommate problem is slightly different from the stable matching problem as there is only one set. In this instance, since Dana is least preferred by all other roommates, she will end up without a match -

Iteration 1:

Proposer	Preference 1	Preference 2	Preference 3
Leslie	Pat	Chris	Dana
Chris	Leslie	Pat	Dana
Pat	Chris	Leslie	Dana
Dana	Chris	Leslie	Pat

- Leslie proposes to Pat, and Pat accepts.
- Chris proposes to Leslie, and Leslie accepts.
- Pat proposes to Chris, and Chris accepts.
- Dana proposes to Chris, but Chris declines as Pat is his higher preference.
- Dana proposes to Leslie, but Leslie declines the proposal as she has accepted Chris who is her higher preference.
- Dana proposes to Pat, but Pat also declines as Pat has already been proposed to by Leslie who is preferred over Dana.
- Hence, Dana has neither received a proposal nor has she got his proposals accepted.

Since Dana did not end up with a match, there is no stable matching existing in this scenario.

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: $n2 - n + 2$

Answer -

The proof of this statement is as follows:

Taking both list sizes = n

Maximum no of proposals a man can make and get rejected = (n-1) (this excludes the day his proposal gets accepted)
This number is also equal to the maximum number of rounds that can occur within the group since none of the men can get rejected more than n-1 times

Maximum no of rejections a woman can make per round assuming she is proposed to by all men = n-1
This is because if a woman rejects all men, she will not be able to find a match, thus ending with the algorithm failing.

Maximum no of rejections a woman can make and achieve a match - (n-1)(n-1) = n^2 - 2n + 1

She still needs one more proposal to achieve a match in the last round - (n^2 - 2n + 1) + 1 = n^2 - 2n + 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.

Answer -

There is always a stable assignment of students to companies in this case and this can be proven using the Gale-Shapley algorithm similar to Resident - Hospital Problem.

Suppose there are k companies -

```
for each company in the list of companies
        while company c1 has at least one position
                c1 offers student s1 found by ranking
                        if s1 is free
                                s1 accepts
                        else ( s1 is already hired by company c[y])
                                if s1 prefers c(k) to c1
                                        then s1 stays with c[y]
                                else s1 accepts c1's offer
                                        decrement c[y] by 1
                                        increment c1 by 1
```

The algorithm terminates because each student will only accept one company, and each company will only make as many proposals as there are available positions.

Time Complexity will be O(mn) as there are two nested loops with m and n length.

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.

Answer -

The statement is False.

Here is a counterexample -

Preferences for men -

Proposer	Preference 1	Preference 2
M1	W1	W2
M2	W2	W1

Preferences for Women -

Proposer	Preference 1	Preference 2
W1	M2	M1
W2	M1	M2

In this case, if M1 is paired off with W1, both W1 and W2 will have to pair with their 2nd preference. However, this is a stable matching, since in both couples (M1, W1), (M2, W2) there is only one participant with a higher preference outside of the match.

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.

Answer -

Suppose the following preferences for men and women:

M1:- W1>W2>W3
M2:- W3>W2>W1
M3:- W1>W2>W3

W1:- M3>M2>M1
W2:- M3>M2>M1
W3:- M3>M1>M2

In this instance, M3 is preferred by all women and prefers W1 himself.

Proposer	Preference 1	Preference 2	Preference 3
M1	W1	W2	W3
M2	W2	W1	W1
M3	W1	W2	W3

Proposer	Preference 1	Preference 2	Preference 3
M1	W1	W2	W3
M2	W2	W1	W1
M3	W1	W2	W3

Proposer	Preference 1	Preference 2	Preference 3
M1	W1	W2	W3
M2	W2	W1	W1
M3	W1	W2	W3

If M1 is paired with W1, M2 with W3, M3 does not match with W1, who is already matched with M1, hence disproving that the man who is the first choice of all the women, will match with his first preference.

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.

Answer -

Yes, there always exists a perfect match without strong instability.
This is true as this case is a modified version of the Gale Shapley Algorithm, with an additional logic of adding preference between the tied suitors on some basis (position, lexicographically, etc)

```
for each man in an unmatched list
    while (man m is unmatched)
      if w is unmatched then
          m proposes, and becomes engaged, to w;
      else (w is already engaged to m') then
          if m is on higher preference than m' then
            m becomes engaged to w
            add m' to the free list;
          else if m is on the same preference with m' then
                check order based on lexicography
                if m is first in order then
                  m becomes engaged to w
                  add m' to the free list;
                else
                  do nothing
```

The algorithm ends when no man is left unmatched.

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.

Answer -

No, there is not always a perfect match without weak instability.

We can prove this with the following counter-example:

⇔ - Represents indifference
> - Represents preference

w: m <=> m`
w`: m`<=> m

m : w` > w
m` : w` > w

If m is matched with w`, and w with m`, in this case, m` prefers w` more and is, therefore, a weak instability, and does not form a perfect matching.

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.

Answer -

In Gale Shapley's Algorithm, it depends if the woman is part of the proposing party or not, based on this they will have control over the outcome through truthfulness. For eg, we take the following preferences:

Chris:- Leslie > Pat > Dana
Brian:- Leslie > Dana > Pat
Donald:- Pat > Leslie > Dana

True Preference
Leslie:- Donald > Chris > Brian (True Preference)
Pat:- Chris > Donald > Brian
Dana:- Donald > Brian > Chris

False Preference
Leslie:- Donald > Brian > Chris (False Preference)

In this case, Leslie prefers Donald, however, she is the first preference of both Chris and Brian

Iteration 1 using True Preference:

Chris proposes to Leslie who accepts him
Brian proposes to Dana as Leslie is engaged
Donald proposes to Pat who accepts

Man	Preference 1	Preference 2	Preference 3
Chris	Leslie	Pat	Dana
Brian	Leslie	Dana	Pat
Donald	Pat	Leslie	Dana

At this point, all men are engaged and the algorithm ends, thus ending with Leslie obtaining her second preference.

Iteration 1 using False Preference:

Chris proposes to Leslie who accepts him
Brian also proposes to Leslie who accepts him and rejects Chris
Donald proposes to Pat who accepts

Man	Preference 1	Preference 2	Preference 3

Chris	Leslie	Pat	Dana
Brian	Leslie	Dana	Pat
Donald	Pat	Leslie	Dana

Iteration 2 using False Preference:

Chris proposes to Pat who rejects Donald (her prior engagement)
Donald who is now free proposes to his 2nd preference Leslie, Leslie accepts rejecting Brian (her prior engagement)

Man	Preference 1	Preference 2	Preference 3
Chris	Leslie	Pat	Dana
Brian	Leslie	Dana	Pat
Donald	Pat	Leslie	Dana

Iteration 3 using False Preference:

Only Brian is left unmatched who proposes to Dana who accepts.

Man	Preference 1	Preference 2	Preference 3
Chris	Leslie	Pat	Dana
Brian	Leslie	Dana	Pat
Donald	Pat	Leslie	Dana

At this point, all men are engaged and the algorithm ends, thus ending with Leslie obtaining her first preference, thereby influencing the outcome. This outcome would be different if the women were proposing in which case it would be a better outcome.

Q9 (5 Points)

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

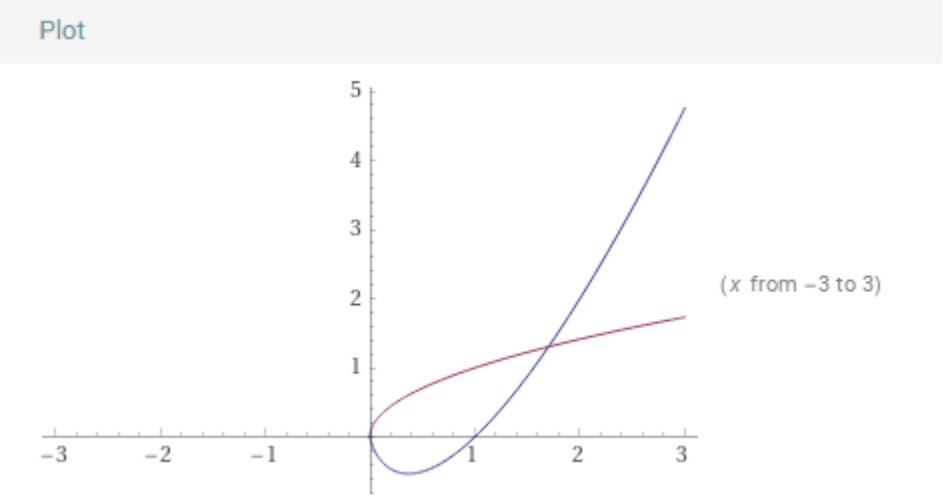
Answer -

√n is asymptotically faster than n log2(n).

Explanation: In the case of the first function, the running time would grow logarithmically with respect to n i.e. as n increases, the running time increases at a slower rate than a linear growth rate.

For the second function, the running time would grow at a rate proportional to the square root of n i.e. as n increases, the time taken would increase at a faster rate compared to logarithmic growth.

As represented by the graph plotting both functions,√n is asymptotically faster than n log2(n).



Link to Wolfram Alpha

Cross-over value of n - 1.7

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.

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.

Answer -

I initialized the teams with randomly generated preferences -

```
#initialize preferences
#rankings[(a, n)] = partner that a ranked n^th

def initializeTeams(n):
  #Super group A
  A = [str(x) for x in range(1,n+1)]
  print("Team A:", A)
  yield A
  #Super group B
  B = list(map(chr, range(97, 97+n)))
  print("Team B:", B)
  yield B
  rank = dict()

  for e in A:
    pref = [x for x in range(1,n+1)]
    random.shuffle(pref)
    rank[e] = tuple(pref)

  for e in B:
    pref = [x for x in range(1,n+1)]
    random.shuffle(pref)
    rank[e] = tuple(pref)

  print("Rank :", rank)
  yield rank
```

Created a set to map preferences for both teams

```python
#create set of preferences/rankings
def createPreferences(A,B, rank):
  Arankings = dict(((a, rank[a][b_]), B[b_]) for (a, b_) in product(A, range(0, len(A))))
  Brankings = dict(((b, rank[b][a_]), A[a_]) for (b, a_) in product(B, range(0, len(B))))

  rankings = Arankings
  rankings.update(Brankings)

  return rankings
```

And used the referenced algorithm to find a stable matching -

```python
def stable(rankings, A, B):

  teams = dict((a, (rankings[(a, 1)], 1)) for a in A)
  is_stable = False # whether the current pairing (given by `partners`) is stable
  while is_stable == False:
      is_stable = True
      for b in B:
          is_paired = False # whether b has a pair which b ranks <= to n
          for n in range(1, len(B) + 1):
              a = rankings[(b, n)]
              a_partner, a_n = teams[a]
              if a_partner == b:
                  if is_paired:
                      is_stable = False
                      teams[a] = (rankings[(a, a_n + 1)], a_n + 1)
                  else:
                      is_paired = True

  return sorted((a, b) for (a, (b, n)) in teams.items())
```

Output:

```python
from itertools import product

data = initializeTeams(8)
A,B,rank=next(data),next(data),next(data)

rankings=createPreferences(A,B, rank)
print("Stable Matching :", stable(rankings, A, B))
```

```
Team A: ['1', '2', '3', '4', '5', '6', '7', '8']
Team B: ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h']
Rank : {'1': (2, 3, 8, 5, 6, 4, 1, 7), '2': (5, 6, 3, 7, 2, 4, 8, 1), '3': (8, 1, 3, 2, 5, 6, 7, 4), '4': (6, 2, 5, 7, 3, 1, 8, 4), '5': (1, 8, 7, 6, 3, 2, 5, 4), '6': (6, 3, 7, 4,
Stable Matching : [('1', 'f'), ('2', 'e'), ('3', 'd'), ('4', 'b'), ('5', 'a'), ('6', 'h'), ('7', 'c'), ('8', 'g')]
```

Code

References: https://github.com/paulgb/Python-Gale-Shapley/blob/master/stable.py

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

Answer -

Shuffling proves that Gale Shapley's Algorithm provides a stable matching every iteration, irrespective of the preference list.

Code

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.

Answer -

Assuming teams from either group can lose, we cannot predict the number of loser teams from each supergroup. This would mean that both sets' size is not guaranteed to be equal in each round. We need both sets to be of equal size as we will check preferences for both teams. This is a preliminary requirement to achieve stable matching.

Therefore, since both teams do not remain of equal size in both cases, Gale Shapley cannot be applied over and over to achieve stable matching.

D. Now combine the lists so that 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.

Answer -

Combining the lists would create one list, where each team would be up against all the teams from both super groups except itself. We will now need to check each team against the entire set of teams, also the rankings are over the complete set. In this case we need to use the stable roommate problem or Irving's Algorithm, which can find stable matches but does not guarantee it.

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?*

Answer -

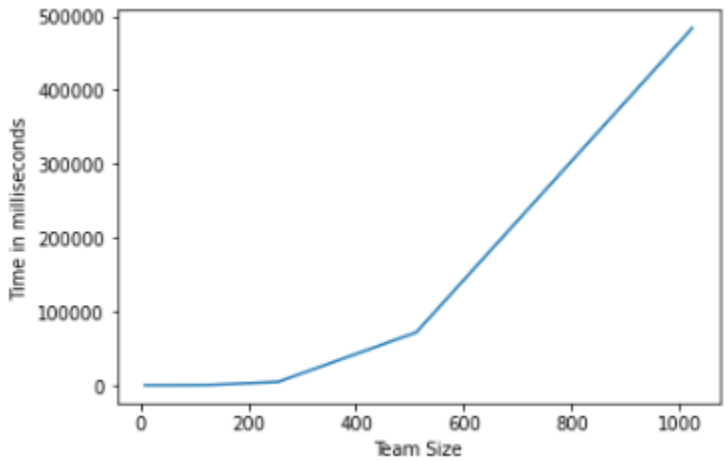
Upon doubling the team size several times we get the processing time incrementing exponentially -

```
# double team size 8 times and measure processing time
n=8
init_size = 8

time_comparison = {}
for ind in range(n):
  team_size = init_size
  init_size = init_size*2
  data = initializeTeams(team_size)
  A,B,rank=next(data),next(data),next(data)

  rankings=createPreferences(A,B, rank)
  start = time.time()
  stable(rankings, A, B)
  end = time.time()
  time_comparison[team_size] = (end - start)*1000
```

```
print([ [k[0],round(k[1], 4)] for k in time_comparison.items()])
```

```
[[8, 0.0768], [16, 1.9016], [32, 12.1024], [64, 25.6896], [128, 412.6358], [256, 4541.105], [512, 71510.9289], [1024, 483666.9855]]
```

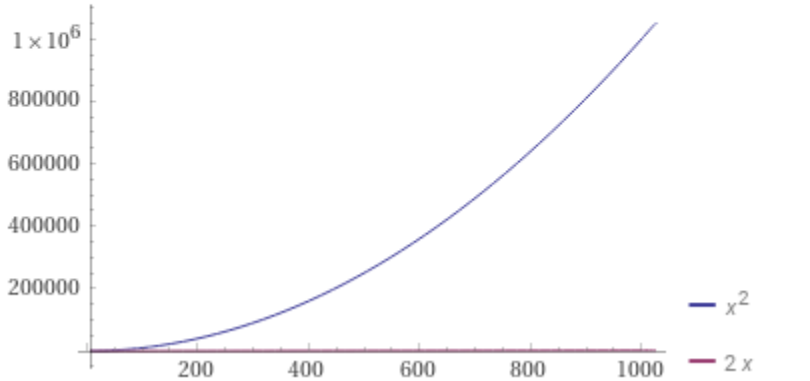
If we plot this time we get a curve similar to the graph for BigO(n^2) -



Similar to O(n^2) -

plot x^2 / $x \times 2$ $x = 8$ to 1024

Plot



[Link to Wolfram Alpha]

[Code]