Name: Connor Morrison
NSID: tvi340
Student Number: 11374770
Course Number: CMPT 145
Lecture Section: 04
Laboratory Section: L02

(a) (3 points)

```
def search(L, target):
    # L is a list
    found = False
    i = 0
    while i < len(L) and not found:
        if L[i] == target:
          found = True
        i += 1
    return found
```

- Big-O Time Complexity: O(N)
- Justification: In the worst case, the function iterates through each element in the list once until it finds the target. Thus, the time complexity is directly proportional to the number of elements ‘N’ in the list ‘L’.

(b) (3 points)

```
def repeat(x):
    # x is an integer
    for i in range(1000):
        print(x)
```

- Big-O Time Complexity: O(1)
- Justification: This function prints the integer ‘x’ 1000 times. The number of iterations (1000) is constant and does not depend on the input size. Thus, the time complexity is constant.

(c) (3 points)

```
def compare(a, b):
    # a and b are integers
    if a > b:
        return 1
    elif b > a:
        return -1
    else:
        return 0
```

- Big-O Time Complexity: O(1)
- Justification: This function directly compares two integers and returns a result based on the comparison. The time to complete the operation does not depend on the size of the input. Thus, it is constant.

(d) (3 points)

```
def is_prime(a):
    # a is an integer
    if a == 1:
        return False
    elif a == 2 or a == 3:
        return True
    result = True
    for i in range(2, a//2):
        if a % i == 0:
            result = False
    return True
```

- Big-O Time Complexity: O(N)
- Justification: In the worst case, this function checks up to ‘a’/2 numbers to determine if a is prime. The number of operations grows linearly with the value of ‘a’, so it simplifies to O(N), where ‘N’ is the value of ‘a’.

(e) (3 points)

.

```
def find_triples(L):
    # L is a list
    if len(L) == 0:
     return 0
    L = sorted(L)
    triples = 0
    repeats = 1
    prev = L[0]
    for i in range(1, len(L)):
     cur = L[i]
     if cur != prev:
       repeats = 1
     else:
       repeats += 1
       if repeats == 3:
         triples += 1
     prev = cur

    return triples
```

- Big-O Time Complexity: O(N log N)
- Justification: The sorting of the list L has a time complexity of O(N log N), and the following iteration through the list has a time complexity of O(N). When combined, the overall complexity is O(N log N), where 'N' is the number of elements in 'L'.

(f) (3 points)

.

```
def interleave(L1, L2):
   # L1 and L2 are lists
   i = 0
   result = []
   while i < min(len(L1), len(L2)):
     result.append(L1[i])
     result.append(L2[i])
     i += 1
   if len(L1) > len(L2):
       longer = L1
   else:
       longer = L2
   for j in range(i, len(longer)):
       result.append(longer[j])
   return result
```

- Big-O Time Complexity: O(N)
- Justification: The function iterates through each of the smaller list's elements once (min(len(L1), len(L2))), and then it iterates through the remaining elements of the

longer list (abs(len(L1) - len(L2))). Overall, this scales linearly with the size of the input lists, hence O(N), where 'N' is the length of the longer list.

(g) (3 points) Consider a single elimination chess tournament. A single elimination format means that as soon as a player loses a single game, they are out of the tournament. Players who win their games keep playing until only a single undefeated player remains.
For example, if we had a tournament with 4 players (Godzilla, Mothra, Gigan and Ghidorah), we would need a total of 3 matches to complete the tournament as follows:

- **Match A: Godzilla vs Mothra**
- **Match B: Gigan vs Ghidorah**
- **Match C: Winner of A vs Winner of B**

Let time be measured in terms of the number of games needed to complete the tournament. What is the big-O notation for the time needed to complete a single elimination tournament with N players?

- Big-O Time Complexity: O(N)
- Justification: In a single elimination tournament, each round halves the number of players until only one remains. The total number of games needed is N-1 (where 'N' is the initial number of players), as each game eliminates one player. Thus, the time complexity is linear O(N) with respect to the number of players.

(h) (3 points) Consider a round-robin chess tournament. A round-robin format means that every player must play each other player exactly once.
For example, if we had a tournament with 4 players (Godzilla, Mothra, Gigan and Ghidorah), we would need a total of 6 matches to complete the tournament as follows:

- **Godzilla vs Mothra**
- **Godzilla vs Gigan**
- **Godzilla vs Ghidorah • Mothra vs Gigan**
- **Mothra vs Ghidorah • Gigan vs Ghidorah**

Let time be measured in terms of the number of games needed to complete the tournament. What is the big-O notation for the time needed to complete a round-robin tournament with N players? Briefly justify your answer.

- Big-O Time Complexity: O(N^2)
- Justification: In a round-robin format, each player must play against every other player exactly once. For 'N' players, this results in N*(N-1)/2 matches, because each match involves a pair of players. This simplifies to (N^2-N)/2, which in Big O notation is O(N^2), reflecting the quadratic relationship between the number of players and the number of games.