Name: _____ Recitation: _____ Andrew Id: _____

15-112 Summer 2019 Quiz 3

Up to 50 minutes. No calculators, no notes, no books, no computers. Style will not be graded on quizzes. Show your work!

1. (15 points) **Big O:** What is the Big-O runtime of each of the following in terms of N?

Built-in Big-O Runtimes					
`len(L)`	$O(1)$	`len(s)`	$O(1)$	`len(lst)`	$O(1)$
`item in S`	$O(1)$	`c in s`	$O(N)$	`item in lst`	$O(N)$
`S.add(item)`	$O(1)$	`list(range(N))`	$O(N)$	`lst[i]`	$O(1)$
`S.remove(item)`	$O(1)$	`L + M`	$O(len(L) + len(M))$	`sorted(lst)`	$O(NlogN)$

```
# S and T are both lists of size N
1: def bigOh1(S, T):                          # Big-O
2:     count = 0                              # _____
3:     for i in range(len(T)):                # _____
4:         new_S = sorted(S + [T[i]]):        # _____
5:         for j in range(len(new_S)):        # _____
5:             count += new_S[j]              # _____
6:     return count                           # _____
```

```
# L is a list of N integers
 1: def bigOh2(L):                                      # Big-O
 2:     count = 0                                       # _____
 3:     S = set()                                       # _____
 4:     for i in range(len(L)):                         # _____
 5:         S.add(L[i])                                 # _____
 6:     for x in L:                                     # _____
 7:         if x in S or x in list(range(len(L)**2)):   # _____
 8:             count += 1                              # _____
 9:     return count                                    # _____
```

2. (15 points) **Code Tracing:** Indicate what the following program prints. Place your answers (and nothing else) in the box under the code.

```
def h(x, y, depth=0):
        print(depth, "h(%d,%d)" % (x,y))
        if (x+y <= 8):
                result = x+y
        else:
                result = 2*h(x-1, y-1, depth+1) + 2*h(x//2, y//2, depth+1)
        print(depth, "-->", result)
        return result

print(h(5,6))
```

3. (30 points) **Free Response**: Write the function mostPopularFootballTeam(prefs) that takes in a dictionary prefs, and returns a set containing the football teams that are liked by the most people in prefs. You can assume prefs is a dictionary mapping people to their favorite football team. As an example, consider the following prefs dictionary:

```
prefs = {"Abhi": "Browns",
         "Sarah": "Eagles",
         "AJ": "Eagles"}
```

In this case, mostPopularFootball team would return the set containing "Eagles", since the Eagles are liked by 2 people, while the Browns are only liked by 1 person. In a case like this:

```
prefs = {"Abhi": "Browns",
         "Sarah": "Eagles",
         "AJ": "Browns",
         "Gabriel": "Eagles"}
```

mostPopularFootball team would return `set(["Eagles", "Browns"])`, since the Eagles and Browns are both liked by 2 people, which is the most number of likes for any given team. Your solution must run in O(N).

Hint: You will want to create a new dictionary in your solution. Any other approach will be considered too inefficient.

4. (40 points) **Free Response: getAirplaneItinerary**

Write the function getAirplaneItinerary(schedule, start, dest) that takes in schedule (a dictionary representing airplane routes), a start city and destination city, and returns a list containing a path from start to dest according to the schedule, and None if no such path exists. The schedule is a dictionary where the keys are source cities and the values are lists of destinations. If a source city has no destinations, its value in the dictionary will be an empty list. An example test case is shown below.

```
schedule = {"PIT": ["DTW", "BOM", "EWR"],
            "EWR": ["MEX", "SFO", "IAH"],
            "MEX": ["DEL", "BOM"],
            "IAH": ["DTW"],
            "DTW": ["BOM", "PIT"],
            "BOM": ["DEL"],
            "DEL": [],
            .... # more keys exist
            }

getAirplaneItinerary(schedule, "PIT", "DEL") = ["PIT", "EWR", "MEX", "DEL"]
getAirplaneItinerary(schedule, "MEX", "PIT") = None # No path!
```

To expand upon the first example, EWR is in PIT's destinations, MEX is in EWR's destinations, and DEL is in MEX's destinations, so a path exists from PIT to DEL.

You may assume that start and dest both appear in the schedule, and that any destinations in the values of the dictionary appear as keys in the schedule as well. Also, your path should not have a destination that is visited twice (i.e. `["PIT", "DTW", "PIT"]` is not a valid path in the above case). If multiple paths exist, your function can return any one of them.

You may use iteration, but you must use recursion in a meaningful way. In particular, to solve this, you must use recursive backtracking.