Homework #1 433 Zachary Hightower

1.1
Q4: Design an algorithm for computing the square root of *n* for any positive integer *n*. Besides assignment and comparison, your algorithm may only use the four basic arithmetical operations.

ANS4: Note, I'm putting the code into a different font for clarity.

```
def square_root_binary_search(n, precision_pointer=1e-20):
   # This rule handles cases for very small values of n approaching 0 or 1
   if n == 0 or n == 1:
      return n

   # This rule defines the search range to be from 0 to n, covering all integers between
   low, high = 0, n

   # These rules perform a simple binary search until we reach a value that satisfies the
condition of    #being a square root, and satisfying our precision defined at the beginning
   while high - low > epsilon:
      mid = (low + high) / 2
      square = mid * mid

      if square == n:
         return mid
      elif square < n:
         low = mid
      else:
         high = mid

   # This ends our function and returns the square root found by the search
   return (low + high) / 2
```

Q6:
A: Find gcd(31415, 14142) by applying Euclid's algorithm.

ANS6-A:

$31415 = 2 \times 14142 + 3131$

$14142 = 4 \times 3131 + 1618$

$3131 = 1 \times 1618 + 1513$

$1618 = 1 \times 1513 + 105$

$1513 = 14 \times 105 + 43$

$105 = 2 \times 43 + 19$

$43 = 2 \times 19 + 5$

$19 = 3 \times 5 + 4$

$5 = 1 \times 4 + 1$

$4 = 4 \times 1 + 0$ //This is where the algorithm stops, since we hit 0

We look to the added number right before to get the gcd, which in this case, is 1. That means that the gcd of our two numbers is 1, which means that they are relatively prime.

B: Estimate how many times faster it will be to find gcd(31415, 14142) by Euclid's algorithm compared with the algorithm based on checking consecutive integers from min{m, n} down to gcd(m, n).

ANS6-B:
If we say that it takes 15 seconds to do one line of the calculation for Euclid's algorithm, and we say that it takes an equal amount of time to do the calculation to find one factor of a number, for small numbers, such as.

4 - 1, 2, 4 - 3 factors
12 - 1, 2, 3, 4, 6, 12 - 6 factors

The calculations might take an equal amount of time, however, for very large numbers, it will take more and more lines to calculate the factors of a number. In addition to this fact, to find the gcd by this alternate method, we would need to find all the factors of each number to be certain of finding the gcd, which could double the workload in the worst case.

So, Euclid's algorithm could be twice as, or hundreds of times faster than an algorithm that checks consecutive integers. With the advantage of Euclid's algorithm growing the larger the two integers become.

So with Euclid's algorithm we would be looking at something in the range of sixty seconds to one hundred and twenty seconds even with relatively large numbers.

With factor checking/checking consecutive integers in a certain range, we could be looking at double that, 120 to 240 seconds, or something ten times, a hundred, a thousand, etc. more, depending on how large the numbers are getting. So it could be a difference of spending two minutes using Euclid's algorithm to spending three hours or more checking each integer or finding every factor.

1.2
Q1:
Old World puzzle A peasant finds himself on a riverbank with a wolf, a goat, and a head of cabbage. He needs to transport all three to the other side of the river in his boat. However, the boat has room for only the peasant himself and one other item (either the wolf, the goat, or the cabbage). In his absence, the wolf would eat the goat, and the goat would eat the cabbage. Solve this problem for the peasant or prove it has no solution. (Note: The peasant is a vegetarian but does not like cabbage and hence can eat neither the goat nor

the cabbage to help him solve the problem. And it goes without saying that the wolf is a protected species.)

ANS1:
Step 1: Take the goat across to the other side, return with nothing
Step 2: Take the wolf across to the other side, return with the goat
Step 3: Take the cabbage across to the other side, return with nothing
Step 4: Take the goat to the other side, continue on journey

Q2: New World puzzle There are four people who want to cross a rickety bridge; they all begin on the same side. You have 17 minutes to get them all across to the other side. It is night, and they have one flashlight. A maximum of two people can cross the bridge at one time. Any party that crosses, either one or two people, must have the flashlight with them. The flashlight must be walked back and forth; it cannot be thrown, for example. Person 1 takes 1 minute to cross the bridge, person 2 takes 2 minutes, person 3 takes 5 minutes, and person 4 takes 10 minutes. A pair must walk together at the rate of the slower person's pace. (Note: According to a rumor on the Internet, interviewers at a well-known software company located near Seattle have given this problem to interviewees.)

ANS1:
Step 1: P1 and P2 cross -2 min, remaining time 15
Step 2: P2 returns w/ flashlight -2 min, remaining time 13
Step 3: P3 and P4 cross -10 min, remaining time 3
Step 4: P1 returns w/ flashlight -1 min, remaining time 2
Step 5: P1 and P2 cross -2 min, remaining time 0

This should get every person across in the exact time limit and conditions described.

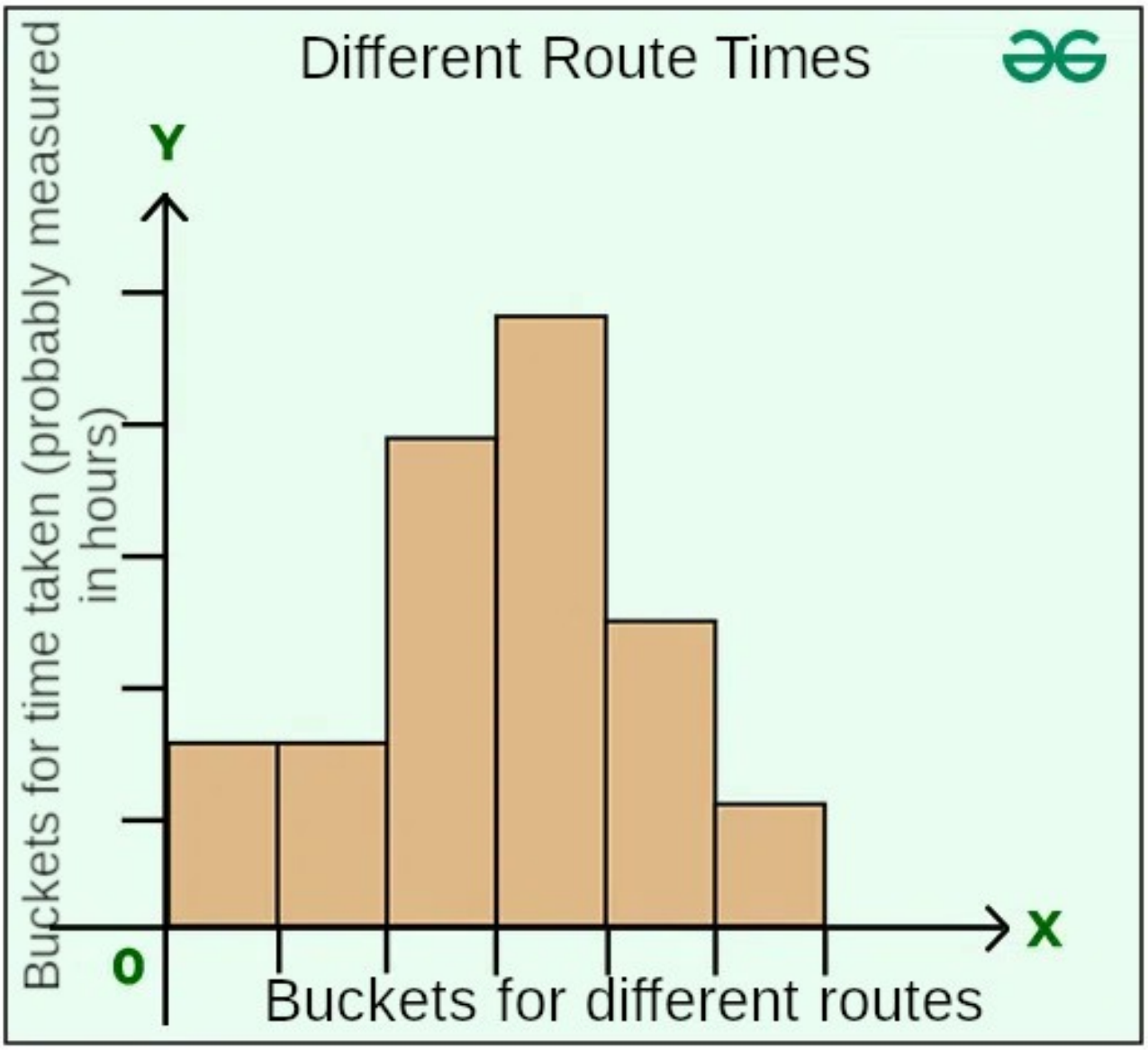
1.3
Q6: Consider the following problem: Design an algorithm to determine the best route for a subway passenger to take from one designated station to another in a well-developed subway system similar to those in such cities as Washington, D.C., and London, UK.

A: The problem's statement is somewhat vague, which is typical of real-life problems. In particular, what reasonable criterion can be used for defining the "best" route?

ANS6-A:
The most reasonable criterion, if we're picking only one, would be time. Other important factors could be comfort, safety, or possible stops that are needed along the way for food and drink.
Time is the one that should always be considered in any scenario to find the best route, though.

B: How would you model this problem by a graph?

ANS6-B:



I think a simple histogram with buckets for our different routes will be good enough for measuring how well they perform on a time scale in hours, or minutes if the route is very short.

I grabbed the image of a histogram off the web and just edited in the title and the labels for the x and y axes.

1.4
Q1: Describe how one can implement each of the following operations on an array so that the time it takes does not depend on the array's size n.

A: Delete the ith element of an array ($1 \leq i \leq n$).

ANS1-A: In order to keep the time complexity constant and delete the ith element of an array, we can use the following method.

Step 1: Swap the ith element, with the last element in the array
Step 2: remove the last element in the array

The order of the array will be changed, but we are not concerned with maintaining a sorted array in this scenario.

Note: The best way to do this in Java is to work with an ArrayList, instead of a base array, which in Java is a fixed size.

B: Delete the ith element of a sorted array (the remaining array has to stay sorted, of course).

ANS1-B:
The best practice in java is to use the same method as before, but with a tweak. We can maintain the sorted order of the array by taking the ith element and shifting it to the end of the array. This will take some time, and in the worst case, where the ith element is at the very beginning of the array, it will take O(n) time. This is the best possible solution using the tools in this scenario, though, and in most cases it will not take O(n) time.

Q3:

A:Show the stack after each operation of the following sequence that starts with the empty stack:

push(a), push(b), pop, push(c), push(d), pop

ANS3-A: //where _ represents the completely empty stack

```
_ --> a --> b --> a --> c --> d --> c
              a             a      c      a
                                   a
```

B: Show the queue after each operation of the following sequence that starts with the empty queue:

enqueue(a), enqueue(b), dequeue, enqueue(c), enqueue(d), dequeue

ANS3-B: //where _ represents the completely empty queue

```
_ --> a --> a --> b --> b --> b --> c
              b             c      c      d
                                   d
```

Q4:

A: Let A be the adjacency matrix of an undirected graph. Explain what property of the matrix indicates that

I: the graph is complete.

ANS4-A-I:
A graph is complete when every element on the adjacency matrix is equal to 1 except for those which make up the main diagonal.

II: the graph has a loop, i.e., an edge connecting a vertex to itself.

ANS4-A-II:
If the graph has a loop, then the adjacency matrix must have an element that equals 1 on the main diagonal.

III: the graph has an isolated vertex, i.e., a vertex with no edges incident to it.

ANS4-A-III:
An isolated vertex is indicated by the adjacency matrix having a row where all the elements are equal to zero.

B: Answer the same questions for the adjacency list representation.

I: the graph is complete.

ANS4-B-I:
A graph in an adjacency list is complete when each of the linked lists contains all the other vertices that make up the graph.

II: the graph has a loop, i.e., an edge connecting a vertex to itself.

ANS4-B-II:
A graph in an adjacency list has a loop when one of the linked lists contains the vertex that defines the list itself.

III: the graph has an isolated vertex, i.e., a vertex with no edges incident to it.

ANS4-B-III:
A graph in an adjacency list with an isolated vertex is shown by one of the adjacency lists being completely empty.