Name: kevin wijaya

Here are some basic rules for calculating the Big O for some $T(n)$ or an algorithm.

1. Only the highest degree of n matters. For example

$$T(n) = n^3 + 5n^2 + 10^7 \rightarrow O(n^3)$$

since once n becomes super-massively huge, the other terms just stop mattering.

2. Constant factors don't matter. $T(n) = 500n$ and $T(n) = 0.005n$ both $O(n)$. Again, as n becomes bigger, these constants stop mattering; what matters is the rate of growth. Example:

$$T(n) = 50n^3 - 2n^2 + 400 \rightarrow O(n^3)$$

3. Counting the number of nested loops usually works.

You can turn in this assignment physically to a TA or me. You can also scan and upload your answers.

 _____ 0 points

For each of the following $T(n)$, write the corresponding Big O time complexity. Some series may require research.

1. (2 points) $T(n) = n^2 + 3n + 2$

1. $O(n^2)$

2. (2 points) $T(n) = (n^2 + n)(n^2 + \frac{\pi}{2})$

2. $O(n^4)$

3. (2 points) $T(n) = 1 + 2 + 3 + \ldots + n - 1 + n$

3. $O(n^2)$

4. (2 points) $T(n) = 1^2 + 2^2 + 3^2 + \ldots + (n-1)^2 + n^2$

4. $O(n^3)$

5. (2 points) $T(n) = 10$

5. $O(1)$

6. (2 points) $T(n) = 10^{100}$

6. $O(1)$

7. (2 points) $T(n) = n + \log n$

7. $O(n)$

8. (2 points) $T(n) = 12\log(n) + \frac{n}{2} - 400$

8. $O(n)$

9. (2 points) $T(n) = (n+1) \cdot \log(n) - n$

9. $O(n \log n)$

10. (2 points) $T(n) = \frac{n^4+3n^2+2n}{n}$

10. $O(n^3)$

_____ 20 points

11. (4 points) What is the time complexity to get an item from a specific index in an ArrayList?

Time complexity is O(1)

12. (3 points) What is the time complexity remove an item in the middle of an ArrayList?

Time complexity is O(n)

13. (3 points) Why? Time complexity is O(n) because the index is shifted left when you remove an element in a list which is not the last index

14. (3 points) What is the **average** time complexity to add an item to the end of an ArrayList?

O(1)

15. (3 points) What is the **worst case** time complexity to add an item to the end of an ArrayList? What if you have to or don't have to reallocate?

assuming no need to reallocate, it would be O(1), O(n) if you have to reallocate.

16. (4 points) Taking this all into account, what situations would an ArrayList be the appropriate data structure for storing your data? I'd say an arraylist would be appropriate if your data doesn't require any shifting of the indexes, i getting, adding and removing at the end of al list

 _____ 20 points

17. (10 points) The above puzzle, while from a children's puzzle book, is actually a very interesting graph theory problem, known as the Rudrata Path or Hamiltonian Path. What is the Rudrata Path problem and how does it correspond to the above puzzle?

The Rudrata Path problem is a problem in which whether or not there exists a Hamiltonian path exists in a graph, it corresponds to this puzzle because Zoomer Duo has to visit every star, because he is in a hurry to save Quirk, so it wouldn't make sense for him to visit a star twice, so he needs to find a path that starts from the entrance and ends at the exit without repeating a star.

- Hr visits vertex once

18. (10 points) Suppose we generalized the above puzzle so that there was any number of stars, rather than just 35. Let the number of stars in the above puzzle be defined as n. If there are n stars and a line between every possible pair of stars, how many paths would an algorithm need to check in order to check to find the solution?

Algorithm would need to check $(n-2)!$ paths the entrance and exit never changes, so there are $(n-2)!$ permutations, and every permutation needs to be checked to see if a path exist between the stars.

 _____ 20 points

`bogosort` attempts to sort a list by shuffling the items in the list. If the list is unsorted after shuffling, we continue shuffling the list and checking until it is finally sorted.

19. (5 points) What is the worst case run time for `bogosort`?

the worst case runtime would be $O(\infty)$

20. (5 points) Why? since bogosort keeps running till the list is sorted then that means that the worst possible runtime would be $O(\infty)$, because if the list is unsorted everytime "Bogosort" runs then that means that Bogosort can go on forever.

21. (5 points) What is the average case run time for `bogosort` (Hint: think about a deck of cards)?

the average case runtime would be $O(n \cdot n!)$

22. (5 points) Why?

since there are n elements, then the possible amount of permutations would be n!, so we can expect the list to be sorted on average after n! times, but then shuffling involves using a for loop to randomize a list so it would be $(n \cdot n!)$

 _____ 20 points

23. (20 points) For each of the methods you wrote in Lab 2, figure out the time complexity of the method you wrote. To turn in this portion, attach a printout of the code and specify the time complexity of each.

_____ 20 points

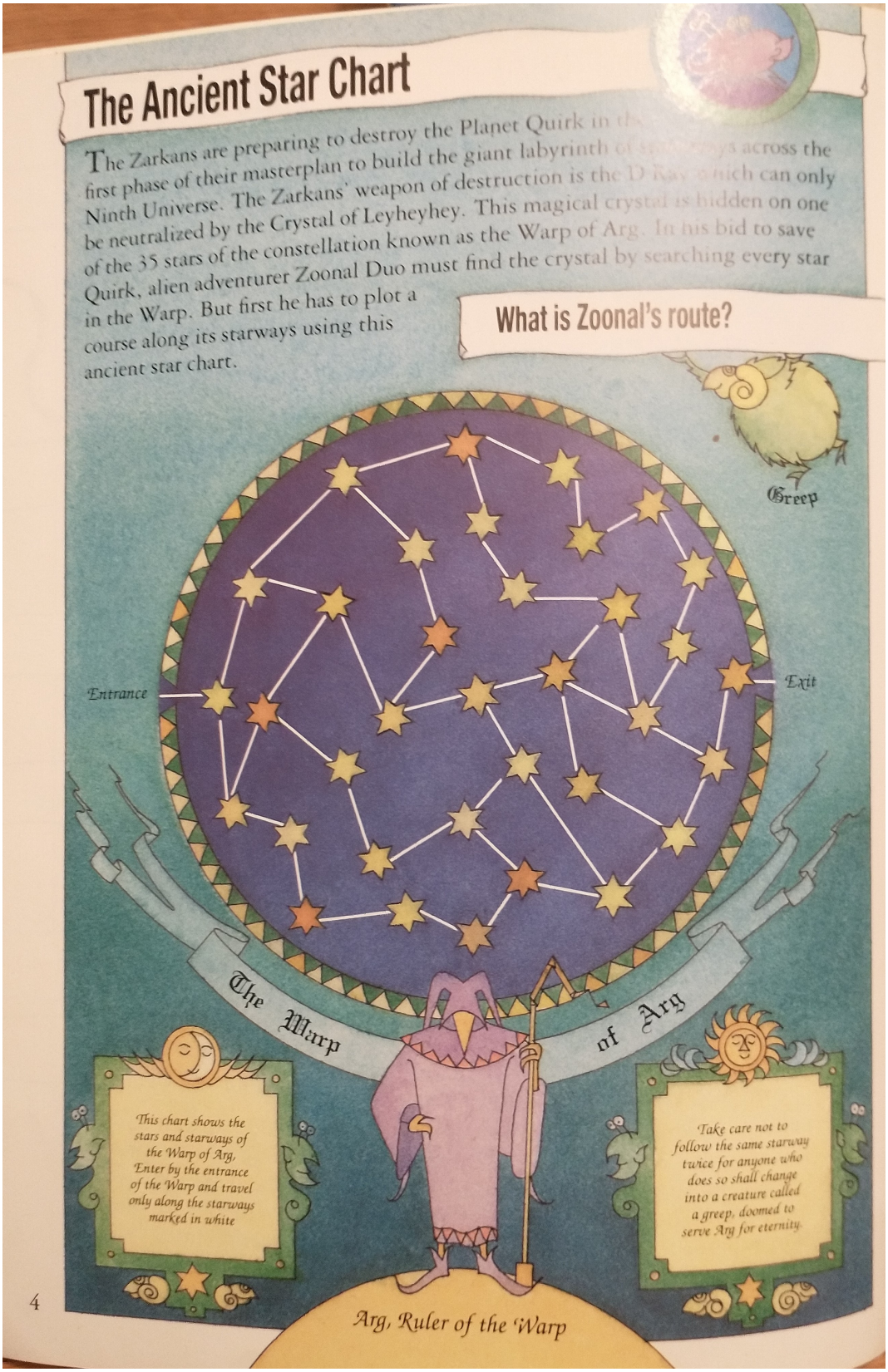

Figure 1: Please look at the above puzzle, taken from Sarah Dixon's *Map & Maze Puzzles* and answer the questions on the next page.

_____ 0 points

1.

```
public static <E> boolean unique(List<E> list) {
      for (int i = 0; i < list.size(); i++) {
         for (int j = 1 + i; j < list.size(); j++) {
            if (list.get(i).equals(list.get(j))) {
               return false;
            }

         }
      }
      return true;
   }
```

time complexity = $O(n^2)$

2.

```
public static List<Integer> allMultiples(List<Integer> multiples, int n) {
      List<Integer> newlist = new ArrayList<>();
      for (int x : multiples) {
         if (x % n == 0) {
            newlist.add(x);
         }

      }
      return newlist;
   }
```

Time complexity = $O(n)$

3.

```
  public static List<String> allStringsOfSize(List<String> duplicate, int n) {
      List<String> Tengu = new ArrayList<>();
      for (String x : duplicate)
         if (x.length() == n) {
            Tengu.add(x);
         }
      }
      return Tengu;
   }
```

time complexity = $O(n)$

4.

```
 public static <E> boolean isPermutation(List<E> newlist1, List<E> newlist2) {
      // checks if the lists are the same size
      if (newlist1.size() == newlist2.size()) {
         for (int i = 0; i < newlist1.size(); i++) {
            for (int j = 0; j < newlist2.size(); j++)
               if (newlist1.get(i).equals(newlist2.get(j)) && i != 0) {
                  newlist1.remove(i);
                  newlist2.remove(j);
                  i--;
                  j--;
               }
```

```
            }
        }
        if (newlist1.equals(newlist2)) {
            return true;
        } else {
            return false;
        }
    } else {
        return false;
    }
}
```

$O(n^3)$

5.

```
public static List<String> StringtoListOfWords(String convertee) {
    List<String> list = new ArrayList<>();
    String[] m = convertee.split("\\s+");
    // loops through the array and adds the elements in the array to a list
    for (String x : m) {
        list.add(x);
    }
    return list;
}
```

$O(n)$

6.

```
public static <E> void removeallInstances(List<E> list, E obj) {
    for (int i = 0; i < list.size(); i++) {
        if (list.get(i) == obj) {
            list.remove(i);
            i--;
        }

    }

}
```

$O(n^2)$