UNIVERSITY OF SASKATCHEWAN

Department of Computer Science
176 Thorvaldson Building
110 Science Place, Saskatoon, SK, S7N 5C9, Canada
Telephine: (306) 966-4886, Facimile: (306) 966-4884

Assignment 9

Searching and Sorting

Date Due: 5:00pm Friday Nov 29 **Total Marks: 37**

General Instructions

- **This assignment is individual work.** You may discuss questions and problems with anyone, but the work you hand in for this assignment must be your own work.
- Each question indicates what to hand in. You must give your document the name we prescribe for each question, usually in the form aNqM, meaning Assignment N, Question M.
- Make sure your name and student number appear at the top of every document you hand in. These conventions assist the markers in their work. Failure to follow these conventions will result in needless effort by the markers, and a deduction of grades for you.
- Do not submit folders, or zip files, even if you think it will help. It might help you, but it adds an extra step for the markers.
- Programs must be written in Python 3.5+
- You may not, in part or in whole, submit any code that was written by generative AI tools, such as ChatGPT
- **Assignments must be submitted to Canvas (the course website).**
- **Canvas will not let you submit work after the assignment hand-in closes.** It is advisable to hand in each answer that you are happy with as you go. You can always revise and resubmit as many times as you like before the deadline; only your most recent submission will be graded.
- Read the purpose of each question. Read the Evaluation section of each question.

Question 1 (5 points):

Purpose: To understand the behaviour of searching algorithms

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
L :	5	10	15	20	25	30	35	40	45	50	55	60	65	70	75	80

Consider the **already sorted list** L shown and above. WHICH elements will be examined and in what ORDER when conducting a...

1. **linear search** when searching for the value 15?
2. **linear search** when searching for the value 42?
3. **binary search** when searching for the value 5?
4. **binary search** when searching for the value 99?
5. **binary search** when searching for the value 42?

For Linear Search, assume the search starts from the front (i.e. index 0). For Binary Search, assume we round the index down if needed to find the middle.

What to Hand In

Hand in your solution in a file called `a9q1.pdf` (.doc, .png, .jpg are also ok).

Evaluation

- 1 mark for each search query

Question 2 (10 points):

Purpose: To understand the behaviour of searching and sorting algorithms

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
L :	25	63	25	30	87	75	50	17	42	42	20	99	80	12	50	42

Consider the list L shown and above and complete the following tasks with regard to it.

- Draw the recursion tree that results from the **divide step of merge sort**
- Draw the recursion tree that results from the **divide step of quick sort**. Assume the last element is used as the pivot. No need to draw branches that would lead to empty lists
- AFTER the list has been sorted, WHICH elements will be examined by **linear search** when searching for the value 77?
- AFTER the list has been sorted, WHICH elements will be examined by **binary search** when searching for the value 77? Assume we round down to find the middle if needed

You will likely find it useful to use diagramming software of some kind for this. There are good ones online that run right in your browser. I used this one: `https://app.diagrams.net/`

What to Hand In

Hand in your solution in a file called `a9q2.pdf` (.doc, .png, .jpg are also ok).

Evaluation

- 3 marks for the merge sort diagaram
- 4 marks for the quick sort diagram
- 1 mark for analyzing linear search
- 2 marks for analyzing binary search

Question 3 (13 points):

Purpose: To practice the concept of dictionary as a database; to implement a problem that has been decomposed into smaller parts

Overview

Pokemon are fantastic creatures that people can catch and train. Pokemon are classified into different "types", such as grass, fire or electric. To catch different types of pokemon, you might have to travel all over the world! Here, you will build a program that will help organize information about where to go to catch the different pokemon.

This part of the assignment is divided into 5 different "questions", but in the end, your work from all of the questions will be combined to complete the overall program. The first four questions will each ask you to define a function that will perform a specific task. Make sure that the inputs (parameters) and outputs (return values) of your functions match the problem description **exactly**, or else the whole thing won't fit together properly!

The following is a brief overview of how the next five questions will fit together:

- **Part 1**: Read data about the pokemon from a text file and organize it as a **database**. **You must fully complete this question before moving on.** All of the other questions will depend on the database that you create here.
- **Part 2**: Build a list of all the **unique** continents where you can find pokemon.
- **Part 3**: Build a list of all the pokemon that can be found on a particular continent.
- **Part 4**: Count how many of each pokemon type occur within a given list of pokemon names.
- **Part 5**: For each continent, display the pokemon type counts.

Individually, none of these questions are particularly difficult, but if you make a mistake anywhere along the way, then the last part won't work. Make sure to perform simple testing, such as printing out lists or dictionaries, as you go to make sure that everything looks right.

One last note of caution: we have given you an input file for this assignment, but be aware that we might test your program on a different input file. Therefore, you must make sure your code is fully general; don't count on knowing exactly how many pokemon there are, or on knowing exactly what the names of the pokemon types or locations might be.

What to Hand In

When you're done, you'll hand in one document, `a9q3.py`, with **ALL** your work for the different parts in it. Make sure you put identification information at the top of your program!

Part 1 (4 marks)

Before starting this question, first, download the data file `pokemonLocations.txt` from the class website. Make sure to put it in the same folder as your python code.

Write a function called `read_pokedata()` that takes as parameter(s):

- a string indicating the name of a pokemon location data file

This function should **return** a database (i.e. a dictionary-of-dictionaries) that stores all of the Pokemon data in a format that we will describe further below. You can review section 11.1.9 of the text for a refresher on what databases look like using dictionaries.

Input File Format

The data file for this question looks like this:

```
bulbasaur,grass,South America
ivysaur,grass,Asia,Antarctica
```

Each line of the file contains all of the data for a single pokemon. The first item on the line is always the pokemon's name; names are guaranteed to be unique. The second item is always the pokemon's type, which in the example above, is the type `grass`.[1] Following that are **one or more** continents where that pokemon can be found. So Bulbasaur can only be found in South America, but Ivysaur can be found in both Asia and Antarctica. Note that the continent names CAN contain spaces. All of the data items on a each line are separated by commas.

Database Format

Your function should open the file given as a parameter and read the data from it into a database. The keys for this database will be pokemon names, and the values will be records (i.e another dictionary) for the matching pokemon.

First, the `read_pokedata()` function should create an empty dictionary to be the overall database.

Then, for each pokemon from the input file, create one record (i.e. one dictionary) with the following fields:

- The pokemon's name, as a string.
- The pokemon's type, as a string.
- The locations where the pokemon can be found, as a **list** of strings.

This record should then be added to the database, using the pokemon's name as a key.

Once all of the records have been added, the function should return the database dictionary.

Evaluation

- 3 marks for correct code
- 1 mark for careful match to the specifications

[1] Yes, we are aware that in fact Pokemon can have more than one type.

Part 2 (2 marks)

Write a function called `find_continents()` which takes as parameter(s):

- A dictionary in the format of the pokemon database as described in Q1

This function should construct and **return** a **list** of all the different continents where it is possible to find pokemon. Each entry in the list should be unique; in other words, don't add the same continent twice even if you can find multiple kinds of pokemon there. For the input file you were given, the resulting list should be (though not necessarily in this order):

```
['Oceania', 'Asia', 'South America', 'North America', 'Africa',
 'Antarctica', 'Europe']
```

Also, note that this function **must not change** the pokemon database in any way.

Evaluation

- 2 marks for correctness

Part 3 (2 marks)

Write a function called `pokemon_in_continent()` which takes as parameter(s):

- A dictionary in the format of the pokemon database as described in Q1
- The name of a continent, as a string

The function should construct and **return** a **list** containing the **names** of all of the pokemon that can be found on the given continent. For example, for the input file you were given, if the continent is `"Africa"`, the resulting list should be (though not necessarily in this order):

```
['gyarados', 'nidorina', 'magmar', 'jigglypuff', 'machoke', 'machamp',
'dragonair', 'slowbro', 'machop', 'venomoth', 'tentacruel', 'parasect',
'omanyte', 'poliwrath', 'muk', 'hitmonlee', 'vaporeon', 'tangela', 'gengar',
'nidoranmale', 'venusaur', 'onix', 'nidoking', 'golem', 'aerodactyl',
'tauros', 'metapod', 'electrode', 'lapras', 'psyduck', 'blastoise',
'nidoqueen', 'alakazam', 'cubone', 'magikarp', 'kabutops', 'kakuna',
'poliwag', 'marowak', 'rhydon', 'kangaskhan', 'zubat', 'dodrio',
 'persian', 'pikachu', 'meowth', 'drowzee', 'gastly']
```

Also, note that this function **must not change** the pokemon database in any way.

Evaluation

- 2 marks for correctness

Part 4 (2 marks)

Write a function called `count_types()` which takes as parameter(s):

- A dictionary in the format of the pokemon database as described in Q1
- A list of pokemon names (as strings)

This function should construct and **return** a **dictionary** where each key is a **pokemon type** (e.g. `grass` or `fire`, etc...) and the value for that key is the number of pokemon of that type in the given list of pokemon names. Recall that **pokemon type** was one of the fields for each pokemon record in the database.

For example, using the sample list of all of the pokemon from `"Africa"` (see Q3 to see that list), the resulting dictionary should be:

```
{'dragon': 1, 'rock': 4, 'normal': 4, 'flying': 3, 'fire': 1, 'electric': 2,
 'bug': 3, 'ghost': 2, 'water': 9, 'poison': 6, 'fairy': 1, 'psychic': 3,
 'ground': 3, 'grass': 2, 'fighting': 4}
```

Also, note that this function **must not change** the pokemon database or the given list of pokemon names in any way.

Evaluation

- 2 marks for correctness

Part 5 (3 marks)

Now it's time to put all your functions together to print out a summary report of how many and what types of pokemon you can find on each continent.

In the main part of your program, write code that will **print to the console** the name of each continent, the TOTAL number of different pokemon that can be found on that continent, followed by the counts for each pokemon type on that continent.

For example, the format for your output might look something like this:

```
Africa: 48 total pokemon
{'ground': 3, 'fire': 1, 'fairy': 1, 'water': 9, 'ghost': 2, 'fighting': 4,
'flying': 3, 'dragon': 1, 'electric': 2, 'grass': 2, 'normal': 4,
'poison': 6,'psychic': 3, 'rock': 4, 'bug': 3}

South America: 55 total pokemon
{'ground': 5, 'fire': 2, 'water': 10, 'fighting': 2, 'flying': 4,
'dragon': 1,  'electric': 5, 'grass': 4, 'normal': 3,
'poison': 8, 'bug': 6, 'rock': 2,  'psychic': 3}
```

There will be more continents, but we're only showing two here to save space.

To do this, you will need to make use of all of the functions you have written so far and make good use of their inputs and outputs. If you've done everything correctly, this part of your program won't require a lot of code: just a few function calls and a single for-loop should be enough.

Evaluation

- 2 marks: For making correct and useful function calls to each of the functions from Parts 1 through 4
- 1 mark: For correct output printed to the console.

Question 4 (4 points):

Purpose: To practice testing the parts of a working program

Write a **test driver** that includes tests for all of the functions from the Pokemon Locations question.

You should **test each function** for parts 1 through 4 independently. This means that for each function, you must invent a variety of test inputs for which you can calculate the correct expected output by hand. You can put all of these tests in the same python code file, but make sure this file is well-organized with regard to where each function is being tested.

Part 5 of the Pokemon Locations question did not involve writing a separate function but instead simply required calling the other functions in sequence. As such, you do not need to do any formal testing for Part 5 here. That's part of the advantage of decomposing a problem into many reasonably small functions!

Below are some tips on creating your test cases for each of the four functions.

- **Testing Part 1 (`read_pokedata()`)**: Since this function reads data from a file, your test inputs must in fact consist of different files! Thus, for each test, create an input file with the same format as `pokemonLocations.txt`. Your test files will likely be MUCH smaller than the provided file. Do not bother testing with files that are not correctly formatted, focus instead on the diversity of possible valid files.
- **Testing Part 2 (`find_continents()`)**: The inputs to this function is a database of Pokemon information. It is ok to use your `read_pokedata()` function to create the database rather than typing in database literals (so long as you confident that function works thanks to testing it!), in which case your test inputs will again consist of input files. Careful! The pokemon in your database might be processed in any order (because it's a dictionary!) and we don't care about the ORDER of the continents in the resulting list. You can deal with this by SORTING the resulting list before comparing it to the expected result!
- **Testing Part 3 (`pokemon_in_continent()`)**: Again this function requires a database, and it is again ok to use your `read_pokedata()` funtion to create it if you find this less tedious. The continent name for each test should however be picked manually. And again, you'll want to sort the resulting list before testing.
- **Testing Part 4 (`count_types()`)**: Again this function requires a database, and it is again ok to use your `read_pokedata()` funtion to create it if you find this less tedious. The list of pokemon names should however be created manually.

Note that it is possible to get full marks for this question even if you did not complete (or attempt) the Pokemon Locations question, so long as you understand WHAT each function was supposed to do and test it adequately.

What to Hand In

- Your test driver in a SINGLE `a9q4.py`, with code for testing **ALL** of the different functions. It's probably best if this file IMPORTS the functions you are testing, i.e `import YOURFILENAME as Q`
- All of the input files that you created to be used in your tests

Evaluation

- 4 marks (1 per function): Function is tested with a thoughtful and adequate number of tests

Question 5 (3 points):

All programs that you submit for this assignment will be assessed with regard to their **style and readability**. Make sure that all of your code:

- includes your name, nsid, and student number as comments
- uses informative variable names
- makes effective use of formatting and white space
- includes a `docstring` for all **function definitions** in the program
- for longer programs, uses helpful and concise single-line comments where appropriate

What to Hand In

There is nothing to hand in for this question. It's simply describing how the rest of your work will be evaluated with regard to style and readability. It is possible to earn these marks even if you don't hand in every other question in this assignment, but you must hand in enough work such that your style and readability can be adequately assessed.

Evaluation

- 3 marks for consistently following the principles listed above

Question 6 (2 points):

In your submission for this assignment, you have an opportunity to earn something we'll call "the wow factor". To earn the wow factor, you must do something creative that goes beyond the expectations laid out in the assignment. This could be an extra feature for one of the previous questions, or it could be an extra little program that you submit that builds on the skills used in the assignment.

You should not aim to do a lot of work for this, but should instead demonstrate in a simple and effective way the depth of your understanding. In short, take ownership of your work and "impress us".

Telling you exactly what to do to earn "the wow factor" would defeat the entire purpose, but here are some very general ideas.

- improve the aesthetic appeal of your program(s)
- improve the user-friendliness of your program(s)
- remove a simplifying assumption and handle the resulting added complexity
- compare two different ways of doing the same task, and submitting data/analysis on which was better
- just doing something generally cool

One quick warning: using fancy extra code libraries in order to AVOID a key learning objective from the assignment is not impressive, it's actually anti-impressive. Focus instead on solving problems using simple clean foundational programming skills. Using extra libraries is fine if they let you do something BEYOND what the assignment asked, but not if they're replacing a core assigned task.

What to Hand In

A plain text file called `wow.txt` describing what you did that you think merits the wow factor. One or two sentences is enough. If you create any additional code files, hand those in too.

Evaluation

- 2 marks for something impressive