# CS 241 DATA STRUCTURES FALL 2021 MIDTERM II

*Instructor Calvin Deutschbein*

| | |
|---|---|
| Roster Name | |
| Sign here to affirm the Honor Code | |

This exam will be timed to take 60 Minutes.

It will be scored out of 200 Points.

It will make up 20% of Final Grade.

## SECTION I: PYTHON 40 Points

*Part 1: Multiple Choice: 4 Questions @ 5 Points each = 20 Points*

Which of the following creates an empty **dictionary** using the built-in Python set type?

A. >>> d = []
B. >>> d = {}
C. >>> d = ()
D. >>> d = “”

Which of the following creates an empty **tuple** using the built-in Python list type?

A. >>> t = []
B. >>> t = {}
C. >>> t = ()
D. >>> t = “”

Which of the following types cannot be added to a dictionary?

A. Boolean
B. Integer
C. List
D. String

For which of the following types are elements not stored in order?

A. List
B. Set
C. String
D. Tuple

*Part 2: Short Response: 2 Questions @ 10 Points each = 20 Points*

Python classes may contain “dunder methods” such as __init__(), __str__(), or __hash__(). What benefits does providing a __hash__() dunder method provide to a given class?

You may wish to discuss sets in your answer.

Suppose you had a file that contained one million entries in some sorted order and wished to add new entries in sorted order to this data set. How would you assess how many new elements could be added at once before re-sorting the entire data set rather than simply adding new elements in sorted order?

You may wish to discuss the complexity of both lists and sorts in your answer.

**SECTION II: ORGANIZING DATA** **40 Points**

*Part 3: True/False:* *4 Questions @ 5 Points each =* *20 Points*

Specify whether the code snippets return boolean values True or False.

```
>>> len([1]) <= 1
```

A. True
B. False

```
>>> type(set()) == type([])
```

A. True
B. False

```
>>> if True:
...       True
...   else:
...       False
```

A. True
B. False

```
>>> if False:
...       True
...   else:
...       False
```

A. True
B. False

*Part 4: Free Response:* *20 Points*

Bubble sort is an algorithm to sort lists by

- Stepping through the list
- Swapping adjacent elements that are not in order
- Repeating this process until the list is sorted

```
def bsort(l):
    def swap(i,j):
        if l[i] > l[j]:
            l[i], l[j] = l[j], l[i]
            return True
        return False
    swapping = True
    while swapping:
        swapping = False
        for i in range(len(l) - 1):
            swapping = swapping or swap(i,i+1)
    return l
```

How would you expect the performance of bubble sort to compare to insertion sort, quick sort, and merge sort? Consider the idea of “Big O notation” when you answer. Consider best, worst, and average cases.

**SECTION III: SORTING DATA** **70 Points**

*Part 5: Multiple Choice:* *4 Questions @ 5 Points each =* *20 Points*

Consider a map of different subregions that border one another, such as states in the U.S., shaded in different colors. To represent this as a graph, the subregions would be represented as:

A. Directions
B. Labels
C. Vertices
D. Weights

Consider a map of different subregions that border one another, such as states in the U.S., shaded in different colors. To represent this as a graph, the **colors of** the subregions would be represented as:

A. Directions
B. Labels
C. Vertices
D. Weights

Consider a map of highways connecting cities. To represent this as a graph, travel times along highways would be represented as:

A. Directions
B. Labels
C. Vertices
D. Weights

Consider a program of a study dependency graph, such as the prerequisite tree for the Data Science major. Which of two connected courses should be completed first would be represented as:

A. Directions
B. Labels
C. Vertices
D. Weights

*Part 6: Written Response: 2 Questions @ 20 Points each = 40 Points*

Describe in brief an *efficient* algorithm for computing whether a graph is *connected*. Recall a graph is connected if and only if there exists a path between any two vertices in the graph.

Describe in brief why your algorithm is *efficient*. You may wish to make a comparison to inefficient techniques such as exhaustively checking every vertex.

## SECTION IV: ALGORITHMS 60 Points

*Part 7: Computing Exercises 2 Questions @ 30 Points each = 60 Points*

MergeSort is a sorting algorithm with optimal worst case time complexity. It sorts a list using comparisons (such as “<”) by dividing lists in half, recursively applying MergeSort to each half, and then “merging” the lists back together. Often merging is done by considering the first element of the two recursively sorted lists, then popping this element off the appropriate list and appending it to a new working list.

Implement MergeSort in Python. While you may use any list implementation to do so, I recommend using the built-in Python list type. You may use any built-in Python functions other sort() and sorted(). MergeSort should accept as input any list and return a list containing the same elements but in sorted order.

Consider the graph represented by the following universities with graduate programs in Oregon:

```
Willamette U, Portland State U, I-5, 62
Willamette U, U of Oregon, I-5, 66
Willamette U, Oregon State U, OR-34, 43
Willamette U, Pacific U, OR-221, 66
Willamette U, George Fox U, River Rd, 45
U of Oregon, Oregon State U, I-5, 51
Pacific U, Portland State U, US-26, 37
Pacific U, George Fox U, OR-47, 29
Portland State U, George Fox U, Pacific Hwy W, 40
```

Given this graph, answer:

1. What is the shortest distance between U of Oregon and Pacific U?
2. What is the path that gives this shortest distance?
3. How do you know this is the shortest distance?