22C:16 Final Exam

May 7th, 5:30 pm to 7:30 pm

Instructions:

- This is an open notes exam and you have 2 hours to complete it. There are 4 problems in the exam and these appear on 9 pages.
- Make sure that you do not have any electronic devices (phones, calculators, etc.) on your desk; you are not allowed to access these during your exam.
- Write as neatly as you can.
- Show all your work, if you want to receive partial credit.

Name:

Circle your section:

SCA (M, W evening)	A01 (9:30-10:20)	A02 (11-11:50)	A03 (12:30-1:20)
	A04 (12:30-1:20)	A05 (2-2:50)	A06 (3:30-4:20)

---

1. **(50 points) On Lists and Strings**.

   (a) **(10 points)** Given the list `L = ["silent", "lives", ["dormitory", "Elvis"], "library", "dirty room", "listen"]` evaluate the following expresssions:

   (i) `L[1][3]`

   (ii) `L[3][2:7]`

   (iii) `"Elvis" in L`

(iv) `L[2][0][1::3]`

(v) `"+".join(L[::3])`

(b) **(15 points)** Anagrams are words in which the same letters appear the same number of times (not counting spaces) but in possibly different positions. For example, `"dormitory"` and `"dirty room"` are anagrams.

A useful way to identify anagrams is to compute a cannonical key that all strings that are anagrams share. This can be done by removing all blanks from the string, translating to lower case, and sorting the remaining letters. For example, the cannonical key for `"dormitory"` and `"dirty room"` is `"dimoorrty"`, while the canonical key for `"Elvis"` and `"lives"` is `"eilsv"`.

Write a function called `anagramKey` that takes a string `s` as its argument and returns its cannonical key as described above.

(c) **(10 points)** What does the function `enrich` given below return when it is called with the list `M` as its argument?

```
M = ["silent", "lives", "dormitory", "listen"]

def enrich(L) :
    result = []
    for i in range(len(L)) :
        result.append(L[i])
        for j in range(i+1, len(L), 2) :
            result.append(L[i] + " " + L[j])
    return result
```

(d) **(15 points)** Write a function called `findAnagrams` that takes a list of strings and prints all pairs of strings that are anagrams. Your function should call the function `anagramKey` defined earlier (part (b)) and should print each pair of anagrams it finds on a new line. Note that your function does not need to return anything, only print output.

2. **(50 points) On Dictionaries**

(a) **(5 points)** Let D1 and D2 be the following dictionaries:

```
D1 = {12:"tape", 76:"rang", 23:"phone", 25:"okay", 37:"early"}
D2 = {"okay":17, "early":34, "rang":12, "phone":76, "tape":23}
```

(i) What is `D1[D2["rang"]]`?

(ii) What is `D2[D1[76]]`?

(b) **(15 points)** Given the function:

```
def trace(x) :
    while (x in D1 or x in D2) :
        print x
        if x in D1:
            x = D1[x]
        elif x in D2:
            x = D2[x]
```

(i) What is the output of `trace("tape")`?

(ii) What is the output of `trace("phone")`?

(c) **(15 points)** Add code to the function `trace` in the locations indicated by underscores so that it stops tracing if it enters an infinite loop,

```
def trace(x) :

    ______________
    while (x in D1 or x in D2) :

        if ______________________
            break


        _______________
        print x
        if x in D1:
            x = D1[x]
        elif x in D2:
            x = D2[x]
```

(d) **(15 points)** A function $f : X \to Y$ and a function $g : Y \to X$ are said to be inverses if for all x in X, $g(f(x)) = x$ and for all y in Y, $f(g(y)) = y$. Write a function `isInverse` that takes two dictionaries as its parameters and returns `True` if they are inverses of each other and `False` if they are not. Note that a dictionary is just a function that maps its set of keys to its set of values.

3. **(50 points) On recursion**

(a) **(15 points)** We want to write a recursive function called `rangeSum` that takes two positive integers, a and b, such that $a \leq b$, and computes the sum $a + (a + 1) + (a + 2) + \cdots + b$. For example, the call `rangeSum(3, 6)` would return 18 (since 3 + 4 + 5 + 6 = 18). Consider the following attempt at implementing `rangeSum`:

```
def rangeSum(a, b):
    if a == b:
        return a
    return rangeSum(a, (a+b)/2) + rangeSum((a+b)/2, b)
```

Suppose we add a main program to the above function that calls the function as:

```
print rangeSum(3, 4)
```

It turns out that we don't get the expected output 7 (since 3 + 4 = 7) and instead something strange happens. Explain in 1-2 sentences *why* we don't get the expected output of 7.

(b) **(15 points)** Rewrite the above function by changing one of the lines *slightly* so that it works correctly.

(c) **(20 points)** Here is the implementation of *binary search* that we discussed in class.

```
def recursiveBinarySearch(L, k, left, right):
    if left > right:
        return False
    mid = (left + right)/2
    if k == L[mid]:
        return True
    if k < L[mid]:
        return recursiveBinarySearch(L, k, left, mid-1)
    if k > L[mid]:
        return recursiveBinarySearch(L, k, mid+1, right)
```

As implemented, this function returns `True` or `False` depending on whether the element `k` is in list `L` or not.

Modify the implementation so that if `k` is in `L` then the function returns the position of `k` in `L`; otherwise if `k` is not in `L` then the function should return a position `i` such that elements in `L` that appear before position `i` are all strictly smaller than `k` and elements in `L` that appear at or after position `i` are all strictly larger than `k`.

For example, suppose that `L = [3, 5, 13, 18, 18, 27, 34, 56, 57, 60, 89]`. If `k` were `18` then the function should return 3 or 4 (both of these are correct). If `k` were `59` then the function should return 9.

4. **(50 points) On classes** Define a class called `priorityQueue`. Each instance of `priorityQueue` is a collection of items, where each item x has an associated positive integer that we refer to as the *priority* of x. In addition to the `__init__` and `__repr__` methods, the `priorityQueue` class is required to provide the following methods:

   - `insert`: Given an element `x` with priority `p`, this method inserts the element `x` with priority `p` into the collection.
   - `get`: This method deletes the element with highest priority from the collection and returns that element. If there is a tie, i.e., if there are multiple elements with the same priority, then an arbitrary element with highest priority is deleted and returned.

   Here is an example of how the `priorityQueue` class might be used:

```
>>> Q = priorityQueue()
>>> Q.insert("text", 10)
>>> Q
text 10
>>> Q.insert("hello", 18)
>>> Q
text 10, hello 18
>>> Q.get()
hello
>>> Q
text 10
```

   Below we provide the implementation of the constructor and the `insert` method.

```
class priorityQueue():

    def __init__(self):
        self.items = []
        self.priorities = []

    def insert(self, x, p):
        self.items.append(x)
        self.priorities.append(p)
```

   (a) **(15 points)** Implement the `get` method.

(b) **(15 points)** Implement the `__repr__` method. As you can see from the example in the previous page, this method produces a string representation of the instance of the `priorityQueue` class consisting of items in the instance along with their priorities. The order in which items appear in the string does not matter.

(c) **(20 points)** Now suppose that we change the implementation of the `priorityQueue` class to be dictionary-based rather than list-based. Specifically, here is the new implementation of the constructor and the `insert` methods.

```
class priorityQueue():
    def __init__(self):
        self.items = {}

    def insert(self, x, p):
        self.items[p] = x
```

Implement the `get` method in the space below for this new dictionary-based implementation.