CSc 110 Sample Midterm Exam #2

1. Collections Mystery

Consider the following function:

```
def mystery(m):
     s = set()
     for key in m.keys():
          if (m[key] != key):
               s.add(m[key])
          else:
               s.add(m[key][0])
     print(s)
```

Write the output that is printed when the function above is passed each of the following dictionaries as its parameter.

```
{'sheep':'wool', 'house':'brick', 'cast':'plaster', 'wool':'wool'}
```

____

```
{'munchkin':'blue','winkie':'yellow','corn':'yellow','grass':'green','emerald':'green'}
```

____

```
{'pumpkin':'peach','corn':'apple','apple':'apple','pie':'fruit','peach':'peach'}
```

____

```
{'lab':'ipl','lion':'cat','terrier':'dog','cat':'cat','platypus':'animal','nyan':'cat'}
```

____

2. 2D List Mystery

Consider the following function:

```
def mystery(data):
    result = set()
    for i in range(1, len(data)):
      for j in range(0, len(data[i]) - 1):
          result.add(data[i][j] - 1)
    return result
```

In the left-hand column below are specific two-dimensional lists. You are to indicate in the right-hand column what values would be stored in the set returned by function mystery if the list in the left-hand column is passed as a parameter to mystery.

Two-Dimensional List	**Contents of List Returned**
`[[0, 1], [2, 3]]`	______________________________
`[[0, 1, 2], [3, 4, 5], [6, 7, 8]]`	______________________________
`[[3, 4], [1, 2, 3], [], [5, 6]]`	______________________________
`[[4, 5], [1, 2, 3, 3, 2, 1], [2, 2, 3]]`	______________________________

3. Searching and Sorting.

(a) Suppose we are performing a **binary search** on a sorted list called `numbers` initialized as follows:

```
# index      0    1    2    3    4    5    6    7    8    9   10   11   12   13
numbers = [-2,   0,   1,   7,   9,  16,  19,  28,  31,  40,  52,  68,  85,  99]

# search for the value 5
index = binary_search(numbers, 5)
```

Write the indexes of the elements that would be examined by the binary search (the `mid` values in our algorithm's code) and write the value that would be returned from the search. Assume that we are using the binary search algorithm shown in lecture and section.

- Indexes examined: ____
- Value Returned: ___________________________

(b) Write the state of the elements of the list below after each of the first 3 passes of the outermost loop of the selection sort algorithm.

```
numbers = [63, 9, 45, 72, 27, 18, 54, 36]
selection_sort(numbers)
```

(c) Trace the complete execution of the merge sort algorithm when called on the list below, similarly to the example trace of merge sort shown in the lecture slides. Show the sub-lists that are created by the algorithm and show the merging of sub-lists into larger sorted lists.

```
numbers = [63, 9, 45, 72, 27, 18, 54, 36]
merge_sort(numbers)
```

4. String Programming

Write a static method called `encode` that takes a string s and an integer n as parameters and that returns a new string that scrambles the order of the characters from s in a particular way. Taking the characters from s in order, imagine filling a grid of n rows column by column. When s is "abcdefghijklmnopqrstuvwxyz" and n is 3, you get:

row 1:	a d g j m p s v y
row 2:	b e h k n q t w z
row 3:	c f i l o r u x

The method should return the result of concatenating these characters together with row 1 first, then row 2, and then row 3. Notice that the final column will not necessarily be complete, as in the example above where the final column has only two characters. So if you make the call:

```
encode("abcdefghijklmnopqrstuvwxyz", 3)
```

You should get back the string "`adgjmpsvybehknqtwzcfilorux`". The string might contain any characters, including spaces. For example, the call:

```
encode("four score and seven", 4)
```

returns "`f rneosedvuc eroasn`" because the following grid would be produced:

row 1:	f r n e
row 2:	o s e d v
row 3:	u c e
row 4:	r o a s n

You may assume that the string passed as a parameter is not empty and that the integer passed as a parameter is greater than or equal to 1 and less than the length of the string. You are not allowed to construct any structured objects other than strings to solve this problem (no list, list of lists, etc).

5. List of Lists Programming

Write a function called `num_unique` that takes a list of lists as a parameter and returns the number of unique values stored in it. For example, if you have the following list of lists:

```
lis = [[1, 2, 3], [4, 3, 2, 1], [6, 7, 7], [8]]
```

a call to `num_unique(lis)` should return 7. You may create one other data structure to help you solve this problem.

6. Collections Programming

Write a function `grade_stats` that accepts as a parameter a list representing the grades given in a course and returns a dictionary mapping each grade to the count of how many were given. For example, if a list `grades` contains the following:

```
['A', 'I', 'C', 'C', 'E', 'B', 'A', 'E', 'E', 'A', 'B', 'B', 'B']
```

The call of `grade_stats(grades)` should return the following dictionary:

```
{'A':3, 'I':1, 'C':2, 'E':3, 'B':4}
```

Note that the grades may not always be A, B, C, D, E and I. For example, A- is a valid grade some places as is 3.7. You should treat whatever is in the passed in list as a valid grade.
If the passed in list is empty, your function should return an empty dictionary.

7. Collections Programming

Write a function `union` that accepts two dictionaries (whose keys and values are both integers) as parameters, and returns a new dictionary that represents a merged union of the two original dictionaries. For example, if two dictionaries `m1` and `m2` contain these pairs:

```
{7=1, 18=5, 42=3, 76=10, 98=2, 234=50}      m1
{7=2, 11=9, 42=-12, 98=4, 234=0, 9999=3}   m2
```

The call of `union(m1, m2)` should return a dictionary that contains the following pairs:

```
{7=3, 11=9, 18=5, 42=-9, 76=10, 98=6, 234=50, 9999=3}
```

The "union" of two dictionaries *m1* and *m2* is a new dictionary that contains every key from *m1* and every key from *m2*. Each value stored in your "union" map should be the sum of the corresponding value(s) for that key in *m1* and *m2*, or if the key exists in only one of the two maps, that map's corresponding value should be used. For example, in the maps above, the key 98 exists in both maps, so the result contains the sum of its values from the two maps, 2 + 4 = 6. The key 9999 exists in only one of the two maps, so its sole value of 3 is stored as its value in the result map.

Either dictionary passed in (or both) could be empty. Though the pairs are shown in sorted order by key above, you should not assume that the dictionaries passed to you store their keys in sorted order.

You may create one collection of your choice as auxiliary storage to solve this problem. You can have as many simple variables as you like. You should not modify the contents of the dictionaries passed to your function.

8. List Programming

Write a function named `longest_sorted_sequence` that accepts an list of integers as a parameter and that returns the length of the longest sorted (nondecreasing) sequence of integers in the list. For example, if a variable named `lis` stores the following values:

```
lis = [3, 8, 10, 1, 9, 14, -3, 0, 14, 207, 56, 98, 12]
```

then the call of `longest_sorted_sequence(lis)` should return `4` because the longest sorted sequence in the list has four values in it (the sequence -3, 0, 14, 207). Notice that sorted means nondecreasing, which means that the sequence could contain duplicates. For example, if the list stores the following values:

```
lis2 = [17, 42, 3, 5, 5, 5, 8, 2, 4, 6, 1, 19]
```

Then the method would return `5` for the length of the longest sequence (the sequence 3, 5, 5, 5, 8). Your method should return 0 if passed an empty list. Your method should return 1 if passed an list that is entirely in decreasing order or contains only one element.

9. Programming

Write a function called `split_pairs` that takes a list of integers as a parameter and that returns a new list containing the result of splitting successive pairs of numbers so that the first values from each pair appear first followed by the second values from each pair. For example, suppose that a variable called `list` stores the following:

```
        [3, 8, 4, 9, 7, 2]
```

This list has three pairs: (3, 8), (4, 9), and (7, 2). Thus, the call `split_pairs(list)` should return the following list:

```
        [3, 4, 7, 8, 9, 2]
```

Notice that this list contains the first values from each pair (3, 4, 7) followed by the second values from each pair (8, 9, 2). If there is an extra value that is not part of a pair, then it should be included with the first set of values in the new list. For example, if list stores:

```
        [7, 5, 3, 2, 8, 4, 6]
```

then the call `split_pairs(list)` should return:

```
        [7, 3, 8, 6, 5, 2, 4]
```

The value 6 in the original list is not part of a pair. Notice that the new list has the first values from each pair (7, 3, 8) followed by 6 followed by the second values from each pair (5, 2, 4).
The function should not construct any extra data structures other than the list to be returned andit should not alter its parameter.

CSc 110 Sample Midterm Exam #2 Solutions

1. Collections Mystery

```
{'plaster', 'wool', 'brick', 'w'}
{'yellow', 'blue', 'green'}
{'a', 'p', 'peach', 'apple', 'fruit'}
{'cat', 'ipl', 'animal', 'c', 'dog'}
```

2. Inheritance Mystery

```
{1}
{2, 3, 5, 6}
{0, 1, 4}
{0, 1, 2}
```

3. Searching and Sorting

(a) Indexes examined: 6, 2, 4, 3 Value returned: -4

(b)
```
[9, 63, 45, 72, 27, 18, 54, 36]
[9, 18, 45, 72, 27, 63, 54, 36]
[9, 18, 27, 72, 45, 63, 54, 36]
```

(c)
```
[63,   9,   45,  72,   27,  18,  54,  36]
[63,   9,   45,  72]  [27,  18,  54,  36]      split
[63,   9]  [45,  72]  [27,  18] [54,  36]      split
[63]  [9]  [45] [72]  [27] [18] [54] [36]      split
[9,  63]  [45,  72]  [18,  27] [36,  54]       merge
[9,  45,   63,  72]  [18,  27,  36,  54]       merge
[9,  18,   27,  36,   45,  54,  63,  72]       merge
```

4. String Programming

```
def encode(s, n):
    result = ""
      for j in range(0, n):
          for i in range(0, len(s) - j, n):
              result += s[i + j]
    return result
```

5. List of Lists Programming

```
def num_unique(lis):
    unique = set()
    for i in range(0, len(lis)):
        for j in range(0, len(lis[i])):
            unique.add(lis[i][j])
    return len(unique)
```

6. Collections Programming

```
def grade_stats(lis):
    grades = {}
    for grade in lis:
        if grade in grades:
            grades[grade] = grades[grade] + 1
        else:
            grades[grade] = 1
    return grades
```

7. Collections Programming

```
def union(m1, m2):
    result = {}
    for key, value in m1.items():
        result[key] = value
    for key, value in m2.items()):
        if (key in result)
            result[key] = result.get(key) + value
        else:
            result[key] = value
    return result
```

8. List Programming

```
def longest_sorted_sequence(list):
    if (len(list) == 0):
        return 0
    max = 1
    count = 1
    for i in range(1, len(list)):
        if (list[i] >= list[i - 1]):
            count += 1
        else:
            count = 1

        if (count > max):
            max = count
    return max
```

9. Programming

```
def split_pairs(lis):
    half = len(lis) // 2
    result = [0] * len(lis)
    for i in range(0, len(lis)):
        if i % 2 == 0:
            result[i // 2] = lis[i]
        elif len(lis) % 2 == 0:
            result[i // 2 + half] = list[i]
        else:
            result[i // 2 + half + 1] = list[i]
    return result
```