CSCI 134 Final Exam Reference Sheet (Spring 2024)

You are free to use any definitions from homeworks, labs, or lectures. Here is a non-exhaustive summary of useful information.

range(start, stop, step):
   range object generating integers starting at start (by default 0), going up to (but not including) stop, using step size step (by default 1)

Slicing uses a similar structure when defining the range of elements to be sliced. If a is a sequence, then:
   a[start:stop:step] returns a slice containing elements from a starting at index start (by default 0), going up to (but not including) stop, using step size step (by default 1)

Slicing examples using sequence a = "abcdefg":
   a[0:len(a):2] -> "aceg"    a[1:] -> "bcdefg"    a[:3] -> "abc"    a[::1] -> "gfedcba"

Useful string class methods:

s.join(lst) -> str
   s is inserted in between each given string in lst. The result is returned as a new string.
   Ex: " ".join(["hello", "world"]) -> "hello world"

s1.split(sep) -> lst
   Return a list of the substrings in the string, using sep as the separator string
   Ex: "a,b,c".split(" ") -> ["a","b","c"]

s.format(args) -> str
   Return a formatted version of s, using substitutions from args, with substitutions identified by '{' and '}'.
   Ex: "Fill in the {}".format("blank") -> "Fill in the blank"

s.strip() -> str
   Returns a string that is equivalent to s, except all leading and trailing whitespace characters are removed.
   Ex: " goodby spaces! ".strip() -> "goodby spaces!"

Useful list class methods:

L.append(object) -> None
   append object to end of list L

L.extend(iterable) -> None
   extend list L by appending each individual element from the iterable sequence iterable

List comprehensions are unnecessary but compact ways to generate a list using the syntax:
   new_list = [expression for item in sequence]
   Ex: [i for i in range(3)] -> [0, 1, 2]    [i*2 for i in range(3)] -> [0, 2, 4]
Handy algorithms from lecture that you are not expected to memorize:

```python
def binary_search(seq, item, start=0, end=len(seq)-1):
    '''Returns True if item is present in seq'''
    if start > end:
        return False
    mid = (start + end) // 2
    if item == seq[mid]:
        return True
    elif item < seq[mid]:
        return binary_search(seq, item, start, mid-1)
    else:
        return binary_search(seq, item, mid+1, end)

def merge(a, b):
    '''Merges two sorted lists a and b, and returns new merged list c'''
    # initialize variables
    i, j, k = 0, 0, 0
    len_a = len(a)
    len_b = len(b)
    c = []
    # traverse and populate new list
    while i < len_a and j < len_b:
        if a[i] <= b[j]:
            c.append(a[i])
            i += 1
        else:
            c.append(b[j])
            j += 1
    # handle any remaining values if one list was exhausted first
    if i < len_a:
        c.extend(a[i:])
    elif j < len_b:
        c.extend(b[j:])
    return c
```