Name: ______________________________ Andrew Id: ______________

15-112 Spring 2024 Quiz 6

Up to 25 minutes. No calculators, no notes, no books, no computers. Show your work!
Do not use dictionaries, sets, try/except, or recursion on this quiz.

1. (6 points) **Code Tracing**: Indicate what the following program prints. Place your answers (and nothing else) in the box below the code.

```
def ct1(a):
    a = sorted(a)
    b = a
    c = copy.copy(a)
    d = b[:]
    a[0] += 5
    b[1] = 6
    c[2] = 7
    d[3] -= 11
    b.append("Cat")
    print(f"a: {a}")
    print(f"b: {b}")
    print(f"c: {c}")
    print(f"d: {d}")

a = [50, 30, 40, 20]
print(ct1(a))
print(a)
```

2. (6 points) **Free Response**: Sub lists

   Write the function `isSubList(L1, L2)` which, given two lists, returns `True` if `L1` is a sub list of `L2` and `False` otherwise.

   L1 is a sub list of L2 if all of its elements are next to each other and in the same order inside of L2. For example, `[6, 3, 8]` is a sub list of `[1, 6, 3, 8, 5]` because the elements `6, 3, 8` can be found together, and in the same order, in the middle of `[1, 6, 3, 8, 5]`. However, `[6, 3, 8]` is *not* a sub list of `[6, 1, 3, 8, 5]` because even though all of the elements are there in order, they are not next to each other.

   Consider the following test cases:

```
# Normal Cases
assert isSubList([6, 3, 8], [1, 6, 3, 8, 5]) == True
assert isSubList([6, 3, 8], [1, 8, 3, 6, 5]) == False
assert isSubList([6, 3, 8], [6, 1, 3, 8, 5]) == False
assert isSubList([6, 3, 8], [6, 3, 8]) == True
assert isSubList(["cat", "dog", "horse"], ["mouse", "monkey", "cat", "dog", "horse"]) == True
assert isSubList([10, 20], [50, 40, 8, 10, 7]) == False

# Strange case
assert isSubList([], [2, 1, 3, 4, 5]) == True
```

   Hint: Slicing out pieces of `L2` to compare to `L1` can be helpful here.

3. (8 points) **Free Response**: Nearly Sorted

We will say that a list is "nearly-sorted" (a coined term) if it is not sorted but it requires exactly one swap of two of its values to become sorted from least to greatest. For example, `a = [5, 9, 7, 8, 6]` is nearly-sorted, since swapping `a[1]` and `a[4]` results in a sorted list. Similarly, `a = [1, 2, 3]` is not nearly-sorted, since it is already sorted. `a = [4, 3, 2, 1, 10, 8]` is also not nearly-sorted, since there is no single swap that could result in it being sorted.

With this in mind, write the function `checkNearlySorted(L)` that takes a list L of integers, and returns `False` if the list is not nearly sorted. If the list is nearly sorted, the function does not return `True`, but rather it returns the tuple `(i, j)`, where swapping the elements at index `i` and j would result in the list being sorted.

Your function must be non-destructive. Also, do not worry about how efficient your algorithm is.

Consider the following test cases:

```
assert checkNearlySorted([50, 90, 70, 80, 60]) == (1, 4)
assert checkNearlySorted([17, 23, 38]) == False
assert checkNearlySorted([47, 21, 13, 35]) == False
assert checkNearlySorted([41, 22, 31, 13, 58, 63, 79]) == (0, 3)
```

Hint: A good strategy is to check every possible `(i, j)` swap and see if it results in a sorted list.