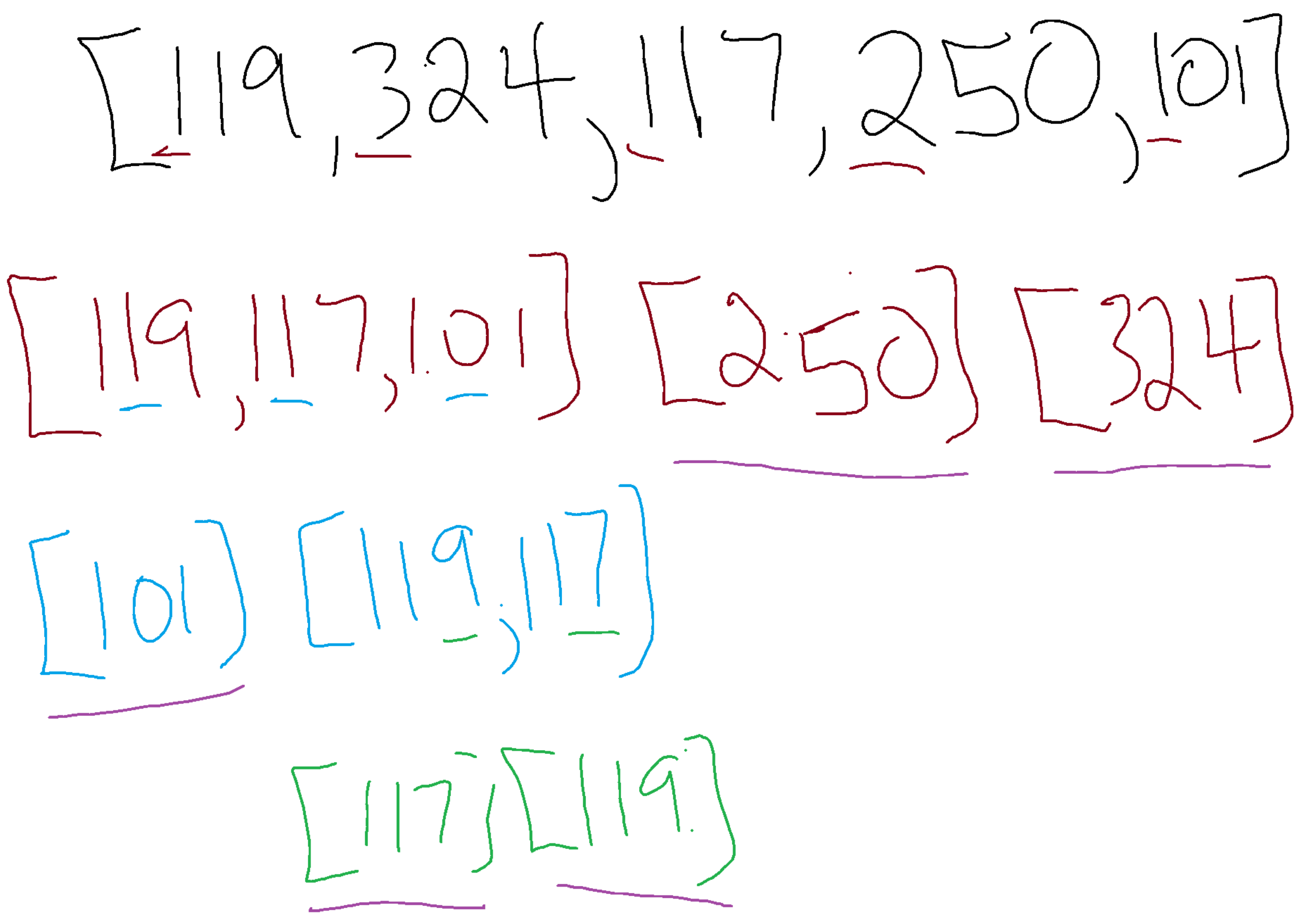
Week 11 - Radix Sort!

As you'll prove in CSC263, if we only allow sorting algorithms that can *compare* and *rearrange* elements of a list, then we actually can't do better than roughly *n*log*n* time for lists of length *n*. However, we can actually do better if we allow some type of structure of the inputs, for example, if we restrict the inputs to be only integers or only strings.

Radix sort

Suppose we have a list of integers. The key idea of radix sort is that we can sort them *digit by digit*, rather than comparing them all at once. There are two variants of radix sort, most-significant digit and least-significant digit; we'll only look at the first variant here.

The idea behind MSD radix sort is to repeatedly sort the numbers into *buckets* by their digits, starting with the digit farthest to the right, and the recursing until the ones digit is reached. (In the diagram below, we use 10 buckets because we humans like to think in decimal; a computer would probably use either 2 or 16 buckets.)



Before looking at the code below, try implementing the algorithm yourself! The basic idea is the same as quicksort, with a few tweaks here and there.

```python
def msd_radix(lst, d=None):
    """ (list of int, int) -> list of int
    Return a new sorted list containing the numbers in lst.
    d is the "next digit to sort by"; assume that the numbers
    in lst agree on all of their digits to the right of the d-th
    digit.
    """
    if len(lst) < 2 or d == 0:
        return lst[:]
    else:
```

```
    if d is None:
        # Determine the maximum number of digits in the numbers
        d = 0
        for num in lst:
            d = max(d, math.floor(math.log10(num)) + 1)

    # Make 10 buckets
    buckets = [[], [], [], [], [], [], [], [], [], [], []]

    # Partition numbers into buckets depending on the d-th digit
    for num in lst:
        # Calculate d-th digit
        digit = (num // 10 ** (d-1)) % 10
        # Add num to corresponding bucket
        buckets[digit].append(num)

    # Recursively sort each bucket and add it to the sorted list
    sorted_lst = []
    for bucket in buckets:
        sorted_lst = sorted_lst + msd_radix(bucket, d - 1)

    return sorted_lst
```

Efficiency

Note that just like quicksort, the parititioning step here takes linear time (*n* steps, where *n* is the size of the list). Moveover, if we now consider recursing on each of the 10 buckets, even though we don't know the size of each individual bucket, we know that their total size is still *n*, because every item had to go somewhere! This means that the total partitioning cost for the second layer in our tree is still *n*, and in fact this is true for every layer.

So then the question is how many layers our recursion tree has; put another way, what is the *recursion depth* of radix sort? Given that we recurse on the parameter `d`, it's not too surprising that the maximum recursion depth is *d*, the largest number of digits of a number in the input list.

Putting these observations together, we see that the total running time of radix sort is *nd*, which is *much* faster even than *n*log*n* if the number of digits *d* is very small compared to *n* (imagine sorting one billion 3-digit numbers).

Reminder: Limitations of Radix Sort

Unlike all the sorting algorithms we've studied so far, radix sort does not simply rely on comparing elements and reordering them in a list. Instead, it uses some special property of numbers - that they have digits - that is not true of all objects we would want to sort.

So be warned: radix sort can be extremely fast, but it doesn't work on all inputs!

Bogosort (Optional)

This one's just for fun:

```
import random

def bogosort(lst):
    """ (list) -> NoneType
    Sort lst using the bogosort algorithm.
    """

    while not is_sorted(lst):
        random.shuffle(lst)

def is_sorted(lst):
    """ (list) -> bool
    Return True if list is sorted.
    """
    for i in range(len(lst) - 1):
        if lst[i] > lst[i+1]:
            return False
    return True
```

And this one's even better:

```
def bogobogosort(lst):
    """ (list) -> NoneType

    Sort lst using the bogobogosort algorithm.
    """
    if len(lst) < 2:
        pass
    else:
        while not is_sorted(lst):
            lst_copy = lst[:-1]
            bogobogosort(lst_copy)
            if lst_copy[-1] < lst[:-1]:
                lst[:-1] = lst_copy[:]
                break
            else:
                random.shuffle(lst)
```

Computer Science
UNIVERSITY OF TORONTO

David Liu (liudavid at cs dot toronto dot edu)
Come find me in BA4260
Site design by Deyu Wang (dyw999 at gmail dot com)