Assignment 1: Answers
Patrick El-Hage
250650822

1. Function growth rates in increasing order
    1. 2^{10}
    2. $4n$
    3. $2^{\log n}$
    4. $3n + 100\log n$
    5. $n \log n$
    6. $4n \log n + 2n$
    7. $n^2 + 10n$
    8. n^3
    9. 2^n

2. Logarithmic - log(n) – this is for the first part of the code, and it is the dominant segment.

3. Below are the lists of most frequent letters and digrams in the CIA text. The program took, on a rough average estimate, around 4 seconds to run. The digrams utilize a 2d array to store the count, while the letters use a 1d array to store the count. The indexes of the arrays correspond to the indexes of the letters in the alphabet:

    1. According to my python program, the following are the **10 most frequent letters** in the 1995 CIA World Fact Book:
        1. e – 213,204
        2. a – 205,925
        3. n – 163,704
        4. i – 160,773
        5. t – 157,275
        6. o – 147,642
        7. r – 143,109
        8. s – 125,545
        9. l – 97,400
        10. c – 80,434

    2. According to my python program, the following are the **10 most frequent digrams** in the 1995 CIA World Fact Book:
        1. an – 39,095
        2. on – 37,350
        3. er – 31,806
        4. in – 30,239

5. al – 27,834
6. at – 27,697
7. ti – 27,091
8. es – 26,612
9. te – 24,337
10. re – 23,236

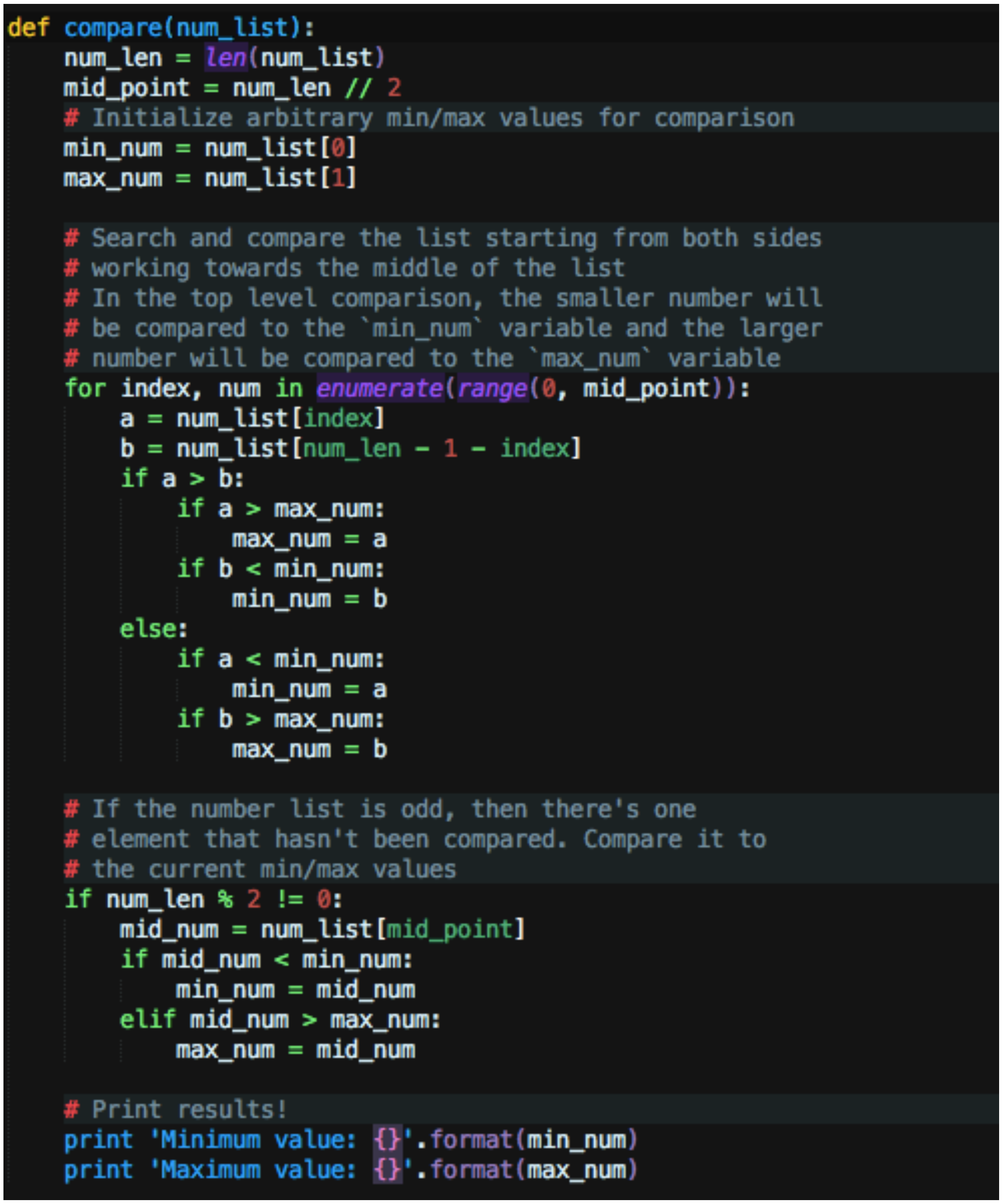
4. **Bonus:**
I initialized `min_num` and `max_num` variables that keep track of the current min/max values throughout the comparisons.

The algorithm starts by going through the list of numbers, starting from both sides of the list (the outside) and moving towards the middle (i.e. comparing first to last, second to second to last, third number to third to last number, etc).

Each number on the left side is compared once to a corresponding number on the right side. The larger number of the two is compared to the current `max_num` value, while the smaller number is compared to the current `min_num` value. If the values are smaller/larger than the `min_num`/`max_num` numbers, then they become the current `min_num`/`max_num` values.

If the number of numbers in the list is odd, then the middle item hasn't been compared yet. So therefore it gets compared to the current min/max values.

The following is the code in action:



```python
def compare(num_list):
    num_len = len(num_list)
    mid_point = num_len // 2
    # Initialize arbitrary min/max values for comparison
    min_num = num_list[0]
    max_num = num_list[1]

    # Search and compare the list starting from both sides
    # working towards the middle of the list
    # In the top level comparison, the smaller number will
    # be compared to the `min_num` variable and the larger
    # number will be compared to the `max_num` variable
    for index, num in enumerate(range(0, mid_point)):
        a = num_list[index]
        b = num_list[num_len - 1 - index]
        if a > b:
            if a > max_num:
                max_num = a
            if b < min_num:
                min_num = b
        else:
            if a < min_num:
                min_num = a
            if b > max_num:
                max_num = b

    # If the number list is odd, then there's one
    # element that hasn't been compared. Compare it to
    # the current min/max values
    if num_len % 2 != 0:
        mid_num = num_list[mid_point]
        if mid_num < min_num:
            min_num = mid_num
        elif mid_num > max_num:
            max_num = mid_num

    # Print results!
    print 'Minimum value: {}'.format(min_num)
    print 'Maximum value: {}'.format(max_num)
```