Problem Set 3

Daniel Wang (S01435533)

1. Assume the red alphabets represent what are being swapped in current iteration:

   E H I S Q U T S T I O N

   E H I S Q U T S T I O N

   E H I S Q U T S T I O N

   E H I I Q U T S T S O N

   E H I I N U T S T S O Q

   E H I I N O T S T S U Q

   E H I I N O Q S T S U T

   E H I I N O Q S T S U T

   E H I I N O Q S S T U T

   E H I I N O Q S S T U T

   E H I I N O Q S S T T U

2. Assume the red alphabets represent what are being swapped in current iteration:

   T H I S Q U E S T I O N

   H T I S Q U E S T I O N

   H I T S Q U E S T I O N

   H I S T Q U E S T I O N

   H I S Q T U E S T I O N

   H I Q S T U E S T I O N

   H I Q S T U E S T I O N

   H I Q S T E U S T I O N

   H I Q S E T U S T I O N

   H I Q E S T U S T I O N

   H I E Q S T U S T I O N

   H E I Q S T U S T I O N

   E H I Q S T U S T I O N

   E H I Q S T S U T I O N

   E H I Q S S T U T I O N

   E H I Q S S T T U I O N

   To abbreviate, let's now only show the final positions given iteration i despite the swapping only works for neighbor.

   E H I I Q S S T T U O N

   E H I I O Q S S T T U N

   E H I I N O Q S S T T U

3. First, consider the 5-length sequence

   I U C H L M N G E R O U E S T Q O N
   I N C H L M O G E R O U E S T Q U N
   I N C H L M O E E R O U G S T Q U N
   I N C E L M O E H R O U G S T Q U N
   I N C E L M O E H R O U G S T Q U N

   Second, consider the 2-length sequence

   C N G E H M I E L R O U O S T Q U N
   C E G E H M I N L N O Q O R T S U U

   Finally, consider the 1-length sequence. The result should be:

   C E E G H I L M N N O O Q R S T U U

4. If all keys are identical, it means the sequence is already sorted. For insertion sort, the inner loop will break immediately, so the time complexity becomes O(N). For selection sort, every iteration should inevitably count the current minimum, which makes the time complexity be O(N^2). Thus, insertion sort work better than selection sort in this situation.

5. If all keys are in reverse order, insertion sort requires N-1 times swapping given a range of N outer loop, so its time complexity is in the worst-case scenario O(N^2). For selection sort, the time complexity is still O(N^2), but the total number of swaps is only N. Thus, insertion sort does not work better than selection sort in this situation.

6. The Python code is shown below:

```python
def remove_duplicates(nums):
    nums.sort()
    n = len(nums)

    left, right = 1, n-1
    while left < right:
        if nums[left] == nums[left-1]:
            for i in range(left, right):
                # swap sequentially to maintain order
                nums[i], nums[i+1] = nums[i+1], nums[i]
            right -= 1
        else:
            left += 1

    # ignore values at the right-hand-side of right
    return nums[:right+1]

nums = [0,4,4,3,2,1,1,5]
print(remove_duplicates(nums)) # [0, 1, 2, 3, 4, 5]
```

7. The Python code is shown below:

```python
from collections import defaultdict

def group_jumbles(words):
    # create a hash-map where key is sorted words
    # and values are a list of original words
    jumble = defaultdict(list)

    for word in words:
        # the sorting could be implemented
        # using insertion sort or selection sort
        # or other kinds of advanced sorting
        key = str(sorted(word))

        jumble[key].append(word)

    for key in jumble:
        print(" ".join(jumble[key]))

words = ["racing", "secura", "saucer", "caring", "random"]
group_jumbles(words)
```

8. Let's use the example: 8 7 6 5 4 3 2 1

The expected result should be 1 2 3 4 5 6 7 8

First, consider the 4-length sequence, then it becomes:

4 7 6 5 8 3 2 1 (swapped 1 time)

4 3 6 5 8 7 2 1 (swapped 1 time)

4 3 2 5 8 7 6 1 (swapped 1 time)

4 3 2 1 8 7 6 5 (swapped 1 time)

Second, consider the 2-length sequence, then it becomes:

2 3 4 1 6 7 8 5 (swapped 3 time)
2 1 4 3 6 5 8 7 (swapped 2 time)

Finally, consider the 1-length sequence, then it becomes:
1 2 3 4 5 6 7 8 (swapped 4 time)

It is an example of worst-case scenario. The problem is that for every k, 2k-length sequence is actually a subproblem of it since k-length sequence covers all candidates swapping positions of the 2k one. However, even swapping 2k-length sequence correctly can just yield a suboptimal solution, as the k-length sequence may not be ordered yet. The intuition is that sorting 2k becomes redundant since we can simply sort k.

9. There is no performance gain when changing the original selection sort to the shell version. When we decrease the sequence length of shell sort, we still require computing minimum to maximum without breaking the loop due to the nature of selection sort implementation. To be more specific, if we have k sequences $[h_1, h_2, \ldots, h_k]$ for the shell sort and a list of N length, then the time complexity becomes $O\left(\sum_{i=1}^{k} \left\lfloor \frac{N}{h_i} \right\rfloor^2\right)$. If $N \gg h_i \ \forall i$, then the complexity will be degraded to $O(kN^2)$, which is even worse then the original selection sort $O(N^2)$.