Complexity

Exercises

Exercise 1 - Time Complexity

Determine the following Python function worst-case scenario upper bound Time Complexity.

```python
def f1(n):
    for i in range(n):
        print("Operation", i)
```

```python
def f2(n):
    for i in range(n):
        for j in range(n):
            print("Operation", i, j)
```

```python
def f3(n):
    print("Single Operation")
```

```python
def f4(n):
    i = 1
    while i < n:
        print("Operation", i)
        i *= 2
```

```python
def f5(n):
    for i in range(n):
        j = 1
        while j < n:
            print("Operation", i, j)
            j *= 2
```

```python
def f6(n):
    if n <= 1:
        return
    f6(n-1)
    f6(n-1)
```

```python
def f7(n):
    for i in range(n):
        for j in range(n):
            for k in range(n):
                print("Operation", i, j, k)
```

```python
def f8(n):
    if n <= 1:
        return n
    return f8(n-1) + f8(n-2)
```

```python
def f9(n):
    i = 0
    while i * i < n:
        print("Operation", i)
        i += 1
```

```python
def f10(n):
    total = 0
    for i in range(1, n+1):
        for j in range(1, i+1):
            total += j
    return total
```

Exercise 2 - Space Complexity

Determine the following Python function worst-case scenario upper bound Space Complexity.

```python
def f1(n):
    counter = 0
    for i in range(n):
        counter += 1
    return counter
```

```python
def f2(n):
    arr = []
    for i in range(n):
        arr.append(i)
    return arr
```

```python
def f3(n):
    matrix = []
    for i in range(n):
        row = [j for j in range(n)]
        matrix.append(row)
    return matrix
```

```python
def f4(n):
    if n <= 1:
        return n
    return f4(n-1) + f4(n-2)
```

```python
def f5(n):
    if n == 0:
        return [0]
    else:
        arr = f5(n-1)
        arr.append(n)
        return arr
```

```python
def f6(n):
    d = {}
    for i in range(n):
        d[i] = i * i
    return d
```

```python
def f7(n):
    s = set()
    for i in range(n):
        s.add(i)
    return s
```

```python
def f8(n):
    arr = [i for i in range(n)]
    s = set(arr)
    return arr, s
```

```python
def f9(n):
    a, b, c = 1, 2, 3
    total = a + b + c + n
    return total
```

```python
def f10(n):
    outer = []
    for i in range(n):
        inner = []
        for j in range(n):
            inner.append((i, j))
        outer.append(inner)
    return outer
```

Exercise 3 - Sorting Algorithm

Find the **best-case scenario upper bound** (Big O) and the **worst-case scenario upper bound** (Big O) and the **average-case scenario upper bound** (Big O) for **Time** and the **worst-case scenario upper bound** (Big O) for **Space** of the following sorting algorithms:

```python
def bubble_sort(arr):
    for i in range(len(arr)):
        for j in range(0, n-i-1):
            if arr[j] > arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[j]
    return arr
```

```python
def insertion_sort(arr):
    for i in range(1,len(arr)):
        while arr[i-1] > arr[i] and i > 0:
            arr[i], arr[i-1] = arr[i-1], arr[i]
            i -= 1
    return arr
```

```python
def selection_sort(arr):
    for i in range(len(arr)):
        min_idx = i
        for j in range(i+1, len(arr)):
            if arr[j] < arr[min_idx]:
                min_idx = j
        arr[i], arr[min_idx] = arr[min_idx], arr[i]
    return arr
```

```python
def merge(left, right):
    arr = []
    i,j = 0,0
    while i < len(left) and j < len(right):
        if left[i] < right[j]:
            arr.append(left[i])
            i += 1
        else:
            arr.append(right[j])
            j += 1
    arr.extend(left[i:])
    arr.extend(right[j:])
    return arr

def merge_sort(arr):
    if len(arr) == 1:
        return arr
    mid_point = len(arr) // 2
    left = merge_sort(arr[:mid_point])
    right = merge_sort(arr[mid_point:])
    return merge(left, right)
```

```python
def partition(arr, low, high):
    pivot = arr[high]
    high_pos = low
    for pointer in range(low, high):
        if arr[pointer] <= pivot:
            arr[high_pos], arr[pointer] = arr[pointer], arr[high_pos]
            high_pos += 1
    arr[high_pos], arr[high] = arr[high], arr[high_pos]
    return arr, high_pos

def quicksort(arr, low, high):
    if low < high:
        arr, pivot = partition(arr, low, high)
        quicksort(arr, low, pivot - 1)
        quicksort(arr, pivot + 1, high)
    return arr
```

Exercise 4 -Timeit

Use the **timeit** module on a few algorithms that have been studied in the course.