Problem Set 04

Assignement 1.1

Slowest

- $\log_4 n$
- $40 \log_2 n$
- $k \log_2 n$
- $n \log_2 n$
- $5 n^2$
- $12 n^6$
- 4^n

Fastest

Assignement 1.2

a) $O(n^2)$
b) $O(n)$
c) $O(n*\log(n))$
d) $O(n^2)$
e) $O(2^n)$
f) $O(n^2)$
g) $O(n*\log(n))$

Assignement 1.3

1)

```
def printCalendar(self):
    print(f'\t  {self.monthName()} {self.year()}\nSu\tMo\tTu\tWe\tTh\tFr\tSa')

    dateArray = []
    spaces = [1, 2, 3, 4, 5, 6, 0]

    startingDate = Date(self.month(), 1, self.year())

    for i in range(spaces[startingDate.dayOfWeek()]):
        dateArray.append(' ')

    while self.month() == startingDate.month():
        dateArray.append(startingDate.day())
        startingDate.advanceBy(1)

    for index, e in enumerate(dateArray):
        print(f'{e}\t', end='')
        if index % 7 == 6:
            print()
```

Lines : 7
First for : n
First while : 2n
Second for : 2n * 3 = 6n

T(n) = 9n + 12 Complexity order: o(n)

2)

```
# Check if a given set is a proper subset of another set
def is_proper_subset(self, setB):
    if self.isSubsetOf(setB):
        if not self.__eq__(setB):
            return True

    return False
```

T(n) = n*n + 2 Complexity order: o(n^2)

3)

```
# Removes an element from the set.
def remove(self, element):
    assert element in self, "The element must be in the set."
    self._theElements.remove(element)
```

In the worst case scenario we have to go through all n elements, and compare every element, then remove it if it's the case. So n to iterate, n to compare and 1 extra to delete the element.
T(n) = n + n + 1 Complexity order: o(n)

```
# Adds a new unique element to the set.
def add(self, element):
    if element not in self:
        self._theElements.append(element)
```

In the worst case scenario we need to add an element at the start of list, making n iterations to shift every other element. So 1 do add it and n to iterate the shifts.
T(n) = n + 1 Complexity order: o(n)

Assignement 1.4

	Best	Worst
a)	O(n)	O(n)
b)	O(n)	O($\log_2 n$)
c)	O(n)	O(n^2)

Assignement 1.5

```
def my_slice(input_list, first, last):
    new_list = []
    for element in range(first, last):
        new_list.append(input_list[element])
    return new_list
```

In the best case the algorithm needs n for the for loop plus 2 extra lines. In the worst case it's the same.

Complexity for both worst and best: O(n)