CSC110 Lecture 24: Analyzing Built-In Data Type Operations

Print this handout

You will find the following formula helpful:

$$\forall n \in \mathbb{N}, \sum_{i=0}^{n} i = \frac{n(n+1)}{2}$$

Exercise 1: Running time of list operations

Each of the following Python functions takes a list as input. Analyse each one's running time in terms of n, the size of its input.

1.

```
def f1(nums: list[int]) -> None:
    list.insert(nums, 0, 10000)
```

2.

```
def f2(nums: list[int]) -> None:
    for i in range(0, 10):
        list.append(nums, 10000)
```

3.

```
def f3(nums: list[int]) -> None:
    for i in range(0, 10):
        list.insert(nums, 0, 10000)
```

Note: the length of `nums` changes at each iteration, and so the running time of `list.insert` does as well!

4.

```
def f4(nums: list[int]) -> None:
    n = len(nums)
    for i in range(0, n * n):
        list.insert(nums, 0, i)
```

Exercise 2: Running-time analysis with multiple parameters

Each of the following functions takes more than one list as input. Analyse their running time in terms of the size of their inputs; do not make any assumptions about the relationships between their sizes.

```
def f1(nums1: list[int], nums2: list[int]) -> None:
    for num in nums2:
        list.append(nums1, num)
```

(Let n_1 be the size of `nums1` and n_2 be the size of `nums2`.)

```
def f2(nums1: list[int], nums2: list[int]) -> None:
    for num in nums1:
        list.insert(nums2, 0, num)
```

(Let n_1 be the size of `nums1` and n_2 be the size of `nums2`.)

Exercise 3: Sets, dictionaries, and data classes

Analyse the running time of each of the following functions.

7.

```
def f7(nums: set[int]) -> bool:
    return -1 in nums or 1 in nums
```

8.

```
def f8(num_map: dict[int, int]) -> None:
    for num in num_map:
        num_map[num] = num_map[num] + 1
```

9.

```
def f9(grades: dict[str, list[int]], new_grades: dict[str, int]):
    for course in new_grades:
        if course in grades:
            list.append(grades[course], new_grades[course])
        else:
            grades[course] = [new_grades[course]]
```

10.

```
from dataclasses import dataclass
import math


@dataclass
class Person:
    '''Docstring omitted'''
    name: str
    age: int

def f10(people: list[Person]) -> int:
    '''Precondition: people != []'''
    max_age_so_far = -math.inf

    for person in people:
        if person.age > max_age_so_far:
            max_age_so_far = person.age

    return max_age_so_far
```

Additional exercises

Analyse the running time of each of the following functions.

1.

```
def extra1(nums: list[int]) -> None:
    for i in range(0, len(nums)):
        nums[i] = 0
```

2.

```
def extra2(nums: list[int]) -> None:
    for i in range(0, len(nums)):
        list.pop(nums)
```

3.

```
def extra3(nums: list[int]) -> None:
    for i in range(0, len(nums)):
        list.pop(nums, 0)
```

Note: the length of `nums` changes at each iteration, and so the running time of `list.pop` does as well!

4.

```
def extra4(num1: list[int], num2: list[int]) -> None:
    for i in range(0, len(nums1)):
        for j in range(0, len(nums2)):
            nums1[i] = nums1[i] + nums2[j]
```

(Let n_1 be the size of `nums1` and n_2 be the size of `nums2`.)

5.

```
def extra5(nested_nums: list[list[int]]) -> None:
    for i in range(0, len(nested_nums)):
        if i >= 34:
            for j in range(0, len(nested_nums[i])):
                nested_nums[i][j] = 0

        else:
            nested_nums[i][0] = 0
```

(Let n be the length of `nested_nums`, and assume every inner list has length n.)

6.

```
def extra6(nums: set[int]) -> list[int]:
    new_nums = []

    for num in nums:
        list.insert(new_nums, 0, num ** 2)

    return new_nums
```

7.

```
def extra7(nums: list[int]) -> dict[int, int]:
    counts_so_far = {}

    for num in nums:
        if num in counts_so_far:
            counts_so_far[num] = counts_so_far[num] + 1
        else:
            counts_so_far[num] = 1

    return counts_so_far
```