UCONN
UNIVERSITY OF CONNECTICUT

Department of Computer Science and Engineering

Data Structures and Object-Oriented Design

(CSE – 2050)

Hasan Baig

Office: UConn (Stamford), 305C
email: hasan.baig@uconn.edu

20

Quick Recap

- How to measure algorithm's performance in terms of timings?
  - Some issues with measuring timings
- We studied timings of an algorithm to find duplicates (duplicates_1) in input data
  - Discussed why duplicates_1 algorithm was not efficient
  - Improved and compared performance of both algos
- Asymptotic Analysis and Atomic operations
- Examples of operations in lists
- Activity # 5 for creating an algorithm to calculate the sum of K integers

21

Activity # 5

Write an algorithm to sum first k integers and perform asymptotic analysis
Example:
k = 3
sum(k)
➔ 6

Activity # 5 Solution

Write an algorithm to sum first k integers and perform asymptotic analysis

```
1  def sum_calc(k):
2      total = 0
3      for i in range(1, k+1):
4          total += i
5      return total
6
7  print(sum_calc(3))
8  print(sum_calc(6))
9  print(sum_calc(7))
10 print(sum_calc(100))
```

```
hasanbaig@HBMAC Prac % python3 sum_calc.py
6
21
28
5050
```

2k + 2

Activity # 5 Solution

More clever approach of calculating the sum of *k* integers

```
12  def sum_clever(k):
13      total = k*(k+1)//2
14      return total
15
16  print(sum_clever(3))
17  print(sum_clever(6))
18  print(sum_clever(7))
19  print(sum_clever(100))
```

```
hasanbaig@HBMAC Prac % python3 sum_calc.py
6
21
28
5050
```

5

Asymptotic Analysis

Lets perform asymptotic analysis of duplicate_2 program

```
10  def duplicates_2(L):
11      n = len(L)
12      for i in range(1,n):
13          for j in range(i):
14              if L[i] == L[j]:
15                  return True
16      return False
```

$n^2/2 - n/2 + 3$

Asymptotic Analysis

- Only higher order terms significantly affects a function
- In asymptotic analysis, we drop lower order terms and constants
  - Consider only higher order terms which grows faster with input size, n

$$5n^2 + 3n + 2$$

Big-O Notation

- The formal mathematical definition which allows us to ignore lower order terms and constants is called **Big-O** notation

Big-O Notation

Notes goes here

28

Big-O Notation

Example:

```
10  def duplicates_2(L):
11      n = len(L)
12      for i in range(1,n):
13          for j in range(i):
14              if L[i] == L[j]:
15                  return True
16      return False
```

$n^2/2 - n/2 + 3$

$O(n^2)$

29

Activity # 6 (1/2)

Calculate the time complexity (Big-O) for the following codes

```
1  #Practice question 1
2  def func(L):
3      x = 0
4      for i in L:
5          for j in L:
6              x += i * j
7      return x
```

$1 + n(n \times 3) + 1 = 3n^2 + 2$

$= O(n^2)$

```
1  # Practice question 2
2  a = 0
3  b = 0
4  for i in range(N):
5      a = a + random()
6
7  for i in range(M):
8      b= b + random()
```

$N + M \Rightarrow O(N+M)$

30

Activity # 6 (2/2)

Calculate the time complexity (Big-O) for the following codes

```
1  #Practice question 3
2  a = 0;
3  for i in range(N):
4      for j in reversed(range(i,N)):
5          a = a + i + j;
```

$N = 10$

$1 + 2 + 3 + \cdots \rightarrow O(N^2)$

$N(N+1)$

```
1  # Practice question 4
2  i = 1
3  while i <= n:
4      print(i)
5      i *= 2
```


31