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

1

Announcements

- Reducing submission deadline for labs

Weeks	Modules	Assignments Schedule
8/29 – 9/2	Mod 1 – Basic Python	
9/5 – 9/9	Mod 2 – Object-oriented Programming & testing	
9/12 – 9/16	Mod 3 – Running Time Analysis	
9/19 – 9/23	Mod 4 – Linear Data Structures	Assignment 1 Due
9/26 – 9/30	**Exam 1**	

2

Quick Recap

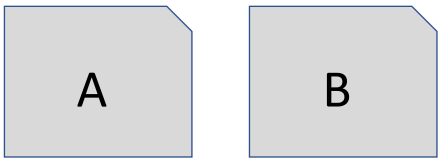
- OOP terminologies: Composition (has-a relationship)
- Inheritance vs Composition
- Polymorphism
- Testing
  - Print()
  - Assert
  - unittest
  - TDD

Module 3

Running Time Analysis

Introduction

- How to determine which algorithm is better to achieve the same task?
- Run both on computer with different inputs and see which takes less time



Problems

1. A may runs faster with specific inputs, while B runs faster with other set of inputs
2. A may runs faster on one specific machine, while B runs faster on another machine

5

Introduction

Factors which affects the running time of an algorithm:

- Hardware (CPU, memory, GPU, etc)
- Software (Multicore, Python implementation 32/64-bits)
- Environment (high temperature)

and…

- Implementation of algorithm itself

6

Running Time Analysis

Example algorithm to detect if there are any duplicates in the input data

duplicates.py

```
1  def duplicates_1(L):
2      n = len(L)
3      for i in range(n):
4          for j in range(n):
5              if i != j and L[i] == L[j]:
6                  return True
7
8      return False
9
10 if __name__ == "__main__":
11     L = [1,2,6,3,4,5,6,7,8]
12     assert(duplicates_1(L))
13
14     L = [1,2,3,4]
15     assert(not duplicates_1(L))
```


Running Time Analysis — time module

Built-in module in python to work with time, example getting current time, giving a delay, etc

timing2.py

```
1  import time
2
3  from duplicates import duplicates_1
4
5  n = 1000
6  for i in range(5):
7      start_time = time.time()
8      duplicates_1(list(range(n)))
9      end_time = time.time()
10     timetaken = end_time - start_time
11
12     print("Time taken for n = ", n, ": ", timetaken)
```

duplicates.py

```
1  def duplicates_1(L):
2      n = len(L)
3      for i in range(n):
4          for j in range(n):
5              if i != j and L[i] == L[j]:
6                  return True
7
8      return False
```

```
hasanbaig@HBMAC Mod3 % python3 timing2.py
Time taken for n =  1000 :  0.08031988143920898
Time taken for n =  1000 :  0.07430481910705566
Time taken for n =  1000 :  0.0741419792175293
Time taken for n =  1000 :  0.07456111907958984
Time taken for n =  1000 :  0.07405304908752441
```

Running Time Analysis | Performance

1. Take an average to smooth out variations
2. The performance of algorithm with the increase in input size

duplicates.py

```
def duplicates_1(L):
    n = len(L)
    for i in range(n):
        for j in range(n):
            if i != j and L[i] == L[j]:
                return True

    return False
```

timetrials.py

```
import time
from duplicates import duplicates_1

def timetrials(dup, n, trials = 10):
    total_time = 0
    for i in range(trials):
        start_time = time.time()
        dup(list(range(n)))
        end_time = time.time()
        total_time += end_time - start_time
    print("Average = %0.7f for n = %d" % (total_time/trials, n))

print("Running duplicates_1 programg")
for n in [50, 100, 200, 400, 800, 1600, 3200]:
    timetrials(duplicates_1, n)
```

```
hasanbaig@HBMAC Mod3 % python3 timetrials.py
Running duplicates_1 programg
Average = 0.0002016 for n = 50
Average = 0.0008012 for n = 100
Average = 0.0029318 for n = 200
Average = 0.0114944 for n = 400
Average = 0.0468351 for n = 800
Average = 0.1910636 for n = 1600
Average = 0.7510886 for n = 3200
```


9

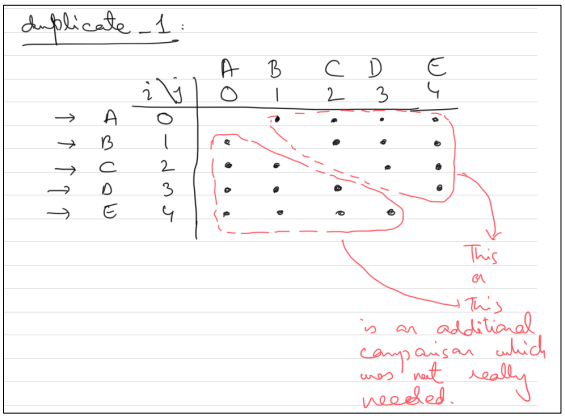
Running Time Analysis | Performance

1. Take an average to smooth out variations
2. The performance of algorithm with the increase in input size

duplicates.py

```
def duplicates_1(L):
    n = len(L)
    for i in range(n):
        for j in range(n):
            if i != j and L[i] == L[j]:
                return True

    return False
```



10

Running Time Analysis | Performance

Improved algorithm to detect duplicates

```
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
```

```
Running duplicates_2 program
Average = 0.0000683 for n = 50
Average = 0.0002734 for n = 100
Average = 0.0011746 for n = 200
Average = 0.0044236 for n = 400
Average = 0.0183014 for n = 800
Average = 0.0753495 for n = 1600
Average = 0.3068655 for n = 3200
```


Running Time Analysis | Performance Comparison

duplicates.py

```
1  def duplicates_1(L):
2      n = len(L)
3      for i in range(n):
4          for j in range(n):
5              if i != j and L[i] == L[j]:
6                  return True
7
8      return False
```

```
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
```

```
hasanbaig@HBMAC Mod3 % python3 timetrials.py
Running duplicates_1 programg
Average = 0.0002016 for n = 50
Average = 0.0008012 for n = 100
Average = 0.0029318 for n = 200
Average = 0.0114944 for n = 400
Average = 0.0468351 for n = 800
Average = 0.1910636 for n = 1600
Average = 0.7510886 for n = 3200
```

```
Running duplicates_2 program
Average = 0.0000683 for n = 50
Average = 0.0002734 for n = 100
Average = 0.0011746 for n = 200
Average = 0.0044236 for n = 400
Average = 0.0183014 for n = 800
Average = 0.0753495 for n = 1600
Average = 0.3068655 for n = 3200
```


Measuring time may not give us the exact performance!

Asymptotic Analysis

In this method, performance is determined by the size of an input and the number of operations executed by algorithm

- The unit to describe number of operations: Atomic Operations
- Atomic Operations:
  - Smallest operation which cannot be split into multiple sub operations
  - Executes in continuous, uninterrupted manner

Asymptotic Analysis — Examples

Atomic Operations

Operation Name	Average Code	Cost
Index access	`L[i]`	1
Index assignment	`L[i] = newvalue`	1
Append	`L.append(newitem)`	1
Pop (from end of list)	`L.pop()`	1
Pop (from index i)	`L.pop(i)`	n – i
Insert at index i	`insert(i, newitem)`	n – i
Delete an item (at index i)	`del(item)`	n – i
Membership testing	`item in L`	n
Slice	`L[a:b]`	b – a
Concatenate two lists	`L1 + L2`	$n_1 + n_2$
Sort	`L.sort()`	$n\log_2 n$

Asymptotic Analysis — Examples

Operation Name	Code	Av. Cost	Worst Case Cost
Index access	`L[i]`	1	1
Index assignment	`L[i] = newvalue`	1	1
Append	`L.append(newitem)`	1	1
Pop (from end of list)	`L.pop()`	1	1
Pop (from index i)	`L.pop(i)`	n – i	n
Insert at index i	`insert(i, newitem)`	n – i	n
Delete an item (at index i)	`del(item)`	n – i	n
Membership testing	`item in L`	n	n
Slice	`L[a:b]`	b – a	
Concatenate two lists	`L1 + L2`	$n_1 + n_2$	$n_1 + n_2$
Sort	`L.sort()`	$n\log_2 n$	$n\log_2 n$

Asymptotic Analysis

1. Assign each operation a cost
2. Focus on **worst case**
3. Count the total cost

18

Asymptotic Analysis — duplicate_1

Lets perform asymptotic analysis of duplicate_1 program

duplicates.py

```
1  def duplicates_1(L):
2      n = len(L)
3      for i in range(n):
4          for j in range(n):
5              if i != j and L[i] == L[j]:
6                  return True
7
8      return False
```

$2n^2 + 3$

19

Activity # 5

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