w12-Lec

Sorting and Searching

Assembled for 204111

by Kittipitch Kuptavanich

Sorting and Searching

- **Sorting คือ Algorithm ในการเรียงสมาชิกของรายการ (List) ให้เป็นไปตามรูปแบบของอันดับที่กำหนด (ตามตัวอักษร, ตามอันดับตัวเลข, ..)**
- **Searching คือ Algorithm ในการค้นหาข้อมูล (Item) ที่ต้องการจากกลุ่มข้อมูล (Collection) ใด ๆ**
- **สิ่งที่ต้องคำนึงถึง**
  - **ความถูกต้อง (this class)**
  - ความซับซ้อนในการคำนวณ (ระยะเวลา)
  - การใช้หน่วยความจำ
  - ...

Efficiency

Efficiency

- **ประสิทธิภาพของโปรแกรมพิจารณาจากการใช้ทรัพยากรต่าง ๆ ของโปรแกรม (เมื่อ Input มีขนาดใหญ่) เช่น**
  - **เวลา**
  - **พื้นที่ (memory)**
  - **Bandwidth..**
  - **อื่น ๆ**

SEARCHING

Searching

- หนึ่งในปัญหาที่พบบ่อยที่สุดใน Computing
- ใน Class นี้เราจะพิจารณาในแง่ที่ว่า Item ที่ต้องการเป็นสมาชิกใน Collection นั้น ๆ หรือไม่
  - True หรือ False
  - สามารถปรับ Algorithm เพื่อใช้บอกตำแหน่งที่พบได้

Searching [2]

- ใน Python การตรวจสอบว่า Item ใด ๆ อยู่ใน List หรือไม่ ทำได้โดยการใช้ `in` operator

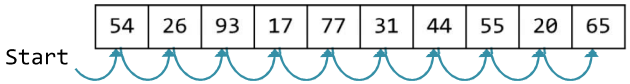
```
>>> 15 in [3, 5, 2, 4, 1]
False

>>> 3 in [3, 5, 2, 4, 1]
True
```

- พิจารณาถึงกระบวนการที่เกิดขึ้นเบื้องหลังเพื่อที่จะได้ผลลัพธ์ดังกล่าว พบมีวิธีในการ Search ที่แตกต่างกันหลายวิธี
  - แต่ละวิธีทำงานอย่างไร
  - เปรียบเทียบ Search Algorithm ในแง่มุมต่าง ๆ

Problem Solving with Algorithms and Data Structures Using Python - Miller and Ranum

Linear Search

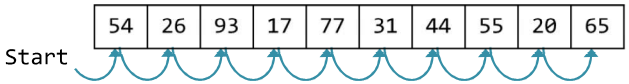


Or Sequential Search

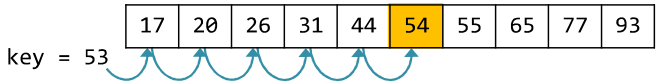
- ไล่ดูทีละ Element จนพบค่าที่ต้องการ

```
09 def linear_search(a_list, key):
10     pos = 0
11     found = False
12     while pos < len(a_list) and not found:
13         if a_list[pos] == key:
14             found = True
15         else:
16             pos = pos + 1
17     return found
18
19 if __name__ == '__main__':
20     test_list = [1, 2, 32, 8, 17, 19, 42, 13]
21     print(linear_search(test_list, 3))
22     print(linear_search(test_list, 13))
```


Linear Search [2]



- ในกรณีที่ List ไม่ได้เรียงตามลำดับ (Unordered List) จะต้องเปรียบเทียบค่ากับทุก Element จนกว่าจะพบ
- กรณี List เรียงตามลำดับ (Ordered List)



  - พิจาณาเฉพาะ Element ที่น้อยกว่าหรือเท่ากับ key
    - เมื่อพบ Element ที่มากกว่า key → หยุดค้นหา

Linear Search [3]

```
09 def ordered_linear_search(a_list, key):
10     pos = 0
11     found = False
12     stop = False
13     while pos < len(a_list) and not found and not stop:
14         if a_list[pos] == key:
15             found = True
16         elif a_list[pos] > key:
17             stop = True
18         else:
19             pos = pos + 1
20         return found
21
22 if __name__ == '__main__':
23     test_list = [0, 1, 2, 8, 13, 17, 19, 32, 42]
24     print(ordered_linear_search(test_list, 3))
```


Linear Search [4]

หากต้องการหา key = 18 ด้วยการทำ Linear Search ใน

Unordered List

[15, 18, 2, 19, 18, 0, 8, 14, 19, 14]

จะต้องทำการเปรียบเทียบทั้งหมดกี่ครั้ง

1. 5
2. 10
3. 4
4. 2

Linear Search [5]

หากต้องการหา key = 13 ด้วยการทำ Linear Search ใน

Ordered List

[3, 5, 6, 8, 11, 12, 14, 15, 17, 18]

จะต้องทำการเปรียบเทียบทั้งหมดกี่ครั้ง

1. 10
2. 5
3. 7
4. 6

Binary Search

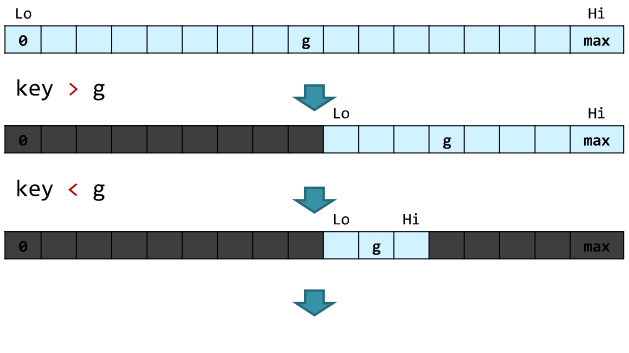
- พิจารณาการเปิดหาคำศัพท์ *key* ใน dictionary
  - ค่าแต่ละค่าในช่วงที่ต้องการหาเป็นค่าที่เรียงกัน
  - หากสุ่มเปิดมาหน้าที่อยู่ในช่วงตัวอักษร*ก่อน*คำที่ต้องการหา
    - แสดงว่าหน้านั้นและ ทุกหน้าก่อนนั้นไม่มีคำตอบ
    - หาคำตอบในหน้าหลังจากนั้นเท่านั้น

- สุ่มตำแหน่ง (Page) g ที่กึ่งกลาง ถ้า key > g แสดงว่า g มีค่าน้อยไป
- คำตอบอยู่ในช่วงทางด้านขวาของ g (ตัดช่วงที่ไม่ใช่คำตอบทิ้งทีละครึ่ง)

Binary Search [2]

หรือ **Bisection Search**

- เราสามารถใช้ **Algorithm การ Search นี้กับการ Search หา Element ใน Ordered List**



Binary Search [3]

- สมมติเรามี ***sorted*** `list` **(array)**

10	11	12	16	18	23	29	33	48	54	57	68	77	84	98

- ต้องการหา key = 23

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]
lo							mid							hi
10	11	12	16	18	23	29	33	48	54	57	68	77	84	98

lo			mid			hi								
10	11	12	16	18	23	29	33	48	54	57	68	77	84	98

				lo	mid	hi								
10	11	12	16	18	23	29	33	48	54	57	68	77	84	98

Binary Search [4]

10	11	12	16	18	23	29	33	48	54	57	68	77	84	98

- ต้องการหา key = 50

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]
lo							mid							hi
10	11	12	16	18	23	29	33	48	54	57	68	77	84	98

							lo		mid			hi		
10	11	12	16	18	23	29	33	48	54	57	68	77	84	98

							lo	mid	hi					
10	11	12	16	18	23	29	33	48	54	57	68	77	84	98

								lo						
								mid						
								hi						
10	11	12	16	18	23	29	33	48	54	57	68	77	84	98

Binary Search [5]

```
09 def binary_search(a_list, key):
10     lo = 0
11     hi = len(a_list) - 1
12     found = False
13     while lo <= hi and not found:
14         mid = (lo + hi) // 2
15         if a_list[mid] == key:
16             found = True
17         elif key < a_list[mid]:
18             hi = mid - 1
19         else:
20             lo = mid + 1
21
22     return found
23
24 if __name__ == "__main__":
25     test_list = [0, 1, 2, 8, 13, 17, 19, 32, 42, ]
26     print(binary_search(test_list, 3))
27     print(binary_search(test_list, 13))
```

Binary Search [6]

หากต้องการหา key = 8 ด้วยการทำ Binary Search ใน

Ordered List

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
3	5	6	8	11	12	14	15	17	18

ลำดับของค่าที่ถูกนำมาเปรียบเทียบเป็นไปตามข้อใด

1. 11, 5, 6, 8
2. 12, 6, 11, 8
3. 3, 5, 6, 8
4. 18, 12, 6, 8

Binary Search [7]

หากต้องการหา key = 16 ด้วยการทำ Binary Search ใน

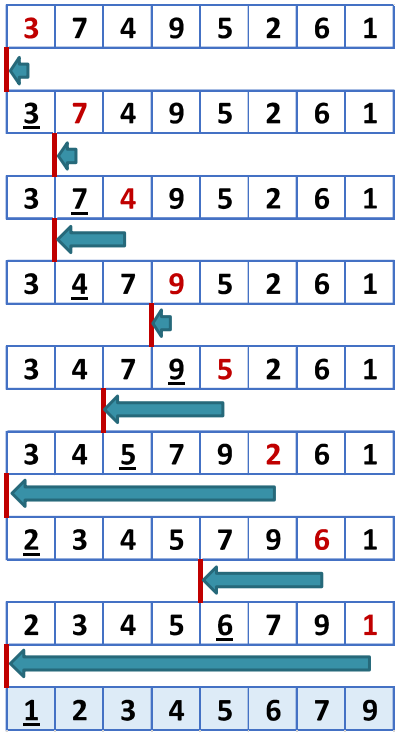
Ordered List

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
3	5	6	8	11	12	14	15	17	18

ลำดับของค่าที่ถูกนำมาเปรียบเทียบเป็นไปตามข้อใด

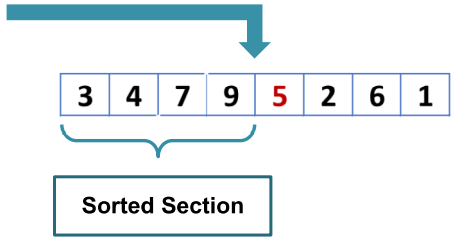
1. 11, 14, 17
2. 18, 17, 15
3. 14, 17, 15
4. 12, 17, 15

SORTING



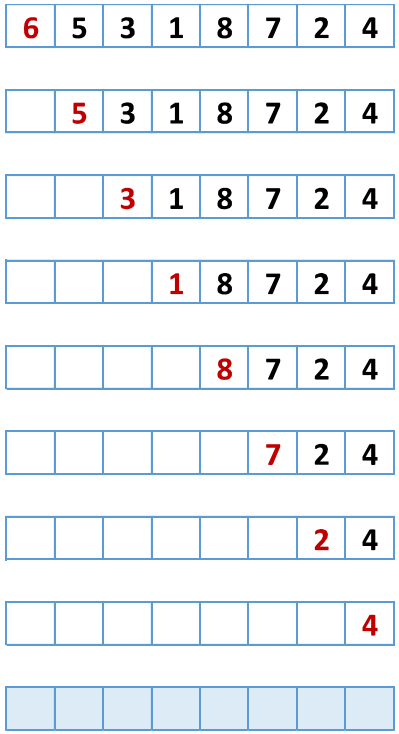
Insertion Sort

- เริ่มจากตำแหน่งซ้ายสุด
- จับวิ่งไปทางซ้ายจนพบค่าน้อยกว่า



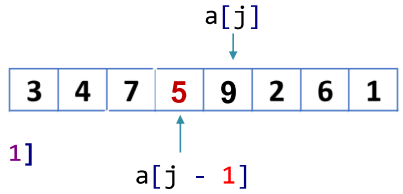
Insertion Sort [2]

Practice 1



Insertion Sort [3]

```
09 def insertion_sort(list_a):
10     size = len(a)
11
12     for i in range(1, size):   # why starts at 1 and not 0?
13         j = i
14         while j > 0 and a[j] < a[j - 1]:
15             a[j], a[j - 1] = a[j - 1], a[j]  # tuple swap!
16             j -= 1
17
18     return list_a
19
20 if __name__ == '__main__':
21     a = [3, 7, 4, 9, 5, 2, 6, 1]
22     print(insertion_sort(a))
```



http://en.wikipedia.org/wiki/Insertion_sort

Insertion Sort [4]

Advantages:

- ง่ายต่อการ **implement**
  Simple implementation
- มีประสิทธิภาพดีกับข้อมูลขนาดเล็ก
  Efficient for (quite) small data sets
- ใช้ได้ดีแม้ **data** มีการเรียงลำดับมาแล้วบางส่วน
  Adaptive, i.e., efficient for data sets that are already substantially sorted:

http://en.wikipedia.org/wiki/Insertion_sort

Insertion Sort [5]

Advantages (cont'd):

- ไม่เปลี่ยนลำดับของสิ่งที่ต้องการเรียงหาก **key** มีค่าเท่ากัน
  Stable; i.e., does not change the relative order of elements with equal keys
- ใช้พื้นที่เพิ่มเติมในการ **sort** ที่คงที่ (ไม่ขึ้นกับขนาดข้อมูล)
  In-place; i.e., only requires a constant amount of additional memory space
- สามารถเรียงข้อมูลไปพร้อม ๆ กับรับข้อมูล (ไม่ต้องรับเข้ามาหมดก่อนถึงจะดำเนินการได้)
  Online; i.e., can sort a list as it receives it

http://en.wikipedia.org/wiki/Insertion_sort

Merge Algorithm

- ในกรณีที่มี list ที่มีการเรียงลำดับไว้แล้วมากกว่าหนึ่ง list
- การนำ list ทั้งสองมารวมกันเพื่อให้ได้ list ใหม่ที่มีการเรียงลำดับ
  - ซึ่งประกอบด้วย element ทั้งหมดจาก list ทั้งสอง
  - เรียกว่า merging
- โดยวิธีในการรวม list ลักษณะดังกล่าว เรียกว่า Merge Algorithm

http://en.wikipedia.org/wiki/Merge_algorithm

Merge Algorithm [2]

1	3	4	5	6

2	5	7	9	10

- ให้ index i และ j ชี้ที่ตำแหน่ง head ของ list A และ list B
- ให้ C เป็น list ว่าง
- ให้ a และ b เป็น value ใน list A และ B ที่ตำแหน่ง i และ j ตามลำดับ
- If $a < b$ เพิ่ม a ไปที่ list C และ increment i
  - Else เพิ่ม b ไปที่ list C และ increment j
- ถ้า list ใดหมดก่อน ให้นำ element ทั้งหมดของ list ที่เหลือ เพิ่มไปที่ list C

http://en.wikipedia.org/wiki/Merge_algorithm

Merge Algorithm [3]

a
1	3	4	5	6		2	5	7	9	10
1										

b
1	3	4	5	6		2	5	7	9	10
1	2									

c
1	3	4	5	6		2	5	7	9	10
1	2	3								

d
1	3	4	5	6		2	5	7	9	10
1	2	3	4							

e
1	3	4	5	6		2	5	7	9	10
1	2	3	4	5						

f
1	3	4	5	6		2	5	7	9	10
1	2	3	4	5	5					

g
1	3	4	5	6		2	5	7	9	10
1	2	3	4	5	5	6				

h
1	3	4	5	6		2	5	7	9	10
1	2	3	4	5	5	6	7	9	10	

Problem Solving with Algorithms and Data Structures Using Python - Miller and Ranum

http://en.wikipedia.org/wiki/Merge_algorithm

Merge Algorithm [4]

```
09 def merge_list(list_a, list_b):
10     len_a = len(list_a)
11     len_b = len(list_b)
12     i = 0
13     j = 0
14     list_c = []
15
16     while i < len_a and j < len_b:
17         if list_a[i] < list_b[j]:
18             list_c.append(list_a[i])
19             i += 1
20         else:
21             list_c.append(list_b[j])
22             j += 1
23
24     if i < len_a:
25         list_c.extend(list_a[i:])
26     if j < len_b:
27         list_c.extend(list_b[j:])
28
29     return list_c
```

bash shell

```
$ python -i merge_list.py
>>> list_a = [1, 3, 4, 5, 6]
>>> list_b = [2, 5, 7, 9, 10]
>>> print(merge_list(list_a, list_b))
[1, 2, 3, 4, 5, 5, 6, 7, 9, 10]
```

Reference

- http://www.kosbie.net/cmu/fall-12/15-112/handouts/notes-efficiency.html
- http://en.wikipedia.org/wiki/Binary_search_algorithm
- http://en.wikipedia.org/wiki/Insertion_sort
- http://en.wikipedia.org/wiki/Merge_algorithm