Review

- Class vs. Class object
- Method vs. Function
- Object-oriented programming
  - Encapsulation
  - Abstraction
  - Inheritance
  - Polymorphism
  - Computing 1 for DS covers OOP more deeply

Linear Search

Lecture 9-1

Hyung-Sin Kim

SNU Graduate School of Data Science

Why Search?

- Searching is a fundamental part of programming, especially in data science

- There are **massive** amount of data in the world and you want to find data you are interested

- You should find data that you want, **efficiently**

Linear Search – Algorithm

- Find if a target value exists in a list
- To do this, search from the first item to the last item sequentially **(linear search)**
- If the target value exists, return the index where the value first occurs
- Otherwise, return -1

Linear Search – Algorithm

- Find if a target value exists in a list
- To do this, search from the first item to the last item sequentially **(linear search)**
- If the target value exists, return the index where the value first occurs
- Otherwise, return -1

target = 4

index	0	1	2	3	4	5	6	7	8	9	10	11
values	5	-2	0	100	-6	7	4	9	-7	50	4	3

Linear Search – Algorithm

- Find if a target value exists in a list
- To do this, search from the first item to the last item sequentially **(linear search)**
- If the target value exists, return the index where the value first occurs
- Otherwise, return -1

target = 4

index	0	1	2	3	4	5	6	7	8	9	10	11
values	5	-2	0	100	-6	7	4	9	-7	50	4	3

Linear Search – Algorithm

- Find if a target value exists in a list
- To do this, search from the first item to the last item sequentially **(linear search)**
- If the target value exists, return the index where the value first occurs
- Otherwise, return -1

target = 4

index	0	1	2	3	4	5	6	7	8	9	10	11
values	5	-2	0	100	-6	7	4	9	-7	50	4	3

Linear Search – Algorithm

- Find if a target value exists in a list
- To do this, search from the first item to the last item sequentially **(linear search)**
- If the target value exists, return the index where the value first occurs
- Otherwise, return -1

target = 4

index	0	1	2	3	4	5	6	7	8	9	10	11
values	5	-2	0	100	-6	7	4	9	-7	50	4	3

Linear Search – Algorithm

- Find if a target value exists in a list
- To do this, search from the first item to the last item sequentially **(linear search)**
- If the target value exists, return the index where the value first occurs
- Otherwise, return -1

target = 4

index	0	1	2	3	4	5	6	7	8	9	10	11
values	5	-2	0	100	-6	7	4	9	-7	50	4	3

Linear Search – Algorithm

- Find if a target value exists in a list
- To do this, search from the first item to the last item sequentially **(linear search)**
- If the target value exists, return the index where the value first occurs
- Otherwise, return -1

target = 4

index	0	1	2	3	4	5	6	7	8	9	10	11
values	5	-2	0	100	-6	7	4	9	-7	50	4	3

Linear Search – Algorithm

- Find if a target value exists in a list
- To do this, search from the first item to the last item sequentially **(linear search)**
- If the target value exists, return the index where the value first occurs
- Otherwise, return -1

target = 4

index	0	1	2	3	4	5	6	7	8	9	10	11
values	5	-2	0	100	-6	7	4	9	-7	50	4	3

Linear Search – Algorithm

- Find if a target value exists in a list
- To do this, search from the first item to the last item sequentially **(linear search)**
- If the target value exists, return the index where the value first occurs
- Otherwise, return -1

target = 4

Found!
Return 6

index	0	1	2	3	4	5	6	7	8	9	10	11
values	5	-2	0	100	-6	7	**4**	9	-7	50	4	3

Linear Search – Algorithm

- Find if a target value exists in a list
- To do this, search from the first item to the last item sequentially **(linear search)**
- If the target value exists, return the index where the value **first occurs**
- Otherwise, return -1

target = 4

Found!
Return 6

index	0	1	2	3	4	5	6	7	8	9	10	11
values	5	-2	0	100	-6	7	**4**	9	-7	50	4	3

Ignored…

Linear Search – Algorithm

- Find if a target value exists in a list
- To do this, search from the first item to the last item sequentially **(linear search)**
- If the target value exists, return the index where the value first occurs
- Otherwise, return -1

target = 1

index	0	1	2	3	4	5	6	7	8	9	10	11
values	5	-2	0	100	-6	7	4	9	-7	50	4	3

Linear Search – Algorithm

- Find if a target value exists in a list
- To do this, search from the first item to the last item sequentially **(linear search)**
- If the target value exists, return the index where the value first occurs
- Otherwise, return -1

target = 1

No target!
Return -1

index	0	1	2	3	4	5	6	7	8	9	10	11
values	5	-2	0	100	-6	7	4	9	-7	50	4	3

Algorithm *vs.* ***Programming***

Algorithm*: A recipe for computers to follow (logical steps)*
Program*: An instruction set in programming languages for a computer to understand and put an algorithm to practice*

There can be ***many different*** *ways to implement (program) a* ***single*** *algorithm!*

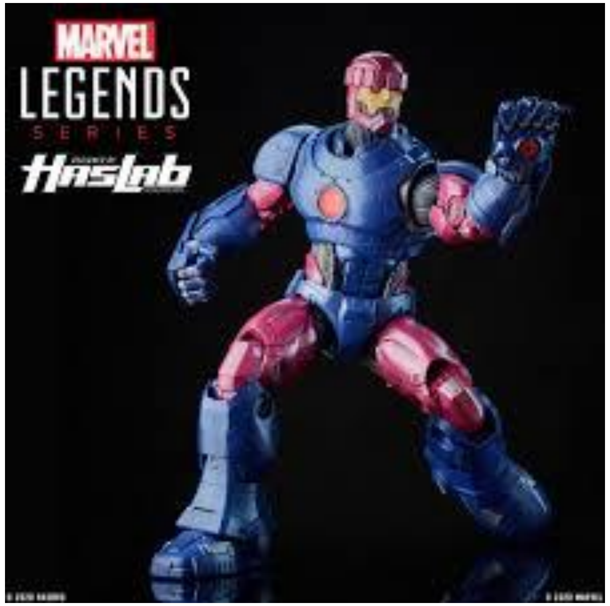
Linear Search – Impl (1): While Loop

```
def linear_search_while(L: list, value: Any) -> int:
    i = 0
    while i < len(L) and L[i] != value:
        i = i + 1
    if i == len(L):
        return -1
    else:
        return i
```

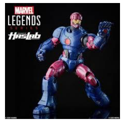
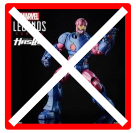
Linear Search – Impl (2): While Loop with Sentinel

- The while loop version needs to do (i < len(L)) every time
  - Because we need to know when the loop reaches the end of the list

- How can we remove this? – by using **sentinel** at the end of the list!



Linear Search – Impl (2): While Loop with Sentinel

- def linear_search_sentinel(L: list, value: Any) -> int:
- **L.append(value)** # Add the sentinel 
- i = 0
- **while L[i] != value**: # This condition is enough!
- i = i + 1
- **L.pop()** # Remove the sentinel 
- if i == len(L):
- return -1
- else:
- return i

Caveat

Some people do not like modifying the input list because it could be dangerous and possibly incur errors

Linear Search – Impl (3): For Loop

```
def linear_search_for(L: list, value: Any) -> int:
    for i in range(len(L)):
        if L[i] == value:
            return i
    return -1
```

- Simple code, no complex conditions
- But some people dislike returning in the middle of a loop
- We have learnt three types of linear search, among which you can choose according to your taste ☺

Linear Search – Time Complexity

- How to measure time spent for an algorithm?
  - import time
  - t_start = time.perf_counter()
  - **<<Your Algorithm>>**
  - t_end = time.perf_counter()
  - return (t_end – t_start) * 1000.0 # the unit becomes milliseconds

Linear Search – Time Complexity (10 M items)

- When the value is located at the end of the list, it takes more time (**linear increase**)
  - This is why the algorithm is called **linear** search!
- Built-in list.index is the **fastest**
  - Python program is notoriously slow since every line of code needs to pass through the Python **interpreter** at run time

Case	while	sentinel	for	list.index
First	0.01	0.01	0.01	0.01
Middle	1261	697	515	106
Last	2673	1394	1029	212

*What if the list is **sorted**?*
Can we do anything better?

index	0	1	2	3	4	5	6	7	8	9	10	11
values	5	-2	0	100	-6	7	4	9	-7	50	4	3

*What if the list is **sorted**?*
Can we do anything better?

index	0	1	2	3	4	5	6	7	8	9	10	11
values	-7	-6	-2	0	3	4	4	5	7	9	50	100

Summary

- Linear search
  - Evaluate the **first** item and cut the **one** evaluated item
  - Time proportional to **len(L)**
  - Applicable to **any** list

Binary Search

Lecture 9-1

Hyung-Sin Kim

SNU Graduate School of Data Science

Binary Search – Motivation

- Linear search does work for a sorted list, but does **NOT** take advantage of the fact that it is sorted

target = 4

Return 5
(Takes similar)

index	0	1	2	3	4	5	6	7	8	9	10	11
values	-7	-6	-2	0	3	4	4	5	7	9	50	100

Binary Search – Motivation

- Linear search does work for a sorted list, but does **NOT** take advantage of the fact that it is sorted

target = 100

Return 11
Takes very long

index	0	1	2	3	4	5	6	7	8	9	10	11
values	-7	-6	-2	0	3	4	4	5	7	9	50	100

Binary Search - Idea

- Idea: Evaluate the **middle** of the sorted list and removes half of candidate entries
- Linear search: one evaluation removes **one** candidate entry
- Binary search: one evaluation removes **half** of candidate entries

index	0	1	2	3	4	5	6	7	8	9	10	11
values	-7	-6	-2	0	3	4	4	5	7	9	50	100

Binary Search – Algorithm

- Idea: Evaluate the **middle** of the sorted list and removes half of candidate entries
- Linear search: one evaluation removes **one** candidate entry
- Binary search: one evaluation removes **half** of candidate entries

target = 9

start = 0
end = 11

mid = (0+11)//2
4 < target

index	0	1	2	3	4	5	6	7	8	9	10	11
values	-7	-6	-2	0	3	4	4	5	7	9	50	100

Binary Search – Algorithm

- Idea: Evaluate the **middle** of the sorted list and removes half of candidate entries
- Linear search: one evaluation removes **one** candidate entry
- Binary search: one evaluation removes **half** of candidate entries

target = 9

*start = **6***
end = 11

mid = (0+11)//2
4 < target

index	0	1	2	3	4	5	6	7	8	9	10	11
values	-7	-6	-2	0	3	4	4	5	7	9	50	100

Binary Search – Algorithm

- Idea: Evaluate the **middle** of the sorted list and removes half of candidate entries
- Linear search: one evaluation removes **one** candidate entry
- Binary search: one evaluation removes **half** of candidate entries

target = 9

start = 6
end = 11

mid = (6+11)//2
7 < target

index	0	1	2	3	4	5	6	7	8	9	10	11
values	-7	-6	-2	0	3	4	4	5	7	9	50	100

Binary Search – Algorithm

- Idea: Evaluate the **middle** of the sorted list and removes half of candidate entries
- Linear search: one evaluation removes **one** candidate entry
- Binary search: one evaluation removes **half** of candidate entries

target = 9

*start = **9***
end = 11

mid = (6+11)//2
7 < target

index	0	1	2	3	4	5	6	7	8	9	10	11
values	-7	-6	-2	0	3	4	4	5	7	9	50	100

Binary Search – Algorithm

- Idea: Evaluate the **middle** of the sorted list and removes half of candidate entries
- Linear search: one evaluation removes **one** candidate entry
- Binary search: one evaluation removes **half** of candidate entries

target = 9

start = 9
end = 11

mid = (9+11)//2
50 >= target

index	0	1	2	3	4	5	6	7	8	9	10	11
values	-7	-6	-2	0	3	4	4	5	7	9	50	100

Binary Search – Algorithm

- Idea: Evaluate the **middle** of the sorted list and removes half of candidate entries
- Linear search: one evaluation removes **one** candidate entry
- Binary search: one evaluation removes **half** of candidate entries

target = 9

start = 9
*end = **9***

mid = (9+11)//2
50 >= target

index	0	1	2	3	4	5	6	7	8	9	10	11
values	-7	-6	-2	0	3	4	4	5	7	9	50	100

Binary Search – Algorithm

- Idea: Evaluate the **middle** of the sorted list and removes half of candidate entries
- Linear search: one evaluation removes **one** candidate entry
- Binary search: one evaluation removes **half** of candidate entries

target = 9

start = 9
end = 9

mid = (9+9)//2
9 >= target

index	0	1	2	3	4	5	6	7	8	9	10	11
values	-7	-6	-2	0	3	4	4	5	7	9	50	100

Binary Search – Algorithm

- Idea: Evaluate the **middle** of the sorted list and removes half of candidate entries
- Linear search: one evaluation removes **one** candidate entry
- Binary search: one evaluation removes **half** of candidate entries

target = 9

start = 9
*end = **8***

Cannot proceed further!

If there is target in L,
L[start] must be the target!

index	0	1	2	3	4	5	6	7	8	9	10	11
values	-7	-6	-2	0	3	4	4	5	7	**9**	50	100

Binary Search – Code

```
def binary_search(L: list, v: Any) -> int:
    start, end = 0, len(L) – 1
    while start != end + 1:
        mid = (start+end) // 2
        if L[mid] < v:
            start = mid + 1
        else:
            end = mid - 1
    if start < len(L) and L[start] == v:
        return start
    else:
        return -1
```

Binary Search – Time Complexity (10 M items)

- Linear search
  - Time delay is proportional to **len(L)**
- Binary search
  - Time delay is proportional to $\boldsymbol{log_2^{len(L)}}$
- A good example why **sorting** is useful!
  - But remember that sorting is **NOT** free either. It also takes non-negligible time…

Case	list.index	binary_search
First	0.007	0.02
Middle	105	0.02
Last	211	0.02 (WoW!)

Summary

- Binary search
  - Evaluate the **middle** item and cut the **half**
  - Time proportional to $\boldsymbol{log_2^{len(L)}}$
  - Applicable to a **sorted** list

Thanks!