Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

b-it Bonn-Aachen International Center for Information Technology

Algorithm Analysis & Data Structures

Mihir Mehta

Foundations Course Summer Semester 2022

18.03.2022

What is an Algorithm?

- Any guess(es)?

What is an Algorithm?

Techincally speaking…

An algorithm is a well-defined procedure that takes a set of values as input and produces set of values as output.

Simply speaking…

An algorithm is just a recipe/guide/manual of simple steps to follow in order to achieve some goal.

Essentially,

An algorithm is a tool to solve a well defined *(computational)* problem.

What is an Algorithm?

In terms of computer science…

An algorithm is a set of steps a program takes to finish a task.

Time for an example – Algorithm in real life…

Goal: To bake a simple cake!

Algorithm/Steps:

- Preheat the oven.
- Gather the ingredients.
- Measure out the ingredients.
- Mix together the ingredients to form a batter.
- Grease/oil a pan.
- Pour the prepared batter into the pan.
- Put the pan into an oven.
- Set the timer.
- When timer rings, take the pan out of the oven.
- Enjoy! *(Not really a part of the algorithm :P)*

A few other examples – Algorithm in real life…

Daily activities such as:

- Driving to a particular destination.
- Doing Laundry.
- Preparing coffee.
- Reaching to office/university.
- Working and completing the assignments.

Have I ever seen/written an Algorithm?

Coming back to the definition of algorithm – computer science…

An algorithm is a set of steps a program takes to finish a task.

What if my goal was just to display “Hello World”?

```
1 print("Hello World")
```

```
Hello World
```

Congrats!

Why should we study about algorithms?

A simple toy example:

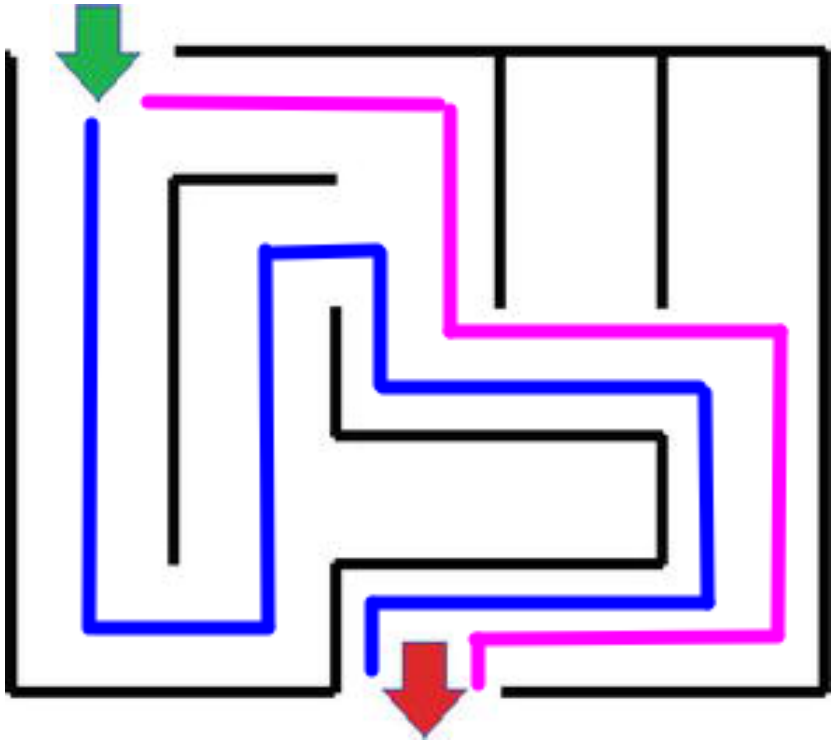


Figure 1: Solving a simple maze [1]

[1] https://www.intechopen.com/chapters/65828

Comparing Algorithms – with example

Suppose …

Figure 2: You have a book to read [2]

[2] https://www.goodreads.com/user_status/show/286383699

Comparing Algorithms – with example

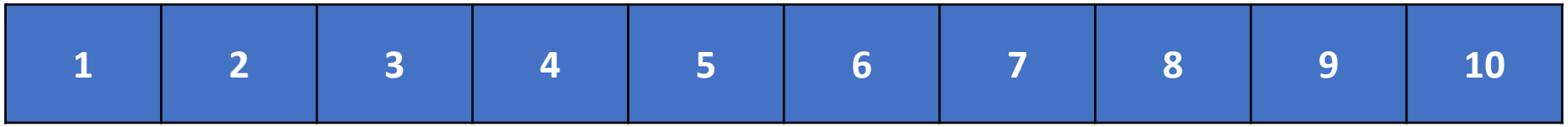
And …

Figure 3: You have to open a target page [3]

[3] https://www.wattpad.com/959498332-origami-a-draco-malfoy-fan-fiction-chapter-1-page

Comparing Algorithms

Keeping it really simple –



Let's say our book only has 10 pages!

As depicted above.

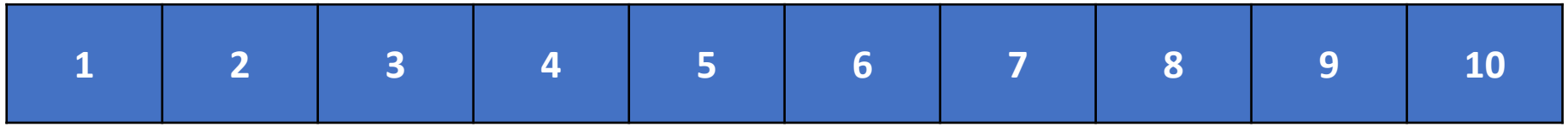
And, there are only two ways to achieve our goal:

1. Simply turn one page at a time sequentially.
2. Dividing the book into half recursively and until we reach our target page.

Comparing Algorithms

Keeping it really simple –

1st Approach.
Target page: 3

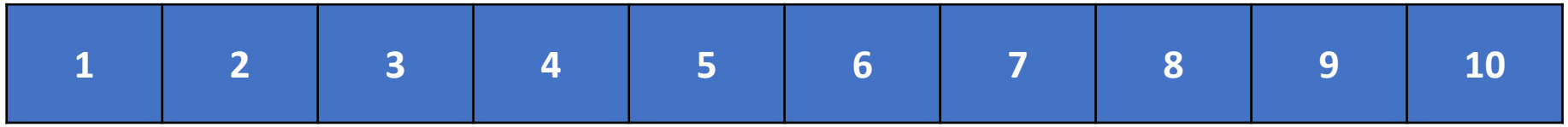


Required steps – 3

Comparing Algorithms

Keeping it really simple –

2nd Approach.
Target page: 3

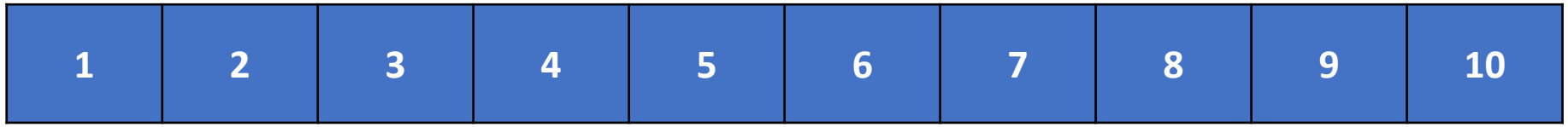


Required steps – 3

Comparing Algorithms

Keeping it really simple –

1st Approach.
Target page: 10

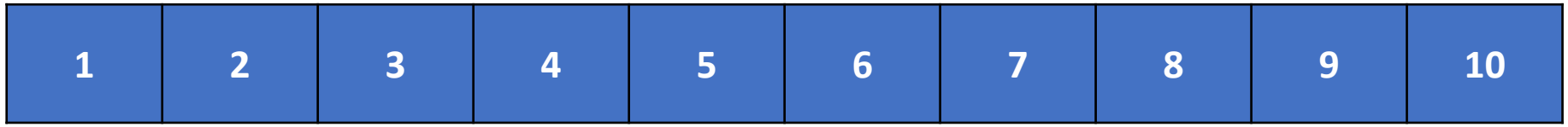


Required steps – 10

Comparing Algorithms

Keeping it really simple –

2nd Approach.
Target page: 10

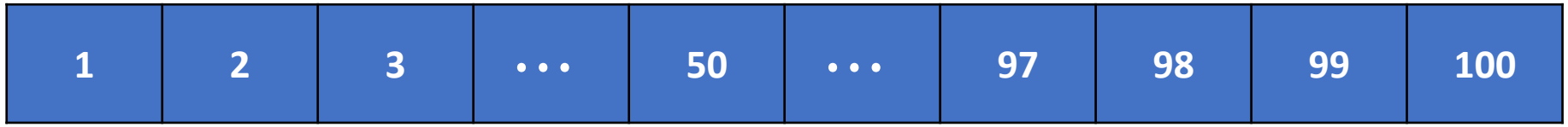


Required steps – 4

Comparing Algorithms

Keeping it really simple –

1st Approach.
Target page: 100

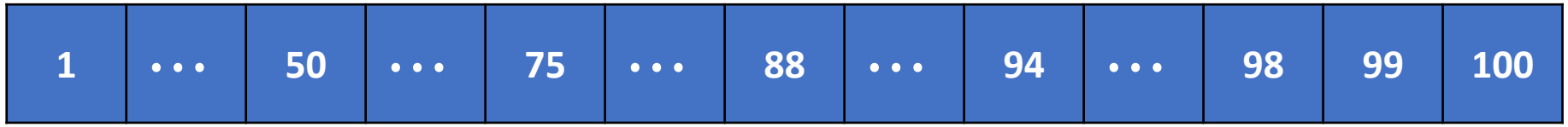


Required steps – ?

Comparing Algorithms

Keeping it really simple –

2nd Approach.
Target page: 100



Required steps – 7

Comparing Algorithms

What did we just perform?

→ Algorithm analysis

Comparisons of algorithms are usually done on the basis of:

Two important factors:

1. Time complexity
2. Space complexity

Ways to evaluate efficiency of an algorithm

Three important cases:

1. Worst case - O (called as Big Oh)
2. Best case - Ω (called as Big Omega)
3. Average case - Θ (called as Big Theta)

Worst Case – Approach 1

Keeping it really simple –

1st Approach.
Input size (n) : 100 Target page : 100

Required steps : 100

Time complexity: O(?)

Worst Case – Approach 1

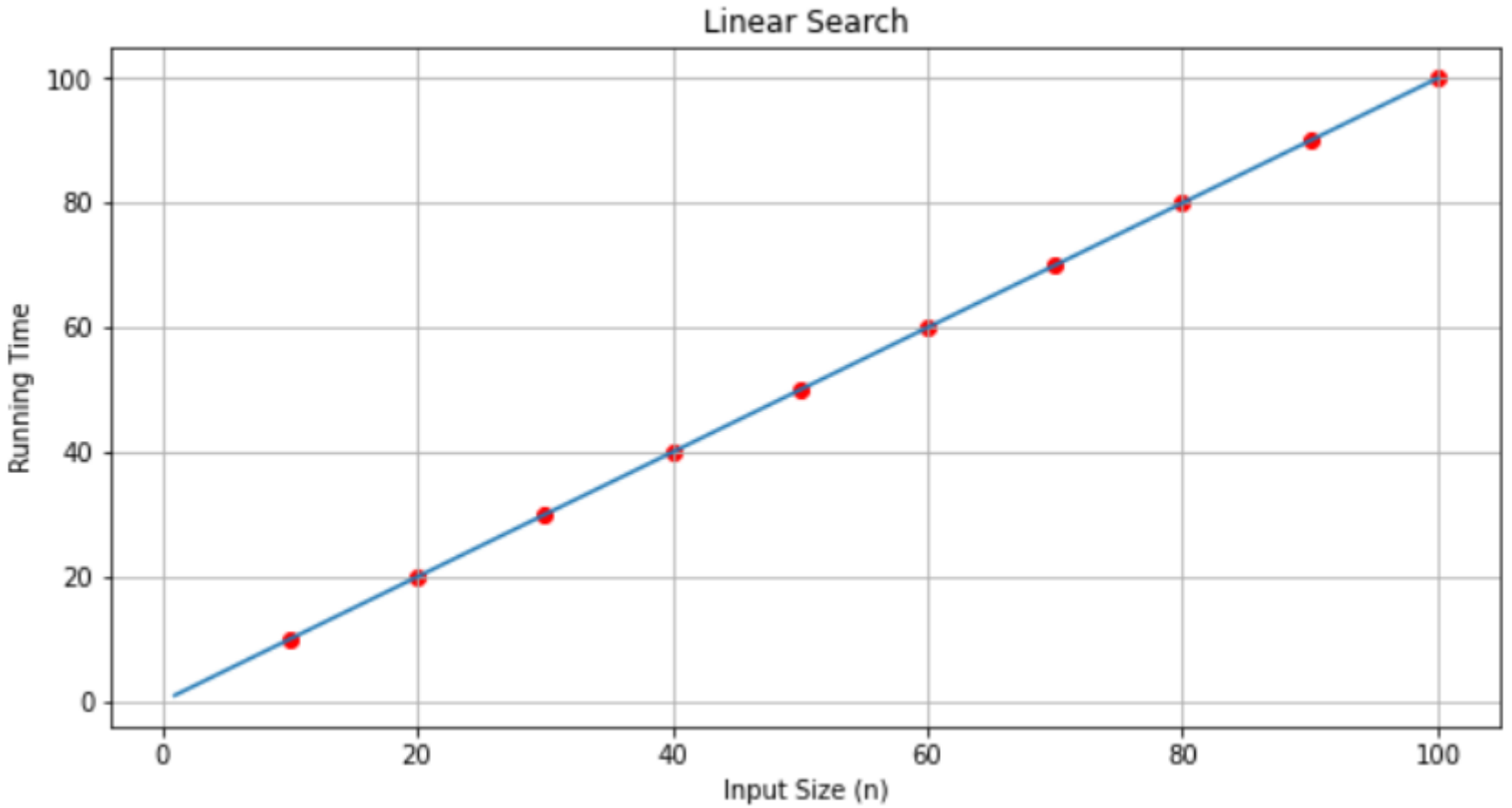


Figure 4: Run time complexity for linear search

Worst Case – Approach 1

Keeping it really simple –

1st Approach.
Input size (n) : 100 Target page : 100

Required steps : 100

Time complexity: O(n)

Worst Case – Approach 2

Keeping it really simple –

2nd Approach.
Input size (n) : 100 Target page : 100

Required steps : 7

Time complexity: O(?)

Worst Case – Approach 2

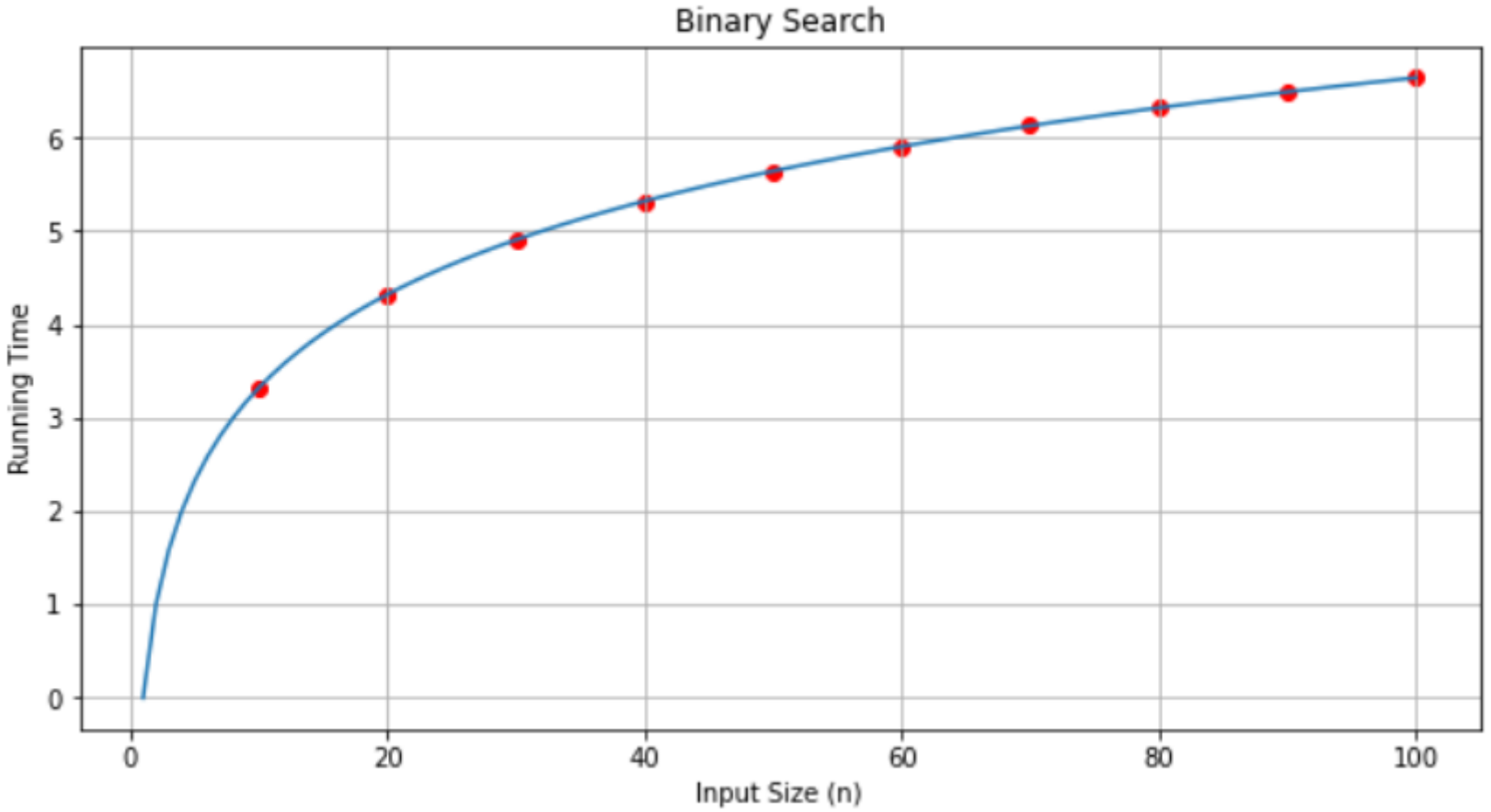


Figure 5: Run time complexity for binary search

Worst Case – Approach 2

Keeping it really simple –

2nd Approach.
Input size (n) : 100 Target page : 100

Required steps : 7

Time complexity: O$(\log_2(n) + 1)$

Time complexity: O$(\log_2(n))$

Some common time complexities

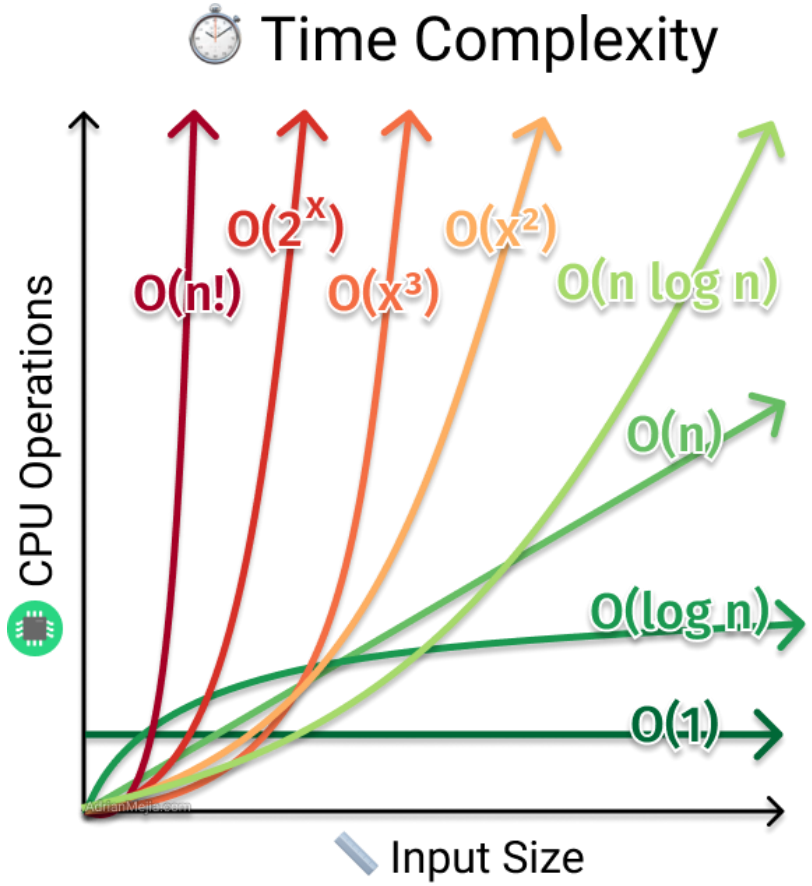


Figure 6: Comparing run-time complexity [4]

[4] https://www.joyk.com/dig/detail/1608665582509127

Taking a step back… to think

How are we storing the data now?

Both the approaches currently use a common way to store data.

A python **List** (few other languages may refer it as “array”)

But, is the only way to store and access data in our programs?

Ways to store data…

- List
- Tuple
- Set
- Dictionary
- Linked Lists
- Stacks
- Queues
- ⋮

A Short Detour

- 3 properties of a python object
- Mutable vs immutable objects
- “is” operator vs “==” operator
- Container objects
- Hashable objects

Properties of a python object

- **Identity (ID) of an object**
  - Memory address
- **Type**
  - Data type *(int, list, set, etc. or user-defined class)*
- **Value**
  - The actual data/value held by the object

Mutable vs Immutable Objects

- **Mutable Objects**
  - Allow changing of values held by the object.
  - Example: set, user-defined classes, lists, etc.

- **Immutable Objects**
  - Do not allow changing of values held by the object.
  - Example: tuple, int, string, etc.

“is” operator vs “==” operator

- **“is” operator**
  - Compares identity of two objects.

- **“==” operator**
  - Compares the values of two objects.

Container Objects

- Objects containing another objects or reference to other objects.
- Example: list, dictionary, tuples, etc.
- **Note:** An immutable container's value which is referencing to a mutable object can be changed if that mutable object is changed.

Hashable Objects

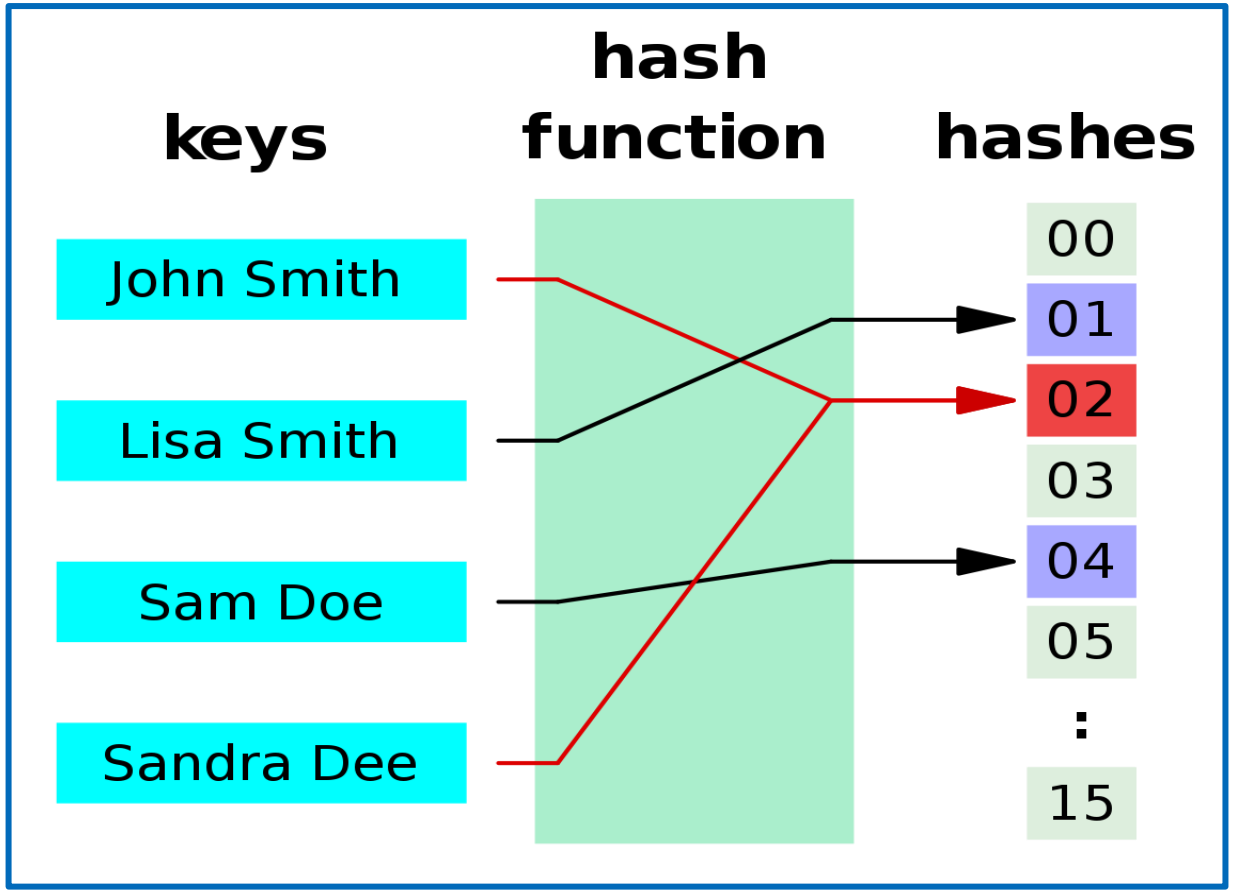


Figure 7: Dictionary as a map [5]

[5] https://en.wikipedia.org/wiki/Hash_function

Hashable Objects

- The main idea is that, you can reduce a complex object to an index in an array.

- Example: Used in implementing dictionary, set, etc. data-structures.

- **Hashable data types:** int, float, str, tuple, NoneType, and user-defined classes.

- **Unhashable data types:** dict, list, and set.

Going back to – Data Structures in Python

- List
- Tuple
- Set
- Dictionary
- Linked Lists
- Stacks
- Queues
- ⋮

List (Array)

Value	1	2	3	4	5	6	7	8	9	10
Index	0	1	2	3	4	5	6	7	8	9

- Stores duplicate data
- Items are ordered
- Mutable object
- Can have items of different types

List (Array)

Operation	Time Complexity (worst Case)
Append	O(1)
Pop	O(1)
Insert	O(n)
Delete	O(n)
Setting value at particular index	O(1)
Getting value from particular index	O(1)
Get length	O(1)
Finding/searching an element in List	O(n)

Tuple

```
1  simple_spells = ("Expelliarmus","Riddikulus","Alohamora")
2  spells = list(simple_spells)
3  spells.append("Wingardium Leviosaa")
4  simple_spells.append("Wingardium Leviosaa")
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-69-0ed7f4ddeee7> in <module>
      2 spells = list(simple_spells)
      3 spells.append("Wingardium Leviosaa")
----> 4 simple_spells.append("Wingardium Leviosaa")

AttributeError: 'tuple' object has no attribute 'append'
```

- Stores duplicate data
- Items are ordered
- Immutable object
- Can have items of different types

Set

```
1 unforgivable_curses = ['Avada Kedavra','Crucio','Imperio',"Crucio","Imperio","Imperio"]
2 print(f"List: {unforgivable_curses}")
3 print("="*30)
4 print(f"Set :{set(unforgivable_curses)}")
```

```
List: ['Avada Kedavra', 'Crucio', 'Imperio', 'Crucio', 'Imperio', 'Imperio']
==============================
Set :{'Crucio', 'Imperio', 'Avada Kedavra'}
```

- Does not store duplicate data
- Items are unordered
- Mutable object
- Can have items of different types

Set

Operation	Time Complexity (worst case)
Finding element in set	O(1)
Intersection of two sets	O(min[len(set1), len(set2)])
Inserting an element	O(1)
Delete an element	O(1)
Iterating all elements in a set	O(n)
Get length	O(1)

Dictionary (hash-tables)

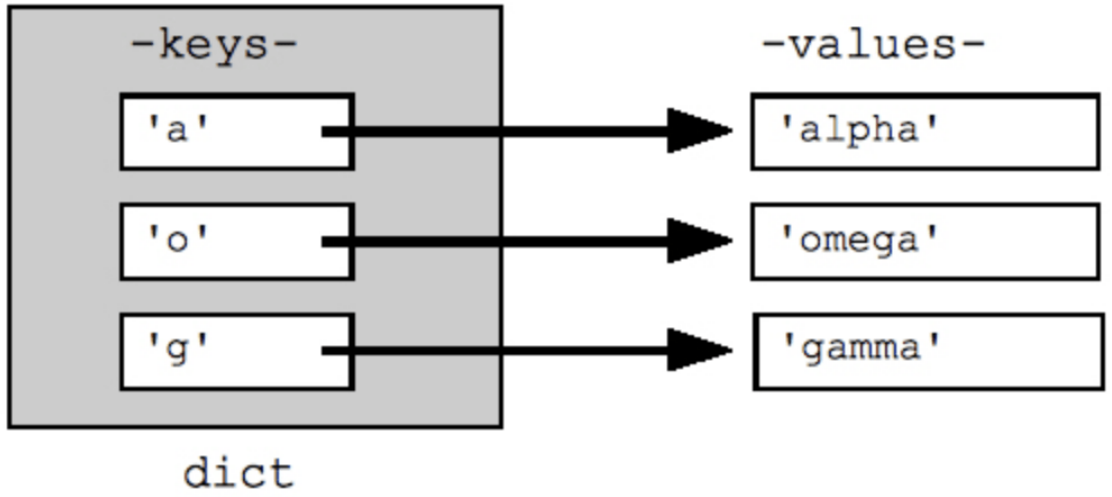


Figure 8: Dictionary as a map [6]

- Does not store duplicate keys
- Mutable object
- Keys should be hashable.

[6] https://www.datacamp.com/community/tutorials/python-dictionaries

Dictionary

Operation	Time Complexity (worst case)
Finding an element	O(1)
Copying the dictionary	O(n)
Inserting an element	O(1)
Get value from key	O(1)
Deleting an element	O(1)
Iterating all elements in a set	O(n)
Get length	O(1)

Linked Lists

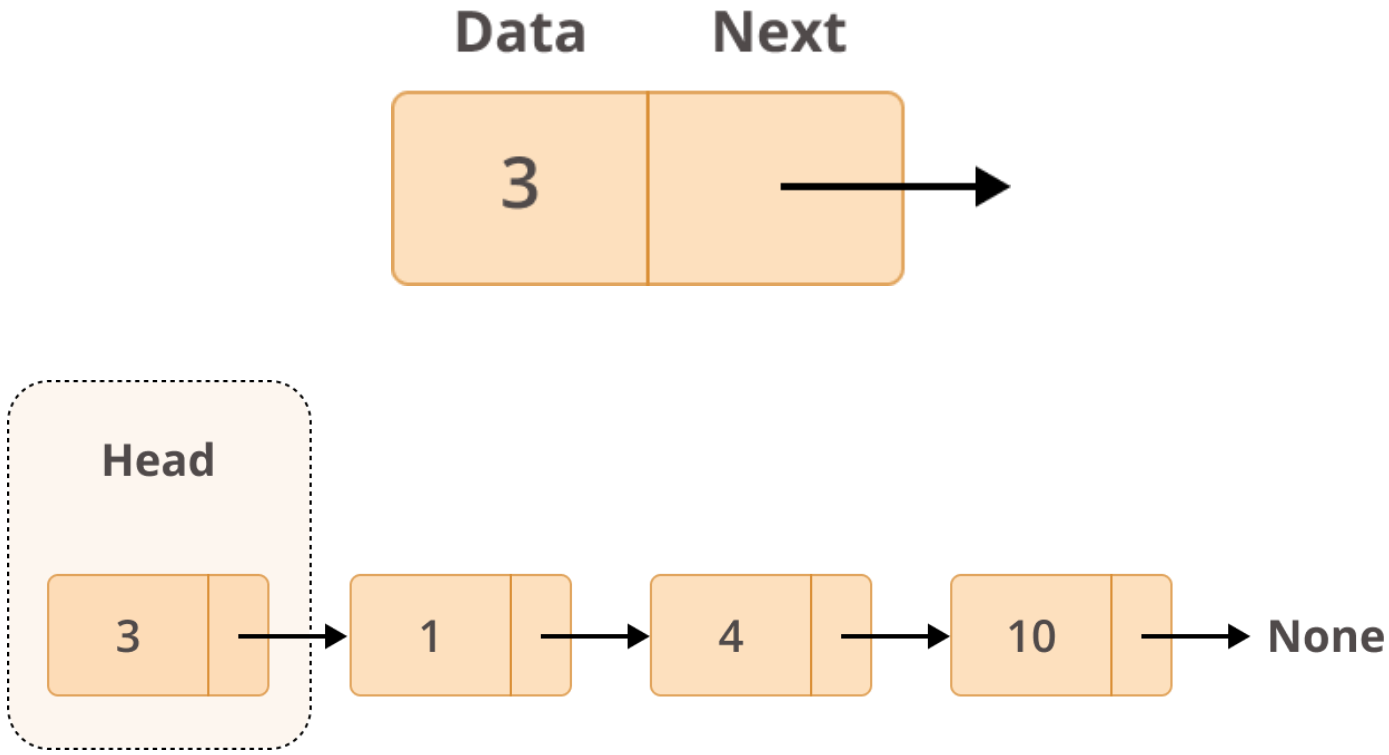


Figure 9: Node of a linked lists, example of singly linked list [7]

[7] https://realpython.com/linked-lists-python/

Linked Lists

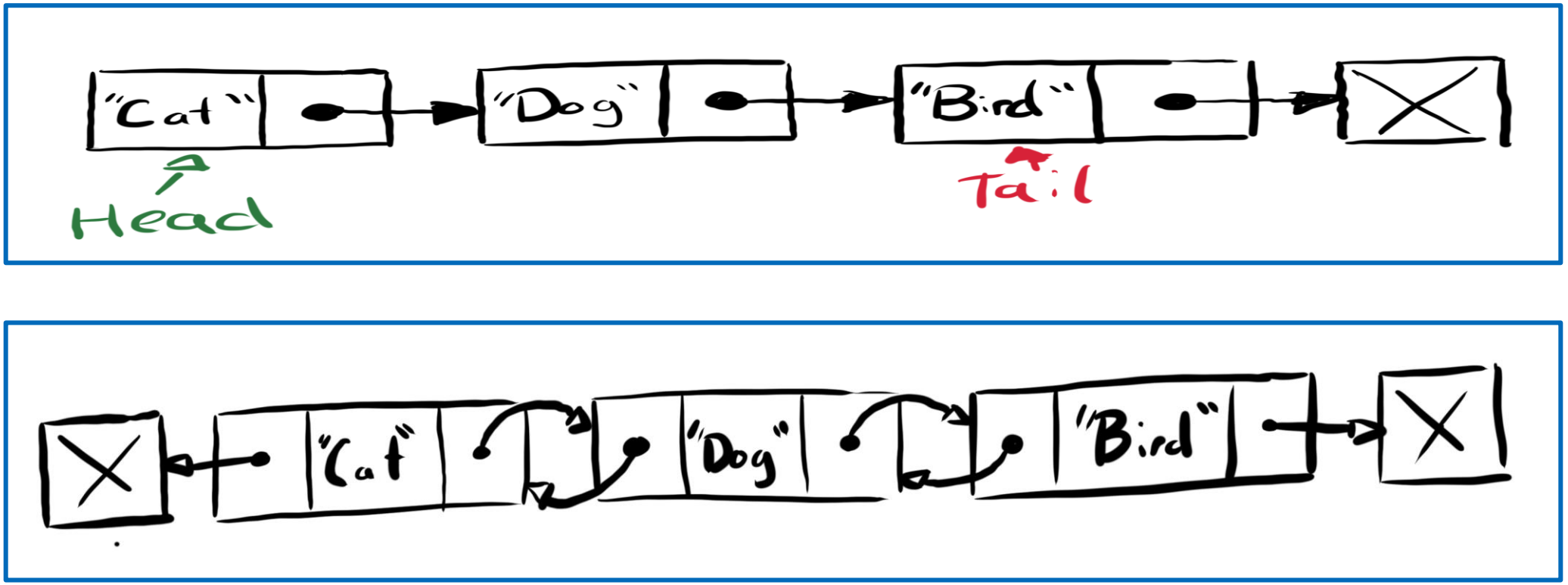


Figure 10: Singly and doubly linked lists [8]

[8] https://www.mongodb.com/developer/how-to/introduction-to-linked-lists-and-mongodb/

Linked Lists

- Similar to lists (arrays) – the name has “List” in it, it was obvious!

- Memory allocation:
  - Lists have sequential memory allocation
  - Linked lists are stored in memory dynamically

- Insertion and deletion at beginning:
  - In Lists, it difficult and is usually O(n)
  - In Linked List, it is very fast – usually O(1)

Stacks

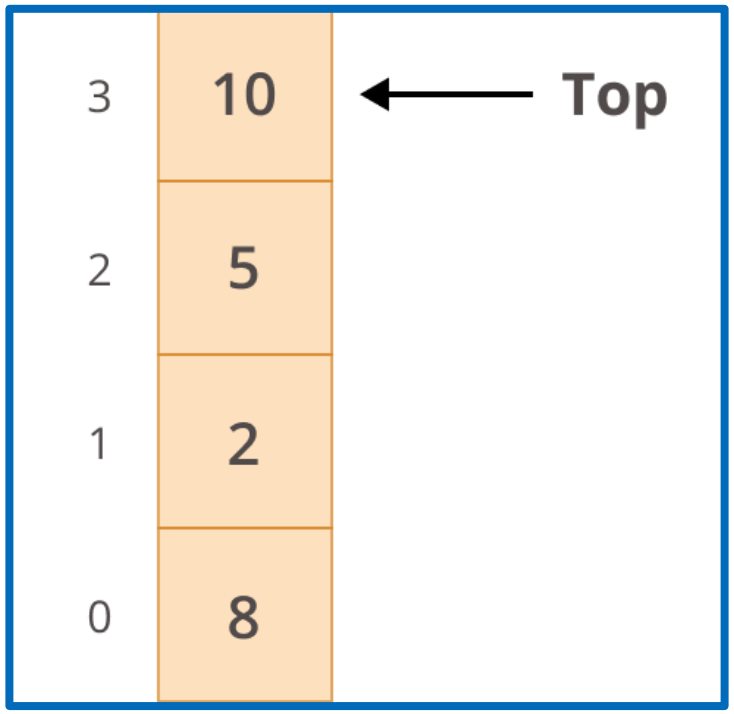


Figure 11: Linked lists as stack [7]

[7] https://realpython.com/linked-lists-python/

Stacks

- Adding and deleting items only to and from one end!
- Follows the principle of LIFO (Last In First Out).
- “Top” always points to the topmost element of the stack.
- Basic stack operations:
  - Push (insertion)
  - Pop (deletion)
  - Top (returns topmost element)
  - IsEmpty (which returns TRUE if stack is empty)

Queues

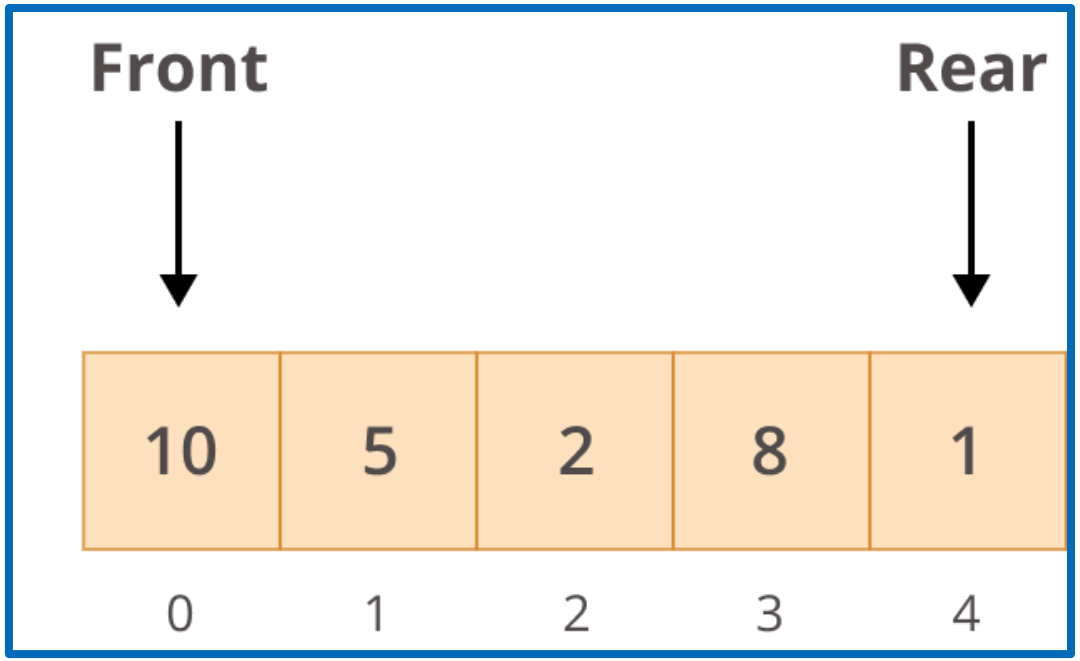


Figure 12: Linked lists as queue [7]

[7] https://realpython.com/linked-lists-python/

Queues

- The 1st element is inserted from one end called the REAR *(also called tail)*, and the removal of existing element takes place from the other end called as FRONT*(also called head)*!

- Follows the principle of FIFO (First In First Out).

- “FRONT” always points to the first element of the queue. “TAIL” always points to the last element in the queue.

Queues

- Basic queue operations:
  - Push (Insertion) (enqueue)
  - PopLeft (Deletion) (dequeue)
  - Peek (returns the first element of queue without popping it)
  - IsEmpty (which returns TRUE if queue is empty)

Linked Lists, Stacks, and Queues

Operation	Time Complexity (worst case)
Finding an element (linked list)	O(n)
Copying the data-structure	O(n)
Appending (queue – enqueue)	O(1)
Append left (pushing into stack)	O(1)
Pop (stack)	O(1)
Pop left (queue – dequeue)	O(1)
Deleting an element (linked list)	O(n)
Iterating all elements in a linked list	O(n)

References (1/2)

[1] https://www.intechopen.com/chapters/65828

[2] https://www.goodreads.com/user_status/show/286383699

[3] https://www.wattpad.com/959498332-origami-a-draco-malfoy-fan-fiction-chapter-1-page

[4] https://www.joyk.com/dig/detail/160866558250912 7

References (2/2)

[5] https://en.wikipedia.org/wiki/Hash_function

[6] https://www.datacamp.com/community/tutorials/python-dictionaries

[7] https://realpython.com/linked-lists-python/

[8] https://www.mongodb.com/developer/how-to/introduction-to-linked-lists-and-mongodb/

Further Reading(s)