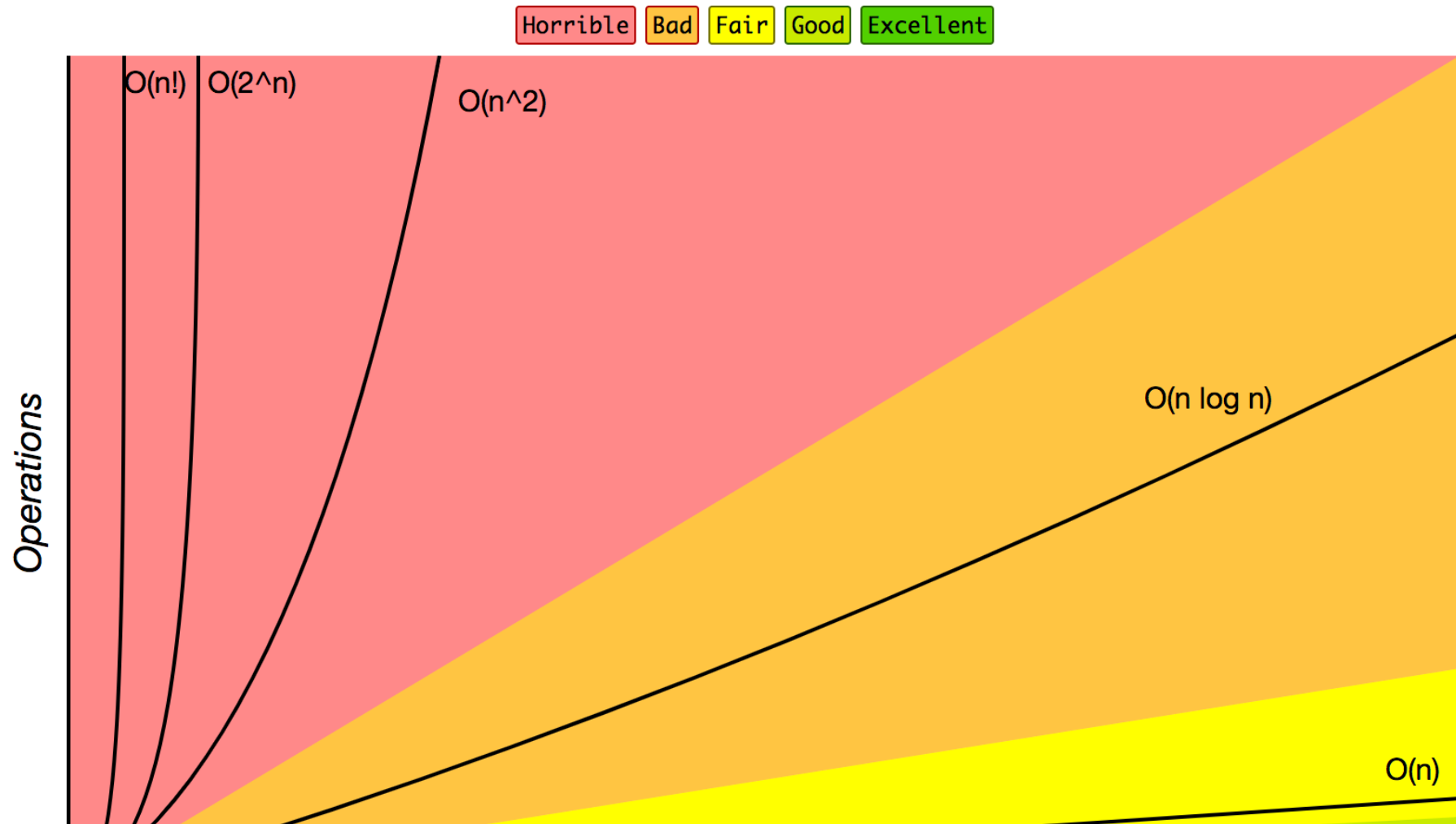


Big O Notation

Big-O Complexity Chart



Elements

Linked Lists

- sequence of nodes where each node stores its own data and a link to the next node
- first node is called the **head**
- last node must have its link pointing to **None**



```
class Node:
    def __init__(self, data, next):
        self.data = data
        self.next = next

class LinkedList:
    def __init__(self):
        self.head = None

    def add_at_front(self, data):
        self.head = Node(data, self.head)

    def add_at_end(self, data):
        if not self.head:
            self.head = Node(data, None)
```

```
        return
    curr = self.head
    while curr.next:
        curr = curr.next
    curr.next = Node(data, None)

def get_last_node(self):
    n = self.head
    while(n.next != None):
        n = n.next
    return n.data

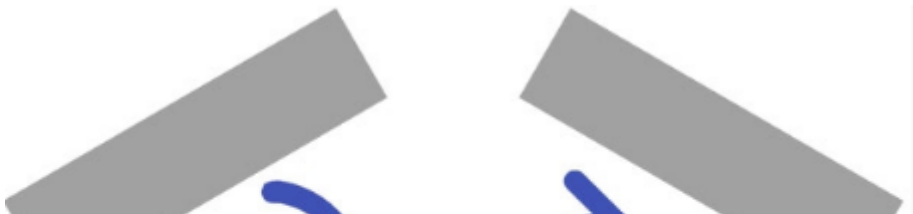
def is_empty(self):
    return self.head == None

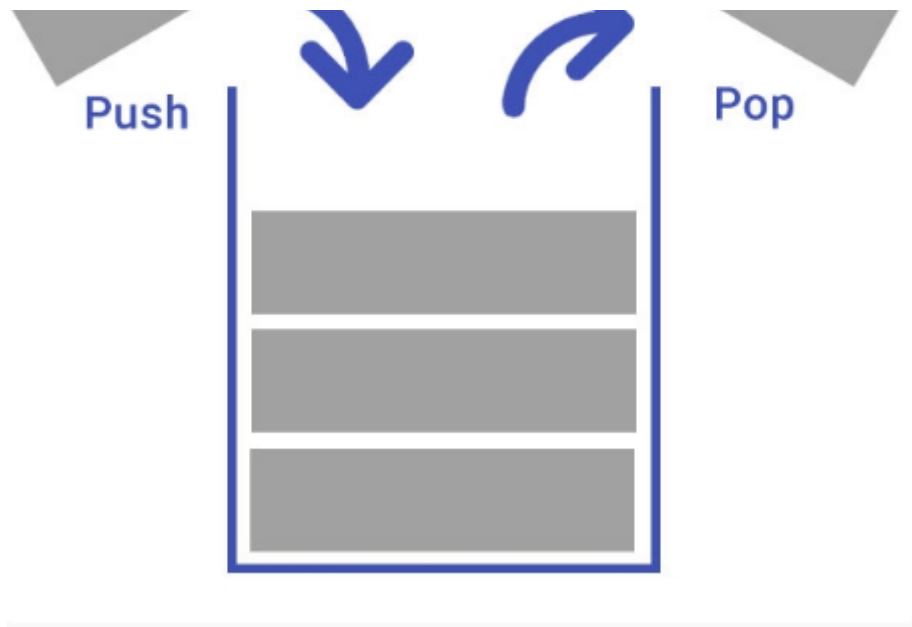
def print_list(self):
    n = self.head
    while n != None:
        print(n.data, end = " => ")
        n = n.next
    print()
```

Stacks and Queues

Stacks

- LIFO - Last In First Out





```
class Stack:
    def __init__(self):
        self.items = []

    def is_empty(self):
        return self.items == []

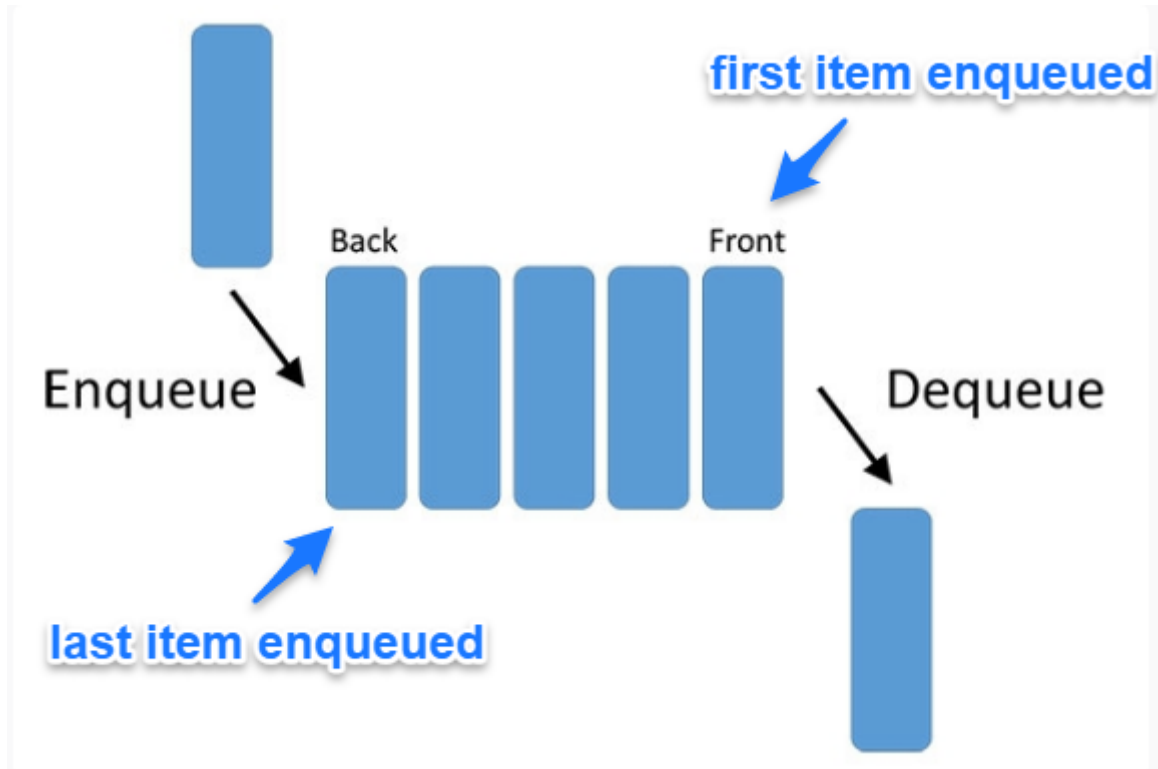
    def push(self, item):
        self.items.insert(0, item)

    def pop(self):
        return self.items.pop(0)

    def print_stack(self):
        print(self.items)
```

Queue

- FIFO - First In First Out



```
class Queue:
    def __init__(self):
        self.items = []
    def is_empty(self):
        return self.items == []
    def enqueue(self, item):
        self.items.insert(0, item)
    def dequeue(self):
        return self.items.pop()
    def print_queue(self):
        print(self.items)
```

Hash Tables

https://www.youtube.com/watch?v=shs0KM3wKv8&ab_channel=HackerRank

Hashtable -> key/value lookup

get a key and associate a value to look easily the data associated

 image.png

442 kB

how to jump from string to index - hash function

1. takes a string
2. convert string into int
3. converts int to an index in the array

Binary Trees:

 YouTube Video



Binary Trees in Python: Introduction and Traversa...



- trees are a way of path
- nodes are the points and the roots are the origin
- 3 types of traversals:
 - pre-order: Root>Left Tree>Right Tree
 - in-order: Left Tree>Root>Right Tree
 - post-order: Left Tree>Right Tree>Root

```
class Node:
    def __init__(self, value):
        self.value = value
        self.left = None
        self.right = None

class BinaryTree(Node):
    def __init__(self, root):
        self.root = Node(root)

    def print_tree(self, traversal_type):
        if traversal_type == 'preorder':
            return self.preorder_print(self.root, '')

        elif traversal_type == 'inorder':
            return self.inorder_print(self.root, '')

        elif traversal_type == 'postorder':
```

```

        return self.postorder_print(tree.root, '')

    else:
        print("traversal type " + str(traversal_type) + "not supported")
        return False

def preorder_print(self, start, traversal):
    # Root>Left Tree>Right Tree
    if start:
        traversal += (str(start.value) + '-')
        traversal = self.preorder_print(start.left, traversal)
        traversal = self.preorder_print(start.right, traversal)

    return traversal

def inorder_print(self, start, traversal):
    # Left Tree>Root>Right Tree
    if start:
        traversal = self.inorder_print(start.left, traversal)
        traversal += (str(start.value) + '-')
        traversal = self.inorder_print(start.right, traversal)

    return traversal

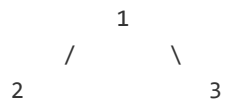
def postorder_print(self, start, traversal):
    # Left Tree>Right Tree>Root
    if start:
        traversal = self.postorder_print(start.left, traversal)
        traversal = self.postorder_print(start.right, traversal)
        traversal += (str(start.value) + '-')

    return traversal

```

"""

Tree Scheme




```

      /      \      /      \
     4        5    6        7
                   \
                   8
"""
```

```
tree = BinaryTree(1)
tree.root.left = Node(2)
tree.root.right = Node(3)

tree.root.left.left = Node(4)
tree.root.left.right = Node(5)
tree.root.right.left = Node(6)
tree.root.right.right = Node(7)

tree.root.right.right.right = Node(8)

print(tree.print_tree('preorder'))
print(tree.print_tree('inorder'))
print(tree.print_tree('postorder'))
```

Depth First Search

 YouTube Video



DEPTH FIRST SEARCH WITH PYTHON





dsdas
.