Alicia Sang's

DCS229 PORTFOLIO

Table of Contents

OOP, Classes, & Inheritance
Binary Tree Search
Big-O / Algorithm Analysis
Stack
Queue
Deque
Priority Queue
Heaps
Hashing
Recursion & Merge Sort
Maze Lab

OOP, CLASSES, & INHERITANCE

Class: Blueprint for objects
Object: Instance of a class
Inheritance: Allows subclasses to use or override methods/attributes from a parent class
Encapsulation: Hide internal state, expose functionality via public methods
Polymorphism: Same interface, different underlying forms

Big O

Object creation: O(1)
Method calls: O(1)

```
class Student:
def __init__(self, name):
     self.name = name
def greet(self):
     print(f"Hello, my name is
{self.name}.")
```

- Defined Shape, Triangle, Rectangle with __slots__
- Overrode methods like foobar() and __str__()
- Demonstrated polymorphism using processShape(shape)
- Used __eq__ to compare objects
- i.e. class: Dog, subclasses: different breeds of dogs (GeeksForGeeks.org)

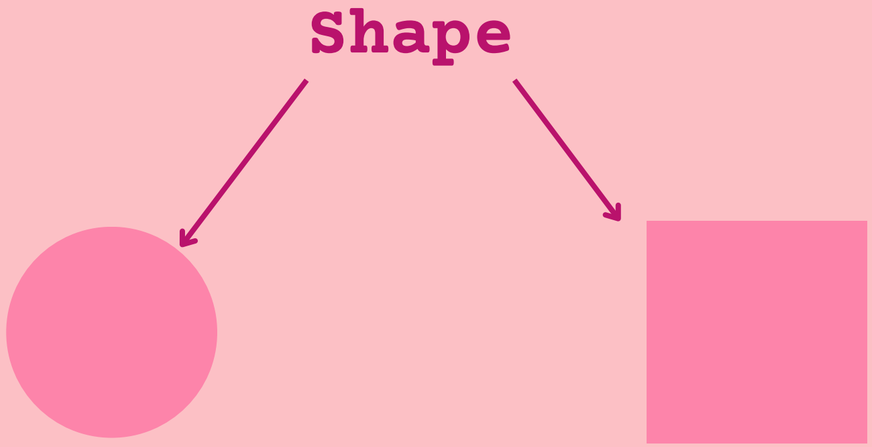
Practice question

Q: Define a Book class with attributes title, author, and isbn, and a method __eq__ to compare two books based on ISBN. Write the class definition and __eq__ method.

A:

```
class Book:
    def __init__(self, title, author, isbn):
        self.title = title
        self.author = author
        self.isbn = isbn

    def __eq__(self, other):
        return self.isbn == other.isbn
```



BINARY TREE SEARCH

BSTs maintain order: left < node < right
Operations include: insert(e), delete(e), search(e)
Can become **unbalanced** and degrade to linked-list behavior

Big O

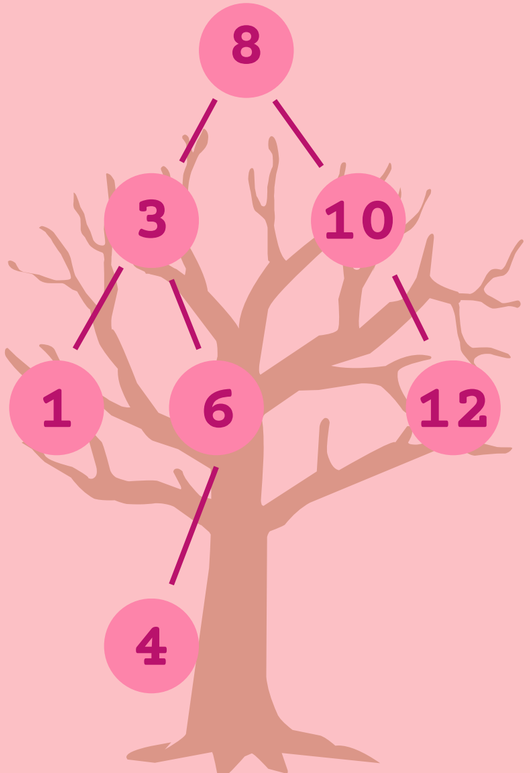
Average case: O(log n)
Worst case (unbalanced): O(n)

- Wrote insert and search functions for BSTs
- Compared balanced vs unbalanced trees
- Practiced inorder, preorder, and postorder traversal

practice question

Q: What's the worst-case time complexity for searching in a BST, and when does it occur?

A: O(n), when the tree is unbalanced and degenerate (e.g., each node only has a right child).



BIG-O ALGORITHM ANALYSIS

Big O: Describes algorithm efficiency
Amortized Analysis: Average performance over time (e.g., dynamic array resizing)

Big O

Sequential search: **O(n)**
Binary search: **O(log n)**
Merge sort: **O(n log n)**
Append to dynamic array: **Amortized O(1)**

- Compared selection sort vs merge sort vs binary search
- Counted operations to estimate runtime
- Analyzed Python list .append() using amortized analysis

```
def find_item(my_list, item):
    for i in range(len(my_list)):
        if my_list[i] == item:
        return i
    return None
```

practice question

Q: What is the Big-O runtime of the following function, and why?

```
def sum_pairs(nums):
    for i in nums:
      for j in nums:
        print(i + j)
```

A: O(n^2): Nested loop over the same list causes quadratic growth in operations as input size increases.

STACK

Operations: push(e), pop(), top(), is_empty(), len()

- Built a custom Stack class
- Used stack for DFS in maze lab
- Related stacks to Python's call stack for recursion

Big O

Push/Pop/Top: O(1)

```
class Stack:
def __init__(self):
     self.data = []
def push(self, e):
     self.data.append(e)
def pop(self):
     return self.data.pop()
```

you can only eat the pancake on the top, it would be hard to eat the one on the bottom first!

Practice question

Q: What is the output of this code?

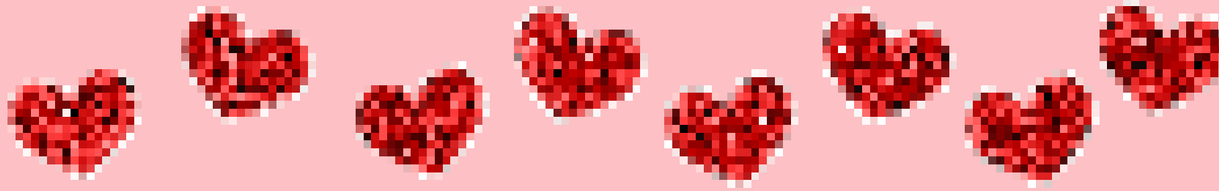
```
s = Stack()
s.push(1)
s.push(2)
print(s.pop())
s.push(3)
print(s.top())
```

A:
2
3

Because stack is LIFO: 2 is popped, then 3 is the new top.

Uses:

- When a function runs, the computer uses a stack to store its info (like local variables).
- Once the function finishes, that info is removed (iQuanta).



QUEUE

Operations: enqueue(e), dequeue(), front(), is_empty()

Big O

Enqueue/Dequeue: O(1)

- Implemented a queue with deque
- Used queue for BFS in maze lab to find shortest paths

Uses:
To run multiple programs simultaneously (GeeksForGeeks)

it would only be fair if the girl in the pink got on the bus first!

Practice question

Q: You are implementing a queue using a Python list. Which operation is slower and why: enqueue() using append() or dequeue() using pop(0)?
A: dequeue() using pop(0) is slower — O(n) — because it requires shifting all remaining elements left after removal.

DEQUE

Big O

All ops: O(1)

Operations: add_first(e), add_last(e), delete_first(), delete_last()

- Used deque to check palindromes
- Applied deque in flexible BFS traversal scenarios

Uses:
Cache: most and least recently used both accessible (Quora)

look! you can drink the juice from the top AND the bottom!

Practice question

Q: Consider a deque implemented via a doubly linked list. What is the Big-O of inserting at both ends, and why?
A: O(1) for both add_first() and add_last() — because insertion only requires updating a constant number of pointers at the head or tail.

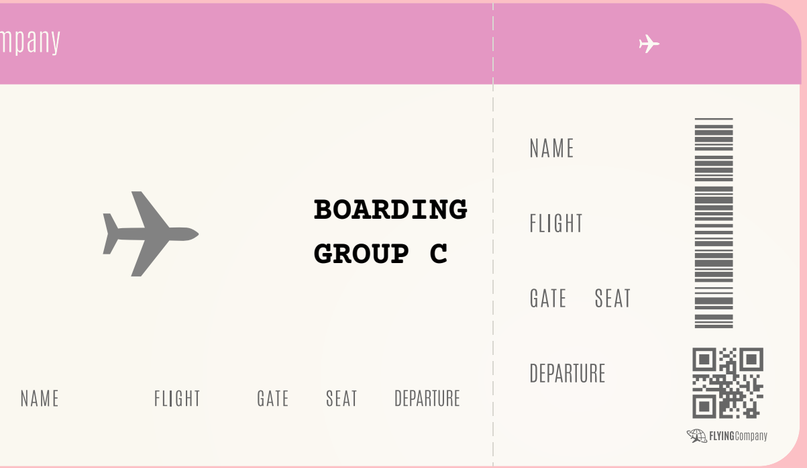
PRIORITY QUEUE

Operations: insert(e), remove_min(), min()

Big O

Insert: O(log n)
Remove Min: O(log n)

- Used priority queue (min-heap) for A* in maze lab
- Compared cost functions: g(n), h(n), f(n)



Uses:
Medical Triage (GeeksForGeeks)

Practice question

Q: You are maintaining a priority queue using a min-heap. What is the cost of removing the highest-priority item and why?
A: O(log n) — removing the root requires bubbling down the last element to restore the heap-order property.

HEAPS

Big O

Insert: O(log n),
Remove Min: O(log n),
Heapify: O(n)

- Explored min-heap operations with heapq
- Tracked heap structure during A* pathfinding

```
# Insert: Add to end, bubble
up
# Remove min: Replace root
with last, bubble down
```

Uses:
Scheduling prioritites (StackOverflow)

Practice question

Q: Given a min-heap represented as an array: [2, 5, 3, 9, 6], show the array after inserting 1 and then reheapifying.
A:
Insert 1 → [2, 5, 3, 9, 6, 1]
Bubble up → [1, 5, 2, 9, 6, 3]

HASHING

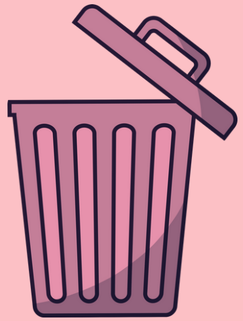
Operations: insert(k, v), get(k), delete(k)

Big O

O(1) average,
O(n) worst

- Used dictionaries to track visited positions in maze
- Learned how hash collisions can degrade performance

Uses:
Password managers, compare password input to expected password (builtin)



Practice question

Q: Explain why Python's in operator on a dictionary is O(1) on average. What would cause it to degrade?
A: Average case: O(1) due to direct indexing via hash functions. Worst case: O(n) if many collisions occur and all items land in the same bucket (poor hash function or hash flooding).

RECURSION & MERGE SORT

Big O

O(n log n) in all cases

- Wrote recursive merge sort
- Used recursion in DFS

Practice question

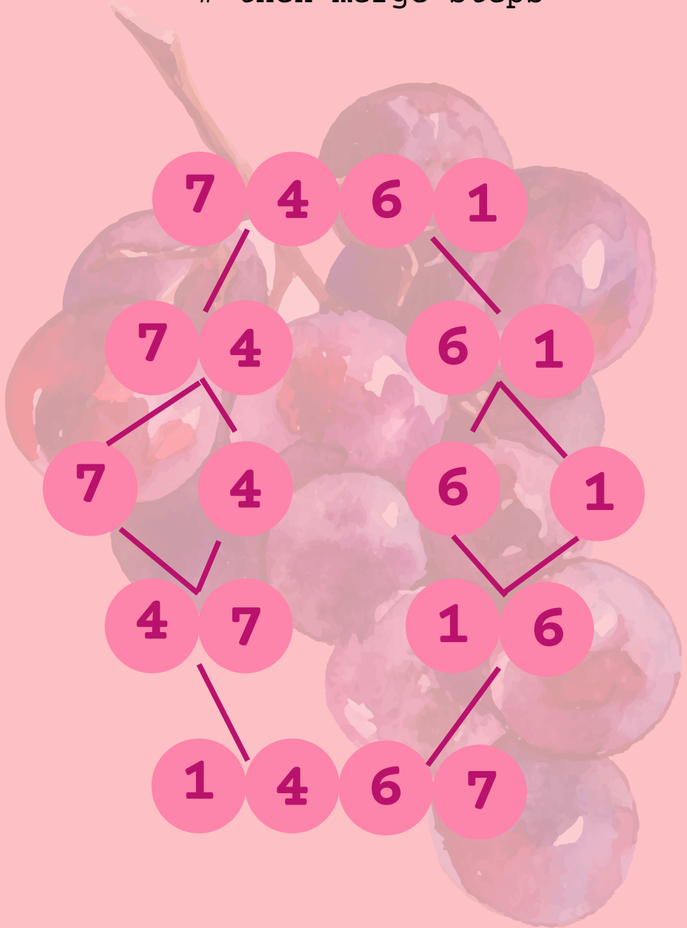
Q: What is the time complexity?

```
def func(n):
    if n <= 1:
        return 1
    return func(n-1) +
func(n-1)
```

A: $O(2^n)$: Each call branches into two more calls, leading to exponential growth.

Uses:
Alphabetizing a pile of resumes (Medium).

```
def merge_sort(arr):
  if len(arr) > 1:
    mid = len(arr)//2
    L = arr[:mid]
    R = arr[mid:]
  merge_sort(L)
  merge_sort(R)
# then merge steps
```



MAZE LAB

DFS: Uses a stack. Goes deep first. Fast, but not guaranteed to find the shortest path.
BFS: Uses a queue. Explores level by level. Guaranteed to find the shortest path.
A*: Uses a priority queue (via a heap). Combines cost-so-far (g(n)) + estimated cost to goal (h(n)) to make smart choices.
Manhattan Distance: Used as a heuristic h(n) for A*. It estimates how far we are from the goal.
Explored Dictionary: Stored previously visited positions as keys (based on Position objects). This made lookup fast and avoided revisiting cells.

Pitfalls:

- Forgetting to check if a neighbor is already explored before pushing/enqueuing
- Not updating the cost properly in A*

- DFS is simple and fast but not optimal
- BFS is great when all steps cost the same and you need the shortest path
- A* is ideal when steps have costs or when a heuristic helps prioritize

Practice question

Q: Why is A* faster and more effective than BFS in many maze-solving t**asks?**

A: Because it uses a heuristic (like Manhattan distance) to guess which paths are more promising, which reduces unnecessary exploration.

References

- https://www.iquanta.in/blog/top-10-applications-of-stack-in-data-structure-in-2025/?utm_source=google.com
- https://www.geeksforgeeks.org/applications-advantages-and-disadvantages-of-queue/
- https://www.quora.com/What-are-some-of-the-real-life-application-of-Deque
- https://www.geeksforgeeks.org/applications-priority-queue/
- https://stackoverflow.com/questions/749199/when-would-i-want-to-use-a-heap
- https://builtin.com/articles/what-is-hashing#:~:text=What%20is%20an%20example%20of,granted%20access%20to%20the%20account.
- https://medium.com/@teamtechsis/understanding-recursion-through-practical-examples-a3c586011f9d#:~:text=Recursion%20in%20the%20Real%20World&text=People%20often%20sort%20stacks%20of,letter%2C%20then%20sort%20each%20pile.