

Double-click (or enter) to edit

```
1 # Basic Operations and Complexity
2 class ArrayOperations:
3     def __init__(self, size):
4         self.size = size
5         self.array = [None] * size
6         self.length = 0
7
8     def access(self, index):
9         if 0 <= index < self.length:
10            return self.array[index]
11        raise IndexError("Index out of bounds")
12
13    def insert(self, index, element):
14        if self.length >= self.size:
15            raise OverflowError("Array is full")
16
17        for i in range(self.length, index, -1):
18            self.array[i] = self.array[i - 1]
19
20        self.array[index] = element
21        self.length += 1
22
23    def delete(self, index):
24        if index >= self.length:
25            raise IndexError("Index out of bounds")
26
27        for i in range(index, self.length - 1):
28            self.array[i] = self.array[i + 1]
29
30        self.array[self.length - 1] = None
31        self.length -= 1
32
33    def display(self):
34        return [self.array[i] for i in range(self.length)]
35
36
37
38 # Shopping Cart System
39
40 # Example 1: Managing a Shopping Cart
41 def shopping_cart_example():
42     print("\n==== Shopping Cart Example ====")
43     # Initialize cart with size 10
44     cart = ArrayOperations(10)
45
46     # Add items to cart
47     items = [
48         {"id": 1, "name": "Laptop", "price": 999.99},
49         {"id": 2, "name": "Mouse", "price": 29.99},
50         {"id": 3, "name": "Keyboard", "price": 59.99}
51     ]
52
53     # Demonstrate insert operation
54     print("Adding items to cart:")
55     for i, item in enumerate(items):
56         cart.insert(i, item)
57         print(f"Added {item['name']} - ${item['price']}")
58
59     print("\nCurrent cart contents:")
60     print(cart.display())
61
62     # Demonstrate delete operation
63     print("\nRemoving second item (Mouse)... ")
64     cart.delete(1)
65     print("Updated cart contents:")
66     print(cart.display())
67
68     # Demonstrate access operation
69     first_item = cart.access(0)
70     print(f"\nFirst item in cart: {first_item['name']} - ${first_item['price']}")
71
72 # Run example
73 shopping_cart_example()
74
75
76
```

```

78 # Example 2: Student Roster Management
79 def student_roster_example():
80     print("\n==== Student Roster Example ====")
81     roster = ArrayOperations(20)
82
83     # Sample student data
84     students = [
85         {"id": "S001", "name": "John", "grade": "A"},
86         {"id": "S002", "name": "Emma", "grade": "B+"},
87         {"id": "S003", "name": "Michael", "grade": "A-"}
88     ]
89
90     # Add initial students
91     print("Adding students to roster:")
92     for i, student in enumerate(students):
93         roster.insert(i, student)
94     print(f"Added {student['name']} (ID: {student['id']}), Grade: {student['grade']}")
```

95

```

96     # Show current roster
97     print("\nCurrent roster:")
98     for student in roster.display():
99         print(f"{student['name']}: {student['grade']}")
```

100

```

101    # Add new student in middle
102    new_student = {"id": "S004", "name": "Sarah", "grade": "B+"}
103    print(f"\nAdding new student {new_student['name']} at position 1")
104    roster.insert(1, new_student)
105
106    print("\nUpdated roster:")
107    for student in roster.display():
108        print(f"{student['name']}: {student['grade']}")
```

109

```

110 # Run example
111 student_roster_example()
112
113
114
115 #1.2 Search Operations in Arrays
116 def linear_search_detailed(array, target):
117     """Detailed linear search implementation"""
118     comparisons = 0
119
120     for index in range(len(array)):
121         comparisons += 1
122
123         if array[index] == target:
124             return {
125                 "index": index,
126                 "comparisons": comparisons,
127                 "found": True,
128                 "search_path": list(range(index + 1))
129             }
130
131     return {
132         "index": -1,
133         "comparisons": comparisons,
134         "found": False,
135         "search_path": list(range(len(array)))
136     }
```

137

```

138 def binary_search_detailed(array, target, key=lambda x: x):
139     """Detailed binary search implementation"""
140     left = 0
141     right = len(array) - 1
142     comparisons = 0
143     steps = []
144
145     while left <= right:
146         mid = (left + right) // 2
147         comparisons += 1
148         current_value = key(array[mid])
149
150         steps.append({
151             "left": left,
152             "right": right,
153             "mid": mid,
154             "current_value": current_value,
155             "action": "comparison"
156         })
```

```

156     .....
157
158     if current_value == target:
159         steps[-1]["action"] = "found"
160         return {
161             "index": mid,
162             "comparisons": comparisons,
163             "steps": steps,
164             "found": True
165         }
166     elif current_value < target:
167         steps[-1]["action"] = "move_right"
168         left = mid + 1
169     else:
170         steps[-1]["action"] = "move_left"
171         right = mid - 1
172
173     return {
174         "index": -1,
175         "comparisons": comparisons,
176         "steps": steps,
177         "found": False
178     }
179
180
181 # Example 1: Library Book Search
182 def library_search_example():
183     print("\n== Library Book Search Example ==")
184
185     # Book database
186     books = [
187         {"id": "B001", "title": "Python Programming", "author": "John Smith"},
188         {"id": "B002", "title": "Data Structures", "author": "Jane Doe"},
189         {"id": "B003", "title": "Algorithms", "author": "Alan Johnson"},
190         {"id": "B004", "title": "Web Development", "author": "Sarah Wilson"},
191         {"id": "B005", "title": "Machine Learning", "author": "Mike Brown"}
192     ]
193
194     # Linear search for a book by ID
195     search_id = "B003"
196     result = linear_search_detailed(books, search_id)
197
198     print(f"\nSearching for book ID: {search_id}")
199     if result["found"]:
200         book = books[result["index"]]
201         print(f"Found: {book['title']} by {book['author']}")
202         print(f"Comparisons made: {result['comparisons']}")
203     else:
204         print("Book not found")
205
206 # Run all examples
207 library_search_example()
208
209
210
211 # Example 2: Student Grade Search
212 def grade_search_example():
213     print("\n== Student Grade Search Example ==")
214
215     # Sorted student grades
216     students = [
217         {"id": "S001", "name": "Alice", "grade": 75},
218         {"id": "S002", "name": "Bob", "grade": 82},
219         {"id": "S003", "name": "Charlie", "grade": 88},
220         {"id": "S004", "name": "David", "grade": 92},
221         {"id": "S005", "name": "Eve", "grade": 95}
222     ]
223
224     # Binary search for grade
225     target_grade = 88
226     result = binary_search_detailed(
227         students,
228         target_grade,
229         key=lambda x: x["grade"]
230     )
231
232     print(f"\nSearching for grade: {target_grade}")
233     if result["found"]:
234         student = students[result["index"]]
235         print(f"Found student: {student['name']}")

```

```

236     ..... print("\nSearch steps:")
237     ..... for i, step in enumerate(result["steps"], 1):
238     .....     print(f"Step {i}:")
239     .....     print(f"Checking range: {step['left']} to {step['right']}")
240     .....     print(f"Middle element: {step['mid']}")
241     .....     print(f"Action: {step['action']}")
242     ..... else:
243     .....     print("Grade not found")
244
245 # Run all examples
246 grade_search_example()
247
248
249
250 # Example 3: Phone Directory Search
251 def phone_directory_example():
252     ..... print("\n==== Phone Directory Search Example ====")
253
254     ..... # Sorted phone directory
255     ..... contacts = [
256     .....     {"name": "Adams, John", "phone": "123-456-7890"},
257     .....     {"name": "Brown, Mary", "phone": "234-567-8901"},
258     .....     {"name": "Davis, Steve", "phone": "345-678-9012"},
259     .....     {"name": "Johnson, Lisa", "phone": "456-789-0123"},
260     .....     {"name": "Wilson, Mike", "phone": "567-890-1234"}
261     ..... ]
262
263     ..... # Binary search by name
264     ..... search_name = "Davis, Steve"
265     ..... result = binary_search_detailed(
266     .....     contacts,
267     .....     search_name,
268     .....     key=lambda x: x["name"]
269     ..... )
270
271     ..... print(f"\nSearching for contact: {search_name}")
272     ..... if result["found"]:
273     .....     contact = contacts[result["index"]]
274     .....     print(f"Found: {contact['name']}")
275     .....     print(f"Phone: {contact['phone']}")
276     .....     print(f"Found in {result['comparisons']} comparisons")
277     ..... else:
278     .....     print("Contact not found")
279
280 # Run all examples
281 phone_directory_example()
282
283
284 #Student Management System
285 class StudentManagementSystem:
286     ..... def __init__(self, capacity):
287     .....     self.roster = ArrayOperations(capacity)
288     .....     self.id_index = {} # For quick lookups by ID
289
290     ..... def add_student(self, student):
291     .....     # Add to array
292     .....     index = self.roster.length
293     .....     self.roster.insert(index, student)
294
295     ..... # Add to index
296     .....     self.id_index[student["id"]] = index
297
298     ..... print(f"Added student: {student['name']}")
299
300     ..... def find_student_by_id(self, student_id):
301     .....     # Direct lookup by ID
302     .....     if student_id in self.id_index:
303     .....         index = self.id_index[student_id]
304     .....         return self.roster.access(index)
305     .....     return None
306
307     ..... def find_students_by_grade(self, grade):
308     .....     # Linear search for all students with matching grade
309     .....     matches = []
310     .....     for i in range(self.roster.length):
311     .....         student = self.roster.access(i)
312     .....         if student["grade"] == grade:
313     .....             matches.append(student)
314     .....     return matches
315

```

```

316 # Usage example
317 def run_student_management_example():
318     print("\n==== Student Management System Example ===")
319
320     # Initialize system
321     sms = StudentManagementSystem(100)
322
323     # Add some students
324     students = [
325         {"id": "S001", "name": "John Doe", "grade": "A", "major": "CS"},
326         {"id": "S002", "name": "Jane Smith", "grade": "B", "major": "Math"},
327         {"id": "S003", "name": "Bob Wilson", "grade": "A", "major": "Physics"},
328         {"id": "S004", "name": "Alice Brown", "grade": "A", "major": "CS"}
329     ]
330
331     for student in students:
332         sms.add_student(student)
333
334     # Find student by ID
335     print("\nSearching for student S003:")
336     student = sms.find_student_by_id("S003")
337     if student:
338         print(f"Found: {student['name']} - {student['major']}")
339
340     # Find students with grade A
341     print("\nSearching for students with grade A:")
342     students = sms.find_students_by_grade("A")
343     for student in students:
344         print(f"{student['name']} - {student['major']}")
345
346 # Run the example
347 run_student_management_example()
348
349
350
351 #Hash Tables
352
353 class SimpleHashTable:
354     """A basic hash table implementation using chaining for collision resolution."""
355
356     def __init__(self, size=100):
357         self.size = size
358         self.table = [[] for _ in range(size)]
359
360     def _hash(self, key):
361         """Simple hash function."""
362         return hash(key) % self.size
363
364     def insert(self, key, value):
365         """Insert a key-value pair"""
366         index = self._hash(key)
367
368         # Check for existing key
369         for item in self.table[index]:
370             if item[0] == key:
371                 item[1] = value # Update value
372                 return
373
374         # Add new key-value pair
375         self.table[index].append([key, value])
376
377     def get(self, key):
378         """Retrieve value for given key"""
379         index = self._hash(key)
380
381         for item in self.table[index]:
382             if item[0] == key:
383                 return item[1]
384         return None
385
386     def delete(self, key):
387         """Delete a key-value pair"""
388         index = self._hash(key)
389
390         for i, item in enumerate(self.table[index]):
391             if item[0] == key:
392                 return self.table[index].pop(i)[1]
393         return None
394
395

```

```

396 #Multi-Index Hash Tables
397
398 class EmployeeDirectory:
399     """
400     ... Employee directory with multiple search indices
401     """
402     def __init__(self):
403         self.by_id = {} # Primary index
404         self.by_department = {} # Department index
405         self.by_salary_range = {} # Salary range index
406
407     def add_employee(self, emp_id, name, department, salary):
408         """Add employee with multiple indices"""
409         employee = {
410             "id": emp_id,
411             "name": name,
412             "department": department,
413             "salary": salary
414         }
415
416         # Add to primary index
417         self.by_id[emp_id] = employee
418
419         # Add to department index
420         if department not in self.by_department:
421             self.by_department[department] = []
422             self.by_department[department].append(employee)
423
424         # Add to salary range index
425         salary_range = (salary // 10000) * 10000
426         if salary_range not in self.by_salary_range:
427             self.by_salary_range[salary_range] = []
428             self.by_salary_range[salary_range].append(employee)
429
430     def find_by_id(self, emp_id):
431         """O(1) lookup by ID"""
432         return self.by_id.get(emp_id)
433
434     def find_by_department(self, department):
435         """O(1) lookup by department"""
436         return self.by_department.get(department, [])
437
438     def find_by_salary_range(self, min_salary, max_salary):
439         """Find employees within salary range"""
440         results = []
441         for range_start in self.by_salary_range:
442             if min_salary <= range_start <= max_salary:
443                 results.extend(self.by_salary_range[range_start])
444
445
446 # Example usage
447 def demonstrate_employee_directory():
448     print("\n== Employee Directory Example ==\n")
449     directory = EmployeeDirectory()
450
451     # Add sample employees
452     employees = [
453         ("E001", "John Doe", "IT", 75000),
454         ("E002", "Jane Smith", "HR", 50000),
455         ("E003", "Bob Wilson", "IT", 88000),
456         ("E004", "Alice Brown", "Finance", 70000)
457     ]
458
459     for emp_id, name, dept, salary in employees:
460         directory.add_employee(emp_id, name, dept, salary)
461         print(f"Added {name} (ID: {emp_id})")
462
463     # Demonstrate different search methods
464     print("\nSearching by ID (E003):")
465     emp = directory.find_by_id("E003")
466     print(f"Found: {emp['name']} - {emp['department']}")
467
468     print("\nIT Department employees:")
469     it_emps = directory.find_by_department("IT")
470     for emp in it_emps:
471         print(f"{emp['name']} - {emp['salary']}")
472
473     print("\nEmployees with salary 70000-80000:")
474     salary_range = directory.find_by_salary_range(70000, 80000)
475     for emp in salary_range:

```

```

476     print(f"{emp['name']} - ${emp['salary']}")
477
478 # Run example
479 demonstrate_employee_directory()
480
481
482 #Linked List
483 class Node:
484     """A single node in the linked list
485
486     Attributes:
487         data: The node's data
488         next: Reference to the next node
489     """
490     def __init__(self, data):
491         self.data = data
492         self.next = None
493
494 class LinkedList:
495     """Linked List implementation with search capabilities
496
497     Attributes:
498         head: First node in the list
499     """
500     def __init__(self):
501         self.head = None
502
503     def append(self, data):
504         """Add a new node at the end O(n)"""
505         if not self.head:
506             self.head = Node(data)
507             return
508
509         current = self.head
510         while current.next:
511             current = current.next
512         current.next = Node(data)
513
514     def insert_front(self, data):
515         """Add a new node at the beginning O(1)"""
516         new_node = Node(data)
517         new_node.next = self.head
518         self.head = new_node
519
520     def delete(self, data):
521         """Delete first occurrence of data O(n)"""
522         if not self.head:
523             return False
524
525         if self.head.data == data:
526             self.head = self.head.next
527             return True
528
529         current = self.head
530         while current.next:
531             if current.next.data == data:
532                 current.next = current.next.next
533                 return True
534             current = current.next
535         return False
536
537     def linear_search(self, target):
538         """Search for an element and return its position"""
539         current = self.head
540         position = 0
541
542         while current:
543             if current.data == target:
544                 return {
545                     "Position": position,
546                     "found": True,
547                     "data": current.data
548                 }
549             current = current.next
550             position += 1
551
552         return {
553             "Position": -1,
554             "found": False,
555             "data": None

```

```

555         added = None
556     .....
557
558     def display(self):
559         """Returns list of all elements"""
560         elements = []
561         current = self.head
562         while current:
563             elements.append(current.data)
564             current = current.next
565         return elements
566
567
568
569
570 # Example: Library Book Management
571 class Book:
572     """Book record for library management"""
573     def __init__(self, isbn, title, author):
574         self.isbn = isbn
575         self.title = title
576         self.author = author
577
578     def __eq__(self, other):
579         """Enable comparison with ISBN string"""
580         if isinstance(other, str):
581             return self.isbn == other
582         return False
583
584     def __str__(self):
585         return f'{self.title} by {self.author} (ISBN: {self.isbn})'
586
587 def demonstrate_library_linked_list():
588     print("\n==== Library Book Management Example ====")
589
590     # Create a LinkedList catalog
591     catalog = LinkedList()
592
593     # Add sample books
594     books = [
595         Book("978-0001", "Python Basics", "John Smith"),
596         Book("978-0002", "Data Structures", "Jane Doe"),
597         Book("978-0003", "Algorithms", "Bob Wilson")
598     ]
599
600     print("Adding books to catalog:")
601     for book in books:
602         catalog.append(book)
603         print(f"Added: {book}")
604
605     # Search for a book
606     print("\nSearching for ISBN: 978-0002")
607     result = catalog.linear_search("978-0002")
608     if result["found"]:
609         book = result["data"]
610         print(f"Found at position {result['Position']}: {book}")
611     else:
612         print("Book not found")
613
614     # Delete a book
615     print("\nDeleting book with ISBN: 978-0001")
616     catalog.delete("978-0001")
617     print("Updated catalog:")
618     for book in catalog.display():
619         print(book)
620
621 # Run the example
622 demonstrate_library_linked_list()
623
624
625
626 #Binary Search Tree
627 class TreeNode:
628     """Binary Search Tree Node
629
630     Attributes:
631         data: Node's data
632         left: Left child
633         right: Right child
634     """
635     def __init__(self, data):

```

```

635     def __init__(self, data):
636         self.data = data
637         self.left = None
638         self.right = None
639
640 class BinarySearchTree:
641     """Binary Search Tree implementation with search operations"""
642
643     def __init__(self):
644         self.root = None
645
646     def insert(self, data):
647         """Insert new node with data"""
648         if not self.root:
649             self.root = TreeNode(data)
650         else:
651             self._insert_recursive(self.root, data)
652
653     def _insert_recursive(self, node, data):
654         """Recursively find correct position and insert"""
655         if data["id"] < node.data["id"]:
656             if node.left is None:
657                 node.left = TreeNode(data)
658             else:
659                 self._insert_recursive(node.left, data)
660         else:
661             if node.right is None:
662                 node.right = TreeNode(data)
663             else:
664                 self._insert_recursive(node.right, data)
665
666     def search(self, target_id):
667         """Search for node with target_id"""
668         result = self._search_recursive(self.root, target_id)
669         return {
670             "found": result is not None,
671             "data": result,
672             "comparisons": self._count_comparisons(self.root, target_id)
673         }
674
675     def _search_recursive(self, node, target_id):
676         """Recursive search operation"""
677         if node is None or node.data["id"] == target_id:
678             return node.data if node else None
679
680         if target_id < node.data["id"]:
681             return self._search_recursive(node.left, target_id)
682         return self._search_recursive(node.right, target_id)
683
684     def _count_comparisons(self, node, target_id, count=0):
685         """Count number of comparisons in search"""
686         if node is None:
687             return count
688
689         count += 1
690         if node.data["id"] == target_id:
691             return count
692         if target_id < node.data["id"]:
693             return self._count_comparisons(node.left, target_id, count)
694         return self._count_comparisons(node.right, target_id, count)
695
696
697
698 def demonstrate_student_bst():
699     print("\n== Student Record System Using BST ==")
700
701     # Create BST
702     student_tree = BinarySearchTree()
703
704     # Add sample students
705     students = [
706         {"id": 101, "name": "Alice", "grade": 95},
707         {"id": 103, "name": "Bob", "grade": 88},
708         {"id": 102, "name": "Charlie", "grade": 92},
709         {"id": 105, "name": "David", "grade": 85},
710         {"id": 104, "name": "Eve", "grade": 90}
711     ]
712
713     print("Adding students to BST:")
714     for student in students:

```

```

715     student_tree.insert(student)
716     print(f"Added: {student['name']} (ID: {student['id']})")
717
718     # Search demonstrations
719     test_ids = [102, 106] # One existing, one non-existing
720     for student_id in test_ids:
721         print(f"\nSearching for student ID: {student_id}")
722         result = student_tree.search(student_id)
723         if result["found"]:
724             student = result["data"]
725             print(f"Found: {student['name']} - Grade: {student['grade']}")
726         else:
727             print("Student not found")
728
729     print(f"Number of comparisons: {result['comparisons']}")
730
731 # Run the demonstration
732 demonstrate_student_bst()
733
734
735
736
737 #Enhanced Binary Search Tree
738 class TreeNode:
739     """
740         Enhanced Binary Search Tree Node
741         Attributes:
742             data: Node's data
743             left: Left child
744             right: Right child
745             height: Height of node for AVL balancing
746     """
747     def __init__(self, data):
748         self.data = data
749         self.left = None
750         self.right = None
751         self.height = 1 # Used for AVL balancing
752
753 class BalancedBST:
754     """
755         Self-balancing Binary Search Tree (AVL Tree) implementation
756         with traversal methods
757     """
758     def __init__(self):
759         self.root = None
760
761     def _get_height(self, node):
762         """Get height of node"""
763         if not node:
764             return 0
765         return node.height
766
767     def _get_balance(self, node):
768         """Get balance factor of node"""
769         if not node:
770             return 0
771         return self._get_height(node.left) - self._get_height(node.right)
772
773     def _update_height(self, node):
774         """Update height of node"""
775         if not node:
776             return
777         node.height = max(self._get_height(node.left),
778                           self._get_height(node.right)) + 1
779
780     def _right_rotate(self, y):
781         """Right rotation for balancing"""
782         x = y.left
783         T2 = x.right
784
785         x.right = y
786         y.left = T2
787
788         self._update_height(y)
789         self._update_height(x)
790
791     return x
792
793     def _left_rotate(self, x):
794         """Left rotation for balancing"""

```

```

795 .....     y = x.right
796 .....     T2 = y.left
797
798 .....     y.left = x
799 .....     x.right = T2
800
801 .....     self._update_height(x)
802 .....     self._update_height(y)
803
804 .....     return y
805
806 def insert(self, data):
807     """Insert new node and maintain balance"""
808     self.root = self._insert_recursive(self.root, data)
809
810 def _insert_recursive(self, node, data):
811     """Recursive insert with balancing"""
812     if not node:
813         return TreeNode(data)
814
815     if data["id"] < node.data["id"]:
816         node.left = self._insert_recursive(node.left, data)
817     else:
818         node.right = self._insert_recursive(node.right, data)
819
820     # Update height
821     self._update_height(node)
822
823     # Get balance factor
824     balance = self._get_balance(node)
825
826     # Balance the tree if needed
827     # Left Left Case
828     if balance > 1 and data["id"] < node.left.data["id"]:
829         return self._right_rotate(node)
830
831     # Right Right Case
832     if balance < -1 and data["id"] > node.right.data["id"]:
833         return self._left_rotate(node)
834
835     # Left Right Case
836     if balance > 1 and data["id"] > node.left.data["id"]:
837         node.left = self._left_rotate(node.left)
838         return self._right_rotate(node)
839
840     # Right Left Case
841     if balance < -1 and data["id"] < node.right.data["id"]:
842         node.right = self._right_rotate(node.right)
843         return self._left_rotate(node)
844
845     return node
846
847     # Traversal Methods
848 def inorder(self):
849     """Inorder traversal: left -> root -> right"""
850     result = []
851     self._inorder_recursive(self.root, result)
852     return result
853
854 def _inorder_recursive(self, node, result):
855     if node:
856         self._inorder_recursive(node.left, result)
857         result.append(node.data)
858         self._inorder_recursive(node.right, result)
859
860 def preorder(self):
861     """Preorder traversal: root -> left -> right"""
862     result = []
863     self._preorder_recursive(self.root, result)
864     return result
865
866 def _preorder_recursive(self, node, result):
867     if node:
868         result.append(node.data)
869         self._preorder_recursive(node.left, result)
870         self._preorder_recursive(node.right, result)
871
872 def postorder(self):
873     """Postorder traversal: left -> right -> root"""
874     result = []

```

```

875     ..... self._postorder_recursive(self.root, result)
876     ..... return result
877
878 def _postorder_recursive(self, node, result):
879     ..... if node:
880         ..... self._postorder_recursive(node.left, result)
881         ..... self._postorder_recursive(node.right, result)
882         ..... result.append(node.data)
883
884 def level_order(self):
885     """Level order (breadth-first) traversal"""
886     if not self.root:
887         ..... return []
888
889     ..... result = []
890     ..... queue = [self.root]
891
892     ..... while queue:
893         ..... node = queue.pop(0)
894         ..... result.append(node.data)
895
896         ..... if node.left:
897             ..... queue.append(node.left)
898             ..... if node.right:
899                 ..... queue.append(node.right)
900
901     ..... return result
902
903 def search(self, target_id):
904     """Search with path tracking"""
905     ..... path = []
906     ..... result = self._search_recursive(self.root, target_id, path)
907     ..... return {
908         ..... "found": result is not None,
909         ..... "data": result,
910         ..... "path": path
911     }
912
913 def _search_recursive(self, node, target_id, path):
914     ..... if not node:
915         ..... return None
916
917     ..... path.append(node.data["id"])
918
919     ..... if node.data["id"] == target_id:
920         ..... return node.data
921
922     ..... if target_id < node.data["id"]:
923         ..... return self._search_recursive(node.left, target_id, path)
924     ..... return self._search_recursive(node.right, target_id, path)
925
926
927 #Example Usage and Demonstration
928 def demonstrate_balanced_bst():
929     ..... print("\n==== Balanced BST Demonstration ====")
930
931     ..... tree = BalancedBST()
932
933     ..... # Sample student data
934     ..... students = [
935         ..... {"id": 5, "name": "Alice", "grade": 95},
936         ..... {"id": 3, "name": "Bob", "grade": 88},
937         ..... {"id": 7, "name": "Charlie", "grade": 92},
938         ..... {"id": 1, "name": "David", "grade": 85},
939         ..... {"id": 9, "name": "Eve", "grade": 90},
940         ..... {"id": 4, "name": "Frank", "grade": 87},
941         ..... {"id": 6, "name": "Grace", "grade": 93}
942     ]
943
944     ..... # Insert students
945     ..... print("Inserting students...")
946     ..... for student in students:
947         ..... tree.insert(student)
948         ..... print(f"Added: {student['name']} (ID: {student['id']})")
949
950     ..... # Demonstrate different traversals
951     ..... print("\nTraversal Demonstrations:")
952
953     ..... print("\nInorder Traversal (sorted by ID):")
954     ..... for student in tree.inorder():

```

```

955 .....print(f"{student['name']}:{student['id']}")
956
957 .....print("\nLevel Order Traversal (by tree level):")
958 .....for student in tree.level_order():
959 .....print(f"{student['name']}:{student['id']}")
960
961 .....# Search demonstration
962 .....print("\nSearch Demonstrations:")
963 .....search_ids = [4, 8] # One existing, one non-existing
964
965 .....for search_id in search_ids:
966 .....result = tree.search(search_id)
967 .....print(f"\nSearching for ID: {search_id}")
968 .....if result["found"]:
969 .....print(f"Found: {result['data']['name']}")
970 .....else:
971 .....print("Student not found")
972 .....print(f"Search path: {'->'.join(map(str, result['path']))}")
973
974 # Run demonstration
975 def main():
976 .....demonstrate_balanced_bst()
977
978 if __name__ == "__main__":
979 .....main()
980
981
982
983 #Array Operations
984 class DynamicArray:
985 .....def __init__(self, initial_size=4):
986 .....self.array = [None] * initial_size
987 .....self.size = initial_size
988 .....self.length = 0
989
990 .....# TODO: Implement these methods
991 .....def append(self, element):
992 ....."""Add element to end of array, resize if needed"""
993 .....pass
994
995 .....def insert(self, index, element):
996 ....."""Insert element at index"""
997 .....pass
998
999 .....def remove(self, index):
1000 ....."""Remove element at index"""
1001 .....pass
1002
1003 .....def get(self, index):
1004 ....."""Get element at index"""
1005 .....pass
1006
1007
1008 # Test cases
1009 def test_dynamic_array():
1010 .....arr = DynamicArray()
1011
1012 .....# Test 1: Append beyond initial size
1013 .....for i in range(6):
1014 .....arr.append(i)
1015 .....assert arr.size >= 6, "Array should resize"
1016
1017 .....# Test 2: Insert in middle
1018 .....arr.insert(2, 10)
1019 .....assert arr.get(2) == 10, "Insert failed"
1020
1021
1022
1023 #Array Search
1024 def find_duplicates(arr):
1025 ....."""
1026 .....Find all elements that appear more than once in the array
1027
1028 .....Example:
1029 .....Input: [1, 3, 4, 2, 2, 1, 5, 3]
1030 .....Output: [1, 2, 3]
1031 ....."""
1032 .....pass
1033
1034 def find_missing_number(arr):

```

```
1031 GET /api/messages/latest (any)
1035 -----
1036 -----Find the missing number in an array containing n-1 numbers from 1 to n
1037
1038 -----Example:
1039 -----Input: [1, 2, 4, 6, 3, 7, 8], n=8
1040 -----Output: 5
1041 -----
1042 -----pass
1043
1044 # Test cases
1045 def test_search_problems():
1046     # Test duplicates
1047     result = find_duplicates([1, 3, 4, 2, 2, 1, 5, 3])
1048     assert set(result) == {1, 2, 3}, "Duplicate finding failed"
1049
1050     # Test missing number
1051     result = find_missing_number([1, 2, 4, 6, 3, 7, 8])
1052     assert result == 5, "Missing number finding failed"
1053
1054
1055
```

```
== Shopping Cart Example ==
Adding items to cart:
Added Laptop - $999.99
Added Mouse - $29.99
Added Keyboard - $59.99

Current cart contents:
[{'id': 1, 'name': 'Laptop', 'price': 999.99}, {'id': 2, 'name': 'Mouse', 'price': 29.99}, {'id': 3, 'name': 'Keyboard', 'price': 59.99}]

Removing second item (Mouse)...
Updated cart contents:
[{'id': 1, 'name': 'Laptop', 'price': 999.99}, {'id': 3, 'name': 'Keyboard', 'price': 59.99}]

First item in cart: Laptop - $999.99

== Student Roster Example ==
Adding students to roster:
Added John (ID: S001), Grade: A
Added Emma (ID: S002), Grade: B
Added Michael (ID: S003), Grade: A-

Current roster:
John: A
Emma: B
Michael: A-

Adding new student Sarah at position 1

Updated roster:
John: A
Sarah: B+
Emma: B
Michael: A-

== Library Book Search Example ==
Searching for book ID: B003
Book not found

== Student Grade Search Example ==
Searching for grade: 88
Found student: Charlie

Search steps:
Step 1:
  Checking range: 0 to 4
  Middle element: 2
Action: found

== Phone Directory Search Example ==
Searching for contact: Davis, Steve
Found: Davis, Steve
Phone: 345-678-9012
Found in 1 comparisons
```

Double-click (or enter) to edit

