DSA

October 6, 2024

[ ]:
```
DSA Assignment
```

[3]:
```
# Problem 1: Reverse a singly linked list
class ListNode:
    def __init__(self, value=0, next=None):
        self.value = value
        self.next = next

def reverse_linked_list(head):
    prev = None
    current = head

    while current is not None:
        next_node = current.next
        current.next = prev
        prev = current
        current = next_node

    return prev
def print_linked_list(head):
    current = head
    while current:
        print(current.value, end=" -> ")
        current = current.next
    print("None")


head = ListNode(1, ListNode(2, ListNode(3, ListNode(4))))

print("Original Linked List:")
print_linked_list(head)

#
reversed_head = reverse_linked_list(head)

print("Reversed Linked List:")
print_linked_list(reversed_head)
```

```
Original Linked List:
1 -> 2 -> 3 -> 4 -> None
Reversed Linked List:
4 -> 3 -> 2 -> 1 -> None
```

[5]:
```python
# Problem 2: Merge two sorted linked lists into one sorted linked list.
class ListNode:
    def __init__(self, value=0, next=None):
        self.value = value
        self.next = next

def merge_two_sorted_lists(l1, l2):

    dummy = ListNode(0)
    current = dummy

    while l1 and l2:
        if l1.value < l2.value:
            current.next = l1
            l1 = l1.next
        else:
            current.next = l2
            l2 = l2.next
        current = current.next

    if l1:
        current.next = l1
    elif l2:
        current.next = l2

    return dummy.next

def print_linked_list(head):
    current = head
    while current:
        print(current.value, end=" -> ")
        current = current.next
    print("None")

l1 = ListNode(1, ListNode(3, ListNode(5)))

l2 = ListNode(2, ListNode(4, ListNode(6)))

print("List 1:")
print_linked_list(l1)

print("List 2:")
```

```
print_linked_list(l2)

merged_head = merge_two_sorted_lists(l1, l2)

print("Merged Linked List:")
print_linked_list(merged_head)
```

```
List 1:
1 -> 3 -> 5 -> None
List 2:
2 -> 4 -> 6 -> None
Merged Linked List:
1 -> 2 -> 3 -> 4 -> 5 -> 6 -> None
```

[7]:
```
# Problem 3: Remove the nth node from the end of a linked list.
class ListNode:
    def __init__(self, value=0, next=None):
        self.value = value
        self.next = next
def remove_nth_from_end(head, n):
    dummy = ListNode(0)
    dummy.next = head
    first = dummy
    second = dummy

    for _ in range(n + 1):
        first = first.next

    while first:
        first = first.next
        second = second.next

    second.next = second.next.next

    return dummy.next

def print_linked_list(head):
    current = head
    while current:
        print(current.value, end=" -> ")
        current = current.next
    print("None")

head = ListNode(1, ListNode(2, ListNode(3, ListNode(4, ListNode(5)))))

print("Original Linked List:")
print_linked_list(head)
```

```
n = 2
modified_head = remove_nth_from_end(head, n)

print(f"Linked List after removing the {n}th node from the end:")
print_linked_list(modified_head)
```

```
Original Linked List:
1 -> 2 -> 3 -> 4 -> 5 -> None
Linked List after removing the 2th node from the end:
1 -> 2 -> 3 -> 5 -> None
```

[9]:
```
# Problem 4: Find the intersection point of two linked lists.
class ListNode:
    def __init__(self, value=0, next=None):
        self.value = value
        self.next = next

def get_intersection_node(headA, headB):
    if not headA or not headB:
        return None

    pointerA = headA
    pointerB = headB

    while pointerA != pointerB:
        pointerA = pointerA.next if pointerA else headB
        pointerB = pointerB.next if pointerB else headA

    return pointerA

def print_linked_list_from(head):
    current = head
    while current:
        print(current.value, end=" -> ")
        current = current.next
    print("None")



intersection = ListNode(6, ListNode(7))

headA = ListNode(1, ListNode(2, ListNode(3, intersection)))

headB = ListNode(4, ListNode(5, intersection))

print("List A:")
```

```
print_linked_list_from(headA)

print("List B:")
print_linked_list_from(headB)

intersection_node = get_intersection_node(headA, headB)

if intersection_node:
    print(f"Intersection Point: {intersection_node.value}")
else:
    print("No intersection point.")
```

```
List A:
1 -> 2 -> 3 -> 6 -> 7 -> None
List B:
4 -> 5 -> 6 -> 7 -> None
Intersection Point: 6
```

[11]:
```
# Problem 5: Remove duplicates from a sorted linked list.

class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next
def deleteDuplicates(head: ListNode) -> ListNode:
    current = head

    while current and current.next:
        if current.val == current.next.val:
            current.next = current.next.next
        else:
            current = current.next

    return head
head = ListNode(1, ListNode(1, ListNode(2, ListNode(3, ListNode(3)))))
new_head = deleteDuplicates(head)

current = new_head
while current:
    print(current.val, end=" -> ")
    current = current.next
```

```
1 -> 2 -> 3 ->
```

[1]:
```
# Problem 6: Add two numbers represented by linked lists (where each node␣
 ↪contains a single digit).
#Input: List 1: 2 -> 4 -> 3, List 2: 5 -> 6 -> 4 (represents 342 + 465)
#Output: 7 -> 0 -> 8 (represents 807)
```

```python
class ListNode:
    def __init__(self, value=0, next=None):
        self.value = value
        self.next = next

def addTwoNumbers(l1, l2):
    dummy_head = ListNode(0)
    current = dummy_head
    carry = 0

    p1, p2 = l1, l2

    while p1 or p2 or carry:
        val1 = p1.value if p1 else 0
        val2 = p2.value if p2 else 0

        total = val1 + val2 + carry
        carry = total // 10
        current.next = ListNode(total % 10)

        current = current.next
        if p1: p1 = p1.next
        if p2: p2 = p2.next

    return dummy_head.next

def create_linked_list(digits):
    head = ListNode(digits[0])
    current = head
    for digit in digits[1:]:
        current.next = ListNode(digit)
        current = current.next
    return head

def print_linked_list(node):
    while node:
        print(node.value, end=" -> " if node.next else "")
        node = node.next
    print()

l1 = create_linked_list([2, 4, 3])
l2 = create_linked_list([5, 6, 4])
result = addTwoNumbers(l1, l2)
print_linked_list(result)
```

```
7 -> 0 -> 8
```

[3]:
```python
#Problem 7: Swap nodes in pairs in a linked list.
#Input: 1 -> 2 -> 3 -> 4
#Output: 2 -> 1 -> 4 -> 3
class ListNode:
    def __init__(self, value=0, next=None):
        self.value = value
        self.next = next

def swapPairs(head):
    dummy = ListNode(0)
    dummy.next = head
    prev = dummy

    while prev.next and prev.next.next:
        first = prev.next
        second = prev.next.next

        # Perform the swap
        first.next = second.next
        second.next = first
        prev.next = second

        # Move to the next pair
        prev = first

    return dummy.next

def create_linked_list(values):
    head = ListNode(values[0])
    current = head
    for value in values[1:]:
        current.next = ListNode(value)
        current = current.next
    return head

def print_linked_list(node):
    while node:
        print(node.value, end=" -> " if node.next else "")
        node = node.next
    print()


head = create_linked_list([1, 2, 3, 4])
print("Original list:")
print_linked_list(head)

swapped_head = swapPairs(head)
```

```
print("Swapped list:")
print_linked_list(swapped_head)
```

```
Original list:
1 -> 2 -> 3 -> 4
Swapped list:
2 -> 1 -> 4 -> 3
```

[5]:
```
# Problem 8: Reverse nodes in a linked list in groups of k
#Input: 1 -> 2 -> 3 -> 4 -> 5, k = 3
#Output: 3 -> 2 -> 1 -> 4 -> 5
class ListNode:
    def __init__(self, value=0, next=None):
        self.value = value
        self.next = next

def reverseKGroup(head, k):

    def reverseLinkedList(start, end):
        prev = None
        current = start
        while current != end:
            next_node = current.next
            current.next = prev
            prev = current
            current = next_node
        return prev


    count = 0
    current = head
    while current and count < k:
        current = current.next
        count += 1

    if count < k:
        return head


    new_head = reverseLinkedList(head, current)



    head.next = reverseKGroup(current, k)

    return new_head
```

```
def create_linked_list(values):
    head = ListNode(values[0])
    current = head
    for value in values[1:]:
        current.next = ListNode(value)
        current = current.next
    return head


def print_linked_list(node):
    while node:
        print(node.value, end=" -> " if node.next else "")
        node = node.next
    print()


head = create_linked_list([1, 2, 3, 4, 5])
k = 3
print("Original list:")
print_linked_list(head)

reversed_head = reverseKGroup(head, k)
print("Reversed in groups of k:")
print_linked_list(reversed_head)
```

```
Original list:
1 -> 2 -> 3 -> 4 -> 5
Reversed in groups of k:
3 -> 2 -> 1 -> 4 -> 5
```

[7]:
```
#Problem 9: Determine if a linked list is a palindrome
# Input: 1 -> 2 -> 2 -> 1
#Output: True

class ListNode:
    def __init__(self, value=0, next=None):
        self.value = value
        self.next = next

def isPalindrome(head):
    if not head or not head.next:
        return True


    slow, fast = head, head
    while fast and fast.next:
```

```
        slow = slow.next
        fast = fast.next.next


    prev = None
    current = slow
    while current:
        next_node = current.next
        current.next = prev
        prev = current
        current = next_node


    first_half, second_half = head, prev
    while second_half:
        if first_half.value != second_half.value:
            return False
        first_half = first_half.next
        second_half = second_half.next

    return True


def create_linked_list(values):
    head = ListNode(values[0])
    current = head
    for value in values[1:]:
        current.next = ListNode(value)
        current = current.next
    return head


head = create_linked_list([1, 2, 2, 1])
print("Is the linked list a palindrome?", isPalindrome(head))
```

```
Is the linked list a palindrome? True
```

[11]:
```
# Problem 10: Rotate a linked list to the right by k places
# Input: 1 -> 2 -> 3 -> 4 -> 5, k = 2
#Output: 4 -> 5 -> 1 -> 2 -> 3class ListNode:
def __init__(self, value=0, next=None):
        self.value = value
        self.next = next

def rotateRight(head, k):
    if not head or not head.next or k == 0:
        return head
```

```
    length = 1
    current = head
    while current.next:
        current = current.next
        length += 1

    # Step 2: Adjust k
    k = k % length
    if k == 0:
        return head

    current = head
    for _ in range(length - k - 1):
        current = current.next

    new_head = current.next
    current.next = None  # Break the list

    tail = new_head
    while tail and tail.next:
        tail = tail.next
    if tail:
        tail.next = head

    return new_head


def create_linked_list(values):
    head = ListNode(values[0])
    current = head
    for value in values[1:]:
        current.next = ListNode(value)
        current = current.next
    return head


def print_linked_list(node):
    while node:
        print(node.value, end=" -> " if node.next else "")
        node = node.next
    print()


head = create_linked_list([1, 2, 3, 4, 5])
k = 2
print("Original list:")
```

```
print_linked_list(head)

rotated_head = rotateRight(head, k)
print("Rotated list:")
print_linked_list(rotated_head)

class ListNode:
    def __init__(self, value=0, next=None):
        self.value = value
        self.next = next

def rotateRight(head, k):
    if not head or not head.next or k == 0:
        return head


    length = 1
    current = head
    while current.next:
        current = current.next
        length += 1


    k = k % length
    if k == 0:
        return head


    current = head
    for _ in range(length - k - 1):
        current = current.next

    new_head = current.next
    current.next = None  # Break the list

    tail = new_head
    while tail and tail.next:
        tail = tail.next
    if tail:
        tail.next = head

    return new_head


def create_linked_list(values):
    head = ListNode(values[0])
    current = head
```

```
    for value in values[1:]:
        current.next = ListNode(value)
        current = current.next
    return head


def print_linked_list(node):
    while node:
        print(node.value, end=" -> " if node.next else "")
        node = node.next
    print()


head = create_linked_list([1, 2, 3, 4, 5])
k = 2
print("Original list:")
print_linked_list(head)

rotated_head = rotateRight(head, k)
print("Rotated list:")
print_linked_list(rotated_head)
```

```
Original list:
1 -> 2 -> 3 -> 4 -> 5
Rotated list:
4 -> 5 -> 1 -> 2 -> 3
Original list:
1 -> 2 -> 3 -> 4 -> 5
Rotated list:
4 -> 5 -> 1 -> 2 -> 3
```

[15]:
```
# Problem 11: Flatten a multilevel doubly linked list.
#Input: 1 <-> 2 <-> 3 <-> 7 <-> 8 <-> 11 -> 12, 4 <-> 5 -> 9 -> 10, 6 -> 13
#Output: 1 <-> 2 <-> 3 <-> 4 <-> 5 <-> 6 <-> 7 <-> 8 <-> 9 <-> 10 <-> 11 <-> 12
 ↪<-> 13
class Node:
      def __init__(self, value=0, prev=None, next=None, child=None):
        self.value = value
        self.prev = prev
        self.next = next
        self.child = child

def flatten(head):
    if not head:
        return None
```

```
    stack = []
    current = head

    while current:
        if current.child:

            if current.next:
                stack.append(current.next)

            current.next = current.child
            current.child.prev = current
            current.child = None


        if not current.next and stack:
            current.next = stack.pop()
            current.next.prev = current

        current = current.next

    return head


def print_flattened_list(head):
    current = head
    while current:
        print(current.value, end=" <-> " if current.next else "")
        current = current.next
    print()




head = Node(1)
node2 = Node(2)
node3 = Node(3)
head.next = node2
node2.prev = head
node2.next = node3
node3.prev = node2


node4 = Node(4)
node5 = Node(5)
node3.next = node4
node4.prev = node3
node4.next = node5
```

```
node5.prev = node4

node6 = Node(6)
node3.child = node6

node7 = Node(7)
node8 = Node(8)
node9 = Node(9)
node10 = Node(10)
node11 = Node(11)
node12 = Node(12)
node13 = Node(13)

node4.next = node7
node7.prev = node4
node7.next = node8
node8.prev = node7

node5.next = node9
node9.prev = node5
node9.next = node10
node10.prev = node9
node10.next = node11
node11.prev = node10
node11.next = node12
node12.prev = node11
node6.next = node13
node13.prev = node6

print("Original multilevel doubly linked list:")
print_flattened_list(head)

flattened_head = flatten(head)
print("Flattened linked list:")
print_flattened_list(flattened_head)
```

```
Original multilevel doubly linked list:
1 <-> 2 <-> 3 <-> 4 <-> 7 <-> 8
Flattened linked list:
1 <-> 2 <-> 3 <-> 6 <-> 13 <-> 4 <-> 7 <-> 8
```

[17]:
```
#Problem 12: Rearrange a linked list such that all even positioned nodes are placed at the end
#Input: 1 -> 2 -> 3 -> 4 -> 5
#Output: 1 -> 3 -> 5 -> 2 -> 4
class ListNode:
    def __init__(self, value=0, next=None):
```

```
        self.value = value
        self.next = next

def rearrangeEvenOdd(head):
    if not head or not head.next:
        return head

    odd_head = odd_tail = head
    even_head = even_tail = head.next


    current = even_tail.next
    is_odd = True

    while current:
        if is_odd:
            odd_tail.next = current
            odd_tail = odd_tail.next
        else:
            even_tail.next = current
            even_tail = even_tail.next

        is_odd = not is_odd
        current = current.next


    odd_tail.next = even_head
    even_tail.next = None

    return odd_head


def create_linked_list(values):
    head = ListNode(values[0])
    current = head
    for value in values[1:]:
        current.next = ListNode(value)
        current = current.next
    return head


def print_linked_list(node):
    while node:
        print(node.value, end=" -> " if node.next else "")
        node = node.next
    print()
```

```
head = create_linked_list([1, 2, 3, 4, 5])
print("Original list:")
print_linked_list(head)

rearranged_head = rearrangeEvenOdd(head)
print("Rearranged list:")
print_linked_list(rearranged_head)  # Should print: 1 -> 3 -> 5 -> 2 -> 4
```

```
Original list:
1 -> 2 -> 3 -> 4 -> 5
Rearranged list:
1 -> 3 -> 5 -> 2 -> 4
```

[19]:
```
# Problem 13: Given a non-negative number represented as a linked list, add one␣
 ↪to it.
# Input: 1 -> 2 -> 3 (represents the number 123)
# Output: 1 -> 2 -> 4 (represents the number 124)
class ListNode:
    def __init__(self, value=0, next=None):
        self.value = value
        self.next = next

def reverse_linked_list(head):
    prev = None
    current = head
    while current:
        next_node = current.next
        current.next = prev
        prev = current
        current = next_node
    return prev

def add_one(head):

    head = reverse_linked_list(head)


    current = head
    carry = 1
    while current and carry:
        current.value += carry
        if current.value == 10:
            current.value = 0
            carry = 1
        else:
            carry = 0
```

```
        current = current.next


    if carry:
        new_node = ListNode(1)
        current = head
        while current.next:
            current = current.next
        current.next = new_node


    head = reverse_linked_list(head)
    return head


def create_linked_list(values):
    head = ListNode(values[0])
    current = head
    for value in values[1:]:
        current.next = ListNode(value)
        current = current.next
    return head


def print_linked_list(node):
    while node:
        print(node.value, end=" -> " if node.next else "")
        node = node.next
    print()


head = create_linked_list([1, 2, 3])
print("Original linked list:")
print_linked_list(head)

result_head = add_one(head)
print("Linked list after adding one:")
print_linked_list(result_head)
```

```
Original linked list:
1 -> 2 -> 3
Linked list after adding one:
1 -> 2 -> 4
```

[21]:
```
# Problem 14: Given a sorted array and a target value, return the index if the␣
 ↪target is found. If not, return the index where it would be inserted.
#Input: nums = [1, 3, 5, 6], target = 5
```

```
#Output:2
def search_insert(nums, target):
    left, right = 0, len(nums) - 1

    while left <= right:
        mid = left + (right - left) // 2

        if nums[mid] == target:
            return mid
        elif nums[mid] < target:
            left = mid + 1
        else:
            right = mid - 1
        return left




nums = [1, 3, 5, 6]
target = 5
index = search_insert(nums, target)
print(f"The index is: {index}")  # Should print: 2
```

```
The index is: 2
```

[23]:
```
# Problem 15: Find the minimum element in a rotated sorted array.
#Input: [4, 5, 6, 7, 0, 1, 2]
#Output: 0
def find_min(nums):
    left, right = 0, len(nums) - 1

    while left < right:
        mid = left + (right - left) // 2

        if nums[mid] > nums[right]:

            left = mid + 1
        else:

            right = mid


    return nums[left]


nums = [4, 5, 6, 7, 0, 1, 2]
min_element = find_min(nums)
```

```
print(f"The minimum element is: {min_element}")  # Should print: 0
```

```
The minimum element is: 0
```

[1]:
```
# Problem 16: Search for a target value in a rotated sorted array.
#Input: nums = [4, 5, 6, 7, 0, 1, 2], target = 0
#Output: 4

def search(nums, target):
    left, right = 0, len(nums) - 1

    while left <= right:
        mid = (left + right) // 2

        if nums[mid] == target:
            return mid

        if nums[left] <= nums[mid]:
            if nums[left] <= target < nums[mid]:
                right = mid - 1
            else:
                left = mid + 1
        else:
            if nums[mid] < target <= nums[right]:
                left = mid + 1
            else:
                right = mid - 1

    return -1


nums = [4, 5, 6, 7, 0, 1, 2]
target = 0
print(search(nums, target))
```

```
4
```

[3]:
```
# Problem 17: Find the peak element in an array. A peak element is greater than␣
 ↪its neighbors.
#Input: nums = [1, 2, 3, 1]
#Output: 2 (index of peak element)

def findPeakElement(nums):
    left, right = 0, len(nums) - 1

    while left < right:
        mid = (left + right) // 2
```

```
        if nums[mid] < nums[mid + 1]:

            left = mid + 1
        else:

            right = mid


    return left


nums = [1, 2, 3, 1]
print(findPeakElement(nums))
```

2

[5]:
```
# Problem 18: Given a m x n matrix where each row and column is sorted in␣
 ↪ascending order, count the number of negative numbers.
#Input: grid = [[4, 3, 2, -1], [3, 2, 1, -1], [1, 1, -1, -2], [-1, -1, -2, -3]]
#Output: 8
def countNegatives(grid):
    m, n = len(grid), len(grid[0])
    count = 0
    row, col = 0, n - 1
    while row < m and col >= 0:
        if grid[row][col] < 0:
            count += m - row
            col -= 1
        else:
            row += 1

    return count


grid = [[4, 3, 2, -1],
        [3, 2, 1, -1],
        [1, 1, -1, -2],
        [-1, -1, -2, -3]]

print(countNegatives(grid))
```

8

[7]:
```
# Problem 19: Given a 2D matrix sorted in ascending order in each row, and the␣
 ↪first integer of each row is greater than the last integer of the previous␣
 ↪row, determine if a target value is present in the matrix.
#Input: matrix = [[1, 3, 5, 7], [10, 11, 16, 20], [23, 30, 34, 60]], target = 3
```

```python
#Output: True
def searchMatrix(matrix, target):
    if not matrix or not matrix[0]:
        return False

    number_of_rows = len(matrix)
    number_of_columns = len(matrix[0])

    left, right = 0, number_of_rows * number_of_columns - 1

    while left <= right:
        mid = (left + right) // 2
        mid_value = matrix[mid // number_of_columns][mid % number_of_columns]

        if mid_value == target:
            return True
        elif mid_value < target:
            left = mid + 1
        else:
            right = mid - 1

    return False

# Example usage:
matrix = [[1, 3, 5, 7],
          [10, 11, 16, 20],
          [23, 30, 34, 60]]
target = 3

print(searchMatrix(matrix, target))  # Output: True
```

```
True
```

[9]:
```python
# Problem 20: Find Median in Two Sorted Arrays
# Problem: Given two sorted arrays, find the median of the combined sorted␣
 ↪array.
# Input: nums1 = [1, 3], nums2 = [2]
#Output: 2.0
def findMedianSortedArrays(nums1, nums2):

    if len(nums1) > len(nums2):
        nums1, nums2 = nums2, nums1

    x, y = len(nums1), len(nums2)
    low, high = 0, x

    while low <= high:
```

```
        partitionX = (low + high) // 2
        partitionY = (x + y + 1) // 2 - partitionX


        maxLeftX = float('-inf') if partitionX == 0 else nums1[partitionX - 1]
        minRightX = float('inf') if partitionX == x else nums1[partitionX]

        maxLeftY = float('-inf') if partitionY == 0 else nums2[partitionY - 1]
        minRightY = float('inf') if partitionY == y else nums2[partitionY]


        if maxLeftX <= minRightY and maxLeftY <= minRightX:

            if (x + y) % 2 == 0:
                return (max(maxLeftX, maxLeftY) + min(minRightX, minRightY)) / 2
            else:
                return max(maxLeftX, maxLeftY)
        elif maxLeftX > minRightY:

            high = partitionX - 1
        else:

            low = partitionX + 1


nums1 = [1, 3]
nums2 = [2]
print(findMedianSortedArrays(nums1, nums2))
```

2

[11]:
```
# Problem 21: Given a sorted character array and a target letter, find the␣
 ↪smallest letter in the array that is greater than the target.
#Input: letters = ['c', 'f', 'j'], target = 'a'
#Output: 'c'
def nextGreatestLetter(letters, target):
    left, right = 0, len(letters) - 1

    while left <= right:
        mid = (left + right) // 2

        if letters[mid] <= target:
            left = mid + 1
        else:
            right = mid - 1
```

```
    return letters[left % len(letters)]


letters = ['c', 'f', 'j']
target = 'a'
print(nextGreatestLetter(letters, target))  # Output: 'c'
```

c

[13]:
```
# Problem 22: Given an array with n objects colored red, white, or blue, sort␣
 ↪them in-place so that objects of the same color are adjacent, with the␣
 ↪colors in the order red, white, and blue.
#Input: nums = [2, 0, 2, 1, 1, 0]
#Output: [0, 0, 1, 1, 2, 2]
def sortColors(nums):
    low, mid, high = 0, 0, len(nums) - 1

    while mid <= high:
        if nums[mid] == 0:
            nums[low], nums[mid] = nums[mid], nums[low]
            low += 1
            mid += 1
        elif nums[mid] == 1:
            mid += 1
        else:
            nums[mid], nums[high] = nums[high], nums[mid]
            high -= 1


nums = [2, 0, 2, 1, 1, 0]
sortColors(nums)
print(nums)
```

[0, 0, 1, 1, 2, 2]

[15]:
```
# Problem 23: Find the kth largest element in an unsorted array.
#Input: nums = [3, 2, 1, 5, 6, 4], k = 2
#Output: 5
import random

def quickselect(nums, left, right, k):
    if left == right:
        return nums[left]


    pivot_index = random.randint(left, right)
    pivot_index = partition(nums, left, right, pivot_index)
```

```
    if k == pivot_index:
        return nums[k]
    elif k < pivot_index:
        return quickselect(nums, left, pivot_index - 1, k)
    else:
        return quickselect(nums, pivot_index + 1, right, k)

def partition(nums, left, right, pivot_index):
    pivot_value = nums[pivot_index]

    nums[pivot_index], nums[right] = nums[right], nums[pivot_index]
    store_index = left

    for i in range(left, right):
        if nums[i] > pivot_value:
            nums[store_index], nums[i] = nums[i], nums[store_index]
            store_index += 1


    nums[right], nums[store_index] = nums[store_index], nums[right]
    return store_index

def findKthLargest(nums, k):
    n = len(nums)

    return quickselect(nums, 0, n - 1, n - k)


nums = [3, 2, 1, 5, 6, 4]
k = 2
print(findKthLargest(nums, k))  # Output: 5
```

2

[17]:
```
# Problem 24: Given an unsorted array, reorder it in-place such that nums[0] <=
 ↪nums[1] >= nums[2] <= nums[3]...
#Input: nums = [3, 5, 2, 1, 6, 4]
#Output: [3, 5, 1, 6, 2, 4]
def wiggleSort(nums):
    n = len(nums)

    for i in range(n - 1):
        if (i % 2 == 0 and nums[i] > nums[i + 1]) or (i % 2 == 1 and nums[i] <
 ↪nums[i + 1]):
            nums[i], nums[i + 1] = nums[i + 1], nums[i]
```

```
nums = [3, 5, 2, 1, 6, 4]
wiggleSort(nums)
print(nums)
```

```
[3, 5, 1, 6, 2, 4]
```

[21]:
```
# Problem 25: Given an array of integers, calculate the sum of all its elements
#Input: [1, 2, 3, 4, 5]
#Output: 15
def sum_of_elements(arr):
    total = 0
    for num in arr:
        total += num
    return total


nums = [1, 2, 3, 4, 5]
print(sum_of_elements(nums))
def sum_of_elements(arr):
    return sum(arr)
```

```
15
```

[23]:
```
# Problem 26: Find the maximum element in an array of integers.
#Input: [3, 7, 2, 9, 4, 1]
#Output: 9
def find_maximum(arr):
    if not arr:
        return None

    max_value = arr[0]
    for num in arr:
        if num > max_value:
            max_value = num
    return max_value


nums = [3, 7, 2, 9, 4, 1]
print(find_maximum(nums))
```

```
9
```

[25]:
```
# Problem 27: Implement linear search to find the index of a target element in␣
 ↪an array.
#Input: [5, 3, 8, 2, 7, 4], target = 8
#Output: 2
def linear_search(arr, target):
```

```
    for index in range(len(arr)):
        if arr[index] == target:
            return index
    return -1


nums = [5, 3, 8, 2, 7, 4]
target = 8
print(linear_search(nums, target))
```

2

[27]:
```
# Problem 28 Calculate the factorial of a given number.
#Input: 5
#Output: 120 (as 5! = 5 * 4 * 3 * 2 * 1 = 120)
def factorial(n):
    if n < 0:
        return None
    result = 1
    for i in range(1, n + 1):
        result *= i
    return result


num = 5
print(factorial(num))
```

120

[29]:
```
# Problem 29: Check if a given number is a prime number.
#Input: 7
#Output: True
import math

def is_prime(n):
    if n <= 1:
        return False
    for i in range(2, int(math.sqrt(n)) + 1):
        if n % i == 0:
            return False
    return True

num = 7
print(is_prime(num))
```

True

[31]:
```python
# Problem 30: Generate the Fibonacci series up to a given number n.
#Input: 8
#Output: [0, 1, 1, 2, 3, 5, 8, 13]
def fibonacci_series(n):
    series = []
    a, b = 0, 1
    while a <= n:
        series.append(a)
        a, b = b, a + b
    return series


n = 8
print(fibonacci_series(n))
```

```
[0, 1, 1, 2, 3, 5, 8]
```

[33]:
```python
# Problem 31: Calculate the power of a number using recursion.
#Input: base = 3, exponent = 4
#Output: 81 (as 3^4 = 3 * 3 * 3 * 3 = 81)
def power(base, exponent):

    if exponent == 0:
        return 1

    return base * power(base, exponent - 1)


base = 3
exponent = 4
print(power(base, exponent))
```

```
81
```

[35]:
```python
#Problem 32: Reverse a given string
#Input: "hello"
#Output: "olleh"
def reverse_string(s):
    return s[::-1]

input_str = "hello"
print(reverse_string(input_str))  # Output: "olleh"
```

```
olleh
```

[ ]:
```python
#
```