In [ ]:
```
3.1 1
```

In [4]:
```
def count_iterations(problemSize):
    iterations = 0
    while problemSize > 0:
        problemSize = problemSize // 2
        iterations += 1
    return iterations

# Test the program
problemSize = int(input("Enter the problem size: "))
iterations = count_iterations(problemSize)
print("Number of iterations:", iterations)
```

```
Enter the problem size: 8
Number of iterations: 4
```

In [ ]:
```
3.1 2
```

In [3]:
```
def count_iterations(problemSize):
    iterations = 0
    while problemSize > 0:
        problemSize = problemSize // 2
        iterations += 1
    return iterations

problem_sizes = [1000, 2000, 4000, 10000, 100000]

for problemSize in problem_sizes:
    iterations = count_iterations(problemSize)
    print("Problem Size:", problemSize)
    print("Number of Iterations:", iterations)
    print()
```

```
Problem Size: 1000
Number of Iterations: 10

Problem Size: 2000
Number of Iterations: 11

Problem Size: 4000
Number of Iterations: 12

Problem Size: 10000
Number of Iterations: 14

Problem Size: 100000
Number of Iterations: 17
```

In [ ]:
```
3.2 3
```

In [8]:
```
import time

# Start the timer
start_time = time.process_time()

# Code segment to measure the processing time
# ...

# Stop the timer
end_time = time.process_time()

# Calculate the elapsed processing time
elapsed_time = end_time - start_time

# Print the elapsed time
print("Elapsed processing time:", elapsed_time, "seconds")
```

```
Elapsed processing time: 0.0 seconds
```

In [ ]:

```
3.2 1
```

In [10]:

```
def analyze_algorithm(expression):
    # Split the expression into terms
    terms = expression.split()

    # Initialize variables
    dominant_term = ""
    degree = 0

    # Iterate over each term to find the dominant term
    for term in terms:
        if term[-1] == "n":
            # Check if the term has a power (degree)
            if "^" in term:
                base, power = term.split("^")
                power = int(power)
            else:
                base, power = term[:-1], 1
            if power > degree:
                degree = power
                dominant_term = term

    # Determine the big-O notation based on the dominant term
    if degree == 1:
        big_o = "O(n)"
    elif degree == 2:
        big_o = "O(n^2)"
    elif degree == 3:
        big_o = "O(n^3)"
    else:
        big_o = "O(n^" + str(degree) + ")"

    return dominant_term, big_o


# Test the expressions
expressions = ["2n - 4n^2 + 5n", "3n^2 + 6", "n^3 + n^2 - n"]

for expression in expressions:
    dominant_term, big_o = analyze_algorithm(expression)
    print("Expression:", expression)
    print("Dominant Term:", dominant_term)
    print("Big-O Notation:", big_o)
    print()
```

```
Expression: 2n - 4n^2 + 5n
Dominant Term: 2n
Big-O Notation: O(n)

Expression: 3n^2 + 6
Dominant Term:
Big-O Notation: O(n^0)

Expression: n^3 + n^2 - n
Dominant Term: n
Big-O Notation: O(n)
```

In [ ]:
```
2
```

In [11]:
```
def algorithm_A(n):
    return n**2

def algorithm_B(n):
    return 0.5 * n**2 + 0.5 * n

problem_sizes = [10, 100, 1000, 10000]

for size in problem_sizes:
    work_A = algorithm_A(size)
    work_B = algorithm_B(size)
    print("Problem Size:", size)
    print("Algorithm A work:", work_A)
    print("Algorithm B work:", work_B)
    print()
```

```
Problem Size: 10
Algorithm A work: 100
Algorithm B work: 55.0

Problem Size: 100
Algorithm A work: 10000
Algorithm B work: 5050.0

Problem Size: 1000
Algorithm A work: 1000000
Algorithm B work: 500500.0

Problem Size: 10000
Algorithm A work: 100000000
Algorithm B work: 50005000.0
```

In [ ]:
```
3
```

In [12]:
```
import time

def algorithm_n4(n):
    return n**4

def algorithm_2n(n):
    return 2**n

def compare_algorithms():
    n = 1
    while True:
        start_time = time.time()
        result_n4 = algorithm_n4(n)
        end_time_n4 = time.time()

        start_time_2n = time.time()
        result_2n = algorithm_2n(n)
        end_time_2n = time.time()

        runtime_n4 = end_time_n4 - start_time
        runtime_2n = end_time_2n - start_time_2n

        if runtime_n4 < runtime_2n:
            return n
        else:
            n += 1

threshold = compare_algorithms()
print("The n^4 algorithm begins to perform better than the 2^n algorithm a
```

```
The n^4 algorithm begins to perform better than the 2^n algorithm at n =
122
```

In [ ]:
```
3.3 1
```

In [15]:
```
def binary_search(arr, target):
    left, right = 0, len(arr) - 1

    while left <= right:
        midpoint = (left + right) // 2
        if arr[midpoint] == target:
            return midpoint, left, right, midpoint
        elif arr[midpoint] < target:
            left = midpoint + 1
        else:
            right = midpoint - 1

    return -1, left, right, midpoint

arr = [20, 44, 48, 55, 62, 66, 74, 88, 93, 99]

target = 90
result, left, right, midpoint = binary_search(arr, target)
print("Target:", target)
print("Result:", result)
print("Left:", left)
print("Right:", right)
print("Midpoint:", midpoint)
print()

target = 44
result, left, right, midpoint = binary_search(arr, target)
print("Target:", target)
print("Result:", result)
print("Left:", left)
print("Right:", right)
print("Midpoint:", midpoint)
```

```
Target: 90
Result: -1
Left: 8
Right: 7
Midpoint: 8

Target: 44
Result: 1
Left: 0
Right: 3
Midpoint: 1
```

In [ ]:
```
2
```

In [16]:
```
def phone_book_search(arr, target):
    left = 0
    right = len(arr) - 1

    while left <= right:
        midpoint = left + (ord(target[0]) - ord(arr[left][0])) * (right -

        if arr[midpoint] == target:
            return midpoint
        elif arr[midpoint][0] < target[0]:
            left = midpoint + 1
        else:
            right = midpoint - 1

    return -1

phone_book = ["Adams", "Brown", "Davis", "Smith", "Williams", "Zhang"]

target = "Smith"
result = phone_book_search(phone_book, target)
print("Target:", target)
print("Result:", result)

target = "Davis"
result = phone_book_search(phone_book, target)
print("Target:", target)
print("Result:", result)
```

```
Target: Smith
Result: 3
Target: Davis
Result: 2
```

In [ ]:
```
3.4 1
```

In [19]:
```
def selection_sort(arr):
    exchanges = 0

    for i in range(len(arr)):
        min_idx = i
        for j in range(i + 1, len(arr)):
            if arr[j] < arr[min_idx]:
                min_idx = j

        if min_idx != i:
            arr[i], arr[min_idx] = arr[min_idx], arr[i]
            exchanges += 1

    return exchanges

# Configuration 1: Smallest number of exchanges (sorted in ascending order
arr1 = [10, 20, 30, 40, 50]
exchanges1 = selection_sort(arr1)
print("Configuration 1:")
print("Input List:", arr1)
print("Number of Exchanges:", exchanges1)
print()

# Configuration 2: Largest number of exchanges (sorted in descending order
arr2 = [50, 40, 30, 20, 10]
exchanges2 = selection_sort(arr2)
print("Configuration 2:")
print("Input List:", arr2)
print("Number of Exchanges:", exchanges2)
```

```
Configuration 1:
Input List: [10, 20, 30, 40, 50]
Number of Exchanges: 0

Configuration 2:
Input List: [10, 20, 30, 40, 50]
Number of Exchanges: 2
```

In [ ]:
```
2
```

In [20]:

```
def selection_sort(arr):
    exchanges = 0

    for i in range(len(arr)):
        min_idx = i
        for j in range(i + 1, len(arr)):
            if arr[j] < arr[min_idx]:
                min_idx = j

        if min_idx != i:
            arr[i], arr[min_idx] = arr[min_idx], arr[i]
            exchanges += 1

    return exchanges

def bubble_sort(arr):
    exchanges = 0
    n = len(arr)

    for i in range(n):
        swapped = False
        for j in range(0, n - i - 1):
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
                exchanges += 1
                swapped = True

        if not swapped:
            break

    return exchanges

# Example data
data = [5, 3, 8, 2, 1, 7, 4, 6]

# Selection Sort
selection_exchanges = selection_sort(data.copy())
print("Selection Sort Exchanges:", selection_exchanges)

# Bubble Sort
bubble_exchanges = bubble_sort(data.copy())
print("Bubble Sort Exchanges:", bubble_exchanges)
```

```
Selection Sort Exchanges: 6
Bubble Sort Exchanges: 14
```

In [ ]:

```
3
```

In [23]:
```
def modified_bubble_sort(arr):
    n = len(arr)

    for i in range(n):
        swapped = False

        for j in range(0, n - i - 1):
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
                swapped = True

        if not swapped:
            break

# Example data
data = [5, 2, 8, 4, 1, 7, 3, 6]

# Modified Bubble Sort
modified_bubble_sort(data)
print("Sorted List:", data)
```

```
Sorted List: [1, 2, 3, 4, 5, 6, 7, 8]
```

In [ ]:
```
4
```

In [24]:
```
def insertion_sort(arr):
    for i in range(1, len(arr)):
        key = arr[i]
        j = i - 1

        while j >= 0 and arr[j] > key:
            arr[j + 1] = arr[j]
            j -= 1

        arr[j + 1] = key

# Example data
data = [5, 2, 8, 4, 1, 7, 3, 6]

# Insertion Sort
insertion_sort(data)
print("Sorted List:", data)
```

```
Sorted List: [1, 2, 3, 4, 5, 6, 7, 8]
```

In [ ]:
```

```