```python
class Employee:
    def __init__(self):
        self.item = []

    def push(self, empid, name, salary):
        self.item.append([empid, name, salary])

    def pop(self):
        return self.item.pop()

    def display(self):
        for i in self.item:
            print(i)


# Usage
print("Push")
e = Employee()
e.push(1, 'a', 1000)
e.push(3, 'c', 5000)
e.push(2, 'b', 2000)
e.display()

print("Pop")
print(e.pop())
print(e.pop())

print("Display")
e.display()
```

Algorithm:

1. **Start**
2. Define class **Employee** with:
    - **`__init__` method** → Initializes an empty list `item`.
    - **`push(empid, name, salary)`** → Adds an employee to `item`.
    - **`pop()`** → Removes and returns the last added employee.
    - **`display()`** → Prints all employees.
3. Create an object **e** of `Employee`.
4. **Push** three employees.
5. **Display** the employee list.
6. **Pop** two employees and print them.
7. **Display** remaining employees.
8. **End**

```python
import time

start = time.time()  # Start time

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

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

    def push(self, item):
        self.items.append(item)

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

    def peek(self):
        return self.items[len(self.items) - 1]

    def size(self):
        return len(self.items)
```

```python
# Usage
s = Stack()
print(s.isempty())

print("Push")
s.push(11)
s.push(12)
s.push(13)

print("Peek:", s.peek())

print("Pop")
print(s.pop())
print(s.pop())

print("Size:", s.size())

end = time.time()  # End time
print("Runtime of the program is", end - start)
```

Algorithm:

1. **Start**
2. Record the **start time**.
3. Define class **Stack**:
   - **`__init__`** → Initialize an empty list `items`.
   - **`isempty()`** → Check if stack is empty.
   - **`push(item)`** → Add an item to the stack.
   - **`pop()`** → Remove and return the last added item.
   - **`peek()`** → Return the top item without removing it.
   - **`size()`** → Return the number of items in the stack.
4. Create a **Stack object `s`**.
5. Check if stack is empty and print result.
6. **Push** three elements (11, 12, 13).
7. **Peek** the top element and print it.
8. **Pop** two elements and print them.
9. Print the **size** of the stack.
10. Record the **end time**.
11. Print the **runtime** of the program.
12. **End**.

```python
def partition(arr, low, high):
    i = low - 1
    pivot = arr[high]

    for j in range(low, high):
        if arr[j] <= pivot:
            i = i + 1
            arr[i], arr[j] = arr[j], arr[i]

    arr[i + 1], arr[high] = arr[high], arr[i + 1]
    return i + 1

def quicksort(arr, low, high):
    if low < high:
        pi = partition(arr, low, high)

        quicksort(arr, low, pi - 1)
        quicksort(arr, pi + 1, high)

arr = [10, 7, 8, 9, 1, 5]
n = len(arr)
quicksort(arr, 0, n - 1)

print("Sorted array is:")
for i in range(n):
    print(arr[i], end=" ")
```

Algorithm for QuickSort:

1. **Start**
2. Define the **partition** function:
    - Select the last element as the pivot.
    - Move elements smaller than the pivot to the left.
    - Swap pivot to its correct position and return its index.
3. Define the **quicksort** function:
    - If the low index is smaller than the high index:
        - Partition the array.
        - Recursively quicksort the left and right subarrays.
4. Initialize an array `[10, 7, 8, 9, 1, 5]`.
5. Call `quicksort()` on the full array.
6. Print the sorted array.
7. **End**.

```python
import matplotlib.pyplot as plt
import numpy as np

xpoint = np.array([1, 8])
ypoint = np.array([3, 10])

plt.plot(xpoint, ypoint)
plt.show()
```

Algorithm for the Code:

1. **Start**
2. Import `matplotlib.pyplot` and `numpy`.
3. Define `xpoint` as an array `[1, 8]`.
4. Define `ypoint` as an array `[3, 10]`.
5. Use `plt.plot(xpoint, ypoint)` to plot the points.
6. Display the plot using `plt.show()`.
7. **End**

```python
import time
import matplotlib.pyplot as plt
import numpy as np
# Start the timer
start = time.time()
# Function for linear search
def linear_search(first, n, key):
    for i in range(n):
        if first[i] == key:
            return 0
    return -1

n = int(input("Enter number of elements: "))
first = []  # Initialize the list

for i in range(n):
    first.append(int(input()))

print("List:", first)
```

```python
key = int(input("Enter key: "))
# Perform search
res = linear_search(first, n, key)

if res == 0:
    print("Element found")
else:
    print("Element not found")
# End the timer
end = time.time()
print("Running time of program is", end - start)
# Plot graph using Matplotlib
xpoints = np.array([1, 8])
ypoints = np.array([3, 10])

plt.plot(xpoints, ypoints)
plt.show()
```

Algorithm for the Given Code

1. **Start the timer** to measure execution time.
2. **Define the `linear_search` function**:
    - Iterate through the list.
    - If the key is found, return `0`.
    - Otherwise, return `-1`.
3. **Take user input** for the number of elements (`n`).
4. **Store `n` elements** in a list (`first`).
5. **Take user input** for the key to search.
6. **Call `linear_search` function** with the list and key.
7. **Check the result**:
    - If key is found, print `"Element found"`.
    - Otherwise, print `"Element not found"`.
8. **Stop the timer** and print execution time.
9. **Plot a simple graph** using Matplotlib.
10. **End the program**.

```python
class IterateCounter:
    def __iter__(self):  # Corrected
        self.a = 1
        return self

    def __next__(self):  # Corrected
        if self.a <= 10:
            x = self.a
            self.a += 1
            return x
        else:
            raise StopIteration
# Create an instance of the class
myclass = IterateCounter()
myiter = iter(myclass)  # Use iter()

# Iterate through the object
for x in myiter:
    print(x)
```

Algorithm for the Given Code

1. **Define a class** `IterateCounter`.
2. **Implement the `__iter__()` method**:
   - Initialize `self.a = 1`.
   - Return `self` as an iterator.
3. **Implement the `__next__()` method**:
   - If `self.a <= 10`, store `self.a` in `x`, increment `self.a`, and return `x`.
   - Else, raise `StopIteration` to end the iteration.
4. **Create an instance** of `IterateCounter`.
5. **Get an iterator** using `iter()`.
6. **Use a `for` loop** to iterate through the object and print values from 1 to 10.
7. **End the program** when iteration is complete.

```python
class Node:
    def __init__(self, data=None):  # Corrected constructor method
        self.data = data
        self.next = None

class SinglyLinkedList:
    def __init__(self):  # Corrected constructor method
        self.head = None

    def add_at_beginning(self, data):
        new_node = Node(data)
        new_node.next = self.head
        self.head = new_node

    def remove_node(self, remove_key):
        head_val = self.head
        # If the head node itself holds the key to be deleted
        if head_val is not None and head_val.data == remove_key:
            self.head = head_val.next
            head_val = None
            return

        prev = None
        while head_val is not None:
            if head_val.data == remove_key:
                break
            prev = head_val
            head_val = head_val.next
```

```python
        if head_val is None:  # If key was not present
            return

        prev.next = head_val.next
        head_val = None

    def print_list(self):
        print_val = self.head
        while print_val:
            print(print_val.data, end=" -> ")
            print_val = print_val.next
        print("None")  # Indicating end of the list

# Creating a linked list and performing operations
linked_list = SinglyLinkedList()
linked_list.add_at_beginning("Mon")
linked_list.add_at_beginning("Tue")
linked_list.add_at_beginning("Thu")
linked_list.print_list()  # Print the list

print("\nRemoving 'Tue' from the list:")
linked_list.remove_node("Tue")
linked_list.print_list()  # Print after deletion
```

Algorithm for the Given Code

1. **Define a `Node` class**:
   - Initialize `data` and `next` pointer.
2. **Define a `SinglyLinkedList` class**:
   - Initialize `head` as `None`.
3. **Add a node at the beginning**:
   - Create a new node.
   - Point `new_node.next` to the current `head`.
   - Update `head` to `new_node`.
4. **Remove a node by key**:
   - Check if the `head` contains the key. If yes, update `head` to `head.next`.
   - Traverse the list to find the key.
   - If found, update `prev.next` to `head_val.next` to remove the node.
5. **Print the list**:
   - Traverse and print each node's data until `None` is reached.
6. **Perform operations**:
   - Insert nodes `"Mon"`, `"Tue"`, and `"Thu"` at the beginning.
   - Print the list.
   - Remove `"Tue"` and print the updated list.

```
def fibo(n):
    if n <= 1:
        return n  # Base case: return n i
    else:
        return fibo(n - 1) + fibo(n - 2)

# Get user input
n = int(input("How many terms? >> "))

# Print Fibonacci sequence
for i in range(n):
    print(fibo(i))
```

Algorithm for Fibonacci Recursive Code

1. **Define the Fibonacci function `fibo(n)`**:
   - If `n ≤ 1`, return `n` (base case).
   - Otherwise, return `fibo(n-1) + fibo(n-2)` (recursive case).
2. **Take user input (`n`)** for the number of terms.
3. **Use a loop** from `0` to `n-1`:
   - Call `fibo(i)` for each term.
   - Print the result.
4. **End the program** when all terms are printed.

```python
import time
start = time.perf_counter()
def binary_search(array, x, low, high):
    if low <= high:
        mid = (low + high) // 2
        if array[mid] == x:
            return mid
        elif array[mid] > x:
            return binary_search(array, x, low, mid - 1)
        else:
            return binary_search(array, x, mid + 1, high)
    return -1
array = [3, 4, 5, 8, 9, 16]
x = 4
result = binary_search(array, x, 0, len(array) - 1)
if result != -1:
    print("Element is present at index", result)
else:
    print("Element not found")
end = time.perf_counter()
print("Running time of program is", (end - start))
```

Algorithm for Binary Search (Recursive)

1. **Start the timer** to measure execution time.
2. **Define `binary_search(array, x, low, high)`:**
   - Find the middle index `mid = (low + high) // 2`.
   - If `array[mid] == x`, return `mid` (element found).
   - If `array[mid] > x`, search the **left subarray** (`low` to `mid - 1`).
   - Else, search the **right subarray** (`mid + 1` to `high`).
   - If `low > high`, return `-1` (element not found).
3. **Initialize a sorted array** and the element `x` to search.
4. **Call `binary_search()`** with `low = 0` and `high = len(array) - 1`.
5. **Print the result**:
   - If `result != -1`, print the found index.
   - Else, print "Element not found".
6. **Stop the timer** and print execution time.

```python
import time
# Start the timer
start = time.perf_counter()
def selection_sort(array):
    n = len(array)
    for i in range(n):
        minimum = i
        for j in range(i + 1, n):
            if array[j] < array[minimum]:
                minimum = j
        # Swap the found minimum element with the first eleme
        array[i], array[minimum] = array[minimum], array[i]
    return array
# Define the array
array = [13, 11, 9, 45, 12]
# Print sorted list
print("Sorted list:", selection_sort(array))
end = time.perf_counter()
print("The time is:", end - start)
```

Short Algorithm for Selection Sort

1. **Start Timer:** Measure execution time.
2. **Iterate through the list:** Start from index `0` to `n-1`.
3. **Find Minimum:** For each position, find the smallest element in the remaining list.
4. **Swap Elements:** Swap the smallest element with the current position.
5. **Repeat Until Sorted:** Continue until all elements are sorted.
6. **Print the sorted list.**
7. **Stop Timer:** Display execution time.

```python
def mergeSort(myList):
    if len(myList) > 1:
        mid = len(myList) // 2  # Find the middle c
        left = myList[:mid]  # Divide into left hal
        right = myList[mid:]  # Divide into right h
        # Recursively sort both halves
        mergeSort(left)
        mergeSort(right)
        # Merge sorted halves
        i = j = k = 0
        # Compare elements and merge
        while i < len(left) and j < len(right):
            if left[i] <= right[j]:
                myList[k] = left[i]
                i += 1
            else:
                myList[k] = right[j]
                j += 1
            k += 1
```

```python
        # Copy any remaining elements from left
        while i < len(left):
            myList[k] = left[i]
            i += 1
            k += 1
        # Copy any remaining elements from righ
        while j < len(right):
            myList[k] = right[j]
            j += 1
            k += 1
# Define the list
myList = [54, 26, 93, 17, 77, 31, 46, 35]
# Print the original list
print("Original List:", myList)
# Call Merge Sort
mergeSort(myList)
# Print the sorted list
print("Sorted List:", myList)
```

Algorithm for Merge Sort

1. **Base Case:** If the list has one or zero elements, return (already sorted).
2. **Divide:** Split the list into two halves (`left` and `right`).
3. **Recursively Sort:** Apply `mergeSort(left)` and `mergeSort(right)`.
4. **Merge:**
   - Compare elements from `left` and `right`, placing the smaller one into `myList`.
   - Copy any remaining elements from `left` and `right` into `myList`.
5. **Repeat until fully sorted.**

```python
import time
start = time.perf_counter()
# Bubble Sort function
def bubble_sort(arr):
    n = len(arr)
    for i in range(n - 1):
        for j in range(0, n - i - 1):
            if arr[j] > arr[j + 1]:
                # Swap elements if they are in the wrong
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
# Define the array
arr = [61, 34, 25, 12]
bubble_sort(arr)
print("Sorted array is:")
for i in arr:
    print(i)
# End the timer
end = time.perf_counter()
print("Time =", end - start)
```

Algorithm for Bubble Sort

1. **Start Timer:** Measure execution time.
2. **Iterate through the list:** Use two nested loops to compare adjacent elements.
3. **Swap elements if necessary:** If the left element is greater than the right, swap them.
4. **Repeat until sorted:** Largest elements "bubble up" to their correct positions.
5. **Print the sorted list.**
6. **Stop Timer:** Display execution time.

```python
import time

# Start the timer
start = time.perf_counter()

# Insertion Sort function
def insertion_sort(array):
    for i in range(1, len(array)):
        key = array[i]
        j = i - 1
        while j >= 0 and key < array[j]:
            should be >= 0)
            array[j + 1] = array[j]
            j -= 1  # Fixed decrement oper
        array[j + 1] = key

# Define the array
arr = [9, 5, 1, 4, 3]  # Fixed missing
```

Algorithm for Insertion Sort

1. **Start Timer:** Measure execution time.
2. **Iterate through the list:** Start from the second element (index `1`).
3. **Compare and shift:** Move elements that are greater than the key one position ahead.
4. **Insert the key:** Place the key in its correct position.
5. **Repeat for all elements:** Continue until the entire list is sorted.
6. **Print the sorted list.**
7. **Stop Timer:** Display execution time.