Linear Search

```
customer_list = [10, 20, 30, 40, 50, 60, 70, 80, 90]
def linear_search(customer_list, target_id):
  for i in range(len(customer_list)):
    if customer_list[i] == target_id:
      return 1
  return 0
search_id = int(input("Enter ID to be searched: "))
found = linear_search(customer_list, search_id)
if found:
  print("Customer ID is found in list")
else:
  print("Customer ID is not found in list")
```

Undo Redo Operations

```
undo_stack = []
redo_stack = []
document = ""
def make_a_change(new_text):
  global document
  undo_stack.append(document)
  document = new_text
  redo_stack.clear()
def undo():
  global document
  if undo_stack:
    redo_stack.append(document)
    document = undo_stack.pop()
  else:
    print("Nothing to undo")
def redo():
  global document
  if redo_stack:
    undo_stack.append(document)  # Save current state for undo
    document = redo_stack.pop()  # Reapply last undone change
  else:
    print("Nothing to redo")
def display():
  print("Current document:", document)
def main():
  while True:
    print("\nChoose an operation")
```

Binary Search

```
customer_id = [222, 111, 333, 555, 800, 400, 862, 504, 320, 777]
customer_id.sort()
def binary_search(customer_list, target_id):
  low = 0
  high = len(customer_list) - 1
  while low <= high:
    mid = (low + high) // 2
    guess = customer_list[mid]
    if guess == target_id:
      return True
    elif guess < target_id:
      low = mid + 1
    else:
      high = mid - 1
  return False
search_id = int(input("Enter the ID to be searched: "))
found = binary_search(customer_id, search_id)
if found:
  print("ID exists in the customer list")
else:
  print("ID does not exist in the customer list")
```

Bubble sorted list

```
salaries = [20000, 45000, 30000, 15000, 50000,55000,32000,89000,76000,120000]
def bubble_sort(arr):
  for i in range(len(arr)):
    for j in range(len(arr) - i - 1):
      if arr[j] > arr[j + 1]:
        arr[j], arr[j + 1] = arr[j + 1], arr[j]
  return arr
bubble_sorted=bubble_sort(salaries.copy())
print(“Top 5 salaries(Bubble Sort):”,bubble_sorted[-5:][::-1])
all_salary_sorted_list=bubble_sort(salaries)
print(“all salaries list by using bubble sort”,all_salary_sorted_list)
```

Selection Sort

```
salaries = [320000.0, 470000.0, 620000.0, 780000.0, 170000.0, 870000.0, 760000.0]
def selection_sort(arr):
  for i in range(len(arr)):
```

```
    print("1. Make a change")
    print("2. Undo Action")
    print("3. Redo Action")
    print("4. Display document content")
    print("5. Exit")
    choice = input("Enter choice (1-5): ")
    if choice == "1":
      new_text = input("Enter new document
content: ")
      make_a_change(new_text)
    elif choice == "2":
      undo()
    elif choice == "3":
      redo()
    elif choice == "4":
      display()
    elif choice == "5":
      break
    else:
      print("Invalid choice, please choose
again")
```

Student management system using linked list

```
class StudentNode:
  def __init__(self, roll_no, name, marks):
    self.roll_no = roll_no
    self.name = name
    self.marks = marks
    self.next = None
class StudentReport:
  def __init__(self):
    self.head = None
  def add_student(self, roll_no, name,
marks):
    new_node = StudentNode(roll_no, name,
marks)
    if self.head is None:
      self.head = new_node
    else:
      current = self.head
      while current.next:
        current = current.next
      current.next = new_node
    print(f"Student {name} added
successfully."
  def display_students(self):
    if self.head is None:
      print("No student records found.")
    min_idx = i
    for j in range(i + 1, len(arr)):
      if arr[min_idx] > arr[j]:
        min_idx = j
    arr[i], arr[min_idx] = arr[min_idx], arr[i]
  return arr
selection_sorted_list =
selection_sort(salaries.copy())
print("Top 5 salaries (selection sort):",
selection_sorted_list[-5:])
all_salary_sorted_list =
selection_sort(salaries.copy())
print("All salaries list by using selection
sort:", all_salary_sorted_list)
```

HashTable class using chaining with lists

```
class HashTable:
  def __init__(self, size):
    self.size = size
    self.table = [[] for _ in range(size)]
  def hash_function(self, key):
    return key % self.size
  def insert(self, key):
    index = self.hash_function(key)
    if key in self.table[index]:
      print(f"{key} already exists at index
{index}")
    else:
      self.table[index].append(key)
      print(f"{key} inserted at index {index}")
  def search(self, key):
    index = self.hash_function(key)
    if key in self.table[index]:
      print(f"{key} found at index {index}")
      return True
    else:
      print(f"{key} not found")
      return False
  def display(self):
    print("\nHash Table:")
    for i, chain in enumerate(self.table):
      print(f"{i}: {chain}")
ht = HashTable(7)
keys = [10, 20, 15, 7, 5, 32]
for key in keys:
  ht.insert(key)
ht.display()
ht.search(43)
ht.search(99)
```

```
        return
    print("\nStudent Report:")
    current = self.head
    while current:
      print(f"Roll No: {current.roll_no},
Name: {current.name}, Marks:
{current.marks}")
      current = current.next
  def search_student(self, roll_no):
    current = self.head
    while current:
      if current.roll_no == roll_no:
        print(f"\nStudent Found:\nRoll No:
{current.roll_no}, Name: {current.name},
Marks: {current.marks}")
        return
      current = current.next
    print(f"\nStudent with Roll No {roll_no}
not found.")
if __name__ == "__main__":
  report = StudentReport()
  report.add_student(101, "Alice", 85)
  report.add_student(102, "Bob", 78)
  report.add_student(103, "Charlie", 92)
  report.display_students()
  report.search_student(102)
  report.search_student(999)
```