```
#University admission process using various data structures.
#1. Student Class(List)
class Student:
    def __init__(self, name, score, department):
    self.name = name
    self.score = score
    self.department = department
    self.next = None  # Pointer for the linked list
    self.left = None  # Pointers for the Binary Search Tree
    self.right = None

#A list(array) to store student details
students = [
    Student("Anu", 98, "Computer Science"),
    Student("Ratna", 85, "BAMMC"),
    Student("Solomon", 95, "Information Technology"),
    Student("Celstine", 78, "Biotechnology"),
    Student("Priya", 88, "Computer Science"),
    Student("Mohan", 90, "Information Technology")]

#2.Admission Process
class AdmittedNode:
    def __init__(self, student):
    self.student = student
    self.next = None
class AdmittedList:
    def __init__(self):
    self.head = None
    def add_student(self, student):
    new_node = AdmittedNode(student)
    if not self.head:
    self.head = new_node
    else:
    current = self.head
    while current.next:
        current = current.next
    current.next = new_node
```

```
#Binary Search Tree (BST)
class BST_Node:
    def __init__(self, key, student):
    self.key = key
    self.student = student
    self.left = None
    self.right = None

#3.Functions for Operations
def display_menu():
    print("\n--- University Admission System ---")
    print("1. View all students")
    print("2. Rank and Admit students (Sorting & Linked List)")
    print("3. Process Interviews and Reviews (Queue & Stack)")
    print("4. Search students (Hash Table, BST & Traversal)")
    print("5. Exit")

def view_students():
    if not students:
    print("No students in the list.")
    else:
    print("\n--- All Students ---")
    for student in students:
    print(f"Name: {student.name}, Score: {student.score}, Department:
{student.department}")

def rank_and_admit():
    # Sorting:descending order
    sorted_students = sorted(students, key=lambda s: s.score, reverse=True)

    # Linked List: Top 3 students
    admitted_list = AdmittedList()
    admitted_count = 0
    print("\n--- Admitting Top Students ---")
    for student in sorted_students:
    if admitted_count < 3:
    admitted_list.add_student(student)
    print(f"Admitted: {student.name} with score {student.score}")
    admitted_count += 1
    print("\n--- Admitted Students List ---")
    current = admitted_list.head
    while current:
    print(f"- {current.student.name}")
    current = current.next

def process_interviews_and_reviews():
    #Queue and Stack
    from collections import deque
    sorted_students = sorted(students, key=lambda s: s.score, reverse=True)
    #Queue:Students for interview (First-In, First-Out).
    interview_queue = deque()
    for student in sorted_students[3:5]:
    interview_queue.append(student)
    print("\n--- Processing Interview Queue ---")
    while interview_queue:
    student = interview_queue.popleft()
    print(f"Interviewing {student.name}...")
    #Stack: Applications for review (Last-In, First-Out).
    review_stack = []
    review_stack.append(sorted_students[5])
    review_stack.append(sorted_students[0])
    print("\n--- Processing Review Stack ---")
    while review_stack:
    student = review_stack.pop()
    print(f"Reviewing {student.name}'s application...")

def search_and_organize():
    # Uses Hash Table, BST, and Tree Traversal
    # Hash Table
    student_database = {student.name: student for student in students}
    search_name = input("Enter the student's name to search for: ")
    print(f"\n--- Searching for {search_name} using Hash Table ---")
    found_student = student_database.get(search_name)
    if found_student:
    print(f"Found: {found_student.name}, Score: {found_student.score},
Department: {found_student.department}")
    else:
    print(f"Student '{search_name}' not found.")
    # BST (Binary Search Tree)
```

```
        def insert_bst(root, key, student):
        if root is None:
        return BST_Node(key, student)
        if key < root.key:
        root.left = insert_bst(root.left, key, student)
        else:
        root.right = insert_bst(root.right, key, student)
        return root
  # Tree Traversal:in-order traversal
        def inorder_traversal(root):
        if root:
        inorder_traversal(root.left)
        print(f"- {root.student.name} ({root.student.department})")
        inorder_traversal(root.right)
        department_tree = None
        for student in students:
        department_tree = insert_bst(department_tree, student.department, student)
        print("\n--- Students Organized by Department (BST In-order Traversal) ---")
        inorder_traversal(department_tree)

def main():
        #menu driven
        while True:
        display_menu()
        choice = input("Enter your choice: ")
        if choice == '1':
        view_students()
        elif choice == '2':
        rank_and_admit()
        elif choice == '3':
        process_interviews_and_reviews()
        elif choice == '4':
        search_and_organize()
        elif choice == '5':
          print("Exiting...")
        break
        else:
        print("Please try again.")
if __name__ == "__main__":
        main()
```