1 Written Questions

Question 1: Priority Queues (15 points)

Simulating Priority Queue Using a Stack

Write a function `push_with_priority(stack, priority, element)` that pushes elements onto a stack based on a given priority. Lower priority elements should always appear closer to the top.
Your Answer:

```
def push_with_priority(stack, priority, element):
    # Temporary stack to hold elements until the right position is found
    temp_stack = []

    # 1) Move elements to temp_stack while maintaining priority order

    # 2) Insert the new element with its priority

    # 3) Move the elements back from temp_stack to stack to maintain
        their order

# Example usage
stack = []
push_with_priority(stack, 3, "A")
push_with_priority(stack, 1, "B")
push_with_priority(stack, 2, "C")
print(stack)  # Output: [(3, 'A'), (2, 'C'), (1, 'B')]
```

```python
def push_with_priority(stack, priority, element):
    temp_stack = []

    # Create the item with its priority
    item = (priority, element)

    if len(stack) == 0:
        stack.append(item)
    else:
        # Move elements to the temporary stack until the right position is found
        while item[0] > stack[-1][0]:
            temp_stack.append(stack.pop())

        # Insert the new element in the correct position
        stack.append(item)

        # Move the elements back from the temporary stack to the main stack
        while len(temp_stack) != 0:
            stack.append(temp_stack.pop())
```

Question 2: Binary Search Tree Operations (20 points)

In this question, you will perform several operations on a Binary Search Tree (BST).

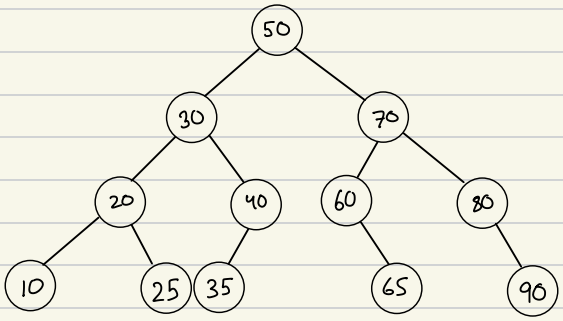
a) (10 points) Draw the binary search tree after inserting the following 12 elements into an empty BST in the given order

$$[50, 30, 70, 20, 40, 60, 80, 10, 25, 35, 65, 90]$$

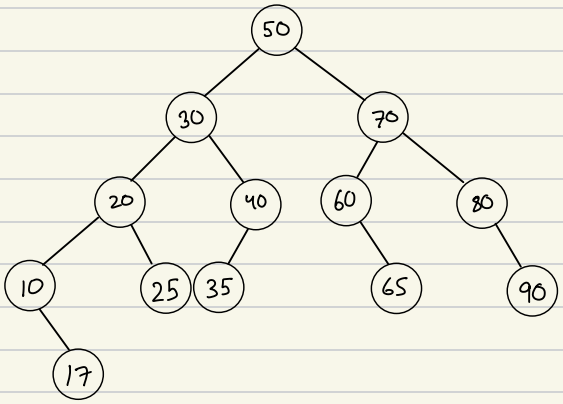
b) (5 points) Insert Element: After constructing the BST with the above elements, draw the updated tree after inserting the element 17 into the tree.

c) (5 points) Delete Element: Draw the binary search tree after deleting the element 70 from the tree and describe how the tree restructures itself.

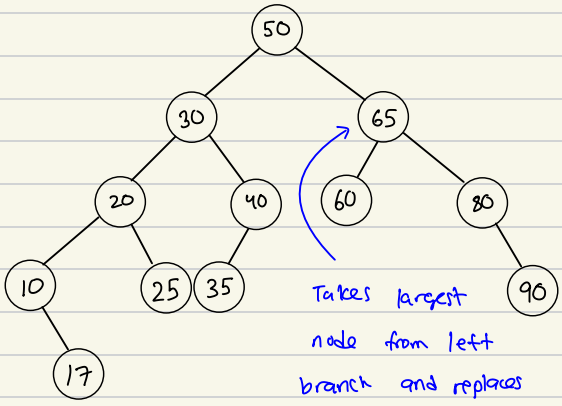
(a)



(b)



(c)



Question 3: Sorting (20 points)

Task 1: [10 points]
Demonstrate the quick-sort algorithm step by step, using **the last element of the current subarray as the pivot**. Given the array:

$$[8, -3, 7, -1, 10, 2, -5]$$

Complete the following table by showing the left partition (L), right partition (G), and the pivot at each step. Perform quicksort recursively on the subarrays and show each intermediate result.

Solution Template:

Step	Pivot	Left Partition	Right Partition
Initial Array	–	–	–
1	-5	[ ]	[8, -3, 7, -1, 10, 2]
2	-2	[-3, -1]	[8, 7, 10]
3	-1	[-3]	[ ]
4	10	[8, 7]	[ ]
5	7	[ ]	[8]

final Array : [-5, -3, -1, 2, 7, 8, 10]

Table 1: Solution Template

Notes for Students:

- Use the solution template above to format your answers.
- Use the last element of the current subarray as the pivot for each step.

Task 2: [2 points]
Complete the missing line in the following code fragment to correctly implement merge-sort's merge function.

```
def merge(S1, S2, S):
    i = j = 0
    while i + j < len(S):
        if j == len(S2) or (i < len(S1) and S1[i] < S2[j]):
            S[i+j] = S1[i]  # copy ith element of S1 as next item of
                S
            i += 1
        else:
            # Missing line
            j += 1
```

S[i+j] = S2[j]

S = [0 : len(S)] = left_partition + [pivot] + right-partition