Question 1

```
#Function accepts Array and k value as input
def problem1(array, k):
    dictionary = {}
    result = 0
    # Initialize a dictionary where key represents the number and count
represents the number of elements
    # which are received entered
    for each in range(0, len(array)):
        value = array[each]
        if value in dictionary:
            count = dictionary.get(value)
            count = count + 1
            dictionary[value] = count
        else:
            dictionary[value] = 1

    # We know if
    # a - b = k, then a = k + b
    # The same idea we can use where we for every 'a' we try to find 'k -
a' which is actually 'b'.
    # There can be multiple numbers present in each key. To compute for the
total number of possible counts
    # we will multiply the count with count - 1
    for key, count in dictionary.items():
        element = key * 2 - k
        if element == key:
            if count > 1:
                result = result + (count * (count - 1))
        else:
            if element in dictionary:
                result = result + count

    return result



array = [2, 6, 4]
k = 2
print(problem1(array, k))
```

Output

```
1
```

Question 2:

```
# Insert the element to the matrix in format [row, column, element]
def insert_element(matrix, row, column):
    element = int(input("Enter element"))
    matrix.append([row, column, element])

# Delete the element for the 'row' and 'column' provided by the user
def delete_element(matrix, row, column):
    pos = 0
    for each in matrix:
        if row == each[0] and column == each[1]:
            matrix.pop(pos)
            break
        pos = pos + 1

# Sorts the sparse matrix with below Algo
# If <Row, 0> value is greater than <Row, 1>
#    swap <Row,0> with <Row, 1>
# If <Row, 0> is same as <Row, 1>
#   Check <Row,1> with <Row, 1>
#   If <Row, 1> value is greater than <Row, 1>
#       swap <Row,0> with <Row, 1>
def sort(matrix):
    max_row = -1
    max_col = -1
    for i in range(0, len(matrix)):
        first = matrix[i]
        for j in range(i + 1, len(matrix)):
            second = matrix[j]

            if max_row < second[0]:
                max_row = second[0]
            if max_col < second[1]:
                max_col = second[1]
            if first[0] > second[0]:
                matrix[i] = second
                matrix[j] = first
                first = matrix[i]

            if first[0] == second[0]:
                if first[1] > second[1]:
                    matrix[i] = second
                    matrix[j] = first
                    first = matrix[i]
            #print(' i = ', i , '; j = ', j, 'matrix = ', matrix)
    return max_row, max_col

#Initialize the matrix
def init(matrix):
    row = int(input("Enter row count"))
    column = int(input("Enter column count"))
    insert_element(matrix, row, column)

#Print matrix
def print_matrix(matrix):
    print('============== Sparse Matrix ==================\n')
    max_row, max_col = sort(matrix)
    pos = 0
```

```python
    for i in range(0, max_row + 1):
        for j in range(0, max_col + 1):
            element = matrix[pos]
            if element[0] == i and element[1] == j:
                print(element[2], end='\t')
                if pos < len(matrix) - 1:
                    pos = pos + 1
            else:
                print(0, end='\t')
        print('\n')



matrix = []
init(matrix)
init(matrix)
init(matrix)
init(matrix)
print_matrix(matrix)
```

Output

```
Enter row count3
Enter column count4
Enter element7
Enter row count2
Enter column count3
Enter element8
Enter row count1
Enter column count2
Enter element6
Enter row count0
Enter column count5
Enter element3
============== Sparse Matrix ==================

0	0	0	0	0	3

0	0	6	0	0	0

0	0	0	8	0	0
```

Question 3:

```python
class BST:

    # Initialize node
    def __init__(self, data=None):
        self.left = None
        self.right = None
        self.data = data

    #Insert element to the array
    def insert(self, data):

        if data == self.data:
            return

        if data < self.data:
            # Add data in left subtree
            if self.left:
                self.left.insert(data)
            else:
                self.left = BST(data)
        else:
            if self.right:
                self.right.insert(data)
            else:
                self.right = BST(data)


    #Find maximum height of the tree
    def max_height(self, node):
        if node is None:
            return 0;

        else:

            left_subtree = node.max_height(node.left)
            right_subtree = node.max_height(node.right)

            if left_subtree > right_subtree:
                return left_subtree + 1
            else:
                return right_subtree + 1

    #Find minimum key
    def getMinimumKey(self, curr):
        if curr is not None:
            while curr.left:
                curr = curr.left

        return curr

    #Delete node from tree
    def delete(self, root, key):
        parent = None
        curr = root

        while curr and curr.data != key:
            parent = curr
            if key < curr.data:
                curr = curr.left
            else:
                curr = curr.right

        if curr is None:
            return root

        if curr.left is None and curr.right is None:

            if curr != root:
                if parent.left == curr:
                    parent.left = None
                else:
                    parent.right = None

            else:
                root = None

        elif curr.left and curr.right:
            successor = self.getMinimumKey(curr.right)
            val = successor.data
            self.delete(root, successor.data)
            curr.data = val

        else:
            if curr.left:
                child = curr.left
            else:
                child = curr.right

            if curr != root:
                if curr == parent.left:
                    parent.left = child
                else:
                    parent.right = child

            else:
                root = child

        return root

    #Search element in tree
    def search(self, root, element):
        height = 0
        width = 1
        while root is not None:
            if root.data == element:
                return height, width

            if root.data < element:
                root = root.right
                width = width * 2
            else:
                root = root.left
                width = (width * 2) - 1

            height = height + 1

        return -1
```

```python
    #Print element of tree
    def print(self, root):
        if root is BST:
            return

        result = []
        queue = [root]

        curr_height = 0
        height = self.max_height(root)

        while True:
            count = len(queue)

            if curr_height >= height:
                break

            result.append('{')
            while count > 0:
                temp = queue.pop(0)

                if temp.data == 'x':
                    if curr_height >= height:
                        count = count - 1
                        continue
                else:
                    result.append(temp.data)

                if count > 1:
                    result.append(',')

                if temp.left:
                    queue.append(temp.left)
                else:
                    queue.append(BST('x'))

                if temp.right:
                    queue.append(temp.right)
                else:
                    queue.append(BST('x'))

                count = count - 1

            result.append('}')
            result.append(',')
            curr_height = curr_height + 1

        del result[-1]
        for each in result:
            print(each, end=' ')


nums = [12, 6, 18, 19, 21, 11, 3, 5, 4, 24, 18]
bst = BST(nums[0])

for num in range(1, len(nums)):
    bst.insert(nums[num])

bst.delete(bst, 12)
bst.print(bst)
```

Output

```
{ 18 } , { 6 , 19 } , { 3 , 11 , , 21 } , { , 5 , , , , , , 24 } , { , , 4 , , , , , , , , , , , , , }
```

Question 4:

```python
class BST:

    def __init__(self, data=None):
        self.left = None
        self.right = None
        self.data = data

    def insert(self, data):

        if data == self.data:
            return

        if data < self.data:
            # Add data in left subtree
            if self.left:
                self.left.insert(data)
            else:
                self.left = BST(data)
        else:
            if self.right:
                self.right.insert(data)
            else:
                self.right = BST(data)

    def max_height(self, node):
        if node is None:
            return 0;

        else:

            left_subtree = node.max_height(node.left)
            right_subtree = node.max_height(node.right)

            if left_subtree > right_subtree:
                return left_subtree + 1
            else:
                return right_subtree + 1

    def get_minimum_key(self, curr):
        if curr is not None:
            while curr.left:
                curr = curr.left

        return curr

    def delete(self, root, key):
        parent = None
        curr = root

        while curr and curr.data != key:
            parent = curr

            if key < curr.data:
                curr = curr.left
            else:
                curr = curr.right

        if curr is None:
            return root

        if curr.left is None and curr.right is None:

            if curr != root:
                if parent.left == curr:
                    parent.left = None
                else:
                    parent.right = None

            else:
                root = None

        elif curr.left and curr.right:
            successor = self.get_minimum_key(curr.right)
            val = successor.data
            self.delete(root, successor.data)
            curr.data = val

        else:
            if curr.left:
                child = curr.left
            else:
                child = curr.right

            if curr != root:
                if curr == parent.left:
                    parent.left = child
                else:
                    parent.right = child

            else:
                root = child

        return root

    def search(self, root, element):
        height = 0
        width = 1
        while root is not None:
            if root.data == element:
                return height, width

            if root.data < element:
                root = root.right
                width = width * 2
            else:
                root = root.left
                width = (width * 2) - 1

            height = height + 1

        return -1

    def search_k_sum(self, root, k):
        queue = [root]

        while len(queue) > 0:
            temp = queue.pop()
            a = temp.data
            b = self.search(root, k - a)
            if isinstance(b, tuple):
                return a, k - a

            if temp.left:
                queue.append(temp.left)

            if temp.right:
                queue.append(temp.right)

        return -1


    def print(self, root):
        if root is BST:
            return

        result = []
        queue = [root]

        curr_height = 0
        height = self.max_height(root)

        while True:
            count = len(queue)

            if curr_height >= height:
                break

            result.append('{')
            while count > 0:
                temp = queue.pop(0)

                if temp.data == 'x':
                    if curr_height >= height:
                        count = count - 1
                        continue
                else:
                    result.append(temp.data)

                if count > 1:
                    result.append(',')

                if temp.left:
                    queue.append(temp.left)
                else:
                    queue.append(BST('x'))

                if temp.right:
                    queue.append(temp.right)
                else:
                    queue.append(BST('x'))

                count = count - 1

            result.append('}')
            result.append(',')
            curr_height = curr_height + 1

        del result[-1]
        for each in result:
            print(each, end=' ')
```

```python
nums = [12, 6, 18, 19, 21, 11, 3, 5, 4, 24, 18]
bst = BST(nums[0])

for num in range(1, len(nums)):
    bst.insert(nums[num])

print(bst.search_k_sum(bst, 1))
```

Output

```
(6, 6)
-1
```