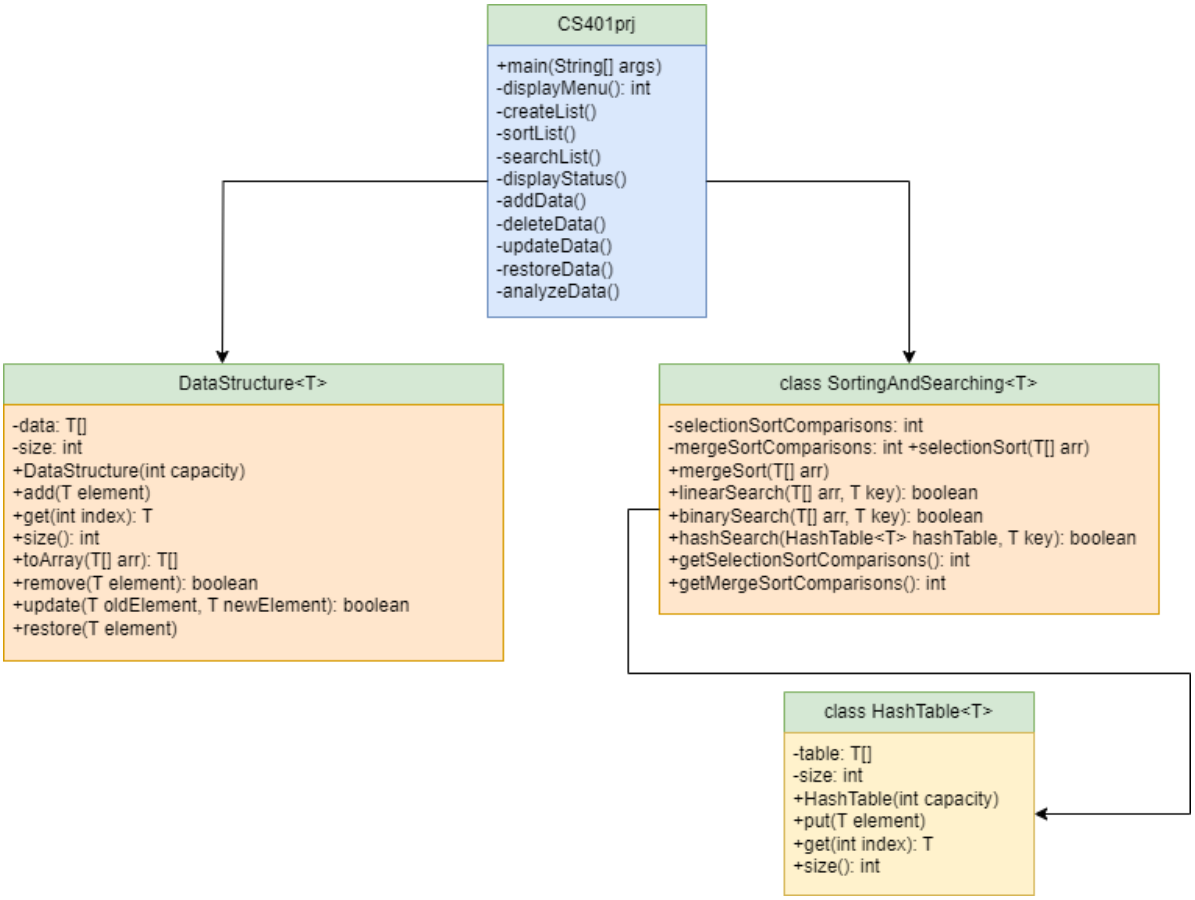
c) Design diagram document (including UML diagram and flow charts or pseudo code)

- UML diagram showing the CS401prj main class and the DataStructure and SortingAndSearching user-defined classes.



- Flow charts or pseudocode for the key algorithms (sorting, searching, and other operations)
  - Here is the pseudo code for entire program. Also, there are different pseudo codes for each operation separately at the end of this.

```
DEFINE class CS401prj

  DEFINE private static DataStructure<Integer> dataStructure

  DEFINE private static SortingAndSearching<Integer> sortingAndSearching

  DEFINE private static boolean listCreated = false


  DEFINE main()

    INITIALIZE dataStructure with capacity 100

    INITIALIZE sortingAndSearching
```

```
    WHILE true
        CALL displayMenu() and store the choice
        SWITCH choice
            CASE 0: CALL createList()
            CASE 1: CALL sortList()
            CASE 2: CALL searchList()
            CASE 3: CALL displayStatus()
            CASE 4: CALL addData()
            CASE 5: CALL deleteData()
            CASE 6: CALL updateData()
            CASE 7: CALL restoreData()
            CASE 8: CALL analyzeData()
            CASE 9: DISPLAY "Exiting program..." and EXIT program
            DEFAULT: DISPLAY "Invalid choice. Please try again."

DEFINE displayMenu()
    DEFINE options array with menu choices
    RETURN JOptionPane.showOptionDialog() to get user's choice

DEFINE createList()
    SET fileName to "C:\Users\chait\Desktop\emp.txt"
    TRY
        OPEN the file and read the number of lines n
        CREATE an Integer array elements of size n
        REOPEN the file and read the lines, parsing each line as an integer and storing in elements array
        FOR each element in elements array
            CALL dataStructure.add(element) to add the element
        DISPLAY the created list
        SET listCreated to true
    CATCH IOException
        DISPLAY "Error reading file: " + e.getMessage()
```

```
DEFINE sortList()
    IF listCreated is false
        DISPLAY "Please create a list first."
        RETURN
    ELSE
        GET the array from dataStructure
        CALL sortingAndSearching.selectionSort(arr)
        CALL sortingAndSearching.mergeSort(arr)
        DISPLAY the sorted list, selection sort comparisons, and merge sort comparisons

DEFINE searchList()
    IF listCreated is false
        DISPLAY "Please create a list first."
        RETURN
    ELSE
        GET the array from dataStructure
        PROMPT user to enter the element to search
        CALL sortingAndSearching.linearSearch(arr, parsedKey) and store the result
        SORT the array
        CALL sortingAndSearching.binarySearch(arr, parsedKey) and store the result
        CREATE a HashTable and add the elements
        CALL sortingAndSearching.hashSearch(hashTable, parsedKey) and store the result
        DISPLAY the search results

DEFINE displayStatus()
    IF listCreated is false
        DISPLAY "Please create a list first."
        RETURN
    ELSE
        GET the array from dataStructure
```

```
        DISPLAY the number of elements and the list of elements


DEFINE addData()
    IF listCreated is false
        DISPLAY "Please create a list first."
        RETURN
    ELSE
        PROMPT user to enter the element to add
        CALL dataStructure.add(parsedElement)
        DISPLAY "Element added successfully."


DEFINE deleteData()
    IF listCreated is false
        DISPLAY "Please create a list first."
        RETURN
    ELSE
        PROMPT user to enter the element to delete
        CALL dataStructure.remove(parsedElement) and store the result
        IF result is true
            DISPLAY "Element deleted successfully."
        ELSE
            DISPLAY "Element not found in the list."


DEFINE updateData()
    IF listCreated is false
        DISPLAY "Please create a list first."
        RETURN
    ELSE
        PROMPT user to enter the old element
        PROMPT user to enter the new element
        CALL dataStructure.update(parsedOldElement, parsedNewElement) and store the result
```

```
        IF result is true
            DISPLAY "Element updated successfully."
        ELSE
            DISPLAY "Element not found in the list."

DEFINE restoreData()
    IF listCreated is false
        DISPLAY "Please create a list first."
        RETURN
    ELSE
        PROMPT user to enter the element to restore
        CALL dataStructure.restore(parsedElement)
        DISPLAY "Element restored successfully."

DEFINE analyzeData()
    IF listCreated is false
        DISPLAY "Please create a list first."
        RETURN
    ELSE
        RECORD the start time
        // Perform analysis operations
        RECORD the end time
        CALCULATE the execution time
        DISPLAY the time complexities and execution time

DEFINE DataStructure<T> class
    DEFINE private T[] data
    DEFINE private int size

    DEFINE constructor(capacity)
        INITIALIZE data with capacity
```

```
    SET size to 0

DEFINE add(element)
    CALL ensureCapacity()
    ADD the element to data at index size
    INCREMENT size

DEFINE ensureCapacity()
    IF size equals data length
        DOUBLE the size of data

DEFINE get(index)
    IF index is out of bounds
        THROW IndexOutOfBoundsException
    ELSE
        RETURN data[index]

DEFINE size()
    RETURN size

DEFINE toArray(arr)
    RETURN copy of data array up to size

DEFINE remove(element)
    GET index of the element
    IF index is -1 (not found)
        RETURN false
    ELSE
        CALL removeAt(index)
        RETURN true
```

```
    DEFINE indexOf(element)
        LOOP through data
            IF data[i] equals element
                RETURN i
        RETURN -1

    DEFINE removeAt(index)
        COPY elements after index to one position before
        DECREMENT size

    DEFINE update(oldElement, newElement)
        GET index of oldElement
        IF index is -1
            RETURN false
        ELSE
            SET data[index] to newElement
            RETURN true

    DEFINE restore(element)
        CALL add(element)

DEFINE SortingAndSearching<T> class
    DEFINE private int selectionSortComparisons
    DEFINE private int mergeSortComparisons

    DEFINE selectionSort(arr)
        SET selectionSortComparisons to 0
        FOR each index i from 0 to length-2
            SET minIndex to i
            FOR each index j from i+1 to length-1
                INCREMENT selectionSortComparisons
```

```
        IF arr[j] < arr[minIndex]
            SET minIndex to j
        SWAP arr[i] and arr[minIndex]

DEFINE mergeSort(arr)
    SET mergeSortComparisons to 0
    CALL auxMergeSort(arr, 0, length-1)

DEFINE auxMergeSort(arr, low, high)
    IF low < high
        SET mid to (low + high) / 2
        CALL auxMergeSort(arr, low, mid)
        CALL auxMergeSort(arr, mid+1, high)
        CALL merge(arr, low, mid, high)

DEFINE merge(arr, low, mid, high)
    CREATE aux array by copying arr
    SET i to low, j to mid+1
    FOR k from low to high
        INCREMENT mergeSortComparisons
        IF i > mid
            SET arr[k] to aux[j] and INCREMENT j
        ELSE IF j > high
            SET arr[k] to aux[i] and INCREMENT i
        ELSE IF aux[j] < aux[i]
            SET arr[k] to aux[j] and INCREMENT j
        ELSE
            SET arr[k] to aux[i] and INCREMENT i

DEFINE swap(arr, i, j)
    SWAP arr[i] and arr[j]
```

```
DEFINE linearSearch(arr, key)
    FOR each element in arr
        IF element equals key
            RETURN true
    RETURN false

DEFINE binarySearch(arr, key)
    SET low to 0, high to length-1
    WHILE low <= high
        SET mid to (low + high) / 2
        IF arr[mid] equals key
            RETURN true
        ELSE IF arr[mid] < key
            SET low to mid + 1
        ELSE
            SET high to mid - 1
    RETURN false

DEFINE hashSearch(hashTable, key)
    SET index to hashFunction(key, hashTable size)
    IF hashTable[index] is not null AND equals key
        RETURN true [3]
    RETURN false

DEFINE hashFunction(key, tableSize)
    RETURN Math.abs(key.hashCode()) % tableSize [3]

DEFINE getSelectionSortComparisons()
    RETURN selectionSortComparisons
```

```
    DEFINE getMergeSortComparisons()
        RETURN mergeSortComparisons

DEFINE HashTable<T> class
    DEFINE private T[] table
    DEFINE private int size

    DEFINE constructor(capacity)
        INITIALIZE table with capacity
        SET size to 0

    DEFINE put(element)
        SET index to hashFunction(element, table length)
        IF table[index] is null
            SET table[index] to element
            INCREMENT size

    DEFINE get(index)
        RETURN table[index]

    DEFINE size()
        RETURN size

    DEFINE hashFunction(element, tableSize)
        RETURN Math.abs(element.hashCode()) % tableSize [3]
```

```
        RETURN the absolute value of key.hashCode() modulo tableSize

    FUNCTION getSelectionSortComparisons()
        RETURN selectionSortComparisons

    FUNCTION getMergeSortComparisons()
        RETURN mergeSortComparisons
END CLASS

CLASS HashTable<T extends Comparable<T>>
    DECLARE table as an array of T
    DECLARE size as an integer

    CONSTRUCTOR HashTable(capacity)
        INITIALIZE table as an array of T with the specified capacity
        SET size to 0

    FUNCTION put(element)
        DECLARE index as the result of hashFunction(element, table.length)
        IF the element at index is null
            SET the element at index to element
            INCREMENT size

    FUNCTION get(index)
        RETURN the element at the specified index in the table

    FUNCTION size()
```

```
    RETURN size

  FUNCTION hashFunction(element, tableSize)
    RETURN the absolute value of element.hashCode() modulo tableSize
END CLASS
```

❖ *Pseudocodes for methods individually.*

- **createList**

```
Function CreateList()
Input: none
 Output: none

 declare n as integer
 prompt user for the number of elements (store in n)
 if n is null then
    return (exit function)  // Added null check
 end if

 try converting input to integer
    n = parseInteger(input)
 catch NumberFormatException
    display message "Invalid input. Please enter a valid integer."
    return (exit function)
 end try

 declare elements as array of integer with size n
```

```
  for i from 0 to n-1
     prompt user for element i (store in elements[i])
     if element i is null then
        return (exit function)  // Added null check
     end if
  end for

  for each element in elements
     call DataStructure.add(element)
  end for

  display message "List created: " + elements.toString()
  set listCreated to true

End Function
```

- **sortList**

```
Function SortList()
Input: none
Output: none

  if listCreated is false then
     display message "Please create a list first."
     return (exit function)
  end if
```

get elements from DataStructure

call SortingAndSearching.selectionSort(elements)

call SortingAndSearching.mergeSort(elements)

display message "Sorted list: " + elements.toString()

display message "Selection Sort Comparisons: " + SortingAndSearching.getSelectionSortComparisons()

display message "Merge Sort Comparisons: " + SortingAndSearching.getMergeSortComparisons()

End Function

- **searchList**

Function SearchList()

Input: none

Output: none

if listCreated is false then

display message "Please create a list first."

return (exit function)

end if

get elements from DataStructure

prompt user for element to search (store in key)

linearSearchResult = call SortingAndSearching.linearSearch(elements, key)

sort elements (using built-in sort function)

binarySearchResult = call SortingAndSearching.binarySearch(elements, key)

```
  create a new HashTable with size equal to elements.length
  for each element in elements
      call HashTable.put(element)
  end for

  hashSearchResult = call SortingAndSearching.hashSearch(HashTable, key)

  display message "Search Results:"
  display message "Linear search result: " + linearSearchResult
  display message "Binary search result: " + binarySearchResult
  display message "Hash search result: " + hashSearchResult
End Function

displayStatus
Function DisplayStatus()
  Input: none
  Output: none

  if listCreated is false then
      display message "Please create a list first."
      return (exit function)
  end if

  get size and elements from DataStructure
  display message "Number of elements: " + size
  display message "List of elements: " + elements.toString()
```

End Function

- **addData**

Function AddData()

Input: none

Output: none

```
if listCreated is false then
    display message "Please create a list first."
    return (exit function)
end if

prompt user for element to add (store in element)
if element is null then
    return (exit function)  // Added null check
end if

call DataStructure.add(element)
display message "Element added successfully."
```

End Function

- **deleteData**

Function DeleteData()

Input: none

Output: none

```
if listCreated is false then
```

```
    display message "Please create a list first."
    return (exit function)
  end if

  prompt user for element to delete (store in element)
  if element is null then
    return (exit function)  // Added null check
  end if

  removed = call DataStructure.remove(element)
  if removed is true then
    display message "Element deleted successfully."
  else
    display message "Element not found in the list."
  end if
End Function
```

- **updateData**

```
Function UpdateData()
Input: none
  Output: none

  if listCreated is false then
    display message "Please create a list first."
    return (exit function)
  end if
```

```
  prompt user for element to update (store in oldElement)
  if oldElement is null then
     return (exit function)  // Added null check
  end if

  prompt user for the new element (store in newElement)
  if newElement is null then
     return (exit function)  // Added null check
  end if

  updated = call DataStructure.update(oldElement, newElement)
  if updated is true then
     display message "Element updated successfully."
  else
     display message "Element not found in the list."
  end if
End Function
```