Algorithms & Data Structures

Use the following algorithms and data structures

- `BubbleSort(A)` - sorts an array A using the bubble sort algorithm.
- `InsertionSort(A)` - sorts an array A using the insertion sort algorithm.
- `SelectionSort(A)` - sorts an array A using the selection sort algorithm.
- `MergeSort(A)` - sorts an array A using the merge sort algorithm.
- `QuickSort(A)` - sorts an array A using the quick sort algorithm.
- `Partition(A,l,h)` - partitions subarray of A from l to h, inclusively, using h's element as the pivot.
- `HeapSort(A)` - sorts an array A using the heap sort algorithm.
- `MaxHeap(A)` - arrange an array A into a max-heap.
- `MinHeap(A)` - arrange an array A into a min-heap.
- `HashCountSort(A,h,k)` - sort an array A using the count sort algorithm with hash function h.
- `Select(A,i)` - returns the ith smallest element of the array A.
- `Random(a,b)` - returns a random integer in the interval [a,b].
- `Swap(a,b)` - swaps the values of a and b.
- `Digit(x)` - returns a hash value for character x associated to the set of digits.
- `Letter(x)` - returns a hash value for character x associated to the set of letters.
- `Alphanumeric(x)` - returns a hash value for character x associated to the set of digits and letters.
- `Search(T,x)` - returns the node of the BST T whose data equals x.
- `Maximum(T)` - returns the node with the maximum data of the BST T.
- `Minimum(T)` - returns the node with the minimum data of the BST T.
- `Successor(n)` - returns the node that has the smallest data greater than the data of n of a BST.
- `Predecessor(n)` - returns the node that has the largest data less than the data of n of a BST.
- `Stack<T>` - follows first in last out principle
  - `push(x)` - inserts into the stack.
  - `pop()` - removes from the stack.
  - `top()` - returns the next item to be removed from the stack.
  - `empty()` - checks if the stack is empty.
- `Queue<T>` - follows first in first out principle
  - `enqueue(x)` - inserts into the queue.
  - `dequeue()` - removes from the queue.
  - `peek()` - returns the next item to be removed from the queue.
  - `empty()` - checks if the queue is empty.

- `Set<T>` - a collection of distinct objects
  - `insert(x)` - inserts into the set if not already present.
  - `remove(x)` - removes item from the set if present.
  - `size()` - returns the number of elements in the set.
  - `contains(x)` - checks if item is in the set.
  - `empty()` - checks if the set is empty.
- `Node<T>` - linked list node
  - `data` - content of the node.
  - `next` - next node link of the node.
  - `prev` - previous node link of the node.
- `TNode<T>` - binary tree node
  - `data` - content of the node.
  - `p` - parent node link of the node.
  - `left` - left child node link of the node.
  - `right` - right child node link of the node.