COIS 3020H Assignment 2: Treaps README Documentation

Description:
The Treap is a hybrid data structure that combines the properties of a binary search tree and a max heap. It maintains the binary search tree property with respect to the keys and the max heap property with respect to the priorities assigned to each key.

Contributors:
Adam Kassana
Alexander Wolf
Carmen Tullio

Content Table

Treaps.cs 3
- TreapNode Class 3
  - Class Data Fields: 3
  - Constructors: 4
    - TreapNode(T Key): 4
    - TreapNode(T Key, int priority): 4
- Treap Class 5
  - Class Data Fields: 5
  - Constructor: 5
    - ● public Treap() 5
  - Public Methods: 5
    - ● public void MakeEmpty() 5
    - ● public bool IsEmpty() 5
    - ● public bool Insert(T item, bool print = true) 5
    - ● public bool InsertP(T item, int priority, bool print = true) 6
    - ● public bool Delete(T item, bool print = true) 6
    - ● public bool Search(T item) 6
    - ● public bool RangeQuery(T MinRange, T MaxRange, bool print = false, T item = default(T)) 6
    - ● public Treap<T> Split(T item) 7
    - ● public Treap<T> Merge(Treap<T> lTreap, Treap<T> rTreap) 7
    - ● public int Size() 7
    - ● public TreapNode<T> SplitSearchUtil(T item) 7

Treaps.cs

The Treaps.cs file is a file that defines a class for representing a Treap data structure. It provides numerous basic functions such as Insertion, Search and Deletion as well as other advanced functions for merging, splitting and range-queries. It consists of the class definition for a treap as well as an internal TreapNode utility class

TreapNode Class

The TreapNode<T> class represents a node in the Treap structure.

Class Data Fields:

- public T item
  A generic T representing a placeholder for the value as determined by the user with read/write properties.

- public Int Priority

Integer variable representing priority assigned to the node, embedded with read/write properties.

- public TreapNode<T> Left
  A field of type TreapNode<T> representing the left child of the current node.

- public TreapNode<T> Right
  A field of type TreapNode<T> representing the right child of the current node.

Constructors:

TreapNode(T Key):

Initializes a node with a key of type T and a randomly generated priority.
E.g. TreapNode<T> node = new TreapNode(T item);

TreapNode(T Key, int priority):

Initializes a node with a specified key of type T and a priority.
E.g. TreapNode<T> node = new TreapNode(T item, int Priority);

Treap Class

The Treap<T> class represents the Treap data structure and implements the ISearchable<T> interface.

Class Data Fields:

- Private TreapNode<T> Root;
  The root of the Treap, which is the topmost node in the tree structure. It serves as the entry point for accessing and navigating the Treap.

Constructor:

- public Treap()
  Description: This constructor initializes an empty Treap with a null root.
  Example of Use: Treap<int> myTreap = new Treap<int>();
  Time Complexity: O(1)

Public Methods:

- public void MakeEmpty()
  Description: This method empties the Treap by setting the root to null.
  Example of Use: myTreap.MakeEmpty();
  Time Complexity: O(1)

- public bool IsEmpty()
  Description: Checks if the Treap is empty by examining the root.Returns true if the Treap is empty; false otherwise.
  Example of Use: bool empty = myTreap.IsEmpty();
  Time Complexity: O(1)

- public bool Insert(T item, bool print = true)
  Description: Inserts a new node with the specified item and a randomly generated priority, maintaining the Treap properties. Returns true on successful insert. Invokes corresponding utility method private TreapNode<T> Insert(T item, TreapNode<T> node)

Example of Use: myTreap.Insert(42);
Time Complexity: O(log (n)) - The time complexity is logarithmic due to the potential recursive call of the insert method

- public bool InsertP(T item, int priority, bool print = true)
  Description: Variation of insert that has a manual declaration of priority instead of randomly generated value for priority Returns true on successful insert. Invokes corresponding utility method private TreapNode<T> InsertP(T item, int priority, TreapNode<T> node)
  Example of Use: myTreap.InsertP(42, 90);
  Time Complexity: O(log (n)) - The time complexity is logarithmic due to the recursive nature of this method

- public bool Delete(T item, bool print = true)
  Description: Deletes the node containing the specified item, maintaining the Treap properties. Invokes corresponding utility method private TreapNode<T> Delete(T item, TreapNode<T> node)
  Example of Use: myTreap.Delete(42);
  Time Complexity: O(log (n)) - The time complexity is logarithmic as this method requires itself to be recursively called.

- public bool Search(T item)
  Description: Searches for the specified item in the Treap using In-Order Traversal. Returns true if the item is found
  Example of Use: bool exists = myTreap.Search(42);
  Time Complexity: O(log (n)) - The time complexity is logarithmic as it involves traversing the height of the Treap during search.

- public bool RangeQuery(T MinRange, T MaxRange, bool print = false, T item = default(T))
  Description: Searches for elements within a specified range and optionally prints them.Performs an in-order traversal and prints or searches for elements within the specified range.
  Example of Use: bool found = myTreap.RangeQuery(10, 50, true, 42);
  Time Complexity: O(n) - The time complexity is linear as it involves traversing all nodes in the Treap.

- public Treap<T> Split(T item)
  Description: Splits the Treap into two Treaps based on the key input, one containing element up until that key item, the other which will contain that key item and all subsequent child nodes. It uses utility functions such as:
    - private void Storage
    - public TreapNode<T> SplitSearchUtil

  Example of Use: Treap<int> targetTreap = sourceTreap.Split(30);
  Time Complexity: O(n) - The time complexity is linear as it involves traversing all nodes in the Treap during the split operation.

- public Treap<T> Merge(Treap<T> lTreap, Treap<T> rTreap)
  Description: Merges two Treaps into a new Treap, preserving the Treap properties. It uses utility functions such as private TreapNode<T> InsertP which allows for insertion of nodes based on key and value pairs.
  Example of Use: Treap<int> mergedTreap = myTreap.Merge(leftTreap, rightTreap);
  Time Complexity: O(m), where m is the total number of nodes in both Treaps. The time complexity is linear as it involves traversing all nodes in the merged Treaps.

- public int Size()
  Description: Counts and returns the number of nodes in the Treap.
  Example of Use: int treapSize = myTreap.Size();
  Time Complexity: O(n) - The time complexity is linear as it involves traversing all nodes in the Treap.

- public TreapNode<T> SplitSearchUtil(T item)
  Description: Search utility to Split method. Searches for break-off point in Treap by checking child nodes of index node and returns it to scope of Split method. Removes break-off node and all subsequent child nodes from source treap by assigning null value to index
  Example of Use: TreapNode<T> breakOff = this.SplitSearchUtil(item);
  Time Complexity: O(log(n)) The time complexity is logarithmic as it involves traversing the height of the Treap during search.

Private Methods:

- private TreapNode<T> LeftRotate(TreapNode<T> root)
  Description: Rotates the nodes in the Treap to the left around the specified root, maintaining the Treap properties. Used by insertion methods and other methods that require balancing to maintain properties.
  Example of Use: node = LeftRotate(node);
  Time Complexity: O(1) - The time complexity is constant as it involves simple pointer adjustments.

- private TreapNode<T> RightRotate(TreapNode<T> root)
  Description: Rotates the nodes in the Treap to the right around the specified root, maintaining the Treap properties. Used by insertion methods and other methods that require balancing to maintain properties.
  Example of Use: node = RightRotate(node);
  Time Complexity: O(1) - The time complexity is constant as it involves simple pointer adjustments.

- private TreapNode<T> Insert(T item, TreapNode<T> node)
  Description: Recursively inserts a new node with the specified item and random priority, maintaining the Treap properties. Returns the current root on successful insert. Utility function for public Insert method
  Example of Use: Root = Insert(item, Root);
  Time Complexity: O(log n) - The time complexity is logarithmic as it involves traversing the height of the Treap during insertion.

- private TreapNode<T> InsertP(T item, int priority, TreapNode<T> node)
  Description: Recursively inserts a new node with the specified item and priority, maintaining the Treap properties.
  Example of Use: Root = InsertP(item, priority, Root);
  Time Complexity: O(log n) - The time complexity is logarithmic as it involves traversing the height of the Treap during insertion.

- private TreapNode<T> Delete(T item, TreapNode<T> node):
  Description: Recursively deletes the node containing the specified item, maintaining the Treap properties.
  Example of Use: Root = Delete(item, Root);
  Time Complexity: O(log n) - The time complexity is logarithmic as it involves traversing the height of the Treap during deletion.

- private void Storage(ref Dictionary<T, int> dict, TreapNode<T> current)
  Description: Performs in-order traversal and stores key-value pairs of Treap nodes in the provided dictionary for later use.
  Example of Use: Storage(ref TreapList, RootNode)
  Time Complexity: O(n) - The complexity is due to the method iterating through all nodes within the treap to add to the dictionary.