Generic Treap (Binary Search Tree + Heap) in C++
Project ID: 5

Team Details:

Darshan D - PES1201801456
Karan Kumar G - PES1201801883
Mayur P L - PES1201801439

Treap is a data structure that stores pairs (X, Y) in a binary tree in such a way that it follows binary search tree property through X and a binary (maximum) heap by Y.

Operations allowed:

- Insert node
    - Function signature: void insert(key_t key)
    - Inserts the key as a node into the treap.

- Delete node
    - Function signature: void delete_(key_t key)
    - Deletes the node with the key passed as argument

- Split treap
    - Function signature: void split(key_t key, Treap_t *left_sub_treap, Treap_t *right_sub_treap)
    - Splits the treap into two subtrees based on the key passed. The two subtrees are stored in the left_sub_treap and right_sub_treap respectively

- Merge treaps
    - Function signature: void merge(Treap_t *left_sub_treap, Treap_t *right_sub_treap)
    - Merges the two treaps left_sub_treap and right_sub_treap into one treap and modifies this pointer(object through which the function is called) to point to this new merged treap

- Union
    - Function signature: void union_treaps(Treap_t *treap1, Treap_t *treap2)
    - Finds the union of the two treaps treap1 and treap2. Duplicates are handled such that the final treap only considers one copy of the key. The

object through which the function is called is made to point to this new treap generated after performing the union operation

- Intersection
    - Function signature: void intersect_treaps(Treap_t *treap1, Treap_t *treap2)
    - Finds the intersection of the two treaps treap1 and treap2. The object through which the function is called is made to point to this new treap generated after performing the intersection operation

- Difference
    - Function signature: void diff_treap(Treap_t *treap1, Treap_t *treap2)
    - Finds the difference of the two treaps treap1 and treap2. The object through which the function is called is made to point to this new treap generated after performing the difference operation

- Traversals:
    - Inorder:
        - Function signature: void inorder()
        - Performs inorder traversal of the treap
    - Preorder:
        - Function signature: void preorder()
        - Performs preorder traversal of the treap
    - Postorder:
        - Function signature: void postorder()
        - Performs postorder traversal of the treap

- Member Find
    - Function signature: Iterator find(Iterator first, Iterator last, key_t search_key)
    - Finds and returns an iterator pointing to the node containing the key passed as argument. The find operation is performed in O(logn) time.
    - If the key is not found, the end iterator is returned.

- Member Replace
    - Function signature: void replace(Iterator first, Iterator last, key_t old_value, key_t new_value)
    - Finds and replaces the node containing the old_value with the new_value

Details about running the software:

- The implementation file consisting of all the implementation code and functions with respect to treaps has been provided as a header file that the client can include in their programs. The way to include it is as follows:
  #include “treap.h”

- Once this is included, clients can create treaps and call functions to perform different operations as explained above using the provided interface.

- The client file is then compiled using the g++ utility.
  g++ client.cpp -o exec

  - Multiple client files have been submitted where each client file checks for a few functionalities
  - **client1.cpp -** Checks for insertion and deletion of nodes and copying treaps
  - **client2.cpp -** Checks for split and merge operations on the treaps
  - **client3.cpp -** Checks for Union, Intersection and Difference set operations
  - **client4.cpp -** Testing of Iterators on member and generic algorithms
  - **client5.cpp -** Checks for move constructor and move assignment operator

- The generated executable is then loaded and executed.
  ./exec