# Data Structures – Project 2

## 1. Project Description

The project requires the design and implementation of a data structure that combines:  
- A binary search tree (BST) based on registration timestamps.  
- A min-heap structure based on player ranking.  
  
You must implement basic functions for inserting, deleting, and searching players, using both ranking and timestamp.

## 2. Data Structures Description

Class participant:  
- Fields: fullname, ID, ranking, origin, timestamp  
- Pointers: left, right, parent (for representing the tree structure)  
  
Class Tree:  
- Field: root  
- Combines min-heap (for ranking) and BST (for timestamp)

## 3. Functions & Complexities

|  |  |  |
| --- | --- | --- |
| Function | Time Complexity | Extra Space |
| insertPlayers | O(h) for BST + heapify | O(1) |
| deletePlayer | O(h) | O(1) |
| findKthBestPlayer | O(n + k log k) | O(k) |
| printIDsInTimeRange | O(#output + h) | O(h) |
| deleteAllWithTimestamp | O(k log k) | O(k) |
| bestBeforeTimestamp | O(h) | O(1) |

## 4. Code Comments & Design Decisions

- Field values are swapped during heap operations instead of changing pointers.  
- A min-heap (priority queue) is used to find the k-th best player.  
- Recursive implementations are used for tree traversal simplicity.

## Code Function Summaries

main()  
- Initializes the tree and displays the user menu.  
- Accepts user input and calls the appropriate tree function until the user chooses to exit.

menu()  
- Displays all available options (1–7, plus 8 for debugging).  
- Includes input validation to ensure numeric input.

actions(int answ, Tree& tree1)  
- Executes the function based on the user’s menu choice:  
 1. Load players from file  
 2. Delete player by name  
 3. Find k-th best player  
 4. Print player IDs in time range  
 5. Delete all players with a given timestamp  
 6. Find best player before a given timestamp  
 7. Exit  
 8. Print all players (for testing)

## Tree Methods Summary

Loading Players from File  
- insertPlayers(filename) inserts each player into the tree as a BST (using insertAsBST()), then applies heapifyUp() to maintain min-heap order by ranking.  
- parseData(str) parses a line from the file and assigns values to a participant object.

Delete Player by Name  
- Finds player by fullname using BST.  
- Performs heapify-down by swapping only ranking-related fields (not fullname).  
- Repeats until the node becomes a leaf, then deletes it.

Find k-th Best Ranked Player  
- Uses a min-heap (priority queue) of size k to track the top-k players.  
- Time complexity: O(k log k)  
 Explanation: At each step, the best candidate is removed and its children are added. This repeats k times.

Print IDs in Time Range  
- Performs in-order traversal and prints players with timestamp ∈ [t1, t2].  
- Time complexity: O(#output + h)  
- Space complexity: O(h) due to recursion stack.

Delete All Players with a Specific Timestamp  
- Collects all nodes with the given timestamp using in-order traversal.  
- Deletes them efficiently (better than repeated deletePlayer calls).  
- Time: O(k log k) (k = number of matches)  
- Space: O(k) (to store the nodes temporarily)

Find Best Player Before Given Timestamp  
- Traverses the BST and keeps track of the best-ranked player where timestamp < target.  
- Time: O(h)  
- Space: O(h) (recursive) or O(1) (iterative)