After the Olympiad in Paris had already commenced, Claude Carmel Cartier, the lead programmer responsible for developing the system, recalled the existence of an additional domain with rules distinct from those supported by the current system. Consequently, he tasked you with constructing an additional system to accommodate this new domain. The system is designed to facilitate the addition and removal of teams and players, maintain statistics, and simulate individual games and tournaments between teams.

Implement a data structure supporting the following operations:

1. `void olympics_t()`: Initializes an empty data structure. Initially, there are no teams or players in the system.
   - Parameters: None
   - Return value: None
   - Time complexity: O(1) in the worst case

2. `virtual ~ olympics_t()`: Releases the data structure, freeing all allocated memory.
   - Parameters: None
   - Return value: None
   - Time complexity: O(n + k) in the worst case, where n is the number of teams and k is the number of players.

3. `StatusType add_team(int teamId)`: Adds a team with a unique identifier teamId to the competition.
   - Parameters:
     - teamId: The unique identifier of the new team.
   - Return value:
     - ALLOCATION_ERROR: Memory allocation or deallocation problem.
     - INVALID_INPUT: If teamId ≤ 0.
     - FAILURE: If teamId is already in use.
     - SUCCESS: If the operation is successful.
   - Time complexity: O(1) in the worst case.

4. `StatusType remove_team(int teamId)`: Removes the team with the identifier teamId from the tournament, along with all its players.
   - Parameters:
     - teamId: The identifier of the team to be removed.
   - Return value:
     - ALLOCATION_ERROR: Memory allocation or deallocation problem.
     - INVALID_INPUT: If teamId ≤ 0.
     - FAILURE: If there is no team with the identifier teamId.
     - SUCCESS: If the operation is successful.
   - Time complexity: O(log(n) + k_players_in_team) in the worst case.

5. `StatusType add_player(int teamId, int playerStrength)`: Adds a new player to the team with the identifier teamId in the competition.

- Parameters:
  - teamId: The identifier of the team the player belongs to.
  - playerStrength: The strength of the player.
- Return value:
  - ALLOCATION_ERROR: Memory allocation or deallocation problem.
  - INVALID_INPUT: If teamId ≤ 0 or playerStrength ≤ 0.
  - FAILURE: If the team with the identifier teamId does not exist.
  - SUCCESS: If the operation is successful.
- Time complexity: O(log(n) + log(k_players_in_team)) in the worst case.

6. `StatusType remove_newest_player(int teamId)`: Removes the newest player from the team with the identifier teamId.
   - Parameters:
     - teamId: The identifier of the team the player belongs to.
   - Return value:
     - ALLOCATION_ERROR: Memory allocation or deallocation problem.
     - INVALID_INPUT: If teamId ≤ 0.
     - FAILURE: If the team with the identifier teamId does not exist or is empty.
     - SUCCESS: If the operation is successful.
   - Time complexity: O(log(n) + log(k_players_in_team)) in the worst case.

7. `output_t<int> play_match(int teamId1, int teamId2)`: Simulates a match between two teams with identifiers teamId1 and teamId2.
   - Parameters:
     - teamId1: The identifier of the first team.
     - teamId2: The identifier of the second team.
   - Return value:
     - The identifier of the winning team.
     - Status:
       - ALLOCATION_ERROR: Memory allocation or deallocation problem.
       - INVALID_INPUT: If teamId1 ≤ 0, teamId2 ≤ 0, or teamId1 = teamId2.
       - FAILURE: If there is no team with identifier teamId1 or teamId2, or if one of the teams is empty.
       - SUCCESS: If the operation is successful.
   - Time complexity: O(log n) in the worst case.

8. `output_t<int> num_wins_for_team(int teamId)`: Returns the total number of wins for the team with the identifier teamId among the matches it has participated in.
   - Parameters:
     - teamId: The identifier of the team.
   - Return value:
     - The total number of matches in which the team participated.
     - Status:
       - ALLOCATION_ERROR: Memory allocation or deallocation problem.
       - INVALID_INPUT: If teamId ≤ 0.
       - FAILURE: If the team with the identifier teamId does not exist.

      - SUCCESS: If the operation is successful.
   - Time complexity: O(log n) in the worst case.

9. `output_t<int> get_highest_ranked_team()`: Returns the rank of the team with the highest rank in the system. If there are no teams in the system, it returns 1.
   - Return value:
     - The rank of the highest-ranked team if there are teams in the system, otherwise 1.
     - Status:
       - ALLOCATION_ERROR: Memory allocation or deallocation problem.
       - SUCCESS: If the operation is successful.
   - Time complexity: O(1) in the worst case.

10. `StatusType unite_teams(int teamId1, int teamId2)`: Merges two teams, where the team with teamId1 acquires the team with teamId2.
    - Parameters:
      - teamId1: The identifier of the acquiring team.
      - teamId2: The identifier of the team being acquired.
    - Return value:
      - ALLOCATION_ERROR: Memory allocation or deallocation problem.
      - INVALID_INPUT: If teamId1 ≤ 0, teamId2 ≤ 0, or teamId1 = teamId2.
      - FAILURE: If the team with identifier teamId1 or teamId2 does not exist.
      - SUCCESS: If the operation is successful.
    - Time complexity: O(log(n)) + k_players_in_team1 + k_players_in_team2 in the worst case.

11. `output_t<int> play_tournament(int lowPower, int highPower)`: Organizes a tournament among all teams whose strengths fall within the range from lowPower to highPower (inclusive).
    - Parameters:
      - lowPower: The minimum power value that allows participation in the tournament.
      - highPower: The maximum power value that allows participation in the tournament.
    - Return value:
      - The identifier of the winning team in the tournament.
      - Additional status:
        - ALLOCATION_ERROR: Memory allocation or deallocation problem.
        - INVALID_INPUT: If lowPower ≤ 0, highPower ≤ 0, or highPower ≤ lowPower.
        - FAILURE: If the number of teams in the given range is not a power of 2.
        - SUCCESS: If the operation is successful.
    - Time complexity: O(log(i) · log(n)) in the worst case, where n is the number of teams in the system and i is the number of teams in the tournament.

The space complexity of the data structure is O(n + k) in the worst case, where n is the number of teams and k is the number of players. This means that at any given moment during execution, the memory consumption of the data structure will be linear in the sum of the number of players and teams in the system. For each operation, there is a need for auxiliary memory, and although the space complexity required for each operation is not specified specifically, it is important not to exceed the overall space complexity defined for the data structure.