- רעיון כללי :
נגדיר שתי אובייקטים Player והשדות שמשתמשים בהם בהסבר:
id-1 הוא מספר מזהה של השחקן. goals-2 מספר השערים לשחקן .gamesPlayed-3 מספר המשחקים שהשחקן שיחק cards-4 הכרטיסים שהשחקן קיבל. team-5 מצביע על אובייקט Team
Team הוא מהווה קבוצה במערכת והשדות שלו הן:
teamId-1 הוא מספר מזהה של הקבוצה. topscorer-2 מכיל מספר מזהה להשחקן שהפקיע הכי הרבה שערים בקבוצה . By_id-3 הוא עץ AVL שמכיל השחקנים בקבוצה ומכניס אותם לפי הסמפר המזהה שלהם . By_goals-4 הוא עץ AVL שמכיל השחקנים בקבוצה ומכניס אותם לפי השערים שלהם. points-5 נקודות של הקבוצה. Xp-6 מוחשב לפי הנוסחה ב- . play_match. is_valid-6 הוא מהווה אם הקבוצה קשירה (לפי הדרוש)

נחזיק 4 עצי AVL כך ש-

1-העץ הראשון מכיל את הקבוצות לפי ID שלהם (כלומר המפתיחים

בעץ הם ID של הקבוצות) שנסמנו idTeamTree .

2-העץ השני מכיל הקבוצות שיכולות לשחק במשחקים ,כלומר קבוצות שיש בהן לפחות 11 שחקנים ואחד מהם לפחות שועיר .גם בעץ הזה הקבוצות נכנסים לפי ID ,שנסנו validTeamTree.

*כל קבוצה בעץ 1,2 מכילה שתי עצי AVL להשחקנים בקבוצה . אחד מכיל השחקנים לפי ID שנסמנו **PladyerIdTree ,והשני מכיל השחקנים לפי השערים שלהם שנסנו PlayerGoaldTree .

3-העץ השלישי מכיל את כל השחקנים במערכת לפי ID שלהם שנסנו PlayerIDTree .

4- העץ הרביעי מכיל את כל השחקנים במערכת לפי השערים שלהם שנסמנו PlayerGoalTree .

**ונחזיק רשימה מקושרת ששמור בתוכה את הקבוצות הקשירות האופן ממוין (וכל איבר בעץ validTeamTree מצביע על מקומו ברשימה)

- world_cup_t() :

מאתחלים את ארבע עצי AVL שלנו(idTeamTree , validTeamTree , PlayerGoaldTree , PlayerIDTree)להיות ריקים .
אתחול של עץ AVL ריק לוקח סיבוכיות של O(1) ולכן הפונקציה לוקחת סיבוכיות זמן ס"הכ O(1) .

- ~world_cup_t() : עוברים על ארבעת העצים ומוחקים איבר איבר ואחר כך מוחקים את העצים .
^זה לוקח סיבוכיות של סכום גודלי העצים שזה O(n+k) כך ש-n הוא מספר השחקנים במערכת ו-k הוא מספר הקבוצות במערכת .

- add_team(int teamId, int points) :

תחילה בודקים אם points <0 או teamId <=0:
-אם כן אז מחזירים INVALID_INPUT .

אם לא אז עושים חיפוש בעץ idTeamTree על קבוצה עם אותו teamId ,חיפוש זה בעץ AVL נעשה בסיבוכיות זמן של-O(log(k)) כך ש-k הוא מספר הקבוצות במערכת :
-אם מצאנו קבוצה עם מזהה teamId , מחזירים FAILURE.

אם לא מצאנו קבוצה עם מזהה teamId אז מוסיפים הקבוצה לעץ idTeamTree , הוספת איבר לעץ AVL נעשה בסיבוכיות זמן של-(O(log(k כך ש-k הוא מספר הקבוצות במערכת, ובסוף ומחזירים SUCCESS .
^נקבל סה"כ סיבוכיות זמן של הפונקציה (O(log(k כך ש-k הוא מספר הקבוצות במערכת .

- remove_team(int teamId) :
תחילה בודקים אם או teamId <=0:
-אם כן אז מחזירים INVALID_INPUT .
עושים חיפוש בעץ idTeamTree על קבוצה עם אותו teamId
(חיפוש זה בעץ AVL נעשה בסיבוכיות זמן של-(O(log(k כך ש-k הוא מספר הקבוצות במערכת) :
-אם לא מצאנו קבוצה עם מזהה teamId אז מחזירים FAILURE .

אם מצאנו קבוצה עם מזהה teamId ,בודקים אם בקבוצה זה יש שחקנים (מפעילים פונקציה שבודקת שדה numOfplayer של הקבוצה הזו וזה לוקח סיבוכיות זמן של (O(1):
-אם יש שחקנים בקבוצה אז מוחקים מה שהקצאנו ומחזירים FAILURE .

אם אין שחקנים בקבוצה אז מוחקים הקבוצה עם מזהה teamId מהעץ (idTeamTree) (מחיקת איבר בעץ AVL נעשה בסיבוכיות זמן של-(O(log(k כך ש-k הוא מספר הקבוצות במערכת) ומחזירים SUCCESS .
--עושים אותו הדבר לעץ validTeamTree(גם (O(logk)
^נקבל סה"כ סיבוכיות זמן של הפונקציה (O(log(k כך ש-k הוא מספר הקבוצות במערכת .

- add_player(int playerId, int teamId, int gamesPlayed, int goals, int cards, bool goalKeeper) :
בודקים אם card<0, goals<0, gamesPlayed<0, teamId <=0, playerId<=0 או אם gamesPlayed=0 אך cards>0, goals>0 :
-אם אחד מהנאים האלה מתקיים אז מחזירים INVALID_INPUT .

אחרת עושים חיפוש בעץ idPlayerTree על שחקן עם אותו playerId
(חיפוש זה בעץ AVL נעשה בסיבוכיות זמן של-(O(log(n כך ש-n הוא מספר השחקנים במערכת) :
-אם מצאנו שחקן עם מזהה playerId ,אז מחזירים FAILURE.

*אם לא מצאנו שחקן עם מזהה playerId אז עושים חיפוש בעץ idTeamTree על קבוצה עם אותו teamId
(חיפוש זה בעץ AVL נעשה בסיבוכיות זמן של-(O(log(k כך ש-k הוא מספר הקבוצות במערכת) :
-אם לא מצאנו קבוצה עם מזהה teamId אז מחזירים FAILURE .

אחרת אז:
1- מוסיפים השחקן שהקצאנו לעץ idPlayerTree
(הוספת איבר לעץ AVL נעשה בסיבוכיות זמן של-(O(log(n כך ש-n הוא מספר השחקנים במערכת) .
2- מוסיפים השחקן שהקצאנו לקבוצה שמצאנו ב-* ע"י הוספתו לשדה players שהוא עץ AVL ,זה לוקח (O(log n-treamID כך ש-n-teamId הוא מספר השחקנים בקבוצה עם מזהה teamId . ונבדוק אם הקבוצה הזו לא הייתה קשירה ועכשיו קשירה ע"י גישה לשדה

is_validלקבוצה, אם כן אז מוסיפים אותה ל-validTeamTree (חיפוש זה בעץ AVL נעשה בסיבוכיות זמן של-O(log(k)) כך ש-k הוא מספר הקבוצות במערכת).

3- מוסיפים השחקן שהקצאנו לעץ goalsPlayerTree
(הוספת איבר לעץ AVL נעשה בסיבוכיות זמן של-O(log(n)) כך ש-n הוא מספר השחקנים במערכת) .
ומחזירים SUCCESS .
^נקבל סה"כ סיבוכיות זמן של הפונקציה O(log(k)+log(n)) כך ש-k הוא מספר הקבוצות במערכת,n מספר השחקנים במערכת .

- remove_player(int playerId) :
בודקים אם playerId<=0 :
אם כן אז מחזירים INVALID_INPUT .
אחרת עושים חיפוש בעץ idPlayerTree על שחקן עם אותו playerId
(חיפוש זה בעץ AVL נעשה בסיבוכיות זמן של-O(log(n)) כך ש-n הוא מספר השחקנים במערכת) :
-אם לא מצאנו שחקן עם מזהה playerId , מחזירים FAILURE.
אם מצאנו השחקן אז מוחקים אותו מהעץ ומכל העצים שהוא נמצא בהן ונעדכן הקבוצה שהוא היה נמצא בעזרת השדה של השחקן שמצביע על הקבוצה שלו ב-idTeamTree ולהקבוצה שלו יש מצביע על הקבוצה עצמה ב-validTeamTree ,נעדכן אותם בהתאם O(1) (מחיקה מעצי AVL נעשה בסיבוכיות O(log(n)) כך ש-n הוא מספר השחקנים במערכת ,כי לכל היותר בכל עץ של שחקנים יש n איברים)ומחזירים SUCCESS .

^נקבל סה"כ סיבוכיות זמן של הפונקציה O(log(n)) כך ש-n הוא מספר הקבוצות במערכת.

- update_player_stats(int playerId, int gamesPlayed, int scoredGoals, int cardsReceived):
--מגדירים פונקציית עזר setPlayer שמקבלת פרמטרים מתאימים כדי לעדכן את השדות של השחקן (לוקחת O(1)).
בודקים אם playerId<=0 ,gamesPlayed<0 ,scoredGoals<0,cardsReceived<0:
-אם אחד מהנאים האלה מתקיים אז מחזירים INVALID_INPUT .
אחרת עושים חיפוש בעץ idPlayerTree על שחקן עם אותו playerId
(חיפוש זה בעץ AVL נעשה בסיבוכיות זמן של-O(log(n)) כך ש-n הוא מספר השחקנים במערכת) :
-אם לא מצאנו שחקן עם מזהה playerId , מחזירים FAILURE.
אם מצאנו השחקן אז מעדכנים השדות שלו בעזרת setPlayer ומעדכנים את המיקום שלו בעצים שממוינים לפי שערים.

^נקבל סה"כ סיבוכיות זמן של הפונקציה O(log(n)) כך ש-n הוא מספר הקבוצות במערכת.

- play_match(int teamId1, int teamId2) :

-מגדירים פונקציית עזר setTeam שמקבלת פרמטרים מתאימים כדי לעדכן את השדות של הקבוצה (לוקחת O(1)).

-מגדירים פונקציית עזר getXp שמחשבת סכום הניקוד הנובחי של הקבוצה עם סכום כל השערים שהשחקנים בה הבקיעו ,פחות מספר הכרטיסים הכולל שהשחקנים קיבלו .(לוקחת O(1)).

תחילה בודקים אם teamId==teamId2, teamId1 ,teamId2<=0 ,teamId<=0 :
-אם אחד התנאים האלה מתקיים אז מחזירים INVALID_INPUT .

אחרת אם קיימים בעץ ValidTeamTree (חיפוש זה נעשה בעץ AVL בסיבוכיות זמן של-O(log(k)) כך ש-k הוא מספר הקבוצות במערכת) :
-אם לא קיים אחד משתי הקבוצות בעץ , מחזירים FAILURE .

אם קיימות שתיהן בעץ אז בודקים הקבוצה עם ה-XP הגדול ביותר (בעזרת getXp):
-אם ה-XP של שתי הקבוצות שווים אז מוסיפים נקודה אחת ל-XP ול-points של שתי הקבוצות, וגם מוסיפים gamesplayed ב-1 .עושים זה בעזרת setTeam .
-אם ה-XP של אחד הקבוצות גדול יותר מ-XP של השנייה אז לקבוצה הזו (הגדולה יותר)מוסיפים 3 ל-XP, points ומוסיפים 1 ל-gamesplayed לשתי הקבוצות .עושים זה בעזרת setTeam.
ובסוף מחזירים SUCCESS .
^פונקציה זו לוקחת סיבוכיות זמן של-O(log(k)) כך ש-k הוא מספר הקבוצות במערכת.

- get_num_played_games(int playerId):
- מגדירים פונקציית עזר GetGamesplayed שמחזירה ערך השדה gamesPlayed של אובייקט שחקן (לוקחת O(1)).

תחילה בודקים אם playerId <=0:
-אם כן אז מחזירים INVALID_INPUT .
אחרת, אז עושים חיפוש בעץ idPlayerTree על שחקן עם אותו playerId
(חיפוש זה בעץ AVL נעשה בסיבוכיות זמן של-O(log(n)) כך ש-n הוא מספר השחקנים במערכת) :
-אם לא מצאנו שחקן עם מזהה playerId , מחזירים FAILURE .

אם מצאנו שחקן עם מזהה playerId אז מחזירים את השדה gamesPlayed (SUCCES) בעזרת GetGamesplayed .

^פונקציה זו לוקחת סיבוכיות זמן של-O(log(n)) כך ש-n הוא מספר השחקנים במערכת.

- get_team_points(int teamId) :
- מגדירים פונקציית עזר getPoints שמחזירה ערך השדה points של אובייקט team (לוקחת O(1)).

תחילה בודקים אם teamId <=0:
-אם כן אז מחזירים INVALID_INPUT .
אחרת, אז עושים חיפוש בעץ idTeamTree על קבוצה עם אותו teamId
(חיפוש זה בעץ AVL נעשה בסיבוכיות זמן של-O(log(k)) כך ש-k הוא מספר הקבוצות במערכת) :
-אם לא מצאנו קבוצה עם מזהה teamId , מחזירים FAILURE .

אם מצאנו קבוצה עם מזהה teamId אז מחזירים את השדה points (SUCCES) בעזרת getPoints .

פונקציה זו לוקחת סיבוכיות זמן של-((O(log(k כך ש-k הוא מספר הקבוצות במערכת.

- get_top_scorer(int teamId) :
--מגדירים פונקציית עזר getMaxvalue של עץ AVL שמחזירה האיבר המקסימלי בעץ (לוקחת O(1) כי מעדכנים אותה בכל הכנסת איבר לעץ).,
--מגדירים פונקציית עזר getId שמחזירה ערך השדה Id של אובייקט player (לוקחת O(1)).
--מגדירים פונקציית עזר getsize של עץ AVL שמחזירה מספר האיברים של העץ (לוקחת O(1)).,

תחילה בודקים אם teamId ==0:
-אם כן אז מחזירים INVALID_INPUT .

אם לא אז בודקים אם teamId<0 :
-אם כן אז בודקים כמה שחקנים ב- goalPlayerTree (בעזרת getsize).אם הוא 0 (כלומר אין שחקנים במערכת)אז מחזירים FAILURE.
-אם מספר השחקנים לא 0 אז מוצאים האיבר המקסימלי בעץ goalPlayerTree (שהוא השחקן עם הכי הרבה שערים) ומחזירים את השדה Id של השחקן הזה בעזרת getId (SUCCES).
^פונקציה זו לוקחת סיבוכיות זמן של-O(1) במקרה הזה .

אם teamId>0 אז עושים חיפוש בעץ idTeamTree על קבוצה עם אותו teamId (חיפוש זה בעץ AVL נעשה בסיבוכיות זמן של-((O(log(k כך ש-k הוא מספר הקבוצות במערכת) :
-אם לא מצאנו קבוצה עם מזהה teamId ,מחזירים FAILURE .

אם מצאנו קבוצה עם מזהה teamId אז בודקים אם יש שחקנים בקבוצה זו ע"י ניגשות ל-By_goals בעזרת getBy_goals ,ובודקים את מספר האיברים ב-By_goals בעזרת getsize :
-אם מספר השחקנים שווה ל-0, מחזירים FAILURE.
-אם מספרם לא-0 ,אז ניגשים ל-By_goals של הקבוצה זו(זה עץ AVL של שחקנים בקבוצה לפי שערים) ומוצאים האיבר המקסימלי ב-By_goals בעזרת getMaxvalue (האיבר הזה הוא שחקן), ומחזירים id שלו בעזרת getId.
^במצב זה נקבל סה"כ סיבוכיות זמן של הפונקציה ((O(log(k כך ש-k הוא מספר הקבוצות במערכת .

- get_all_players_count(int teamId):
--מגדירים פונקציית עזר getsize של עץ AVL שמחזירה מספר האיברים בעץ (לוקחת O(1)).,

תחילה בודקים אם teamId ==0:
-אם כן אז מחזירים INVALID_INPUT .

אם לא אז בודקים אם teamId<0 :
-אם כן אז מחזירים כמה איברים יש ב-idPLayerTree בעזרת getsize.
^ פונקציה זו לוקחת סיבוכיות זמן של-O(1) במקרה הזה.

-אם לא (teamId>0) אז עושים חיפוש בעץ idTeamTree על קבוצה עם אותו teamId
(חיפוש זה בעץ AVL נעשה בסיבוכיות זמן של-((O(log(k כך ש-k הוא מספר הקבוצות במערכת) :
-אם לא מצאנו קבוצה עם מזהה teamId מחזירים FAILURE .

אם מצאנו קבוצה עם מזהה teamId אז ניגשים ל-By_id בעזרת getBy_id ומחזירים מספר האיברים ב-By_id בעזרת getsize.
^במצב זה נקבל סה"כ סיבוכיות זמן של הפונקציה ((O(log(k כך ש-k הוא מספר הקבוצות במערכת .

- get_all_players(int teamId, int * const output) :
----מגדירים פונקציית עזר getBy_goals של קבוצה Team שמחזירה השדה By_goals של הקבוצה (לוקחת (O(1).,
--הגדרנו אטירטור של עץ AVL שעובר על האיברים בעץ בשיטת inorder כלומר באופן ממוין.((O(n כך ש-n הוא מספר האיברים בעץ)
תחילה בודקים אם teamId ==0:
-אם כן אז מחזירים INVALID_INPUT .

אם לא אז בודקים אם teamId<0 :
-אם כן אז בודקים אם יש שחקנים במערכת כמו ע"י שימוש ב-getsize כמו get_all_players_count :אם אין שחקנים במערכת אז מחזירים FAILURE .
--אם יש שחקנים במערכת אז עוברים על עץ-goalPlayerTree באיטרטור שלנו וממלים את המערך output ב-id של השחקנים שעוברים עליהם באיטרטור, ניגשים ל-id של השחקנים בעזרת getId, ואחרי שעוברים על כל העץ מחזירים SUCCESS.
^ פונקציה זו לוקחת סיבוכיות זמן של -(O(n-teamId במקרה הזה, כך ש-n-teamId הוא מספר השחקנים בקבוצה עם מזהה teamId.(כי עוברים על כל השחקנים פעם אחד)

-אם (teamId>0) אז עושים חיפוש בעץ idTeamTree על קבוצה עם אותו teamId
(חיפוש זה בעץ AVL נעשה בסיבוכיות זמן של-((O(log(k כך ש-k הוא מספר הקבוצות במערכת) :
-אם לא מצאנו קבוצה עם מזהה teamId , מחזירים FAILURE .

אם מצאנו קבוצה עם מזהה teamId אז ניגשים ל-By_goals בעזרת getBy_goals ועוברים על עץ זה באיטרטור שלנו וממלים את המערך output ב-ID של השחקנים שעוברים עליהם באיטרטור, ניגשים ל-ID של השחקנים בעזרת getId, ואחרי שעוברים על כל העץ מחזירים SUCCESS.
^פונקציה זו לוקחת סיבוכיות זמן של (O(n במקרה הזה, כך ש-n הוא מספר השחקנים במערכת

- get_closest_player(int playerId, int teamId) :
תחילה בודקים אם played<=0 או teamId<=0 :
-אם אחד מהם מתקיים אז מחזירים INVALID_INPUT .
(בודקים אם יש רק שחקן אחד במערכת בשימוש ב-getsize על idPlayerTree,אם יש רק אחד אז מחזירים FAILURE)
-אם לא אז עושים חיפוש בעץ idTeamTree על קבוצה עם אותו teamId
(חיפוש זה בעץ AVL נעשה בסיבוכיות זמן של-((O(log(k כך ש-k הוא מספר הקבוצות במערכת) :
-אם לא מצאנו קבוצה עם מזהה teamId, מחזירים FAILURE .

-אם מצאנו קבוצה עם מזהה teamId אז ניגשים לשדה By_goals של הקבוצה הזו בעזרת getBy_goals ועושים חיפוש בעץ By_goals על שחקן עם אותו playerId
(חיפוש זה בעץ AVL נעשה בסיבוכיות זמן של-((O(log(n-teamId כך ש-n-teamID הוא מספר השחקנים בקבוצה עם המזהה teamId).
ניגשים להשדות previous, next של השחקן בעזרת getNext/Previous שהם שתי השחקנים הקרובים ביותר לשחקן עם מזהה teamId ובודקים מי מהם קרוב יותר להשחקן שלנו ומחזירים את המספר המזהה שלו(מספר מזהה של השחקן הקרוב ביותר להשחקן עם playerId).
^ פונקציה זו לוקחת סיבוכיות זמן של-((O(log (k)+log(n-tealId כך ש-k מספר הקבוצות במדרכת ו-n-teamId הוא מספר השחקנים בקבוצה עם מזהה teamId.

- knockout_winner(int minTeamId, int maxTeamId) :
תחילה בודקים אם minTeamId<0,naxTeamID<0,maxTeamId<minTeamId :
-אם אחד מהם מתקיים אז מחזירים INVALID_INPUT .
-אם לא אז בודקים מספר הקבוצות הקשירות בעזרת getsize על העץ validTeamTree :
--אם מספר הקבוצות הקשירות הוא 0 אז מחזירים FAILURE.
--אחרת , עושים חיפוש בעץ validTeamTree על קבוצה עם מספר מזהה maxTeamId ואם לא מצאנו קבוצה כזו ניקח הקבוצה עם המזהה הגדול ביותר שקטן מ- maxTeamId נסמן קבוצה זו ב-Tmax .
ועושים חיפוש עוד פעם בעץ validTeamTree על קבוצה עם מספר מזהה minTeamId ואם לא מצאנו קבוצה כזו ניקח הקבוצה עם המזהה הקטן ביותר שגדול מ-minTeamId , ונסמן קבוצה זו ב-Tmin.
(חיפושים אלה בעץ AVL נעשים בסיבוכיות זמן של-(O(log(min{n,k} כך ש-k הוא מספר הקבוצות במערכת ו-n הוא מספר השחקנים במערכת-הסבר ב-*) .
Tmin ,Tmax מצביעות על מקומם ברשימה מקושרת שמכילה את הקבוצות הקשירות בסדר עולה ,נמלה שתי מערכים בגודל r בקבוצות שבין Tmax, Tmin(כולל) זה יקח סיבוכיות של r.
שתי המערכים החדשים הן Xp_array,id_array ה-Xp,id של הקבוצות שבין Tmin Tmax,(כולל) בהתאמה .
נעבור על Xp_array בשתי הקבוצות הראשונות ונבדוק איזה קבוצה יש לה Xp גדול יותר ונשמור את ה-id שלה בתא הראשון ב-id_array ונשמור גם סכום ה-Xp של שתי הקבוצות +3 בתא הראשון ב-Xp_array . נמשיך ככה על כל זוג סמוכים (אין חיתוכים) ואם בסוף יש קבוצה בדידה נשמור את ה-id ו-Xp שלה .
נעבור אותו אלגורתים על מערך Xp_array עד שנגיע לקבוצה אחד מנצחת (סיבוכיות זמן O(r)=r+r/2+r/4+r/8+...) ונחזיר את ה-id שלו הקבוצה המנצחת .

^ פונקציה זו לוקחת סיבוכיות זמן של- O(log(min{n,k})+r) כאשר k הוא מספר הקבוצות במערכת ו-n הוא מספר השחקנים במערכת ו-r מספר הקבוצות המתחרות .
*למה log(min{n,k}) :
מספר הקבוצות הקשירות kv מקיים kv<=k ,ומספר השחקנים n הוא לפחות 11*kv כי בכל קבוצה כשירה יש לפחות 11 שחקנים לכן kv<=n ולכן חיפוש ב-validTeamTree לוקח סיבוכיות זמן log(min{n,k}) .

- unite_teams(int teamId1, int teamId2, int newTeamId) :
תחילה בודקים אם teamId1<=0,teamId1==temaId2 ,teamID2<=0 ,newTeamId<=0 :
-אם אחד מהם מתקיים אז מחזירים INVALID_INPUT.
-אם לא אז עושים חיפוש בעץ idTeamTree על קבוצות עם אותו teamId1,teamId2 ,כלומר עושים חיפוש פעמיים

(חיפוש זה בעץ AVL נעשה בסיבוכיות זמן של-(O(log(k כך ש-k הוא מספר הקבוצות במערכת) :
- אם לא מצאנו קבוצות עם מזהים teamId2,teamId1 אז ,מחזירים FAILURE .
-אחרת עוברים על By_id של כל אחד מהקבוצות בסיור inorder (באמצעות האיטרטור שלנו) וממלאים שני מערכים באיברים של שתי הקבוצות בהתאמה (זה לוקח O(n-teamId1+n-teamId2) כך ש-n-teamIdi הוא גודל הקבוצה ה-i),ועושים merge לשתי המערכים למערך אחד (משום שהמערכים ממוינים אז זה לוקח O(n-teamId1+n-teamId2) כך ש-n-teamIdi הוא גודל הקבוצה ה-i).עוברים על המערך החדש שלנו וממלים את העץ By_value של newTeamId כמו שלמדנו בהרצאה\תרגול (גם סיבוכיות של O(n-teamId1+n-teamId2) .עושים אותו הדבר ל-By_goals .מעדכנים השדות של newTeamId כמו שדרוש (O(1) באמצעות פונקציה updateTeamstats). מוחקים שתי הקבוצות מהעצים שלנו ואחר כך מכניסים את הקבוצה החדשה(O(logk), ומחזירים SUCCESS .
^ פונקציה זו לוקחת סיבוכיות זמן של-(O(n-teamId1+n-teamId2+log(k כך ש-n-teamIdi הוא גודל הקבוצה ה-i ו-k מספר הקבוצות במערכת .

^^^סיבוכיות מקום (k הוא מספר הקבוצות במערכת ו-n הוא מספר השחקנים במערכת.):

שתי עצי שחקנים זה 2n מקום ועוד שתי עצי קבוצות זה 2k מקום ורשימה מקושרת של קבוצות זה k מקום ובכל עץ של קבוצות נשמרים השחקנים בכל קבוצה כלומר זה 2n מקום.
ולכן ס"הכ נקבל O(n+k) כאשר k הוא מספר הקבוצות במערכת ו-n הוא מספר השחקנים במערכת.