מבני נתונים 1

234218

תרגיל רטוב 2

מספר

הוגש ע"י:

עמרי בן ארי	209540483
שם	מספר זהות
גל שור	208253559
שם	מספר זהות

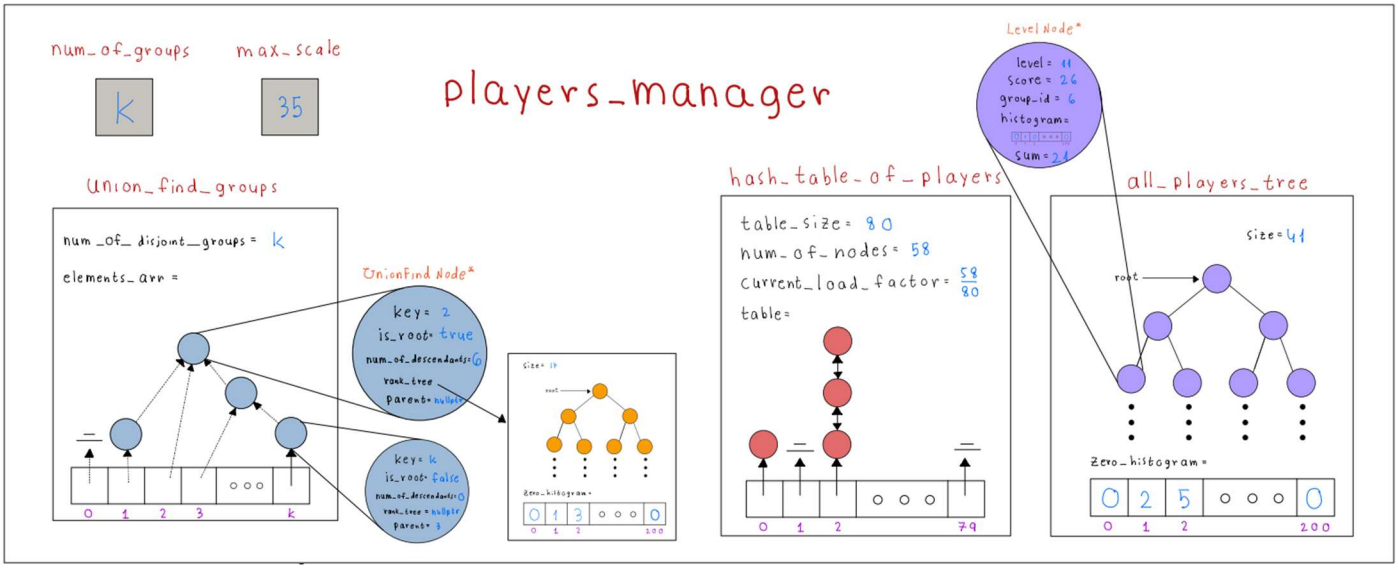
ציון:

לפני בונוס הדפסה:

כולל בונוס הדפסה:

נא להחזיר לתא מס':

חלק א' – הצגת מבנה הנתונים באמצעות ציור



חלק ב' – הצגת רכיבי מבנה הנתונים באופן מילולי

נכנה את מבנה הנתונים שלנו בשם PlayersManager.
על מנת להבין את השדות המרכזיים של המבנה יש לציין תחילה כמה מחלקות עזר בהן השתמשנו:

1. **Node**: מחלקה זו מייצגת צומת כללי בעץ ומאגדת בתוכה את השדות שרלוונטיים לכלל סוגי הצמתים והם: הערך השמור בצומת, גובה הצומת, גורם האיזון של הצומת, ומצביעים לבן השמאלי הימני והאבא של הצומת.
2. **LevelNode** – צומת היורשת מ- Node ומשמשת לתיאור שחקן במבנה שלנו. היא מכילה בנוסף לשדות הכלליים של Node גם שדות הרלוונטיים לשחקן בלבד והם הרמה (level) שלו, הניקוד (score) שלו, ID של הקבוצה אליה משתייך השחקן ושתי שדות נוספים עליהם נרחיב מיד. חשוב לציין כי יחס הסדר המוגדר על שני צמתים מטיפוס זה נקבע קודם כל לפי הרמה שלהם (הגדול יותר הוא בעל הרמה הגבוהה יותר) ורק אז לפי ה-id שלהם (הגדול יותר הוא בעל ה-id הגדול יותר).
3. **RankTree**: מחלקה המתארת את מבנה הנתונים עץ דרגות שנלמד בהרצאות ובתרגולים. מחלקה זו יורשת מהמחלקה AVLTree (המתארת עץ דרגות מאוזן ואותה מימשנו בתרגיל הבית הקודם) ועל כן כל עץ דרגות בו נעשה שימוש במבנה הנתונים שלנו הוא עץ דרגות מאוזן. צמתי העץ הם מטיפוס LevelNode.

מספר פרטים חשובים אודות המחלקה:

- בכל LevelNode (שיסומן ב-) v בעץ שמורים שתי שדות נוספים (המאפשרים לו לתפקד כעץ דרגות) והם:
  - היסטוגרמה של scores –היסטוגרמה בגודל 201. ערכה המינימלי הוא 0 כערך ה- score המינימלי וערכה המקסימלי הוא 200 כערך ה- score המקסימלי. בהינתן i בין 0 ל- 200 ערכו של האיבר ה- i בהיסטוגרמה שווה למספר השחקנים בעלי הניקוד i בתת העץ ששורשו v (כולל ה- score של v).
    - נעיר כי אם נסמן ב- max_scale את ה- score המקסימלי של שחקן במבנה אז מספרי הרשומות בהיסטוגרמה שאותן ייתכן ונערוך ינוע בין 1 (כולל) ל- max_scale (כולל).
  - Sum – ערך מספרי השווה לסכום כל ה- levels בתת העץ ששורשו v (כולל ה- level של v).
- הבחנה: לא קיימים שחקנים בעלי level ששווה ל-0 בעץ. הדבר נעשה על מנת לעמוד בדרישות הסיבוכיות שעליהם נפרט בהמשך.
- בעץ קיים שדה בשם zero_histogram – מדובר במערך בגודל 201 בעל ייעוד דומה להיסטוגרמה של ה-scores השמורה בכל צומת. בהיסטוגרמה זו, בהינתן i בין 0 ל- 200 ערכו של האיבר ה- i בהיסטוגרמה שווה למספר השחקנים **ברמה 0** בעלי הניקוד i **בעץ כולו**. שני ההבדלים העיקריים בין היסטוגרמה זו להיסטוגרמה השמורה בכל צומת הן:
  - Zero_histogram היא היסטוגרמה של כל העץ (ולא של צומת בודדת).
  - Zero_histogram מכילה רק scores של שחקנים בעלי level=0 ואילו שאר ההיסטוגרמות מכילות scores של שחקנים בעלי level>0.

4. **HashTable**- מחלקה המתארת טבלת ערבול מסוג chain hashing שנלמד בהרצאות ובתרגולים.
מספר פרטים שחשוב לציין אודות המחלקה:
   a. מדובר בטבלת ערבול דינאמית ולכן מספר האיברים בה גדל/ קטן בהתאם לפקטור עומס.

b. פונקציית הערבול היא הפונקציה modk כאשר k הוא מספר האיברים בטבלה ברגע נתון. זו פונקציה שמתאימה לתרגיל מכיוון ודרישות הסיבוכיות הן בממוצע על הקלט ולא בממוצע הסתברותי ולכן אין צורך בפונקציית ערבול אוניברסלית.
c. כל תא בטבלת הערבול הוא מצביע לרשימה מקושרת שצמתיה הם מטיפוס *LevelNode.
d. המפתחות לפיהם אנחנו מבצעים את פעולות טבלת הערבול (כדוגמת הכנסה/ חיפוש/ הוצאה של איברים) הם ה-ID של השחקן.

5. **UnionFind** – מחלקה המתארת מימוש לפתרון בעיית UnionFind באמצעות עץ הפוך בדומה לנלמד בהרצאות בתרגולים. המחלקה מחזיקה בתוכה מערך של איברים שממפה בין איבר לצומת המתאים לו בעץ ההפוך. כל צומת בעץ ההפוך הוא מטיפוס **UnionFindNode**. מימשנו צומת זו כך שתכיל את השדות הבאים: הערך (key) של הצומת, מצביע לאבא (parent) של הצומת, מספר הצאצאים של הצומת, מצביע ל- RankTree ומשתנה שתפקידו להגיד האם צומת זו היא שורש בעץ ההפוך או לא. נעיר כי במימוש שבחרנו אנחנו מבצעים איחוד קבוצות זרות לפי גודל וכיווץ מסלולים במהלך ביצוע הפעולה find וזאת במטרה לעמוד בדרישות הסיבוכיות של התרגיל.

כעת נוכל להציג את השדות במבנה הנתונים שלנו, **השדות המרכיבים את המבנה שלנו הם**:

- **all_players_tree *RankTree:** מצביע לעץ דרגות המאגד את כלל השחקנים במבנה. כל צומת בעץ מייצגת שחקן. נציין שלא כל שחקן מיוצג על ידי צומת בעץ מכיוון שיש את מערך ה- zero_histogram שסופר שחקנים עם level=0.

- **union_find_groups *UnionFind:** מצביע למבנה הנתונים UnionFind שמומש כפי שתואר לעיל. כל איבר במבנה מייצג קבוצה בעלת מזהה group_id. כל קבוצה זרה במבנה מיוצגת על ידי שורש יחיד בעץ ההפוך.

נשים לב כי כל קבוצה במבנה מיוצגת על ידי צומת מטיפוס **UnionFindNode** ולכן מכילה מצביע לעץ דרגות. עץ דרגות זה מכיל את כל השחקנים שנמצאים בקבוצה זו ונשמר בשורש של העץ ההפוך של קבוצה זו. בנוסף, מהאופן בו מימשנו את המבנה, לאחר איחוד קבוצות, גישה לאחת מן הקבוצות שאוחדו שקולה לגישה לקבוצה המאוחדת.

- **hash_table_of_players *HashTable:** מצביע לטבלת ערבול כפי שתוארה לעיל המאחסנת את כלל השחקנים במשחק.

- **int num_of_groups**: שדה השומר את מספר הקבוצות במשחק ומתקבל כקלט בעת אתחול מבנה הנתונים.

- **int score_scale:** שדה השומר את ה-score המקסימלי ששחקן יכול לקבל ומתקבל כקלט בעת אתחול מבנה הנתונים.

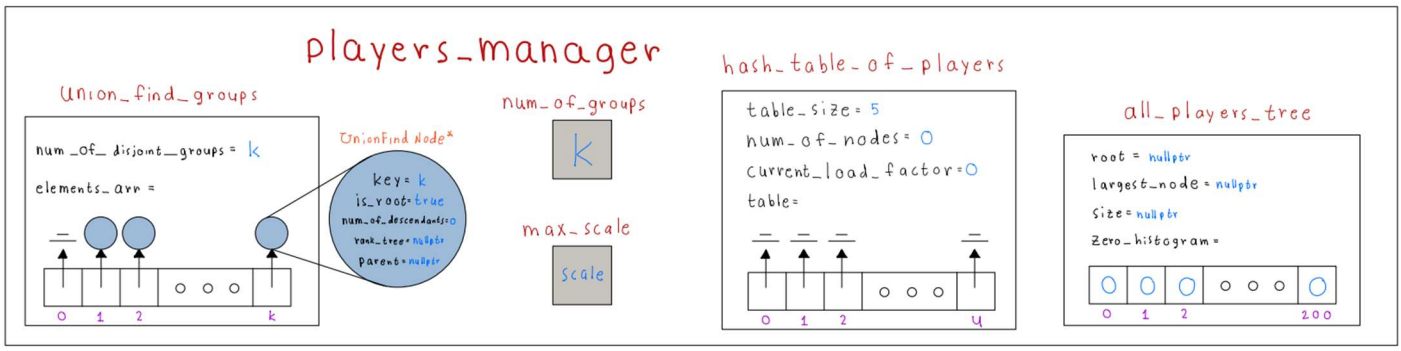
חלק ג' – הצגת הפעולות במבנה הנתונים

1) **הפעולה:**

void* Init()

- **מטרה**: מאתחל מבנה נתונים עם k קבוצות, כל קבוצה מוגדרת להיות ריקה בהתחלה. Scale מתאר את התוצאה המקסימלית במשחק עבור שחקן מסוים.
- **דרישות סיבוכיות זמן**: O(k) במקרה הגרוע.
- **מימוש**:
נתאר את עיקרי המימוש בשלבים:
  - בדיקה שהקלט שקיבלנו עומד בדרישות הקלט והחזרת ערך מתאים אם לא – **O(1)**.
  - קריאה לבנאי של המחלקה PlayersManager.
בנאי זה אחראי להקצאת זיכרון עבור עץ השחקנים, טבלת הערבול ומבנה ה-UnionFind שצויינו קודם.
במצב ההתחלתי אנחנו מאתחלים את השדות בהתאם לנראה בציור למטה.
נעיר כמה הערות על האתחול:
  - all_players_tree – מאותחל להיות עץ ריק עם שדות כפי שניתן לראות בציור. פעולת האתחול של העץ מתבצעת ב- O(1) כי גודלו של ה- zero_histogram קבוע ושווה ל-201. אתחול שאר השדות בעץ גם כן מבוצע ב- O(1) ולכן אתחול העץ נעשה ב- O(1).
  - union_find_groups – נקצה מערך של מצביעים לצמתים מסוג UnionFindNode בגודל k+1. ולאחר מכן לכל i בין 1 ל-k נקצה באיבר ה-i מצביע לצומת חדשה מטיפוס UnionFindNode שמאותחלת עם group_id = i. שדות כל צומת מאותחלים ב-O(1). ולכן השדה מאותחל בסיבוכיות O(k).
  - hash_table_of_players –נקצה מערך של *Node בגודל התחלתי קבוע לתיאור טבלת הערבול. מכיוון ומדובר בגודל קבוע ללא תלות בקלט אז פעולת ההקצאה והאתחול (ל-nullptr) יבוצעו ב-O(1). את שאר השדות נעדכן בהתאם למופיע בציור. לכן אתחול טבלת הערבול מבוצע בסיבוכיות זמן של O(1).
- **מצב מבנה הנתונים לאחר ביצוע הפעולה**: המבנה מכיל טבלת ערבול ריקה, עץ דרגות ריק משחקנים ומבנה UnionFind שאותחל עם k קבוצות זרות.

- **מדוע אנחנו עומדים בסיבוכיות הזמן הדרושה**: פעולת ה-Init מורכבת מ-3 פעולות אתחול מרכזיות כפי שהצגנו קודם לכן שמתבצעות בסיבוכיות זמן כוללת של O(k), שאר הפעולות בפונקציה מבוצעות ב-O(1) ולכן סיבוכיות הזמן הכוללת היא O(k).



2) **הפעולה:**

StatusType mergeGroups(void *DS, int GroupID1, int GroupID2)

- **מטרה**: הפעולה מאחדת בין שתי הקבוצות עם המזהים, GroupID1, GroupID2 כל השחקנים משתי הקבוצות עוברים להיות בקבוצה החדשה. לאחר איחוד שתי הקבוצות, שני המזהים GroupID1, GroupID2 יתייחסו לקבוצה המאוחדת.
- **דרישות סיבוכיות זמן**: O(log* (k) +n) משוערך, בממוצע על הקלט, כאשר k הוא מספר הקבוצות ו-n מספר השחקנים בשתי הקבוצות המאוחדות יחד.
- **מימוש**: נרצה לאחד את הקבוצות הזרות אליהן שייכות הקבוצות בעלות המזהים group_id1 ו- group_id2 במבנה UnionFind על ידי שימוש בפונקציות find ו-union. נבצע את הפעולות הבאות:
  - בדיקת תקינות הקלט לפונקציה ב- O(1).
  - קריאה לפונקציה union של union_find_groups על מנת לאחד את הקבוצות הזרות אליהן משתייכות הקבוצות בעלות המזהים group_id1 ו- group_id2. פעולה זה מומשה בהתאם לנלמד בהרצאות ובתרגולים ובנוסף על ידי שימוש בפעולה find פעמיים למציאת השורש של הקבוצה המאוחדת. נשים לב כי כחלק מפעולת ה- union ולאחר מציאת עצי הדרגות המצויים בשורשים של הקבוצות המאוחדות נרצה לבצע איחוד של עצים. פעולת איחוד עצים מבוצעת בהתאם לנלמד בתרגול תוך שאנחנו מקפידים לתקן את המזהים הקשורים בדרגת כל צומת במקביל לתיקון הגובה וגורם האיזון של כל אחד מהצמתים. לכן סיבוכיות הזמן של פעולה זו זהה למימוש הנלמד בתרגול ומבוצעת בסיבוכיות O(n) כאשר n הינו מספר השחקנים בשתי הקבוצות המאוחדות יחד.
- **מצב מבנה הנתונים לאחר ביצוע הפעולה**: השדה היחידי במבנה שבו חל שינוי לאחר השימוש בפעולה הוא union_find_groups. שינוי זה מתבטא בכמה רבדים: השורש של אחת הקבוצות הקודמות כעת מצביע לשורש השני שמהווה את השורש של הקבוצה המאוחדת החדשה בעץ. ישנו עץ אחד ומאוחד של כל הקבוצות בתת העץ ששמור בשורש. נשים לב גם כי אנחנו עושים שימוש בפעולה find כחלק מה-union למציאת השורשים של כל אחת מהקבוצות ולכן ביצענו כיווץ מסלולים בעץ ההפוך כחלק מתהליך האיחוד.
- **מדוע אנחנו עומדים בסיבוכיות הזמן הדרושה**: הסיבוכיות של פעולה זו היא O(log* (k) +n) משוערך עם שאר הפעולות במבנה ועל כך נפרט בהרחבה לאחר שנציג את שאר הפעולות במבנה. נציין כי סיבוכיות הזמן שנובעת ממספר השחקנים במשחק היא O(n) כי אנו מבצעים איחוד של עצי דרגות לפי האלגוריתם שנלמד בתרגול, נדגיש שסיבוכיות זו אינה ממוצעת ואינה משוערכת אלא סיבוכיות במקרה הגרוע. שאר פעולות איחוד הקבוצות, שינוי המזהים והשדות הפנימיים של הקבוצות והחלפת המצביעים (כל אלה בדומה לנלמד בתרגול) מבוצעים ב- O(1).

3) **הפעולה:**

StatusType AddPlayer(void *DS, int PlayerID, int GroupID, int Score)

- **מטרה**: הוספת שחקן חדש שמתחיל את המשחק עם level=0 במשחק, עם תוצאה התחלתית score ומשתייך לקבוצה בעל המזהה GroupID.
- **דרישות סיבוכיות זמן**: O(log* k) משוערך, בממוצע על הקלט, כאשר k הוא מספר הקבוצות.
- **מימוש**: נרצה לעדכן את שלושת השדות העיקריים במבנה כי יש שחקן חדש. נבצע את הפעולות הבאות:
  - נוודא את תקינות הקלט, ובפרט שלא קיים שחקן עם id כזה על ידי חיפוש ב- HashTable ב-O(1) בממוצע על הקלט ומשערך כפי שלמדנו בהרצאה.
  - ניצור שחקן חדש עם השדות שקיבלנו בקלט ולאחר מכן נכניס את השחקן ל- HashTable בתא המתאים לפי פונקציית ה-hash. בסך הכול פעולות אלה מבוצעות בסיבוכיות O(1) משוערך בממוצע על הקלט.
  - נעדכן את ההיסטוגרמה ב- all_players_tree, משמע נוסיף שחקן חדש לעץ עם level=0 עם ה- score הנתון. סיבוכיות O(1).
  - מציאת השורש של הקבוצה GroupID ב- UnionFind בסיבוכיות O(log*k) משוערך. ואז עדכון ההיסטוגרמה בעץ הדרגות הפנימי שבשורש בסיבוכיות O(1).

- **מצב מבנה הנתונים לאחר ביצוע הפעולה**: ב- HashTable נוסף שחקן חדש עם הנתונים המתאימים לקלט. ההיסטוגרמה zero_histogram בעץ all_players_tree ובעץ הפנימי המתאים לקבוצה GroupID ב- UnionFind עודכנו כי קיים שחקן נוסף במערכת עם level=0 וה- score הנתון.
- **מדוע אנחנו עומדים בסיבוכיות הזמן הדרושה**: הסיבוכיות של פעולה זו היא O(log*k) בממוצע על הקלט ומשוערך עם שאר הפעולות במבנה ועל כך נפרט בהרחבה לאחר שנציג את שאר הפעולות במבנה. נציין כי בנוסף סיבוכיות הזמן הנובעת ממספר השחקנים היא O(1) בממוצע על הקלט ומשוערך ונפרט על כך בהמשך.

4) **הפעולה:**

StatusType RemovePlayer(void *DS, int PlayerID)

- **מטרה**: השחקן בעל המזהה PlayerID "נפסל" מהמשחק, וניתן למחוק אותו מהמערכת.
- **דרישות סיבוכיות זמן**: O(log*k+log n) משוערך, בממוצע על הקלט, כאשר k זה מספר הקבוצות ו-n הוא מספר השחקנים הכולל במשחק כרגע.
- **מימוש**: בעת הסרת שחקן מהמערכת נרצה להסיר את כל המופעים שלו משלושת השדות במבנה הנתונים שלנו. נתאר את עיקרי המימוש בשלבים:
  - נוודא את תקינות הקלט ובפרט שקיים שחקן עם ה- PlayerID הנתון על ידי חיפוש ב- HashTable בסיבוכיות של O(1) בממוצע על הקלט ובמשוערך כפי שלמדנו בהרצאה.
  - נסיר את השחקן שמצאנו בסעיף הקודם מה- HashTable בעזרת פעולת ה- remove כפי שנלמדה בתרגול. סיבוכיות של O(1) בממוצע על הקלט ובמשוערך.
  - ניקח את מספר הקבוצה אליה משתייך השחקן ונחפש את שורש קבוצה זו באמצעות שימוש בפעולה find ב- UnionFind. סיבוכיות O(log*k) משוערך.
  - נסיר את השחקן מן העץ הפנימי בצומת המתאימה ב- UnionFind כאשר יש לבצע הפרדה למקרים:
    - אם level==0 של השחקן אז יש לבצע --zero_histogram[score] מכיוון שהצומת לא קיימת עדיין בעץ (בעץ יש אך ורק שחקנים עם level>0). סיבוכיות O(1).
    - אחרת נבצע אלגוריתם הסרה מעץ דרגות כפי שנלמד בתרגול. סיבוכיות $O(\log n_k)$ כאשר n_k הוא מספר השחקנים בקבוצה.
  - נסיר את השחקן מן העץ all_players_tree באמצעות אותו אלגוריתם והפרדה למקרים שתוארה בסעיף הקודם. סיבוכיות O(1) אם level=0 וסיבוכיות O(log n) אם level>0.
- **מצב מבנה הנתונים לאחר ביצוע הפעולה**: מבנה הנתונים שלנו לא מכיל מידע אודות השחקן עם ה-ID הנתון כלומר: ה- HashTable אינו מכיל יותר את השחקן הדרוש, והעצים all_players_tree והעץ הפנימי המתאים לקבוצה של השחקן אינם מכילים מידע אודות אותה שחקן כי הוסר מכל המחלקות האלו.
- **מדוע אנחנו עומדים בסיבוכיות הזמן הדרושה**: הסיבוכיות של פעולה זו היא O(log*k + log n) בממוצע על הקלט ומשוערך עם שאר הפעולות במבנה ועל כך נפרט בהרחבה לאחר שנציג את שאר הפעולות במבנה. נוסיף כי הסיבוכיות הנובעת ממספר השחקנים במשחק היא O(log n) עבור פעולת ההסרה בשני העצים, גודלם חסום על ידי n. זוהי סיבוכיות שאינה משוערכת ונכונה עבור המקרה הגרוע. בנוסף ישנה סיבוכיות משוערכת בממוצע על הקלט O(1) שנובעת משימוש בטבלת הערבול, נפרט על כך בהמשך.

5) **הפעולה:**

StatusType increasePlayerIDLevel (void *DS, int PlayerID, int LevelIncrease)

- **מטרה**: הגדלת השלב במשחק של השחקן בעל המזהה PlayerID ב- LevelIncrease.
- **דרישות סיבוכיות זמן**: O(log*k+log n) משוערך, בממוצע על הקלט, כאשר k זה מספר הקבוצות ו-n הוא מספר השחקנים הכולל במשחק כרגע.
- **מימוש**: נתאר את עיקרי המימוש בשלבים:
  - נוודא את תקינות הקלט ובפרט שקיים שחקן עם ה- PlayerID הנתון על ידי חיפוש ב- HashTable בסיבוכיות של O(1) בממוצע על הקלט ובמשוערך כפי שלמדנו בהרצאה.
  - נרצה תחילה להסיר את כל המופעים של השחקן מהמבנה בדומה לנעשה בפעולה remove. נשים לב שפעולה זו מופרדת למקרים לפי level==0. כפי שראינו קודם פעולה זו מבוצעת בסיבוכיות O(log*k + log n) בממוצע על הקלט ומשוערך עם שאר הפעולות במבנה.
  - נעדכן את ה- level של השחקן ב- O(1).
  - נבצע הכנסה של השחקן לעץ הפנימי של הקבוצה לפי אלגוריתם הכנסה של עץ דרגות שנלמד בתרגול. סיבוכיות $O(\log n_k)$ כאשר n_k הוא מספר השחקנים בקבוצה.
  - נבצע הכנסה של השחקן לעץ all_players_tree לפי אלגוריתם הכנסה של עץ דרגות שנלמד בתרגול. סיבוכיות O(log n).
- **מצב מבנה הנתונים לאחר ביצוע הפעולה**: לשחקן עם ה-id הנתון יש Level מעודכן. אם לפני העדכון level==0 אז לאחר פעולה זו הוספנו לעץ all_players_tree והעץ הפנימי של הקבוצה המתאימה צומת חדשה עם שחקן זה. אם level>0 ביצענו עדכון של העצים שכן שינוי הערך של ה-level משנה את מיקום השחקן בעץ.

- **מדוע אנחנו עומדים בסיבוכיות הזמן הדרושה**: הסיבוכיות של פעולה זו היא O(log*k + log n) בממוצע על הקלט ומשוערך עם שאר הפעולות במבנה ועל כך נפרט בהרחבה לאחר שנציג את שאר הפעולות במבנה. נוסיף כי הסיבוכיות הנובעת ממספר השחקנים במשחק היא O(log n) עבור פעולת ההסרה וההכנסה בשני העצים, שגודלם חסום על ידי n. זוהי סיבוכיות שאינה משוערכת ונכונה עבור המקרה הגרוע. בנוסף ישנה סיבוכיות משוערכת בממוצע על הקלט O(1) שנובעת משימוש בטבלת הערבול, נפרט על כך בהמשך.
- **שאלת בונוס**: תארו בחלק היבש כיצד ניתן היה לממש את הפעולה הנ"ל בסיבוכיות של O(log*k + log n) משוערך (שימו לב שכאן הסיבוכיות אינה בממוצע על הקלט).
- **פתרון:** כדי לבצע פעולה זו באותה סיבוכיות אבל במקרה הכי גרוע, ניאלץ לוותר על ה- HashTable בפעולה זו ובמקום זאת לבצע חיפוש של השחקן ב- all_players_tree בסיבוכיות של O(log n) במקרה הגרוע. כעת נוכל לדעת מהי הקבוצה אליה שייך השחקן ולחפש את הקבוצה ב- UnionFind בסיבוכיות של O(log*k) משוערך. מכאן האלגוריתם זהה לחלוטין לאלגוריתם הקודם (נפריד למקרים לפי level=0 ונבצע את הפעולות הכתובות לעיל בהתאם) רק שהפעם אין שימוש ב- HashTable לכן סיבוכיות הזמן היא לכל היותר O(log*k + log n) משוערך במקרה הגרוע ולא בממוצע על הקלט.

6) **הפעולה:**

StatusType changePlayerIDScore (void *DS, int PlayerID, int NewScore)

- **מטרה**: שינוי התוצאה של השחקן בעל המזהה PlayerID ל- NewScore.
- **דרישות סיבוכיות זמן**: O(log*k+log n) משוערך, בממוצע על הקלט, כאשר k זה מספר הקבוצות ו-n הוא מספר השחקנים הכולל במשחק כרגע.
- **מימוש**: נתאר את עיקרי המימוש בשלבים:
  - נוודא את תקינות הקלט ובפרט שקיים שחקן עם ה- PlayerID הנתון על ידי חיפוש ב- HashTable בסיבוכיות של O(1) בממוצע על הקלט ובמשוערך כפי שלמדנו בהרצאה.
  - נרצה תחילה להסיר את כל המופעים של השחקן מהמבנה בדומה לנעשה בפעולה remove. נשים לב שפעולה זו מופרדת למקרים לפי level==0. כפי שראינו קודם פעולה זו מבוצעת בסיבוכיות O(log*k + log n) בממוצע על הקלט ומשוערך עם שאר הפעולות במבנה.
  - נעדכן את ה- score של השחקן ב- O(1).
  - נפריד למקרים:
    - אם level==0 נעדכן את ההיסטוגרמות של העצים בהתאם.
    - אחרת:
      - נבצע הכנסה של השחקן לעץ הפנימי של הקבוצה לפי אלגוריתם הכנסה של עץ דרגות שנלמד בתרגול. סיבוכיות $O(\log n_k)$ כאשר n_k הוא מספר השחקנים בקבוצה.
      - נבצע הכנסה של השחקן לעץ all_players_tree לפי אלגוריתם הכנסה של עץ דרגות שנלמד בתרגול. סיבוכיות O(log n).
- **מצב מבנה הנתונים לאחר ביצוע הפעולה**: עדכנו את הscore של השחקן עם ה-id הנתון בכל המופעים שלו במבנה. אם level = 0 אז עידכנו את ההיסטוגרמות של העץ all_players_tree ושל העץ הפנימי בקבוצה המתאימה. אם level>0 אז צריך לעדכן את דרגות העצים בהתאם במסלול ההסרה וההכנסה.
- **מדוע אנחנו עומדים בסיבוכיות הזמן הדרושה**: : הסיבוכיות של פעולה זו היא O(log*k + log n) בממוצע על הקלט ומשוערך עם שאר הפעולות במבנה ועל כך נפרט בהרחבה לאחר שנציג את שאר הפעולות במבנה. נוסיף כי הסיבוכיות הנובעת ממספר השחקנים במשחק היא O(log n) עבור פעולת ההסרה וההכנסה בשני העצים, גודלם חסום על ידי n. זוהי סיבוכיות שאינה משוערכת ונכונה עבור המקרה הגרוע. בנוסף ישנה סיבוכיות משוערכת בממוצע על הקלט O(1) שנובעת משימוש בטבלת הערבול, נפרט על כך בהמשך.

7) **הפעולה:**

StatusType getPercentOfPlayersWithScoreInBounds(void *DS, int GroupID, int score, int lowerLevel, int higherLevel, double* players)

- **מטרה**: הפעולה מחשבת את אחוז השחקנים בעלי תוצאה השווה בדיוק ל score- מבין השחקנים שנמצאים ברמה שהינה בטווח [higherLevel, lowerLevel] בקבוצה עם המזהה GroupID, אם $GroupID == 0$ אז נבצע את הפעולה על כל שחקני המשחק.
- **דרישות סיבוכיות זמן**: O(log*k+log n) משוערך, כאשר k זה מספר הקבוצות ו-n הוא מספר השחקנים הכולל במשחק כרגע.
- **מימוש**: נתאר את עיקרי המימוש בשלבים:
  - נבדוק את תקינות הקלט ב- O(1).
  - נמצא את העץ המתאים עליו נרצה לעבוד בהתאם לקלט שהוכנס (all_players_tree או עץ דרגות פנימי של קבוצה ב- UnionFind). אם GroupID > 0 אז הסיבוכיות זמן היא סיבוכיות O(log*k) משוערך כי צריך למצוא את הקבוצה המתאימה בשביל לשלוף את העץ ואם GroupID=0 אז הסיבוכיות היא O(1).

- אם 0 נמצא בתחום השלבים הנתון אז ניגש ל- zero_histogram של העץ וניקח את הנתונים הרלוונטיים על השחקנים (סכום ההיסטוגרמה והרשומה במיקום score). סיבוכיות O(1) כי גודל ההיסטוגרמה קבוע.
- כעת נבצע פעולת scoreRank על higherLevel כדי לקבל את כל השחקנים עם ה- score הנתון שהשלב שלהם קטן או שווה ל- higherLevel. נחסיר מתוצאה זו את scoreRank על lowerLevel כדי לקבל בדיוק את כל השחקנים עם ה- score הנתון בתחום השלבים הנתון. פעולה scoreRank היא פעולת rank כפי שנלמד בתרגול שעובדת על extra_data שנמצא בהיסטוגרמה באינדקס score. סיבוכיות O(log n) בגלל שפעולת rank תלויה בגובה העץ כפי שנלמד בתרגול.
- נבצע פעולת nodeRank על higherLevel כדי לקבל את כל השחקנים שהשלב שלהם קטן או שווה ל- higherLevel. נחסיר מתוצאה זו את nodeRank על lowerLevel כדי לקבל בדיוק את כל השחקנים בתחום השלבים הנתון. הפעולה nodeRank היא פעולת rank כפי שנלמדה בתרגול שעובדת על extra_data שבמקרה זה הוא סכום כל ההיסטוגרמה בצומת מסוים ששווה למספר השחקנים מכל score בתחום. סיבוכיות O(log n) בגלל שפעולת rank תלויה בגובה העץ כפי שנלמד בתרגול וגודל ההיסטוגרמה קבוע.
- נבדוק אם total_rank > 0, אם לא אז נחזיר שגיאה ואם כן אז נחזיר 100*(score_rank/total_rank).

- **מצב מבנה הנתונים לאחר ביצוע הפעולה** : אין כל שינוי.
- **מדוע אנחנו עומדים בסיבוכיות הזמן הדרושה** : הסיבוכיות של פעולה זו היא O(log*k + log n) משוערך עם שאר הפעולות במבנה ועל כך נפרט בהרחבה לאחר שנציג את שאר הפעולות במבנה. נציין כי הסיבוכיות שנובעת ממספר השחקנים במשחק היא log(n) מכיוון שאנו מבצעים מספר קבוע של פעולות ה- rank כל אחת בסיבוכיות log(n) עבור עצים מאוזנים.

8) **הפעולה:**

StatusType averageHighestPlayerLevelByGroup (void *DS, int *GroupID*, int m, double *avgLevel)

- **מטרה** : הפעולה תחשב את הרמה הממוצעת בה נמצאים m השחקנים ברמות הגבוהות ביותר בקבוצה GroupID, אם $GroupID == 0$ הפעולה תחזיר את הרמה הממוצעת של m השחקנים ברמות הגבוהות ביותר במשחק.
- **דרישות סיבוכיות זמן** : O(log*k+log n) משוערך, כאשר k זה מספר הקבוצות ו-n הוא מספר השחקנים הכולל במשחק כרגע.
- **מימוש** : נתאר את עיקרי המימוש בשלבים :
  - ראשית נבדוק את תקינות הקלט ב- O(1).
  - נמצא את העץ המתאים עליו נרצה לעבוד בהתאם לקלט שהוכנס (all_players_tree או עץ דרגות פנימי של קבוצה ב- UnionFind). אם GroupID > 0 אז הסיבוכיות זמן היא סיבוכיות O(log*k) משוערך כי צריך למצוא את הקבוצה המתאימה בשביל לשלוף את העץ ואם GroupID=0 אז הסיבוכיות היא O(1).
  - כעת נבדוק אם מספר השחקנים בקבוצה גדול או שווה ל-m, על ידי סכימת גודל העץ עם הסכום של כל ההיסטוגרמה. אם מספר השחקנים קטן יותר נחזיר שגיאה, אחרת נמשיך. סיבוכיות O(1) כי גודל ההיסטוגרמה קבוע.
  - נבצע את פעולת ה- sumRank על הצומת הגדולה ביותר בעץ כדי לקבל את סכום כל השלבים בעץ ונחסיר מערך זה את sumRank על הצומת במיקום max{n-m,0} כאשר n מספר הצמתים בעץ כדי לקבל את סכום כל השלבים של M השחקנים ברמות הגבוהות ביותר. פעולת sumRank היא פעולת ה-rank שהוגדרה בתרגול שעובדת על extra_data של sum (הסוכם את כל ה-levels בתת העץ כולל השורש של אותו תת עץ). הפעולה select(r) היא אותה פעולת select שהוגרה בתרגול שמחזירה את הצומת במיקום ה- r בדרגתה. מכיוון שפעולות rank ו- select הן בסיבוכיות O(log n) כפי שנלמד בתרגול הסיבוכיות הכוללת היא O(log n) כי פעולות אלו מתבצעות מספר סופי של פעמים ושאר הפעולות ב- O(1) שכן ההיסטוגרמה בגודל קבוע.
  - נשים לב כי יכול להיות ש-m גדול ממספר הצמתים בעץ וזה אומר שנסכום את כל הצמתים בעץ ואילו שאר הצמתים שיש לסכום הם בהכרח עם level=0 כלומר לא תורמים בכלל לסכום.
  - נחלק את הסכום שקיבלנו ב- m ונקבל את הממוצע הדרוש. סיבוכיות O(1).
- **מצב מבנה הנתונים לאחר ביצוע הפעולה** : אין כל שינוי.
- **מדוע אנחנו עומדים בסיבוכיות הזמן הדרושה** : הסיבוכיות של פעולה זו היא O(log*k + log n) משוערך עם שאר הפעולות במבנה ועל כך נפרט בהרחבה לאחר שנציג את שאר הפעולות במבנה. נציין כי הסיבוכיות שנובעת ממספר השחקנים במשחק היא log(n) מכיוון שפעולת ה- rank ו- select שאנו מבצעים מספר סופי של פעמים היא בסיבוכיות log(n) עבור עצים מאוזנים.

9) **הפעולה:**

void Quit(void **DS)

- **מטרה** : הפעולה משחררת את המבנה.
- **דרישות סיבוכיות זמן** : O(n+k) במקרה הגרוע.
- **מימוש** : לאחר בדיקת קלט ב- **O(1)**, נקרא להורס של PlayersManager שמבצע קריאה לכל אחד מההורסים של השדות שלו :

א. הרס העץ all_players_tree: מבוצע על ידי מעבר על צמתי העץ באמצעות סיור postorder שמבוצע בסיבוכיות לינארית למספר הצמתים בעץ. בעת הגעה לצומת מסוימת בסיור הרס הצומת מבוצע בסיבוכיות זמן O(1) תוך ביצוע מספר סופי של פעולות. לכן הרס כלל העץ מבוצע בסיבוכיות O(n).

ב. הרס hash_table_of_players: מבוצע על ידי מעבר על מערך התאים ועבור כל תא הריסת שרשרת הצמתים שבתא. מספר הצמתים בשרשרות הפנימיות שווה בדיוק למספר השחקנים במבנה ולכן מבוצע בסיבוכיות זמן לינארית של O(n) שכן הרס כל צומת מבוצע ב- O(1). בנוסף הרס מערך התאים גם כן מבוצע ב- O(n). הנימוק לכך הוא שבמימוש שלנו פקטור העומס חסום בין שני מספרים קבועים המכתיבים את הטווח שלו. מכיוון ו- load_factor=n/table_size אז נקבל כי table_size<(n/MIN_LOAD_FACTOR כאשר MIN_LOAD_FACTOR הוא גבול תחתון חיובי של פקטור העומס. לכן: table_size=O(n) ומכאן שהרס המערך מבוצע בסיבוכיות זמן של O(n).

ג. הרס union_find_groups: מבוצע על ידי מעבר על כלל הצמתים בעץ ההפוך (בדיוק k צמתים כמספר הקבוצות) והרס כל צומת אליה הגענו בסיור. נשים לב כי אם צומת בסיור היא גם שורש בעץ ההפוך אז היא תכיל מצביע לעץ דרגות של כלל השחקנים המשתייכים לקבוצה המאוחדת המיוצגת על ידי השורש. מכיוון וכל שחקן במבנה שלנו משתייך בדיוק לקבוצה אחת אז כחלק מהרס העץ ההפוך נבצע הריסה של בדיוק n צמתים פנימיים של שחקנים ו-k צמתים של קבוצות. לכן בסך הכל הרס ה- union_find_groups מבוצע בסיבוכיות O(n+k).

- לסיום, מבצעים השמה ל nullptr ל-DS*.
- **מצב מבנה הנתונים לאחר ביצוע הפעולה**: המבנה לא מוגדר ומכיל ערכי זבל.
- **מדוע אנחנו עומדים בסיבוכיות הזמן הדרושה**: **O(n)+O(n)+O(n+k) = O(n+k)**

חלק ד' – סיבוכיות הזמן של הפעולות המשוערכות במבנה

נחלק את ניתוח סיבוכיות הזמן לשני חלקים לפי הדרישה: ניתוח סיבוכיות משוערכת הנובעת ממספר הקבוצות K וניתוח סיבוכיות משוערכת הנובעת ממספר השחקנים במשחק n.

ניתוח סיבוכיות לפי קבוצות

יהי רצף באורך t פעולות מהסוג: mergeGroups, averageHighestPlayerLevelByGroup, increasePlayerIDLevel, getPercentOfPlayersWithScoreInBounds, changePlayersIdScore, removePlayer, addPlayer.

נראה כי הסיבוכיות הכוללת של רצף פעולות זה היא $O(t*\log^*(k) + t*f(n))$ כאשר f(n) היא פונקציה התלויה במספר השחקנים בלבד.

נשים לב כי לפנינו 7 פעולות שונות. נמספר אותן באופן שרירותי.

טענה: בהינתן i בין 1 ל-7 הפעולה ה- i מתבצעת בסיבוכיות זמן $O(a_i(\log^*(k) + f_i(n)))$.

נימוק: בכל פעולה מבין 7 הפעולות אנחנו מבצעים קריאה לפעולות union ו- find מספר סופי של פעמים. לפי משפט מהרצאה a פעולות find ו- union עם איחוד לפי גודל וכיווץ מסלולים מתבצע בסיבוכיות זמן של $O(a*\log^*k)$. נעיר כי $f_i(n)$ היא פונקציה התלויה במספר השחקנים בלבד ועליה נרחיב בחלק הבא או שהרחבנו בחלק הקודם.

לכן סיבוכיות הזמן המשוערכת של t פעולות מהסוג שצוין קודם לכן היא:

$$\sum_{i=1}^{7} O\left(a_i\left(\log^*(k) + f_i(n)\right)\right) = O\left(\sum_{i=1}^{7}(a_i)\cdot\left(\log^*(k) + f_{\max\{i\}}(n)\right)\right) = O\left(t\left(\log^*(k) + f_{\max\{i\}}(n)\right)\right)$$

המעבר האחרון נובע מכך שסכום כל ה- a_i הוא t.

סיבוכיות הזמן כנדרש היא: $\boldsymbol{O\left(t\left(\log^*(k) + f(n)\right)\right)}$.

ניתוח סיבוכיות לפי מספר השחקנים במשחק

יהי רצף באורך t_1 פעולות מהסוג: increasePlayerIDLevel, changePlayersIDScore, removePlayer ויהי רצף של t_2 פעולות addPlayer.

נראה כי הסיבוכיות הכוללת של רצף פעולות זה היא $O(f(k)+t_1*\log^*(n) + t_2)$ כאשר f(k) היא פונקציה התלויה במספר הקבוצות בלבד.

נמספר את הפעולות increasePlayerIDLevel, changePlayersIDScore, removePlayer באופן שרירותי.

טענה: בהינתן i בין 1 ל-3 הפעולה ה- i מתבצעת בסיבוכיות זמן $O(a_i(\log(n) + f_i(k)+1))$ בממוצע על הקלט.

נימוק: בכל פעולה מבין 3 הפעולות אנחנו מבצעים קריאה לפעולות add, remove או find של טבלת הערבול מספר סופי של פעמים. מהרצאה אנו יודעים כי בממוצע על הקלט פעולות אלו הן O(1). מכיוון שטבלת הערבול מומשה עם מערך דינאמי שהוא $\Theta(n)$ כפי שיוצג בניתוח סיבוכיות המקום בהמשך, מתקיים לפי ההרצאה כי a פעולות מסוג זה הן בסיבוכיות של $O(a*1)$ בממוצע על הקלט. נעיר כי $f_i(k)$ היא פונקציה התלויה במספר השחקנים בלבד ואילו הפונקציה log n נובעת משימוש בעץ בכל אחת מהפעולות מספר סופי של פעמים.

לכן סיבוכיות הזמן המשוערכת של t פעולות מהסוג שצוין קודם לכן היא:

$$\sum_{i=1}^{3} O\big(a_i(\log(n) + f_i(k) + 1)\big) = O\big(\sum_{i=1}^{3}(a_i) \cdot \big(\log(n) + f_{\max\{i\}}(k) + 1\big) = O\big(t_1\big(log^*(k) + f_{\max\{i\}}(n) + 1\big)\big)$$

המעבר האחרון נובע מכך שסכום כל ה- a_i הוא t_1.

סיבוכיות הזמן כנדרש של פעולות אלה היא: $O\big(t_1(\log(n) + f(k) + 1)\big) = O\left(t_1\big(\log(n) + f(k)\big)\right)$ משוערך ממוצע על הקלט.

טענה: בהינתן t_2 פעולות add של טבלת הערבול נקבל כי סיבוכיות הזמן היא $O(t_2)$ משוערך בממוצע על הקלט.

הוכחנו טענה זו בתרגול בתנאי שאכן גודל המערך הדינאמי הוא $\Theta(n)$ כפי שנציג בהמשך.

כלומר עבור t_2 פעולות addPlayer נקבל כי סיבוכיות הזמן היא $O\left(t_2\big(1 + f(k)\big)\right) = O(t_2 + f(k))$ משוערך בממוצע על הקלט.

בסה"כ נקבל: $O(f(k) + t_1 * \log n + t_2)$ בממוצע על הקלט כנדרש עבור הרצף הנתון.

חלק ד' – סיבוכיות המקום של המבנה

דרישת סיבוכיות מקום: $O(n + k)$ במקרה הגרוע, כאשר n הוא מספר השחקנים ו-k הוא מספר הקבוצות.

מדוע אנחנו עומדים בדרישה זו:

במבנה ישנם שלושה שדות עיקריים: עץ דרגות של כל השחקנים, טבלת ערבול דינמית של כל השחקנים ו- UnionFind של הקבוצות. נציין כי כל שחקן נשמר בצורה של צומת בכמה מבנים: בעץ הדרגות all_players_tree, ב- HashTable ובעץ הפנימי של הקבוצה אליה הוא שייך. לכן נקבל כי לכל שחקן יש בדיוק שלושה צמתים לכל היותר (אם level = 0 אז רק צומת אחת נשמרת בטבלת הערבול) ולכן מדובר בסיבוכיות מקום נוסף של O(n). בנוסף, כל קבוצה מיוצגת על ידי תא אחד ב- UnionFind ולכן סיבוכיות המקום הנוסף המתאימה היא O(k). נסקור מדוע השדות הנוספים של המבנה אינם פוגעים בסיבוכיות מקום:

- המערך הדינמי בטבלת הערבול הוא $\Theta(n)$ שכן:
  MIN_LOAD_FACTOR<load_factor<MAX_LOAD_FACTOR וידוע כי load_factor=n/table_size ולכן בסה"כ נקבל כי
  (n/MAX_LOAD_FACTOR)<table_size<(n/MIN_LOAD_FACTOR) כאשר
  MIN_LOAD_FACTOR ו- MAX_LOAD_FACTTOR הם קבועים חיוביים כלומר
  table_size= $\Theta(n)$ ולכן בסיבוכיות מקום נוסף של O(n).
- נשים לב כי בעצים ישנה היסטוגרמה בגודל קבוע של 201 ולכן זו סיבוכיות מקום נוסף O(1). בנוסף בכל צומת בעץ ישנה גם היסטוגרמה בגודל קבוע של 201 שאינה דורשת סיבוכיות מקום נוסף.
- נציין כי בכל קבוצה ב- UnionFind קיים עץ דרגות, נשים לב כי סכום כל הצמתים של כל עצי הדרגות הפנימיים הוא לכל היותר n מספר השחקנים במשחק ולכן סיבוכיות המקום הנוסף של עצים אלו היא O(n).

עד כה, הראינו כי סיבוכיות המקום הנוסף של מבנה הנתונים היא O(n+k).

נציין כי קיימות פעולות לאורך הקוד שיש להן סיבוכיות מקום גדולה מ- O(1), נראה כי בכל זאת, בכל שלב לאורך הקוד שלנו סיבוכיות המקום עומדת בדרישות. בפרט, בכל מקום בקוד בו ישנה פעולת חיפוש, הכנסה, הוצאה, rank, select וסיור מכל סוג שהוא מתבצעות קריאות רקורסיביות כתלות בגובה העץ (כלומר log(n) עבור עצי השחקנים). נעבור בקצרה על הפעולות השונות:

- Init- סיבוכיות מקום O(1) כי יש מספר סופי של הקצאות זיכרון במבנה.
- AddPlayer, RemovePlayer, IncreaseLevel, ChangeScore - המשותף לפעולות אלה הוא שמלבד מספר סופי של הקצאות לטובת הכנסת ו/או הסרת שחקן ו/או קבוצה מהמבנה, יש שימוש חוזר בפעולות insert, remove ו- find באופן בלתי תלוי (אין רקורסיה מקוננת), קרי סיבוכיות מקום של O(log(n)) במקרה הגרוע כנדרש.
- MergeGroups- כחלק ממימוש פעולה זו ישנה הקצאה של שלושה מערכים בגודל המתאים למספר השחקנים בקבוצות, כלומר הקצאות אלו חסומות על ידי O(n). בנוסף ישנה הקצאה של עץ חדש בגודל של מספר השחקנים בשתי הקבוצות כלומר הקצאה החסומה על ידי O(n). בנוסף על כל זה מתבצעים בפעולה מספר סיורי Inorder שזו סיבוכיות מקום של O(log(n)) במקרה הגרוע.
- getPercentOfPlayersWithScoreInBounds, averageHighestPlayerLevelByGroup– בפעולות אלו מתבצע שימוש בפעולות rank ו- select של עץ דרגות. פעולות אלו מתבצעות בצורה רקורסבית על העץ בעומק מקסימלי של גובה העץ כלומר O(log n) מכיוון שזה עץ מאוזן, לכן סיבוכיות המקום הנוסף בפעולות אלו היא O(log n).

לסיכום בכל זמן בפונקציה, סיבוכיות המקום הנוסף היא לכל היותר O(n+k).