ראשית נתאר את מבנה הנתונים בו נשתמש בפתרון התרגיל: מבנה הנתונים $PlayerManager$ אשר יחזיק מבנה $Union-Find$ של קבוצות בגדול $k+1$ כך שהקבוצה באינקס 0 במבנה תהיה קבוצת כל השחקנים, את כמות הקבוצות - k ואת $scale$. על קבוצה 0 לא נפעיל את פעולת $mergeGroups$ ולכן היא תמיד תהיה נפרדת משאר הקבוצות.

מחלקות העזר "שחקן", ו-"קבוצה": עבור כל שחקן ישמרו המספר המזהה שלו, רמתו ,התוצאה שלו ובאיזה קבוצה הוא נמצא, ולקבוצה יהיה עץ של כל השחקנים שרמתם אינה 0 הממויין לפי התוצאה, לאחר מכן הרמה ולבסוף המספר המזהה של השחקן, עץ דרגות נוסף של שחקנים שרמתם אינה 0 ממויין לפי השלב ואז המספר המזהה, טבלת ערבול של כל השחקנים ומערך היסטוגרמות של התוצאה של שחקנים שרמתם 0 שגודלו קבוע ואינו משתנה במהלך התוכנית.

מבנה הנתונים ישתמש במבני הנתונים הבאים:

1. עץ מסוג דרגות מסוג AVL אשר מומש בתרגיל הראשון (בתוספת דרגות הצמתים - כל צומת ידע מה כמות הצמתים בתת העץ החל ממנו). נשתמש לאורך התרגיל בכך שסיבוכיות הזמן של פעולות העץ היא ידועה ולפי מה שנלמד בהרצאות ובתרגולים - מציאת איבר\מחיקה\הכנסה בסיבוכיות של $O(log(n))$ ופעולות סיור על העץ -$inorder, postorder, preorder$ יבוצעו כולן ב- $O(n)$ כאשר n היא כמות האיברים בעץ. כמו כן, נשתמש בפעולת $combineTrees$ אשר מומשה בתרגיל הקודם לצורך איחוד שני עצים (חלק מפונקציית $ReaplceGroups$) וסיבוכיות הזמן שלה היא $O(n_1+n_2)$ כאשר זו כמות האיברים הכוללת בשני העצים. כמו כן, הראנו בתרגולים ובהרצאות כי ה"שדרוג" של העץ להיות עץ דרגות אינו משנה את סיבוכיות הזמן והמקום של פעולות העץ. מכיוון שהיעוד היחיד של העץ בתרגיל הוא לאחסון המשתתפים , לכל צומת בעץ תהיה שמורה הרמה הממוצעת בתת העץ שצומת זו היא השורש שלו. עדכון שדה זה קורה ע"י חישוב פשוט בדומה לדרגה או גובה צומת ולכן אינו משפיע על שיקולי סיבוכיות. סיבוכיות המקום של העץ היא $O(n)$ כאשר n היא כמות האיברים בו.

2. טבלת ערבול - $Hash-Table$ בעלת פקטור עומס של 1 הממומשת ע"י "ערבול כפול" - במידה ואיבר יותאם לתא כלשהו במערך אך התא תפוס, נבצע איטרציה נוספת ונמצא לאיבר תא נוסף אליו הוא יותאם. הטבלה ממומשת כפי שנלמד בהרצאה כך שבכל תא יהיה איבר יחיד ובכך סיבוכיות הזמן הממוצעת על הקלט להכנסה\מציאה\מחיקה של איבר היא $O(1)$. כאשר הטבלה מלאה, נגדיל אותה פי 2 בקירוב, למספר הראשוני הראשון שקרוב לגודל הבטלה הנוכחי כפול 2. המטרה בכך היא לשמור על פונקציית הערבול ועל פקטור העמוס לפי הדרישות. לאחר מספר גדול של מחיקות ביחס לטבלה, נקצה טבלה חדשה ונוותר על הישנה ובה דגלים שאינם רלוונטיים יותר לחיפוש איבר בתאים ריקים. סיבוכיות המקום של הטבלה היא $O(n)$ כאשר n זו כמות האיברים בה.

3. מבנה $Union-Find$ שיאותחל לגודל $k+1$ וישאר קבוע למשך כל ריצת התוכנית. המבנה ימומש כפי שנלמד בהרצאה 8 שקופיות 17-21, וישתמש בעצים הפוכים, באיחוד לפי גודל ובכיווץ מסלולים על מנת לפעול בסיבוכיות משוערכת של $log^*(k)$ כאשר k היא כמות האיברים בו. סיבוכיות המקום של המבנה היא $O(k)$.

כעת נתאר את מימוש כל אחת מהפעולות וננתח את סיבוכיות הזמן שלה.

את חישוב סיבוכיות הזמן המשוערכת הנוגעת לחלק שנובע ממספר הקבוצות - k, של הפעולות:

$$averageHighestPlayerLevelByGroup,\ mergeGroups,\ increasePlayerIDLevel,\ changePlayerIDScore,$$
$$getPercentOfPlayersWithScoreInBounds,\ removePlayer,\ addPlayer,\ getPlayersBound$$

כמו גם את חישוב הסיבוכיות המשוערכת הנוגעת לחלק של מספר השחקנים - n, של הפעולות:

$$removePlayer,\ increasePlayerLevel,\ changePlayerIDScore$$
$$addPlayer$$

נחשב בסוף במרוכז לאחר שנתאר כל אחת בנפרד.

תיאור הפעולות:

1. $\boldsymbol{void * init(intk, intsclae)}$: הפעולה תאתחל $PlayerManager$ חדש ומבנה תאתחל את מבנה ה- $Union-Find$ בגדול $k+1$ ל-$k+1$ הקבוצות. סיבוכיות הזמן של הפעולה זו היא $O(k+1) = O(k)$ כי הפעולה יוצרת את כל הקבוצות ויצירת כל קבוצה היא ב- $O(1)$. הפעולה תחזיר את מנהל השחקנים החדש, בהנחה והקלט היה תקין וכי לא הייתה בעיה בהקצאת זיכרון. סיבוכיות הזמן הכוללת של היא $O(k)$כנדרש.

2. $\boldsymbol{StatusType\ mergeGroups(void*\ DS, int\ GroupID1, int\ GroupID2)}$: הפעולה תמצא ותאחד את שתי הקבוצות במבנה ה-$Union-Find$ בסיבוכיות משוערכת של $O(log^*k)$ כפי שהוסבר בהרצאה 8 והצג למעלה, תאחד את עצי השחקנים של שתי הקבוצות בסיבוכיות $O(n)$ כאשר n זוהי כמות השחקנים בשתי הקבוצות ביחד על ידי שימוש במתודה $combineTrees$ שמוגדרת על העץ (תוך שהיא מעדכנת את דרגות העץ באופן רלוונטי), תעדכן את מערך ההיסטוגרמות של הקבוצה הרלוונטית להכיל את התוכן של המערך מהקבוצה השנייה (מבוצע ב-$O(1)$ כי גודל המערך קבוע) ותאחד את טבלאות השחקנים לכדי טבלה אחת, פעולה שממומשת על ידי המבנה $Hash-Table$ ומבוצעת ב-$O(n)$ בממוצע. יש לשים לב בעת התייחסות לקבוצה שכבר אוחדה בעבר עם קבוצה אחרת, שחקני הקבוצה הם שחקני הקבוצות המאוחודת ואיחודה עם קבוצה נוספת יהיה איחוד של הקבוצה הנוספת אל הקבוצות שכבר אוחדו. כמו כן, פעולות מבנה ה-$Union-Find$ אינן תלויות בכמה שחקנים יש לכל קבוצה, אלא כמה קבוצות אוחדו ביחד לכדי קבוצה אחת. אין דבר זה משפיע על סיבוכיות פעולות העצים וזה נועד על מנת לממש את המבנה בצורה היעילה ביותר. במידה ושתי הקבוצות אותן רוצים לאחד כבר אוחדו בעבר הפעולה לא תעשה דבר. במידה והקלט לא יהיה תקין או תהיה שגיאת זיכרון תוחזר שגיאה מתאימה. בסה"כ, סיבוכיות הזמן של הפעולה היא $O(log^*k+n)$ כנדרש.

3. $\boldsymbol{StatusType\ addPlayer(void*\ DS, int\ PlayerID, int\ GroupID, int\ score)}$: הפעולה תיצור שחקן חדש, תכניס אותו בקבוצת כל השחקנים לטבלת כל השחקנים בקבוצה ב- $O(1)$ בממוצע על הקלט ותעדכן את התא במקום המתאים לתוצאה שלו במערך ההיסטוגרמות, תמצא את קבוצות השחקן בסיבוכיות של $O(log^*k)$ במשוערך ותכניס אותו לטבלת שחקני הקבוצה, גם כן ב-$O(1)$ בממוצע על הקלט. במידה והקלט לא יהיה תקין,

השחקן כבר קיים או תהיה שגיאת זיכרון תוחזר שגיאה מתאימה.סה"כ, סיבוכיות הזמן המשוערכת בממוצע על הקלט של הפעולה היא $O(log^*k)$.

4. $StatusType\ removePlayer(void*\ DS, int\ PlayerID)$: הפעולה תמצא את השחקן בקבוצת כל השחקנים (בטבלת הערבול בקבוצה, סיבוכיות משוערת $O(1)$ בממוצע), משם תמצא את קבוצתו דרך השדה השמור אצלו, תמצא את קבוצתו במבנה ה-$Union-Find$ של הקבוצות (סיבוכיות משוערכת $O(log^*k)$) ותסיר אותו מטבלת השחקנים בקבוצה (בסיבוכיות $O(1)$ בממוצע) ומעצי השחקנים במידה ורמתו אינה 0 (היא תחפש אותו לפי התוצאה שלו ולפי רמתו בשני העצים השונים, בסדר בו יכנסו שחקנים לעץ ותוך כדי ההוצאה תעדכן את דרגות הצמתים, סיבוכיות הזמן היא $O(log(n))$ כאשר יש n איברים בעץ) ותעדכן את התא המתאים לתוצאה שלו במידה ורמתו כן 0. לבסוף הפעולה תסיר אותו מקבוצת כל השחקנים באותו האופן. במידה והקלט לא יהיה תקין, השחקן לא קיים או תהיה שגיאת זיכרון תוחזר שגיאה מתאימה. בסה"כ, סיבוכיות הזמן של הפעולה היא $O(log^*k+log(n))$ כנדרש.

5. $StatusType\ increasePlayerIDLevel(void*\ DS, int\ PlayerID, int\ LevelIncrease)$: הפעולה תמצא את השחקן בקבוצת כל השחקנים (בטבלת הערבול בקבוצה, בסיבוכיות $O(1)$ בממוצע), במידה ורמת השחקן היא 0, תעדכן את רמתו, תקטין ב-1 את התא המתאים לתוצאה שלו בהיסטוגרמה ותכניס אותו לעצי השחקנים שרמתם אינה 0, במידה והיא אינה 0, תעדכן אותו, תמצא אותו בעצים, תסיר אותו, תעדכן את רמתו ואז תכניס אותו מחדש (בכך לא תיפגע בתקינות סדר האיברים בעץ ובדרגות הצמתים). לאחר מכן היא תמצא את קבוצתו דרך השדה השמור אצל השחקן ותבצע את פעולה זו שוב באותו האופן רק בקבוצת השחקן ולא בקבוצת כל השחקנים. במידה והקלט לא יהיה תקין, השחקן אינו קיים במערכת או תהיה שגיאת זיכרון תוחזר שגיאה מתאימה. הסיבוכיות הכוללת של הפעולה נקבעת לפי מציאתו ועדכנו בטבלאות, בעצים ומציאת קבצתו במבנה הקבוצת ולכן היא בהס"כ תהיה $O\left(log^*k+log(n)\right)$כנדרש.

6. $StatusType\ changePlayerIDScore(void*\ DS, int\ PlayerID, int\ NewScore)$: הפעולה תמצא את השחקן בקבוצת כל השחקנים (בטבלת הערבול בקבוצה, בסיבוכיות $O(1)$ בממוצע), תעדכן את התוצאה שלו, במידה ורמת השחקן אינה 0, תמצא אותו בעץ הרלוונטי, תסיר אותו, תעדכן את התוצאה שלו ואז תכניס אותו מחדש (בכך לא תיפגע בתקינות סדר האיברים בעץ) ובמידה ורמתו כן 0, תעדכן במערך ההיסטוגרמות את התאים המתאים לתוצאתו הישנה (תחזיר מהכמות השמורה שם 1) ולתוצאתו החדשה (תוסיף 1). לאחר מכן היא תמצא את קבוצתו דרך השדה השמור אצל השחקן ותבצע את פעולה זו שוב באותו האופן רק בקבוצת השחקן ולא בקבוצת כל השחקנים. במידה והקלט לא יהיה תקין, השחקן אינו קיים במערכת או תהיה שגיאת זיכרון תוחזר שגיאה מתאימה. הסיבוכיות הכוללת של הפעולה נקבעת לפי מציאתו בטבלאות, בעצים, מציאת קבצתו במבנה הקבוצת ועדכון התאים במערך ולכן היא בהס"כ תהיה $O\left(log^*k+log(n)\right)$כנדרש.

7. $getPercentOfPlayersWithScoreInBounds(DS,\ GroupId,\ score,\ lowerLevel\ higherLevel,\ players)$ $StatusType$: הפעולה תעבור על העץ הממויין תחילה לפי תוצאה, תמצא כמה שחקנים יש בתוצאה המבוקשת כאשר רמתם היא בין שני החסמים (תעשה זאת בכך שתחפש את השחקן שתוצאתו היא המבוקשת ורמתו הכי קרובה מלמעלה לחסם התחתון, ואת השחקן שתוצאתו היא המבוקשת ורמתו הכי קרובה מלמטה לחסם העליון) כאשר את כמות השחקנים היא תחשב לפי דרגות הצמתים, תעשה אותה הפעולה בעץ הממויין רק לפי הרמה והמספר המזהה בו תקבל את כמות השחקנים הכוללת של השחקנים בין רמות אלה ותחזיר את היחס בין שתי הקבוצות. הפעולה תתייחס בנפרד למקרה ובו החסם התחתון הינו 0, ובמקרה זה תסכום גם את השחקנים שנמצאים ברמה 0 כחלק מהסכום הכולל ולכמות השחקנים בתוצאה הזאת תוסיף את כמות השחקנים ברמה 0 בעזרת ההיסטוגרמה.

במידה והקלט לא יהיה תקין, אין שחקנים בקבוצה בין רמות אלו או תהיה שגיאת זיכרון תוחזר שגיאה מתאימה. הפעולה תפעל על הקבוצה המבוקשת או קבוצת כל השחקנים - קבוצה מספר 0, לפי הקלט. סיבוכיות הזמן הכוללת מתבטאת בחיפוש הקבוצה הרלוונטית ובה חיפוש של ארבעה איברים שונים בעצים כאשר מציאת כל אחד מהם היא ב-$O\left(log(n)\right)$ וחישוב אמפירי ולכן בסה"כ תהיה $O\left(log^*k + log(n)\right)$ כנדרש.

8. $\boldsymbol{StatusType\ averageHighestPlayerLevelByGroup(\ DS, GroupID, m, avgLevel)}$: פעולה זו תשתמש בשדה נוסף מסוג $double$ שיהיה שמור בכל צומת בעץ. הפעולה תרוץ על השחקנים הממויין לפי רמתם, כאשר תתחיל מהשורש ותחפש בתת העץ הימני כל פעם את הצומת בה כמות הצמתים (השחקנים) בתת העץ הנוכחי היא בדיוק הכמות של השחקנים הטובים ביותר (שרמתם הגבוהה ביותר) להם נרצה לחשב את הממוצע. השדה הנוסף של הצומת יחזיק את ממוצע השחקנים שנמצאים בתת העץ כאשר צומת זו היא השורש. במידה לא קיימת צומת המחזיקה את המספר המדויק של השחקנים אותו נרצה לחשב, נמצא את אבא שלה (בהנחה שהצומת אינה השורש כי אז הדרישה היא על כמות שחקנים הגודלה מזו שבעץ, במקרה זה נשתמש גם בשחקנים שרמתם 0 ובמידה וגם אז לא יהיו מספיק שחקנים בקבוצה תוחזר שגיאה). ממנו, נרד לבן הימני, נחשב את סכום הדרגות הכולל שם (ע"י הכפלת הדרגה הממוצעת השמורה בו כפול גודל תת העץ שהשורש שלו הוא הבן הימני) ונספור כמה שחקנים עברנו - כאלה שהיו בתת העץ הימני כולל שורש תת העץ, אחר כך נלך לתת העץ השמאלי שלו ונתחיל את החיפוש מההתחלה כאשר השורש הוא הצומת בה אנו נמצאים כרגע (הבן השמאלי של האבא) וכומת השחקנים היא הכמות שחיפשנו בהתחלה פחות כמה שהיו בתת העץ הימני פחות 1 עבור השורש עצמו של תת עץ זה. לדוגמה, נניח ונחפש מה ממוצע הרמות של 9 השחקנים ברמה הגבוהה ביותר. בעצם נחפש את הצומת האינדקס הוא 9 ונחזיר את הערך שם. תמיד מתחילים מלמעלה והולכים ימינה לכן נניח ונעצרנו בצומת שגודל תת העץ בה הוא 11, ובבן הימני שלו יש 5. אז נלך לבן הימני, נקח את הממוצע שיש בו, נכפיל ב-5, נחזור לאבא, ניקח את הרמה השמורה בו ונלך ממנו שמאלה ואז ימינה עד הסוף ונחפש צומת עם 3. אם אין, וקיימת נגיד צומת עם 5 ומימינה יש רק 2, נחזור על התהליך.

עדכון העץ יבוצע בכל פעם שנעלה את רמתו של שחקן והכנסה ראשונית מבוצעת בפעם הראשונה שמעלים את רמתו מ-0. בעץ השחקנים ימוינו לפי הרמה. במידה והקלט לא יהיה תקין, יש בקבוצה פחות שחקנים מאשר הכמות לה אנו מתבקשים לחשב ממוצע או תהיה שגיאת זיכרון תוחזר שגיאה מתאימה. הפעולה תפעל על הקבוצה המבוקשת או קבוצת כל השחקנים - קבוצה מספר 0, לפי הקלט. סה"כ, סיבכיות הזמן של הפעולה היא מציאת הקבוצה הרלוונטית, ופעולת הסריקה של העץ המבוצעת בסיבוכיות זמן של $O\left(log(n)\right)$ כאשר n זו כמות האיברים בעץ, ולכן בסה"כ, סיבוכיות הזמן של הפעולה היא $O\left(log^*k + log(n)\right)$כנדרש.

9. $\boldsymbol{StatusType\,getPlayersBound(\ DS, GroupID, score, m, LowerBoundPlayers, HihgerBoundPlayers)}$: לא מומש.

10. $\boldsymbol{voidQuit(void**DS)}$: הפעולה תקרא להורס של $PlayerManager$ שבתורו יקרא להורסים של כל שאר המבנים במימוש, כאשר כל אחד מהם מוחק את הזיכרון שמוחזק בו. כל הורס מוחק את הזיכרון שמוכל בו בלבד בסיבוכיות זמן של כמות הזכרון שמכול בו (העץ עושה זאת דרך סיור $postorder$, טבלת הערבול ומבנה ה-$Union-Find$ דרך מעבר על מערך האיברים) ובגלל שסיבוכיות הזיכרון במבנה היא $O\left(n+k\right)$(נסביר בהמשך) פעולה זו תפעל בסיבוכיות זו כנדרש.

חישוב הסיבוכיות המשוערכת:

1. חישוב סיבוכיות הזמן המשוערכת הנוגעת לחלק שנובע ממספר הקבוצות - k, של הפעולות:

$$averageHighestPlayerLevelByGroup,\ mergeGroups,\ increasePlayerIDLevel,\ changePlayerIDScore,$$
$$getPercentOfPlayersWithScoreInBounds,\ removePlayer,\ addPlayer,\ getPlayersBound$$

בפעולות אלה, סיבוכיות הזמן היא כפי שהוצגה מעלה ועומדת בדרישות מכיוון שאנו נסמכים על כך (והמימוש תומך בכך) שפעולת $find$ במבנה $Union-Find$ היא בסיבוכיות משוערכת של $O\left(log^*k\right)$. במימוש שלנו, בניגוד למה שרואים בהרצאות או בתרגול, במקום לשלוח לפעולת $Union$ שתי קבוצות לאחר שהשתמשנו פעמיים בפעולת $find$, הפעולה עצמה קוראת ל-$find$. לכן כדי לעמוד בדרישות הסיבוכיות האלו, עלינו לדאוג לכך שהמבנה באמת ישתמש בעצים הפוכים, יאחד לפי גודל (כן מבוצע בפעולת ה-$Union$) ויכווץ מסלולים (אשר מתבצע בפעולת ה-$find$). לכן בסה"כ, הסיבוכיות המשוערכת היא אכן $O\left(t*log^*k+t*f(n)\right)$ כאשר $f(n)$ תלויה בכמות השחקנים בלבד, כנדרש.

2. כמו גם את חישוב הסיבוכיות המשוערכת הנוגעת לחלק של מספר השחקנים - n, של הפעולות:

$$removePlayer,\ increasePlayerLevel,\ changePlayerIDScore$$
$$addPlayer$$

בפעולות אלה, סיבוכיות הזמן המתייחסת לחלק של השחקנים תלויה בשני דברים:

(א) פעולות על העצים (מציאה, הכנסה והסרה) אשר מתבצעות בסיבוכיות של $O\left(log(n)\right)$במקרה הגרוע ולכן גם במשוערך.

(ב) מציאת שחקנים\הסרתם מטבלת הערבול. ראינו בהרצאות שהסיבוכיות הממוצעת לפעולות הכנסה, הוצאה ומציאה היא ב-$O\left(1\right)$ושהסיבוכיות המשוערכת היא על כן ב-$O(1)$ גם כן.

לכן בסה"כ, הסיבוכיות של פעולות אלה תהיה $O\left(f(k)+t_1*log(n)+t_2\right)$כאשר $f(k)$ תלוי רק במספר הקבוצות, t_1 זו כמות הפעולות הכוללת של רצף הפעולות $removePlayer,\ increasePlayerLevel,\ changePlayerIDScore$ כי הן אלה הפועלות המשתמשות בפעולות העץ ו-$addPlayer$ אינה פועלת על העץ ומבצעת t_2 פעמים.

סיבוכיות המקום הכוללת של מבנה הנתונים מתבטאת בדברים הבאים:

1. החזקת מבנה ה-$Union-Find$ בגדול $k+1$ של קבוצות. סיבוכיות המקום של מבנה זה היא $O\left(k+1\right)=O\left(k\right)$כאשר אאת סיבוכיות המקום הנוספת לכל קבוצה נסכום בנפרד.

2. כל קבוצה מחזיקה שני עצים של שחקנים שרמתם אינה 0 ו, מערך של מספרים שלמים בגדול קבוע וחסום ע"י 201 זטבלת ערבול ובה כל השחקנים. לכן, סיבכויות המקום של כמות השחקנים בכל קבוצה חסומה מלמעלה ע"י $3n$ כאשר n זוהי כמות השחקנים בקבוצה (כפול כמות המקום הקבועה הנשמרת עבור

כל שחקן). בסה"כ, מכיוון שבכל k הקבוצות יחד יש אותה כמות שחקנים כמו בקבוצה מספר 0 - קבוצת כל החשקנים, סה"כ, סיבוכיות המקום עבור השחקנים חסומה ע"י $O\left(6n\right) = O\left(n\right)$. רק נזכיר כי סיבוכיות המקום הכוללת של העצים וטבלת העירבול היא $O\left(n\right)$כאשר n זוהי כמות האיברים במבנה.

לכן בסה"כ, סיבוכיות המקום הכוללת של מבנה הנתונים היא $O(k+n)$ כנדרש.