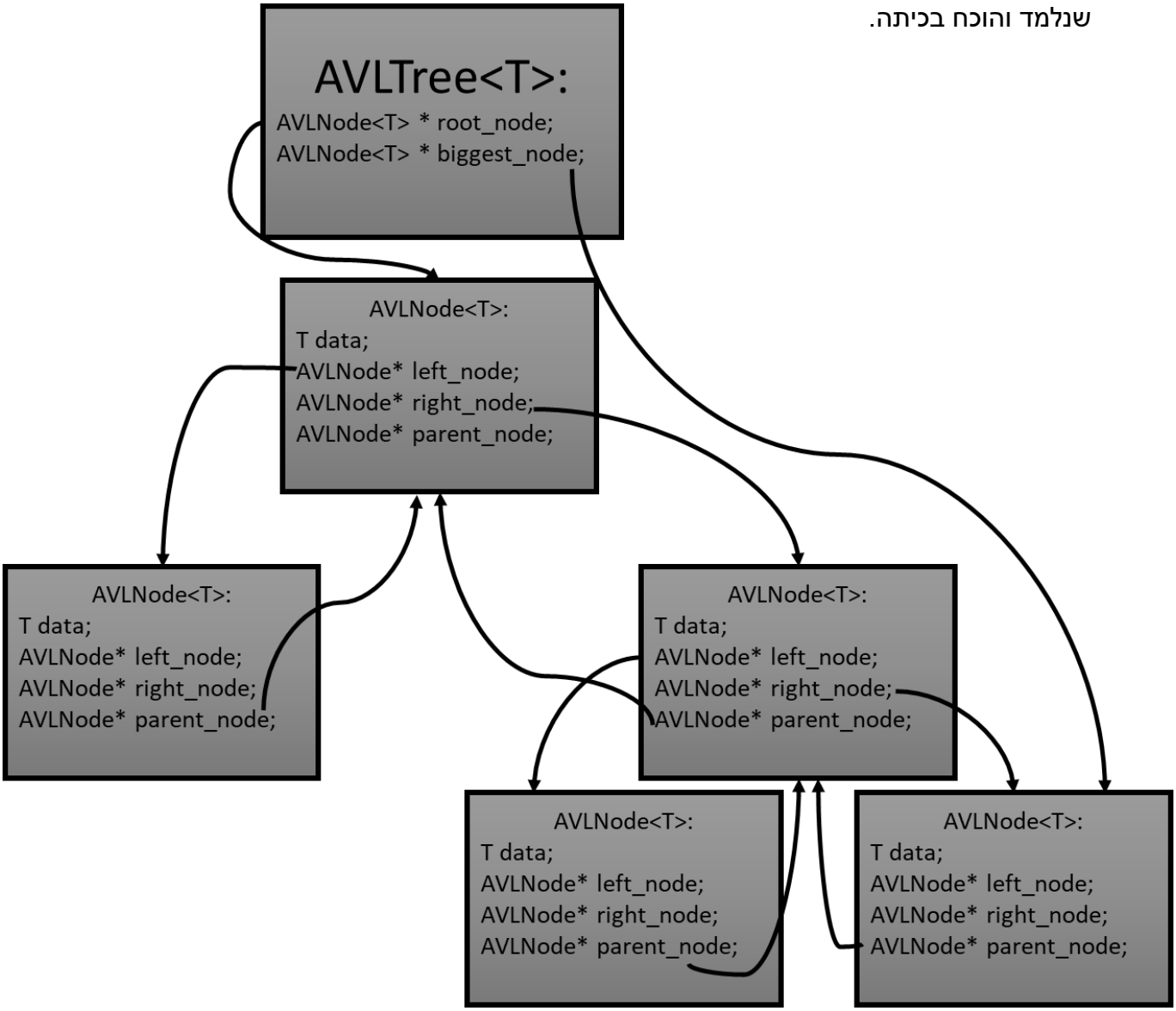
תרגיל רטוב 1: מגישים: אסף לובטון 209844414 ועדן דמבינסקי 212227888.

מאחר וראינו כי ברצוננו להחזיק 2 עצי חיפוש על מנת לממש את התרגיל החלטנו לממש עץ חיפוש (מסוג **AVL**) גנרי. לחוליה הגנרית בעץ החיפוש, יש מצביע לחוליה ימנית, שמאלית, אבא, ושדה המכיל מצביע לטיפוס הגנרי שיכיל את התוכן.

המימוש הגנרי של העץ כולל את המשתנים והפעולות הבאים: מצביע לשורש (**root**) מסוג החוליה הגנרית שתוארה לעיל, ומצביע לערך הגדול ביותר בעץ מסוג החוליה הגנרית, נשים לב כי עלינו לעדכן מצביע זה במקרה ומתבצעת הכנסה של ערך גדול יותר או הוצאה של הערך הגדול ביותר בעץ, (בכל מקרה פעולת עדכון של המצביע היא לוגוריתמית שכן מדובר בחיפוש בעץ כלומר במקרה הגרוע מספר הפעולות הוא מסדר גודל של גובה העץ).

לעץ הפעולות הסטנדרטיות שראינו בהרצאה ובתרגולים: המימוש תואם לאופן שבו למדנו ולכן לא נסביר את אלגוריתמי הכנסה הוצאה וחיפוש שכן מדובר בחומר שנלמד. הכנסה (**insert** מבוססת על העמסת אופרטור =<,== עבור הטיפוס הגנרי), חיפוש (**find** מבוסס על העמסת אופרטור = עבור הטיפוס הגנרי), מחיקה (**delete** מבוסס על אופרטור <,>), גלגול לימין, גלגול לשמאל, חישוב גובה, חישוב פקטור איזון.

הכנסה חיפוש ומחיקה כל אלה בסיבוכיות זמן לוגוריתמית למספר הצמתים בעץ(שווה לגובה העץ) כפי שנלמד והוכח בכיתה.

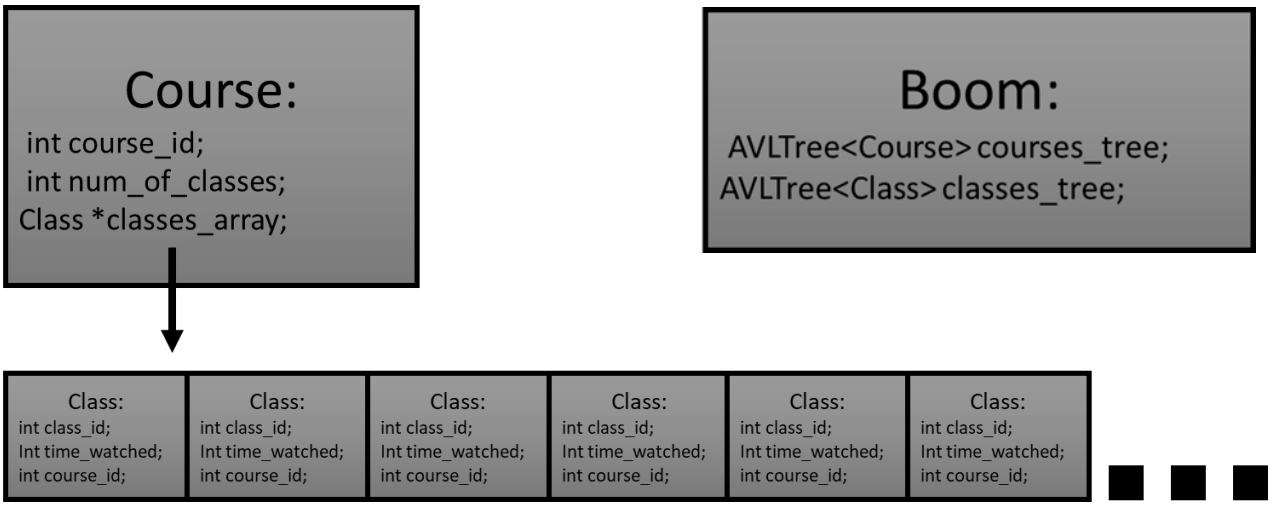


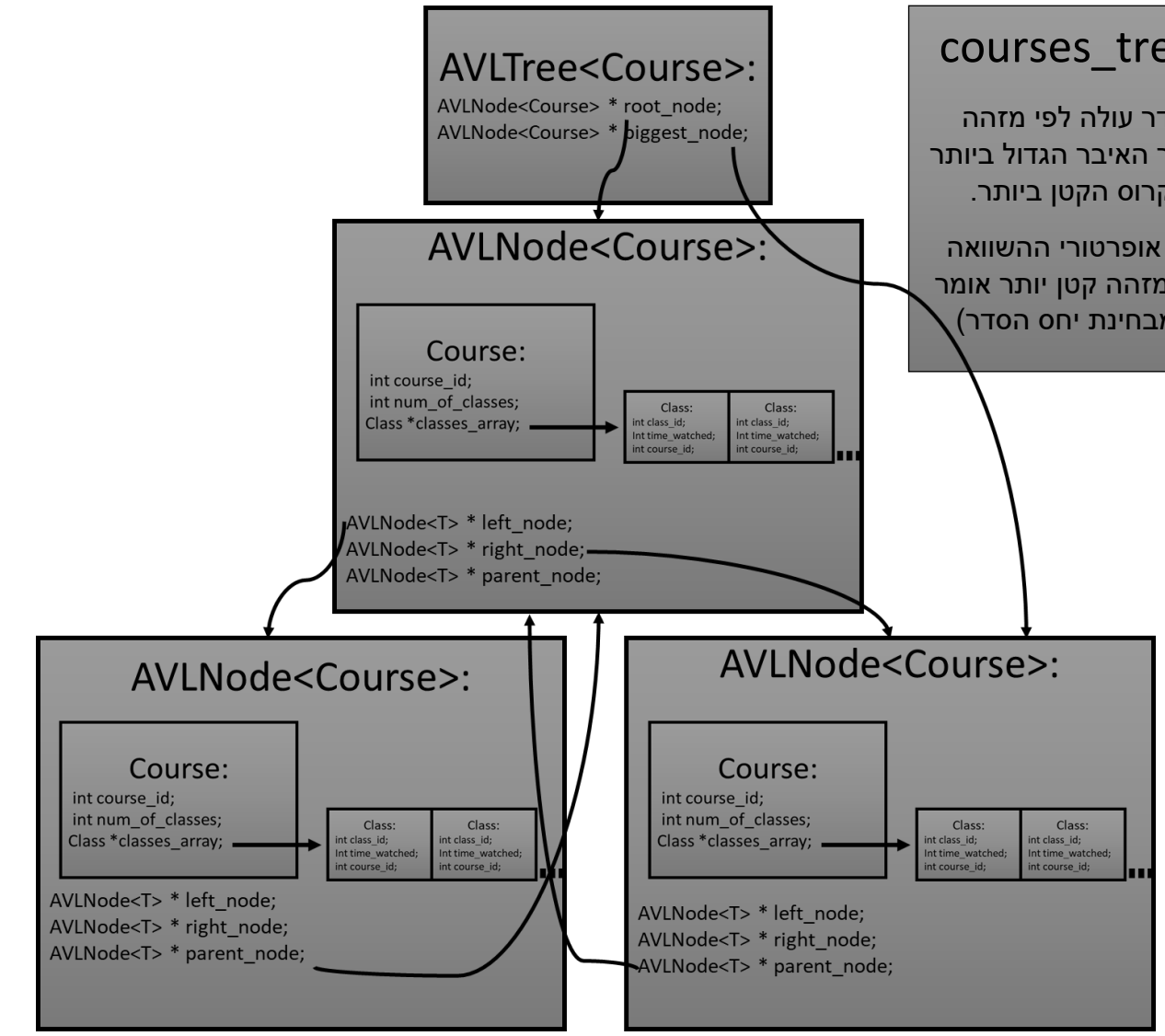
<u>בנינו טיפוס מסוג קורס (**Course**):</u> עבור כל קורס שמרנו מספר שיהיה מזהה הקורס, מספר עבור מספר הכיתות בקרוס ומערך של כיתות בגודל מספר הכיתות.

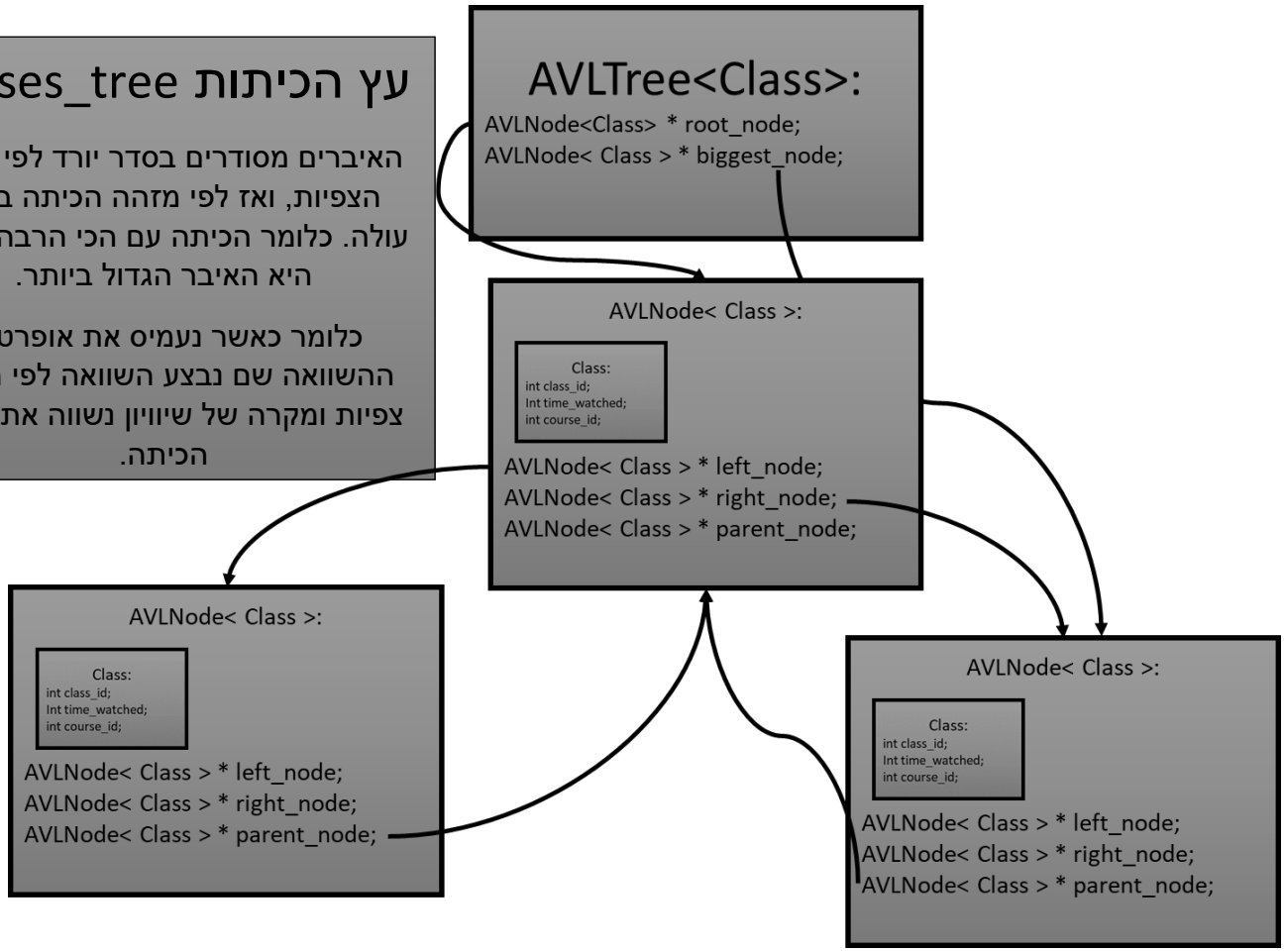
<u>טיפוס מסוג כיתה (**Class**):</u> עבור כל כיתה שמרנו מספר שיהיה מזהה כיתה, כמות הזמן הנצפה, ואת מזהה הקרוס אליו היא שייכת.

<u>טיפוס (**Boom**):</u> מחזיר עץ חיפוש מסוג **Course** (העץ ממוין לפי המזהה קורס) ועץ חיפוש מסוג **Class** (העץ ממויין לפי זמני הצפייה ובמקרה של שיוויון לפי לפי מזהה השיעור).

עבור מבנה הנתונים **DS** נבנה מופע אחד של **Boom**. ניתן לראות איור של שני העצים בעמוד הבא.







עץ הקורסים courses_tree

האיברים מסודרים בסדר עולה לפי מזהה הקרוס בסדר עולה, כלומר האיבר הגדול ביותר בעץ הוא עם מזהה הקרוס הקטן ביותר.

כלומר כאשר נעמיס את אופרטורי ההשוואה שם נהפוך סימן ונקבע שמזהה קטן יותר אומר שהקורס "גדול יותר" (מבחינת יחס הסדר)

```
AVLTree<Course>:
AVLNode<Course> * root_node;
AVLNode<Course> * biggest_node;
```

```
AVLNode<Course>:
Course:
 int course_id;
 int num_of_classes;
Class *classes_array;
Class:            Class:
int class_id;     int class_id;
Int time_watched; Int time_watched;
int course_id;    int course_id;
...
AVLNode<T> * left_node;
AVLNode<T> * right_node;
AVLNode<T> * parent_node;
```

```
AVLNode<Course>:
Course:
 int course_id;
 int num_of_classes;
Class *classes_array;
Class:            Class:
int class_id;     int class_id;
Int time_watched; Int time_watched;
int course_id;    int course_id;
...
AVLNode<T> * left_node;
AVLNode<T> * right_node;
AVLNode<T> * parent_node;
```

```
AVLNode<Course>:
Course:
 int course_id;
 int num_of_classes;
Class *classes_array;
Class:            Class:
int class_id;     int class_id;
Int time_watched; Int time_watched;
int course_id;    int course_id;
...
AVLNode<T> * left_node;
AVLNode<T> * right_node;
AVLNode<T> * parent_node;
```

עץ הכיתות classes_tree

האיברים מסודרים בסדר יורד לפי מספר הצפיות, ואז לפי מזהה הכיתה בסדר עולה. כלומר הכיתה עם הכי הרבה צפיות היא האיבר הגדול ביותר.

כלומר כאשר נעמיס את אופרטורי ההשוואה שם נבצע השוואה לפי מספר צפיות ומקרה של שיוויון נשווה את מזהה הכיתה.

```
AVLTree<Class>:
AVLNode<Class> * root_node;
AVLNode< Class > * biggest_node;
```

```
AVLNode< Class >:
Class:
int class_id;
Int time_watched;
int course_id;
AVLNode< Class > * left_node;
AVLNode< Class > * right_node;
AVLNode< Class > * parent_node;
```

```
AVLNode< Class >:
Class:
int class_id;
Int time_watched;
int course_id;
AVLNode< Class > * left_node;
AVLNode< Class > * right_node;
AVLNode< Class > * parent_node;
```

```
AVLNode< Class >:
Class:
int class_id;
Int time_watched;
int course_id;
AVLNode< Class > * left_node;
AVLNode< Class > * right_node;
AVLNode< Class > * parent_node;
```

הסבר על הפעולות אותן נדרשנו לממש: **n** מספר הקורסים במערכת, **m** מספר ההרצאות של קורס מסויים, **M** מספר ההרצאות במערכת.

void * Init():

הפעולה מחזירה מצביע לטיפוס מסוג **Boom**, הפעולה קוראת לבנאי שלו אשר קורא לבנאי של שני העצים האחד עץ הקורסים והשני עץ הכיתות. כאשר הבנאי של העץ הגנרי שם ערך **null** בשורש העץ על מנת לאתחל אותו. לכן מדובר בסהכ במספר קבוע של פעולות לכן סיבוכיות הזמן היא מסדר גודל **O(1)**.

StatusType AddCourse (void *DS, int courseID, int numOfClasses):

על מנת ליצור מופע של הטיפוס קורס מדובר בסיבוכיות מסדר גודל **O(m)** מאחר ואנו מאתחלים מערך כיתות כאשר הבנאי של כיתה מאתחל את מספר הצפיות לאפס ואת מזהה הכיתה להיות זהה למיקומו במערך הכיתות של הקורס, ואת מזהה הקורס של הכיתה לערך של מזהה הקורס המסופק בחתימת הפונקציה. *אנחנו לא יכולים להסתפק במזהה כיתה בלבד עבור תיאור חד חד ערכי של הכיתות מאחר ויהיו הרבה כיתות עם אותו מזהה כיתה וזמן צפייה, לכן החלטנו להוסיף לכל כיתה מזהה קורס וכך קיימת כיתה אחת בכל מבנה הנתונים שלנו שלה יש אותו מזהה קורס ומזהה כיתה. בחירה זו תעזור לנו להגדיר את היחס סדר בעץ הכיתות שיהיה השוואה לפי זמני צפייה ואז לפי מזהה כיתה ואז לפי מזהה קורס.

על מנת להכניס את הקורס לעץ הקורסים עלינו לבצע חיפוש בעץ הקורסים, על פי המימוש שלנו על מנת להכניס קורס סיבוכיות הזמן תהיה לוגוריתמית למספר הקורסים בעץ. לכן אם נסכום נראה שאכן קיבלנו את סיבוכיות הזמן המבוקשת **O(log(n)+m)**.

נדגיש כי כאשר אנו מוסיפים קורס חדש למערכת אנחנו לא מוסיפים את הכיתות של הקורס לעץ הכיתות שכן הן כולן מאותחלות לאפס. לכן נזכור כי אם כיתה לא נמצאת בעץ הכיתות היא מכילה אפס צפיות.

StatusType RemoveCourse(void *DS, int courseID):

על מנת למחוק קורס עלינו למחוק את הקורס מעץ הקורסים, וגם לדאוג למחיקה של כל כיתה שבקורס מעץ הכיתות.

תחילה נחפש את הקורס בעץ הקורסים (העמסנו את אופרטור > ו= שיש בהם שימוש במימוש הגנרי של עץ החיפוש), חיפוש זה ייקח לנו במקרה הרע כגובה העץ כלומר לוגוריתמי ביחס למספר הקורסים בעץ הקורסים (מובטח לנו כי מספר ההרצאות=מספר הקורסים כפול מספר הכיתות גדול ממש ממספר הקורסים) אנו עושים חיפוש זה פעם אחת והוא בסיבוכיות קטנה ממש מ**log(M)**>**log(n)**). ייתכן ונצטרך לבצע גלוגלים כגובה העץ על מנת לאזן אותו אך עדיין אנו שומרים על סיבוכיות לוגוריתמית למספר הקורסים מאחר וכל גלגול מבצע מספר קבוע של פעולות ולכן הוא מסיבוכיות **O(1)**.

כעת מכשמצאנו את הקורס אותו אנו רוצים למחוק נעבור על מערך הכיתות של הקורס, עלינו למחוק כל אחת מהכיתות מעץ הכיתות. המחיקה של כל כיתה מעץ הכיתות היא לוגוריתמית למספר הכיתות בעץ (ל**M**), לכן עבור כל כיתה נבצע **log(M)** סהכ **O(mlog(M))** פעולות על מנת למחוק את כל הכיתות מעץ הכיתות.

מחיקת מערך הכיתות של הקורס תיקח סדר גודל של סיבוכיות **O(m)**.

בנוסף נבדוק האם הערך אותו אנו רוצים למחוק הוא הערך הגדול ביותר בעץ, אם כן עלינו לאחר המחיקה לרוץ על עץ הכיתות (מקסימום כגובה העץ שהוא לוגוריתמי למספר הכיתות) למצוא את הערך הגדול ביותר בעץ, ולעדכן את המצביע לערך הגדול ביותר בעץ. לכן נקבל כי דרוש מספר פעולות מסדר גודל **O(log(M))** על מנת לעדכן את המצביע במקרה הגרוע ביותר.

לכן סה"כ שמרנו על הסיבוכיות המבוקשת

מאחר ו-

O(mlog(M))= k_1log(n) + k_2(m) + K_3mlog(M)+k_4log(M) , (**log(n)<log(M)**)** .

<u>StatusType WatchClass(void *DS, int courseID, int classID, int time)</u>:

מאחר ואנחנו לא מוסיפים כיתה לעץ הכיתות אם מספר הצפיות של הכיתה עומד על אפס היא לא נמצאת בעץ הכיתות, מאחר ומסופק לנו המזהה קורס והמזהה כיתה אנו נבצע חיפוש בעץ הכיתות (החיפוש מתבצע באמצעות האופרטור = ו> שאותם העמסנו בשתי המחלקות של כיתה ושל קורס). יש שתי אפשרויות, הכיתה נמצאת בעץ ויש לעדכן אותה, הכיתה לא נמצאת בעץ ויש להוסיף אותה לעץ. למצוא כיתה בעץ הכיתות לוקח סדר פעולות לוגוריתמי למספר הכיתות במערכת כלומר **O(log(M))** פעולות על מנת למצוא את הכיתה שאותה יש לעדכן, העדכון עצמו לוקח מספר קבוע של פעולות. לאחר שמצאנו את הכיתה אנחנו שומרים העתק שלה עם זמן הצפייה החדש, מוחקים את הכיתה מהעץ הכיתות ומוספים אותה מחדש לעץ עם הערכים העדכניים, מאחר וחיפוש מחיקה והכנסה כולם מתבצעים בסדר גודל לוגוריתמי למספר הכיתות בעץ הכיתות סך הכל לקח לנו **O(log(M))**. אם הכיתה מלכתחילה לא הייתה קיימת בעץ עלינו ליצור מופע של הטיפוס כיתה עם הערכים המסופקים ולהוסיף אותו לעץ הכיתות. (גם אפשרות זו נעשית בסדר גודל לוגוריתמי למספר הכיתות בעץ).

פעולה נוספת שנעשה היא לבדוק האם לאחר העדכון הכיתה בעלת הכי הרבה צפיות בעץ הכיתות עדיין אותה הכיתה, אין צורך לעשות חיפוש, מאחר ואנו מחזיקים מצביע לערך הגדול ביותר בעץ כל שיש לעשות הוא להשוות בין הערכים של החוליה שהכנסנו בשני המקרים ואם יש צורך לעדכן את המצביע לכיתה הגדולה ביותר. עדכון המצביע נעשה על ידי מספר קבוע של פעולות ולכן הוא מסיבוכיות **O(1)**.

כעת עלינו לעדכן את הכיתה במערך הכיתות עבור הקורס ששמור בעץ הקורסים. (בחרנו להפריד בין העצים ולא לשמור מצביע בין האיברים על מנת לשמור על פשטות, ומאחר ונראה כי אנו נעמוד בדרישה בכל מקרה).

באופן דומה לסעיף הקודם מאחר ו**M** גדול ממש מ**n** ואנו עושים פעולת חיפוש על עץ הקורסים כעת ייקח לנו סדר גודל לוגריתמי ל**n** שהוא קטן ממש מ**O(log(M))** , הגישה לעדכן איבר במערך היא ב**O(1)**, ועדכון מספר הצפיות עבור כיתה במערך גם דורש מספר קבוע של פעולות ולכן גם הוא מסיבוכיות **O(1)**. את כל אחד מהעדכנים עושים פעם אחת לכן סהכ נקבל כי הסיבוכיות היא **O(log(M)).**

StatusType TimeViewed(void *DS, int courseID, int classID, int *timeViewed):

על מנת לקבל את משך הצפייה של של כיתה מסויימת, נחפש בעץ הקורסים לפי מזהה הקורס, פעולה זו תיקח לנו סדר גודל לוגוריתמי למספר הקורסים כלומר **O(log(n))** לאחר שמצאנו את הקורס המבוקש ניגש למערך הכיתות של הקורס מאחר ומזהה הכיתה מתאים למיקום של הכיתה במערך הכיתות של הקורס כל שעלינו לעשות הוא לגשת למערך הכיתות בסיבוכיות **O(1)**. לכן סהכ שמרנו על **O(log(n))** פעולות.

StatusType GetMostViewedClasses(void *DS, int numOfClasses, int *courses, int *classes):

על מנת לבצע חלק זה שמרנו את המצביע לערך הגדול ביותר בעץ הכיתות (דאגנו לשמור אותו מעודכן בכל מחיקה ועדכון נתוני צפייה של כיתה מסויימת). אנחנו נעבור על עץ הכיתות כאשר אנו מתחילים מהערך הגדול ביותר לפי האלגוריתם הבא: יש לנו מונה שערכו שווה למיקום בו אנחנו נמצאים במערך שעלינו למלא. הצומת ממנו אנחנו מתחילים את הסיור בעץ הוא הצומת העל הערך הגדול ביותר נקרא לו **v**.

1.אם v ריק או שמספר הכיתות שנותר להביא הוא אפס סיים.

2. הכנס למערך במקום המונה את הערך הרלוונטי מv.

3. מצא את הצומת הגדול ביותר בתת העץ השמאלי של v (אם קיים אחרת עבור ל4) וחזור לשלב 1 עם v כערך שמצאת בחיפוש בתת העץ השמאלי, כאשר אתה מגדיל את המונה באחד.

4.השם את אבא של v להיות v אם קיים אחרת סיים. חזור לשלב 1.

האופציה היחידה בה אנו עוברים על כל העץ היא שמספר הכיתות שביקשו מאיתנו להביא הוא שווה למספר הכיתות בעץ.

אם מספר הכיתות בעץ גדול ממספר הכיתות שביקשו מאיתנו להביא נראה באמצעות אינדוקציה על מספר הכיתות שעלינו להביא (**m=num_of_classes**) כי האלגוריתם שהצענו עומד בסיבוכיות המבוקשת.

בסיס: **m=1**

מבקשים להביא כיתה אחת, נשתמש במצביע לכיתה הגדולה ביותר בעץ הכיתות ונסיים.

הנחה:

אנחנו מביאים את **m** הכיתות בסיבוכיות **O(m)**, נראה כי אנחנו מביאים את **m** הכיתות ב**O(m+1)**.

נחלק לשני מקרים:

1. האיבר הגדול הבא הוא האבא של הצומת הנוכחי (העוקב ב**in-order**), נשתמש בהנחת האינדוקציה להביא את ה**m** הכיתות הגדולות ביותר בעץ ואת הכיתה הבאה נביא דרך המצביע

לאבא של הצומת הנוכחי. לכן סה"כ קיבלנו סדר גודל של 1 **+O(m)** פעולות לכן סה"כ **O(m)<= O(m+1)**.

2. האיבר הגדול הבא נמצא בתת העץ השמאלי (הכי ימני בתת העץ השמאלי). נשתמש בהנחת האינדוקציה להביא את **m** האיברים הגדולים ביותר. כעת עלינו לרדת ימינה לכל היותר את גובה הצומת בו אנחנו נמצאים (פחות אחד) בשלב זה באינדוקציה, כלומר לכל היותר עלינו לרדת **(log(m+1-1****. מאחר והעץ מאוזן (**AVL**) ההפרש בין גובה תת העץ השמאלי לבין תת העץ הימני קטן שווה בערך מוחלט מ1. עד כה הבאנו את כל תת העץ הימני לצומת בו אנחנו נמצאים בשלב זה באינדוקציה ולכן הבאנו מספר שהוא בהכרח גדול שווה מגובה הצומת ולכן גדול שווה ***מ- **log(m)**. לכן סה"כ עשינו **O(m)+log(m)<=O(m+1)** לכן לכל היותר נבצע **O(m+1)** פעולות.

ייתכן כי מספר הכיתות שיבקשו מאיתנו להביא גדול ממספר הכיתות שאותחלו, ניזכר כי בעץ הכיתות נמצאות רק כיתות שמספר הצפיות בהן הוא גדול מאפס. האופן שבו מימשנו את את אלגוריתם החיפוש שתיארנו לעיל טיפה יותר מרוכב שכן הפעולה שלנו החזירה בסופו של דבר את מספר האיברים שהבאנו (ראינו לנכון לחסוך את התתעסקות בהוספות והורדות של אחדים על ערך ההחזרה כדי לשמור את הרעיון פשוט ומובן...), כעת נותר לנו להביא את **num_of_classes** פחות מספר הכיתות שכבר הבאנו. נעשה זאת באופן דומה לאלגוריתם שבאמצעותו עברנו על עץ הכיתות, כעת נעבור על עץ הקורסים. רק שהפעם בכל צומת שנבקר (כל קורס) נעבור על מערך הכיתות שלו, אם מספר הצפיות של הכיתה הוא אפס נוסיף אותה למערך שלנו ונוריד אחד ממונה הכיתות שעלינו להביא. אם מספר הצפיות שונה מאפס, הכיתה הופיעה בעץ הכיתות ולכן כבר בהכרח נמצאת בין הכיתות שהבאנו. אנחנו במקרה הגרוע ביותר עוברים על מספר הכיתות שהתבקשנו להביא פעמיים, מאחר והמקרה הגרוע ביותר הוא שאנחנו רוצים להביא את האיבר האחרון במערך בלבד (כלומר כל הכיתות שהופיעו לפניו בעלות מספר חיובי של צפיות) בכל מקרה, במקרה הגרוע ביותר אנחנו עוברים שוב על מספר הכיתות שצריך להביא, כי לכל כיתה שאנחנו עוברים עליה במערך בין אם יש לה אפס צפיות ואז נוסיף אותה ובין אם יש לה יותר מאפס צפיות ואז לא נוסיף אותה(כי כבר הוספנו אותה מעץ הכיתות) אחרי מספר הכיתות המבוקש נסיים.

void Quit(void **DS):

על מנת לשחרר את מבנה הנתונים עלינו למחוק את שני העצים, פשוט נסייר בכל עץ ונמחוק אותו, עשינו זאת בסדר של **post-prder**. לכל קורס נניח **k** כיתות ושחרור של כל כיתה נעשה על ידי מספר קבוע של פעולות לכן נקבל כי על מנת לשחרר את עץ הקורסים נבצע סדר גודל של **kn** פעולות , שזה במילים אחרות בסדר גודל של מספר השיעורים במערכת. כלומר .**O(m)** במידה ומספר הכיתות עבור כל קורס הוא 1 אז נקבל כי הסיבוכיות היא מסדר גודל .**O(n)**

על מנת לשחרר את עץ הכיתות נצרטך לבצע מספר קבוע של פעולות כפול מספר הכיתות בעץ שזהו מספר השיעורים במערכת לכן מדובר בסדר גודל של **O(m)**.

סה"כ נקבל כי הסיבוכיות היא מסדר גודל **O(m+n)**.

<u>סיבוכיות מקום:</u>

אנחנו מחזיקים 2 עצים, עץ הכיתות הוא בגודל הכיתות שיש להן יותר מ0 צפיות, לכן גודלו חסום על ידי **m** שזה מספר כלל הכיתות במערכת. גודל הטיפוס בעץ זה הוא 3 אינטים, ואנו שומרים מצביע לטיפוס ו3 מצביעים עבור בן שמאלי בן ימני ואב, לכן סהכ סיבוכיות המקום עבור עץ הכיתות היא לכל היותר **O(m)**.

עבור עץ הקורסים אנחנו מחזיקים **n** צמתים כאשר **n** הוא מספר הקורסים במערכת. לכל קורס אנחנו מחזיקים מערך של הכיתות, מלבד המערך אנחנו מחזיקים מספר קבוע של משתנים לכל חוליה לכן סיבוכיות המקום עבור עץ הקורסים היא מספר הקורסים כפול מספר הכיתות בכל קורס, קרי מספר השיעורים שיש במערכת לכן סה"כ סיבוכיות המקום היא מסדר גודל **O(m)**, כאשר אם יש רק כיתה אחת לכל קורס הסיבוכיות מקום היא **O(n)**.

לכן סה"כ מדובר בסיבוכיות מקום **O(n+m)**.