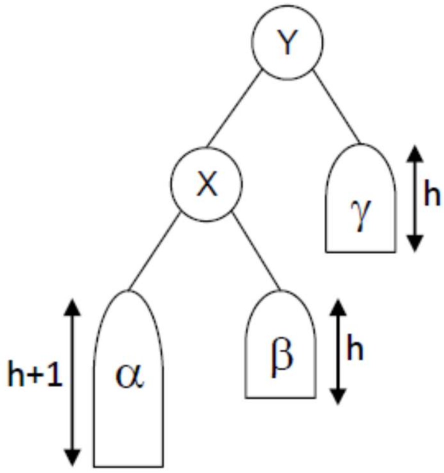
<u>שאלה מספר 3 (25%)</u>

<u>סעיף א' (12%)</u>: נתון עץ AVL הבא, מיד אחרי פעולת Insert אבל (כפי שרואים) לפני פעולת תיקון העץ.



נניח שבעץ הנ"ל בכל קודקוד יש גם שדה Size (השומר את גודל תת העץ ששורשו בקודקוד). נסמן את הגודל של העצים α, β, γ ב - S_γ, S_β, S_α , בהתאמה, וכן ב- S_x וב- S_y את הגודל של העצים ששורשיהם X ו- Y בהתאמה (כל הגדלים הנ"ל <u>אחרי</u> פעולת ה- Insert <u>ולפני</u> תיקון העץ). מה יהיה השדה Size[Y] <u>אחרי</u> פעולת ה-Insert ופעולת תיקון העץ?

1) $S_y + 1$

2) $S_y + 2$

3) $S_\alpha + S_\beta + 1$

4) $S_\alpha + S_\beta + 2$

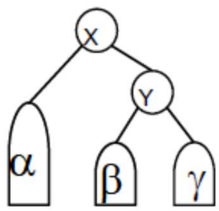
5) $S_\beta + S_\gamma + 1$

6) $S_\beta + S_\gamma + 2$

<u>תשובה לשאלה 3</u>

<u>סעיף א'</u>:

העץ אחרי הרוטציה יראה כמו בציור למטה. לכן הגודל של תת העץ ששורשו Y הוא הגודל של β ועוד הגודל של γ ועוד אחד (בשביל השורש).



<u>סעיף ב' (13%)</u>: נתונה ערימת מינימום בגודל n, ונתון מספר טבעי $1 < k < n$. רוצים להדפיס את k האיברים הקטנים ביותר בערימה. תאר אלגוריתם העושה זאת בזמן $O(k \log k)$. רמז: אפשר להשתמש בערימה נוספת.

<u>סעיף ב':</u>

האלגוריתם:

נחזיק ערימת עזר. בערימת העזר נחזיק חלק מהמפתחות של הערימה המקורית, עם מצביעים לצמתים המתאימים בערימה המקורית. בהתחלה נעתיק לערימת העזר את שורש הערימה המקורית.

בכל שלב נבצע *ExtractMax* מערימת העזר. נדפיס את האיבר שיצא x. את שני בניו בערימה המקורית נכניס לערימת העזר. נחזור על הנ"ל עד שהודפסו k איברים.

הסבר נכונות:

אם המפתח של צומת הוא לא בין k המפתחות הגדולים, אז לא צריך להדפיס אותו, ובהכרח גם את בניו לא צריך להדפיס. נשים לב שבכל שלב ערימת העזר מכילה את כל האיברים הפוטנציאלים להדפסה, כלומר את כל הבנים (מהערימה המקורית) של הצמתים שהודפסו. כל צומת שאביו יודפס – יכנס בעצמו לערימה, וכל צומת שאביו לא יודפס – לא יכנס לערימה. לכן האלגוריתם ידפיס את k הערכים הגדולים ביותר, ואפילו בסדר יורד.

ניתוח זמן ריצה:

בכל שלב מוציאים מערימת העזר איבר אחד ומכניסים במקומו שניים. כלומר גודל ערימת העזר הוא לכל היותר k+1. לכן זמן פעולה בסיסית על הערימה הוא $O(\lg k)$.

מציאת שני הבנים מבוצעת בזמן $O(1)$ (כי יש מצביע לאיבר המתאים בערימה המקורית). סה"כ $O(k)$

בכל שלב של הריצה מבצעים שלוש פעולות על ערימת העזר (פעם אחת Extract ופעמיים Insert). יש k שלבים כאלה. סה"כ $O(k\lg k)$.

לכן סה"כ זמן ריצה $O(k\lg k)$.

נשים לב שב-k/2 השלבים האחרונים הערימה כבר בגודל לפחות k/2, לכן זמן פעולה בסיסית על הערימה הוא $\Omega(\lg k)$, כלומר זמן הריצה הכולל הוא גם $\Omega(k\lg k)$, ולכן $\Theta(k\lg k)$.

<u>סעיף א' (12%)</u>: סמן **נכון** / **לא נכון** : בעץ AVL ההפרש בעומק בין שני עלים כלשהם הוא לכל היותר 1.

לא נכון

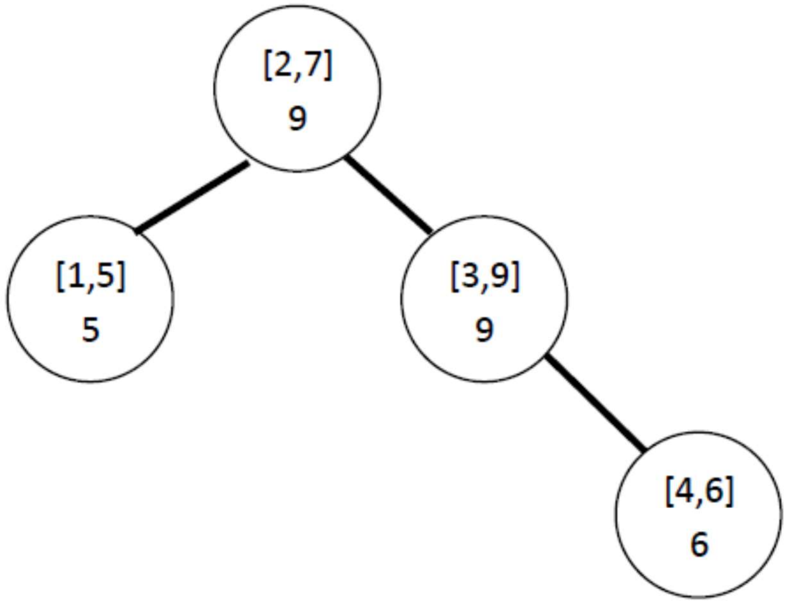
שאלה מספר 1 (20 נקודות)

סעיף א (13 נקודות):

נייצג אינטרוול $x = [a,b]$ על הישר על ידי שני מספרים ממשיים a,b

כעת בהינתן קבוצה של אינטרוולים בעלי ערכי a **שונים** זה מזה נבנה עץ חיפוש בינארי (עץ אינטרוולים) באופן הבא: נכניס את האיברים לעץ כאשר המיון יתבצע לפי השדה a. בנוסף לכל צומת v בעץ נשמור שדה נוסף max אשר שומר בתוכו את הערך המקסימלי של השדה b מבין כל השדות בתת העץ המושרש ב - v (כולל v).

לדוגמא : בהינתן האינטרוולים הבאים $\{[2,7],[3,9],[4,6],[1,5]\}$ בהכנסה משמאל לימין יווצר העץ הבא:



בהינתן עץ חיפוש בינארי כנ"ל T ואינטרוול נוסף $x = [a,b]$, תארו אלגוריתם לבדיקה האם קיים אינטרוול בעץ שהיתוכו עם x אינו ריק בזמן $O(h)$ כאשר h הוא גובה העץ.

<u>סעיף ב' (7 נקודות):</u>

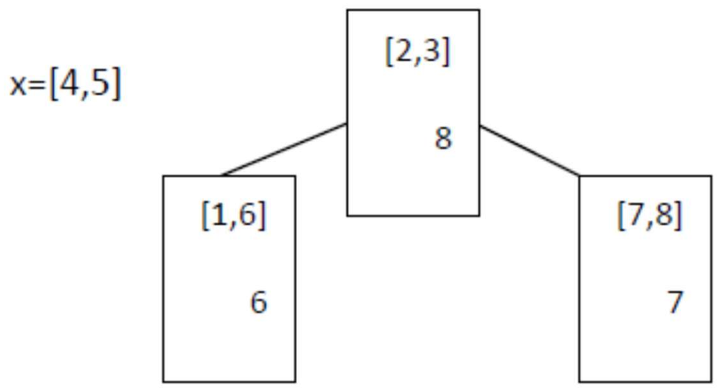
כעת נמחק מכל קודקוד בעץ האינטרוולים את השדה max. **תנו דוגמא** לעץ אינטרוולים T ולאינטרוול נוסף [a',b']=x כך שקיים חיתוך בין x לבין אחד מהאינטרוולים בעץ, אך קטע הקוד הבא לא ימצא אותו.

```
Search(T,x)
y←root(T)
if (y==null) return "not found"
if (x.a≥y.a)
{
   if (x.a ≤ y.b)
       return y
   else
       Search(right(T),x)
}
else
      Search(left(T),x)
```

סעיף א תשובה

```
Intersection(v,x):
 If v = NULL return false
 If b[x] ≥ a[v] and a[x] ≤ b[v] return true
 If left[v] ≠ NULL and a[x] ≤ max[left[v]]
   return Intersection(left[v],x)
 else
   return Intersection(right[v],x)
```

סעיף ב תשובה



<u>שאלה מספר 3 (28 נקודות):</u>

נתונה קבוצה S של n מספרים **<u>שלמים שונים זה מזה</u>**, תאר מבנה נתונים עבור S ואת הפעולות הבאות בזמנים המבוקשים (עליך להסביר גם את זמני הריצה בעת תיאור הפעולות):

פעולה	**תיאור**	**זמן**
<u>Init(S)</u>	בניית מבנה נתונים מקבוצה S המכילה n מספרים	**O(n) במקרה הגרוע**
<u>extractMax()</u>	מחיקת מספר מקסימאלי מהמבנה	**O(log n) בממוצע**
<u>extractMin()</u>	מחיקת מספר מינימאלי מהמבנה	**O(log n) בממוצע**
<u>Insert(k)</u>	הכנסת מספר חדש k למבנה. ניתן להניח ש-k לא קיים במבנה.	**O(log n) בממוצע**
<u>Delete(k)</u>	מחיקת מספר k מהמבנה.	**O(log n) בממוצע**

<u>שימו לב:</u> [O(g(n)+f(n)) בממוצע] = [O(f(n)) בממוצע] + [O(g(n)) במקרה הגרוע]

הקבוצה S מכילה n מספרים שלמים ושונים זה מזה, אין כל ידע על הטווח של המספרים או ההתפלגות שלהם, ולכן לא ניתן להשתמש בשום סוג של מיון ליניארי.

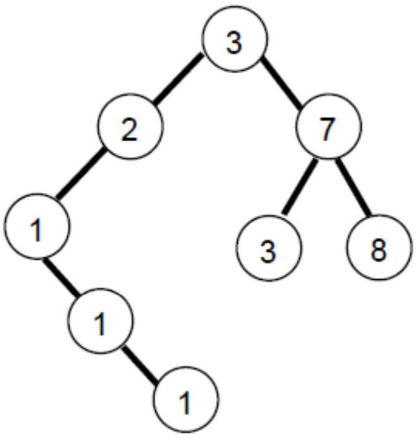
אנו מתחילים בהעתקת האיברים לשני מערכים ולאחר מכן, בניית שתי ערימות מינ'/מקס' בעזרת **buildHeap** , כפי שהוכח בכיתה זמן הריצה הוא O(n) (בנייה דינמית של ערימה בהכנסות היא כמובן O(n*Log(n)) ולא תעבוד כאן).

בעזרת הערימות נוכל לבצע הוצאת מקס'/מינ' בזמנים המבוקשים, אך לא נוכל לבצע מחיקה של איבר בזמן המבוקש (**ערימה היא לא עץ חיפוש בינארי**) , לכן אנו ניעזר גם בטבלת גיבוב. כדי לעמוד בזמן הריצה של האתחול אנו מחויבים להשתמש בשיטת ה – chaining כדי שנוכל להכניס כל איבר בראש הרשימה בתא המתאים לו ב – O(1) במקרה הגרוע. לכל איבר בטבלת הגיבוב נחזיק מצביעים לאיברים בערימות כדי שנוכל למחוק איברים בזמן המבוקש (החיפוש בטבלה יהיה O(1) בממוצע). צריך לשים לב שאם נכניס איברים לטבלת הגיבוב לפני שנבנה את הערימות ונקשר בין האיברים, נדרש לבצע חיפוש בטבלה לכל איבר כדי לעדכן את מצביעיו ושוב לא נוכל לעמוד בזמן הריצה של האתחול (במקרה בקרוע).

<u>סעיף ג' (10 נקודות):</u>

נתון עץ בינרי T המכיל n מספרים שלמים. בהינתן מספר x המופיע ב T, נגדיר את הריבוי של x בתור מספר המופעים של x ב-T. רוצים לייצר מערך ממוין B בגודל $\leq n$, שיכיל את הריבויים של המספרים המופיעים ב T.

דוגמה:



בעץ T, הריבוי של 1 הוא 3, הריבוי של 8 הוא 1, הריבוי של 3 הוא 2, הריבוי של 2 הוא 1, הריבוי של 7 הוא 1.

מערך הריבויים הממוין הוא B :

3	2	1	1	1

הציעו אלגוריתם שפותר את הבעיה בזמן O(n).

נעבור בסריקת inorder על העץ, ונשים את המפתחות המתקבלים מהסריקה (ממוינים מהקטן לגדול) לתוך מערך A. על מנת לקבל את הריבויים נשתמש במונה שמאותחל ל- 1. נשתמש במערך נוסף B לשמירת הריבויים. עוברים על המערך A, אם התא הנוכחי שווה לתא הקודם נגדיל את המונה ב-1, אם התא הנוכחי שונה מהתא הקודם נשמור את ערכו של המונה בתא הפנוי הבא ב- B, ונאתחל את המונה שוב ל-1. ברגע שיש לנו את הריבויים בתוך מערך B, נשתמש בעובדה שהריבויים הם מספרים שלמים בתחום [1..n] ונמיין אותם בזמן ליניארי ע"י מיון מנייה.

<u>סעיף ג' (18 נקודות): (אין קשר לסעיפים הקודמים)</u>

נתון מערך בגודל n המכיל מספרים ממשיים. יש לבדוק האם קיימים במערך 2 ערכים **<u>שמספר המופעים</u>** של שניהם יחד הוא **בדיוק** 2011. הציעו אלגוריתם הפותר את הבעיה בזמן ריצה **ממוצע** $\Theta(n)$. מספר המופעים של ערך הוא מספר הפעמים שהוא מופיע ברשימת הקלט.

ג. לסעיף זה יש מספר פתרונות. כולם מלבד המסובכים ביותר דורשים שימוש בטבלת גיבוב. עבור מערך מקורי A נשתמש בטבלת גיבוב H עם שרשור שתשתמש בפונקציית גיבוב h. גודל הטבלה יהיה $m = O(n)$. כל איבר בטבלת הגיבוב יכיל בנוסף לערכו גם שדה count שיציין את מספר הפעמים שהאיבר נמצא ב-A. בנוסף נשתמש במערך עזר B בגודל 2010 שהאינדקסים שלו הם בין 1 ל-2010. האלגוריתם יפעל בצורה הבאה:

- **נעבור על A ולכל איבר A[i] נחפש אותו ברשימה H[h(A[i])]. אם לא מצאנו, ניצור ברשימה H[h(A[i])] איבר חדש בעל ערך A[i] ו-count של 1. אם מצאנו אז נוסיף ל-count של האיבר שמצאנו 1.**

  זמן ריצה: $O(n)$ במקרה הממוצע.
- **נאתחל את תאי B לאפסים בזמן $O(1)$.**
- **נעבור על כל התאים בכל הרשימות שב-H ולכל תא x:**

  אם x.count<2011 נבצע: $B[x.count] \leftarrow 1$

  זמן ריצה: $O(n)$ במקרה הגרוע.
- **נרוץ עם i על הערכים 1 עד 1005. אם נמצא i עבורו $B[i] = B[2011-i] = 1$ נחזיר *כן*. אם לא מצאנו נחזיר *לא*.**

  זמן ריצה: $O(1)$ במקרה הגרוע.

סה"כ זמן ריצה: $O(n)$ במקרה הממוצע.

סעיף ג' (15 נקודות)

תארו אלגוריתם אשר בהינתן 2 עצי AVL (שסכום מספר צמתיהם הוא n) מייצר עץ AVL יחיד המכיל את כל n אברי שני העצים. ניתן להשתמש ב $O(n)$ זיכרון נוסף וזמן הריצה צריך להיות $O(n)$ במקרה הגרוע ביותר. נתחו את זמן הריצה של האלגוריתם.

סעיף ג'

נסרוק את העץ הראשון בסריקת inorder **ונשמור את התוצאה במערך עזר** A – $O(n)$ – A **ממויין.**

נסרוק את העץ השני בסריקת inorder **ונשמור את התוצאה במערך עזר** B – $O(n)$ – B **ממויין.**

מיזוג המערכים A **ו-**B **למערך עזר** C **בדומה למיזוג של** Mergesort – $O(n)$ – C **ממויין ומכיל את כל** n **האיברים.**

בניית עץ AVL **ע"פ האלגוריתם הרקורסיבי הבא:**

שורש – איבר אמצעי במערך - $O(1)$

בנה באופן רקורסיבי תת-עץ שמאלי של השורש מחלק C[1..n/2-1] **של המערך.**

בנה באופן רקורסיבי תת-עץ ימני של השורש מחלק C[n/2+1..n] **של המערך.**

זמן ריצה של אלגוריתם רקורסיבי: $T(n) = 2T(n/2)+O(1)=O(n)$

שאלה 4 (30 נקודות)

ממשו מבנה נתונים ששומר איברים שונים כאשר לכל איבר x יש את השדות הבאים:

- color[x] - מספר שלם בין 1 ל 6 המציין את צבע האיבר.
- Priority[x] - מספר שלם המציין את העדיפות של האיבר.

הפעולות על מבנה הנתונים הן

פעולה	תיאור	זמן ריצה במקרה הגרוע
init()	אתחול מבנה הנתונים כמבנה ריק	$O(1)$
insert(x)	הכנסת האיבר x למבנה הנתונים	$O(\log n)$
extractMax()	הוצאת האיבר בעל העדיפות הגבוהה ביותר מתוך מבנה הנתונים	$O(\log n)$
increase(c,p)	הגדלת העדיפות של כל האיברים הצבועים בצבע c אשר נמצאים בתוך מבנה הנתונים ב p	$O(\log n)$

כאשר n הוא מספר האיברים במבנה.

תארו בקצרה את מבנה הנתונים, ספקו אלגוריתם לכל אחת מהפעולות ונתחו בקצרה את זמן הריצה של כל אלגוריתם.

שאלה 4

מבנה:

6 ערימות מקסימום. אחת לכל צבע. המפתח – העדיפות.

6 משתנים. אחד לכל צבע. מאותחלים ב- 0. לשמירת שינויי העדיפות המצטבר לצבע.

פעולות:

init – מאתחלים הכול ב- O(1).

insert(x) – מורידים מהעדיפות של x את שינוי העדיפות המצטבר לצבע של x. מכניסים את x לערימה של הצבע של x. O(1+logn) = O(logn).

extractMax() – בוחנים את האיברים המקסימליים של כול הערימות ע"פ העדיפות הרשומה שלהם פלוס שינוי העדיפות המצטבר של הצבע שלהם ושולפים ומחזירים את בעל העדיפות המקסימלית אחרי שמוסיפים לעדיפות הרשומה שלו את שינוי העדיפות המצטבר לצבע שלו. O(6 + 6 + logn) = O(logn).

increace(c, p) – מוסיפים לשינוי העדיפות המצטבר של הצבע c את p. O(1).

הערה:

אפשר היה גם להשתמש בעץ AVL ולקבל את התוצאה.