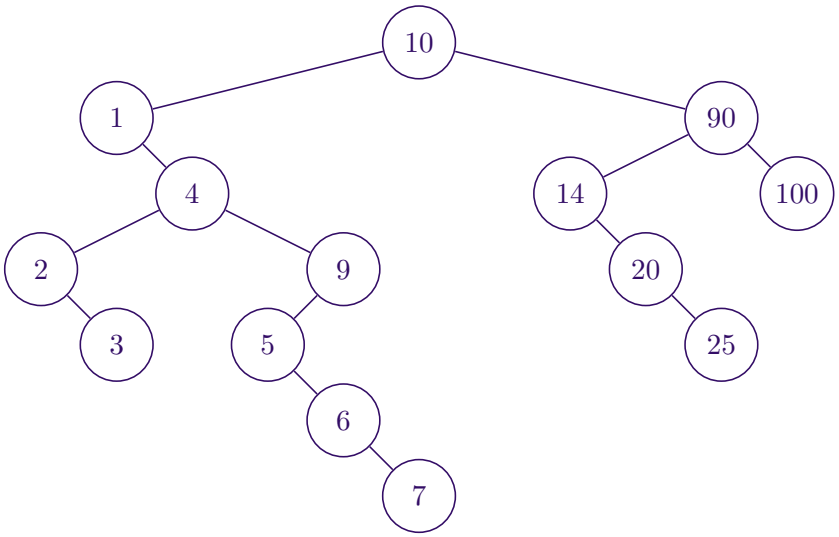
מבני נתונים
תרגיל 5 - פתרונות

1. בנו עץ חיפוש בינארי בעזרת השיטה להוספת קודקודים שלמדנו בכיתה. סדר הכנסת הקודקודים הוא:

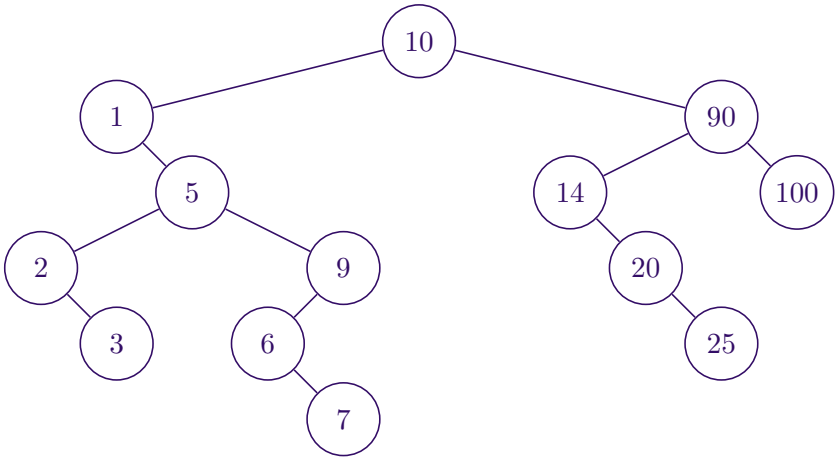
$$10, 1, 4, 9, 5, 2, 6, 90, 14, 20, 3, 25, 100, 7$$

מה תגרום מחיקת הקודקוד 4?

פתרון:



לאחר המחיקה:



2. נתון האלגוריתם המקבל שורשים של שני עצים:

```
boolean func(Node t1, Node t2)
{
  if (t1 == null || t2 == null)
    return (t1 == null && t2 == null);
  return func(t1.getLeft(), t2.getLeft()) &&
         func(t1.getRight(), t2.getRight());
}
```

מה האלגוריתם עושה? מה הסיבוכיות שלו?

פתרון: האלגוריתם בודק האם המבנה של שני עצים הוא זהה לחלוטין. אבל הוא מתעלם לגמרי מהערכים שנמצאים בקודקודי העצים. הסיבוכיות שלו היא לינארית בגודל הקלט - כי במקרה הגרוע - בו שני העצים הם באמת זהים, הוא עובר על קודקודי כל אחד מהעצים.

3. כתבו אלגוריתם רקורסיבי המחשב את עומקו של עץ בינארי - מספר הקודקודים המקסימלי במסלול מהשורש עד לעלה.

פתרון:

```
int depth(Node t)
{
  if (t == null)
    return 0;
  return 1 + Math.max(depth(t.getLeft()), depth(t.getRight());
}
```

4. כתבו אלגוריתם רקורסיבי הבודק הם עץ בינארי הוא מלא - עץ הוא מלא אם לכל קודקוד שאינו עלה יש שני בנים, וכל עלי העץ הם באותו עומק.

פתרון: כדי לפתור, נכתוב שיטה שבודקת הם עץ הוא מלא ומחזירה את עומקו של העץ. נשתמש בערך החזרה כדי להשוות את עומקם של שני התת עצים שלנו. השיטה תחזיר -1 אם העץ אינו מלא.

```
int isFull(Node t)
{
  if (t == null)
    return 0;
  int s1 = isFull(t.getLeft());
  int s2 = isFull(t.getRight());
  if (s1 == -1 || s2 == -1 || s1 != s2)
    return -1;
  return s1 + 1;
}
```

5. כתבו את השיטה המקבלת קודקוד בעץ ומחזירה את הקודקוד המחזיק את הערך הכי קרוב אליו וקטן ממנו - כלומר את הקודם של הקודקוד הנתון אילו היינו מדפיסים את העץ בצורה ממוינת.

פתרון:

```
Node prev(Node t)
{
  if (t.getLeft() != null) {
    for (Node p = t.getLeft(); p.getRight() != null;
         p = p.getRight());
    return p;
  }
  for (p = t; p.getFather().getLeft() == p; p = p.getFather());
  return p.getFather();
}
```

6. כיתבו אלגוריתם המקבל שורשי שני עצים $t1$, ו-$t2$, וממזג אותם, כלומר מחזיר שורש של עץ חדש שמכיל את קודקודי שני העצים יחד.

יש להניח שכל הערכים בעץ $t1$ קטנים מכל הערכים בעץ $t2$. עובדה זו יכולה להקל מאוד על האלגוריתם.

מצאו אלגוריתם שמבצע מיזוג זה בזמן קצר ככל האפשר, ושעומק העץ שהוא יוצר הוא לכל היותר המקסימום בין עומקי שני העצים המקוריים פלוס 1. האלגוריתם שלכם יכול לשבש את העצים המקוריים אם הוא צריך.

פתרון: הרעיון הוא למצוא את המקסימום של העץ הראשון ולשים אותו בתור שורש. אז נשים את העץ הראשון כולו משמאלו ואת העץ השני כולו מימינו וקיבלנו מה שרצינו. זמן הריצה יהיה $O(d)$ כאשר d הוא עומקו של העץ הראשון.

יש כל מיני מקרי קצה. למשל אם המקסימום הוא כבר בשורש, אבל אז זה אומר שלשורש אין בן ימני, אז אפשר לחבר את העץ השני שם וסיימנו.

איך מנתקים את קודקוד המקסימום מהעץ? נשים לב שיש לו רק בן שמאלי, ולכן צריך לקחת את אבא שלו ובמקום שישב המקסימום, לשים במקומו את הבן השמאלי של המקסימום.

```
Node merge(Node t1, Node t2)
{
  Node newRoot = t1.max();

  if (newRoot.getFather() == null) {
     // newRoot is the root of t1.
    newRoot.setRight(t2);
    return newRoot;
  }

  // remove the newRoot node from the t1 tree.
  if (newRoot.getFather().getLeft() == newRoot) // newRoot was left child
    newRoot.getFather().setLeft(newRoot.getLeft());
  else // new Root was right child
    newRoot.getFather().setRight(newRoot.getLeft());

  newRoot.setLeft(t1);
  newRoot.setRight(t2);
  return newRoot;
}
```

7. כתבו אלגוריתם רקורסיבי הבודק האם עץ בינארי הוא עץ חיפוש חוקי.

הדרכה: לשיטה שלכם תהיה חתימה:

```
boolean check(Node t, int lower, int upper)
```

כאשר היא תחזיר $true$ רק אם כל קודקודי העץ ששורשו t נמצאים בטווח בין $lower$ ו־$upper$.

הקריאה הראשונה תהיה

```
check(Node t, -infinity, +infinity)
```

כלומר על השורש אין הגבלה. חישבו למשל איזה מגבלה יש על הבן הימני של השורש, ואיזה תהיה על הבנים שלו, הן הימני והן השמאלי.

פתרון:

```
boolean check(Node t, int lower, int upper)
{
  if (t == null)
    return true;
  if (t.getValue() < lower || t.getValue() > upper)
    return false;
  return check(t.getLeft(), lower, t.getValue()) &&
         check(t.getRight(), t.getValue(), upper);
}
```

8. **בונוס:** השתמשו במחסנית וכתבו אלגוריתם שאינו רקורסיבי להדפסת עץ בינארי בצורה ממוינת. כדי להקל, הניחו שלכל קודקוד יש משתנה בוליאני שאתם יכולים לסמן בעזרתו קודקודים שטיפלתם בהם. כלומר, בהנתן קודקוד n אתם יכולים להפעיל את השיטה:

```
n.setMark();
```

שמסמנת אותו. בנוסף ניתן להפעיל את השיטה:

```
n.isMarked();
```

שבודקת האם הוא מסומן. הניחו שבתחילה כל הקודקודים אינם מסומנים.

פתרון:

```
void printSorted(Node t)
{
  Stack s = new Stack();
  s.push(t);
  while(!s.isEmpty()) {
    Node n = s.pop();
    if (!n.isMarked()) {
      if (n.getRight() != null)
        s.push(n.getRight());
      s.push(n);
      n.setMark();
      if (n.getLeft() != null)
```

```
        s.push(n.getLeft());
    } else
      print n.getValue();
  }
}
```