1. ננסה לבנות עצים המכילים את סדרות המספרים המדוברות:

שני העצים תקינים.

1. נכתוב פונקציה רקורסיבית, המקבלת כפרמטר את שורש העץ (או צומת כלשהי), ואינה מחזירה ערכים:

print\_tree(x) {

print key[x]

if left-child[x] != nil

print\_tree (left-child[x])

if right-sibling[x] != nil

print\_tree (right-sibling[x])

}

ניתוח סיבוכיות: הפונקציה מבצעת מספר קבוע של פעולות על כל צומת בעץ, ולכן הסיבוכיות היא .

1. אמנם סיבוכיות הפונקציה tree\_successor היא log(n), ואנו קוראים לה n פעמים, אך בפועל מה שמתבצע הוא מעבר על כל הצמתים בעץ עם הפקודות: הדפס את תת העץ השמאלי, הדפס את ערך הצומת, הדפס את תת העץ הימני. זה בדיוק כמו הפונקציה in\_order שסיבוכיותה .
3. מחיקת המספר 90:
4. מחיקת המספר 0:
5. מחיקת המספר 50:
6. מחיקת המספר 4:
7. מחיקת המספר 5:
8. מחיקת המספר 12:

leaves(x) {

if x = nil

return 0

else

return 1 + leaves(left[x]) + leaves(right[x])

}

הפונקציה רצה על כל הצמתים בעץ ומבצעת בכל אחד מהם מספר קבוע של פעולות, ולכן סיבוכיותה היא .

length[x] {

if left[x] = nil OR key[left[x]] < key[x]

y ← 0

else

y ← length[left[x]]

if right[x] = nil OR key[right[x]] < key[x]

z ← 0

else

z ← length[right[x]]

if y > z

return 1 + y

else

return 1 + z

}

סיבוכיות: במקרה הגרוע (לדוגמא: ערימת-מינימום) הפונקציה צריכה לעבור על כל הצמתים בעץ, ולכן סיבוכיותה היא .

1. (כתבנו את הפונקציה לפי איך שהבנו את ההגדרה "עצים איזומורפיים", בתקווה שלא טעינו…)

הפונקציה הרקורסיבית מקבלת שני צמתים לבדיקה האם תתי העצים שלהם איזומורפיים.

isomorphic(x, y) {

if rank[x] != rank[y]

return false

if rank[left[x]] = rank[left[y]] {

if isomorphic(left[x], left[y])

AND isomorphic(right[x], right[y])

return true

else

return false

} else {

if isomorphic(left[x], right[y])

AND isomorphic(right[x], left[y])

return true

else

return false

}

}

לפונקציה זו אין צורך לבדוק פעמיים תת-עץ אחד, בגלל התנאי ההתחלתי "if rank[left[x]] = rank[left[y]]", ולכן הסיבוכיות שלה היא .