## שאלה 3

### A

ב- fibonacci הסיבוכיות זמן ריצה היא O(n) ובnaive\_fibonacci היא O(2^n).  
הסביבה היא ש Fibonacci משתמש ב- memoization – כלומר הוא שומר חישובים של מספרים שהוא כבר חשב.  
נקח דוגמא קטנה, כדי לחשב את מספר פיבוניצי 4 נוצר העץ הבא:

על מנת לחשב את מספר פיבונצי נצטרך לדעת את שני המספרים הקודמים לו  
ולכן נוצר העץ הזה כאשר 0,1 ידועים לנו מראש, ולכן כאשר מגיעים עליהם מפסיקים  
את הירידה בעץ.  
  
במקרה שלנו בשני הפוקנציות הרקרוסיה הולכת קודם לעץ השמאלי   
(ל n-1). כלומר כאשר חישבנו כבר את תת העץ השמאלי, כחלק מהחישוב   
גם חישבנו את מספרי פיבונאצי 2,3.  
  
ההבדל בין השיטה בלי memo לעם memo, היא מה קורה כאשר מגיעים  
למספר שלא הוגדר לנו מראש אבל כבר חישבנו אותו במהלך הריצה על העץ.  
במקרה של naive\_fibonacci כאשר נעבור לחישוב תת העץ הימני נצטרך לחשב עכשיו את 2 מחדש.  
לעומת זאת ב- Fibonacci נשמור את החישוב של 2, ונוכל מיד לחזור.  
(בדוגמאות עם מספרים יותר גדולים הדבר כמובן קורה גם בתוך כל תת עץ של השורש).

### B

המטרה של המשתנה memo היא בדיוק לשמור את מספרי פיבונאצי שחישבנו כבר על מנת לא לחשב אותם שוב כאשר נתקל בהם בריצה על העץ

### C

המשתנה memo מוגדר בתוך הscope של הפונקציה ההאונונימית. אך מיכוון שהפונקציה האנונונימית מחזירה פונקציה (שמוגדר בתוכה) ומשתמש בmemo, נוצר העתק על הheap של memo. זאת מיכוון שבסיום הריצה של הפונקציה האנונמית המשתנה memo שלה שהוגדר על המחסנית נהרס, ולכן הפונקציה המוחזרה לא יכולה לגשת אליו.

### D

לאורך כל הריצה של התוכנית (או עד להחלטתו הלא צפויה של ה-GC), זאת מיכוון שאנחנו לא יודעים מתי הפונקציה תהיה בשימוש, לכן נצטרך להחזיק את memo לזמן לא ידוע מראש.

### E

1. הפונקציה האנונמית מאפשרת לנו לא לחשוף את memo לכלל התוכנית (הוא נגיד רק מתוך הפונקציה)
2. מיכוון שmemo לא נמחק כאשר אנחנו מסיימים את ריצת הפונקציה, ונוצר מחדש כאשר נכנסים אליה, נשמרים לנו כל מספרי פיבונאצי שחושבו כבר על ידי ריצות קודמות של הפונקציה.  
   לדוגמא אם הרצנו פעם ראשונה Fibonacci(10) – וחושבו כל מספרי פיבונאצי מ-2 עד 10. כעת בהרצה השניה של Fibonacci(10) לא יחושב דבר, וישר יחזור מספר פיבונאצי העשירי!

### F – בקובץ נפרד.