**מטלה 1 - מערכות הפעלה**

**215011537\_208309153\_324953900**

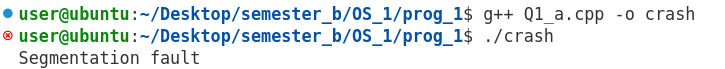
תרגיל 1:

**הרצת התרגיל :**- יש לטעון את התיקיה ע"י הזנת cd Q1 לטרמינל  
- להרצת מקרה א/ ב/ ג יש להזין לטרמינל את הפקודה : make a/ b / c (בהתאמה למקרה הרצוי)  
- להרצת מקרה א/ ב/ ג עם gdb יש להזין לטרמינל את הפקודה : make a\_gdb/ b\_gdb / c\_gdb   
- להרצת מקרה א/ ב / ג עם ddd יש להזין לטרמינל את הפקודה : make a\_ddd/ b\_ddd / c\_ddd   
- למחיקת הקבצים יש להזין לטרמינל את הפקודה : make clean  
\*\*הפקודות מורצות עם דגל -g, במידה ותרצו להריץ ללא, יש למחוק אותו מה-FLAGS שב-makefile.

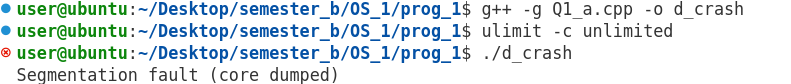
**מקרה א' -> גלישה מהמחסנית עקב רקורסיה אינסופית.**

ייצור core

פתיחת הcore- באמצעות הdebugger- ללא debug info

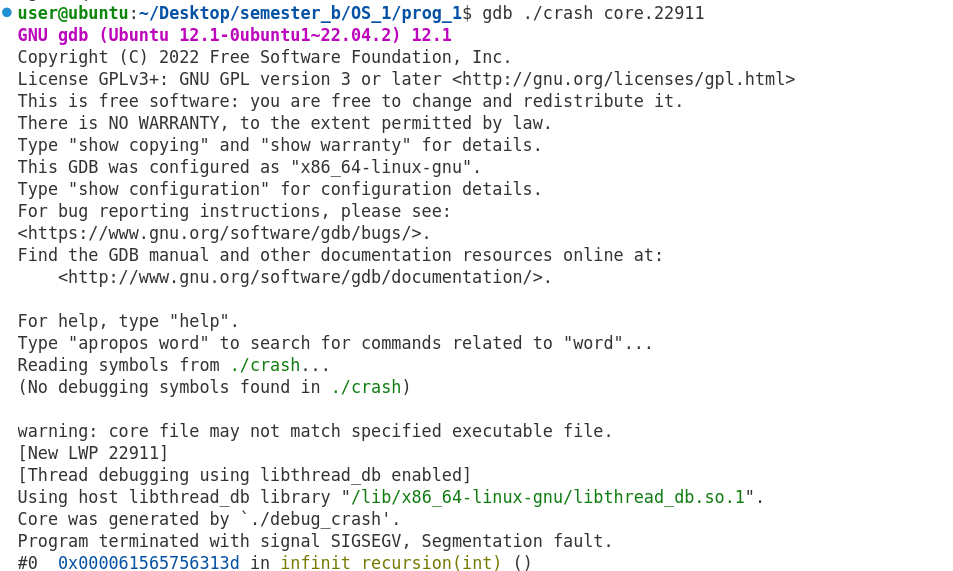


פתיחת הcore- באמצעות הdebugger- עם debug info



פתיחת הcore- בעזרת debugger טקסטואלי (gdb)

נריץ את הראשון (crash)



נמצא איפה הנפילה באמצעות where:

A screenshot of a computer code

Description automatically generated

ננסה למצוא את ערך המשתנה num ע"י print:

בעצם gdb לא מסוגל לספק את הערך של המשתנה מאחר ש-num לא מוגדר בהקשר של מחסנית קריאת הפונקציה מאחר שלא הוקצה לו ערך ספציפי לפני הקריסה.  
infinit\_recursion אינו מחזיר מספר חוקי לפני הקריסה, ולכן num למעשה אינו מאתחל או אינו נגיש.

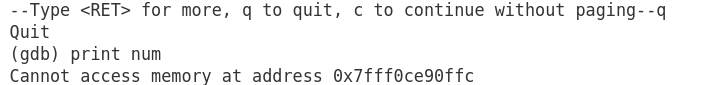
נריץ את השני (debug\_crash) –

A screenshot of a computer code

Description automatically generated

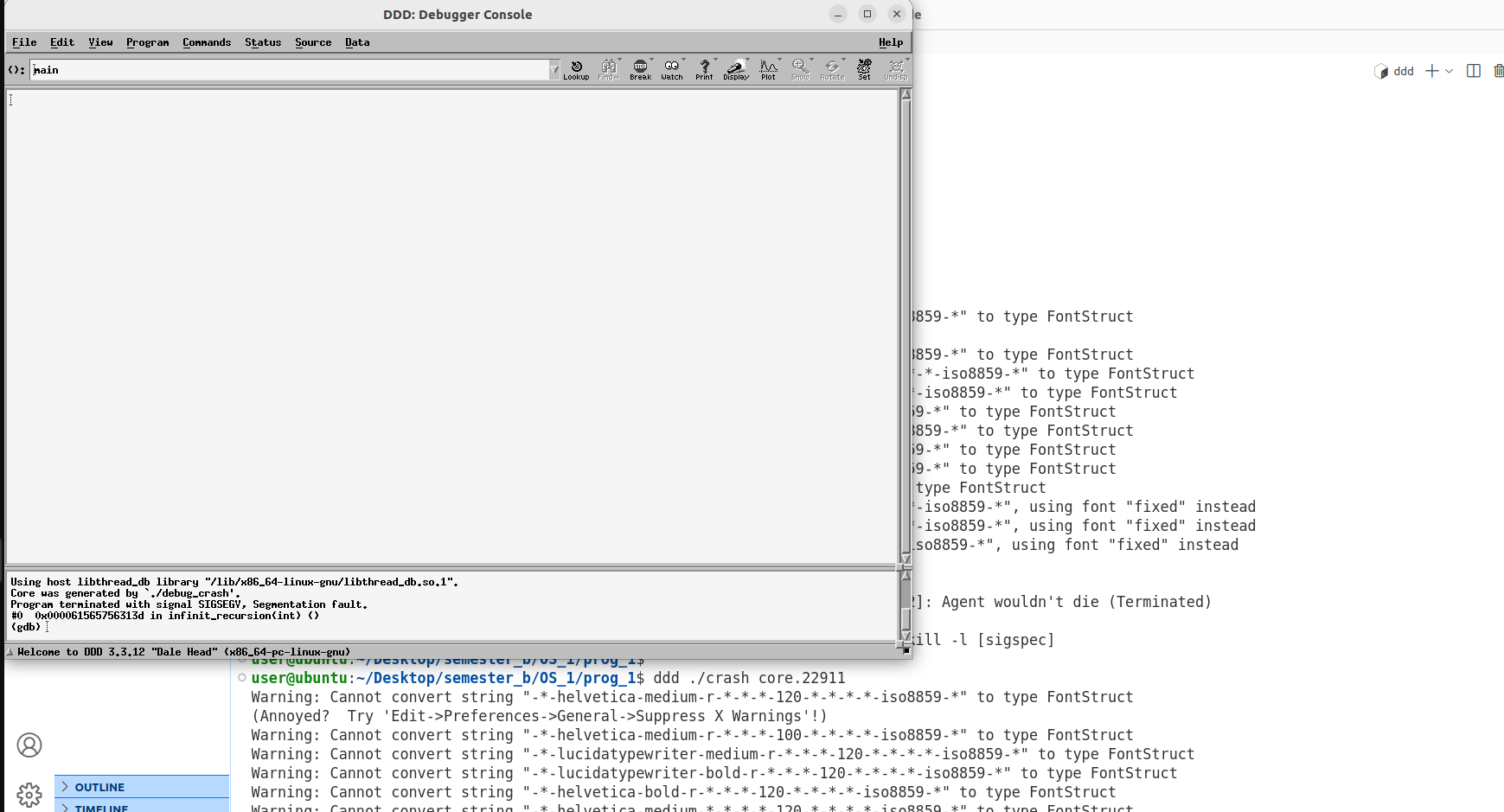
A screen shot of a computer

Description automatically generated



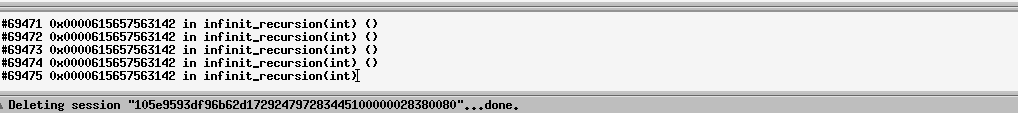
בעצם gdb לא מסוגל לספק את הערך של המשתנה מאחר ש-num לא מוגדר בהקשר של מחסנית קריאת הפונקציה מאחר שלא הוקצה לו ערך ספציפי לפני הקריסה.  
infinit\_recursion אינו מחזיר מספר חוקי לפני הקריסה, ולכן num למעשה אינו מאתחל או אינו נגיש.

פתיחת הcore- בעזרת debugger גרפי(ddd) :

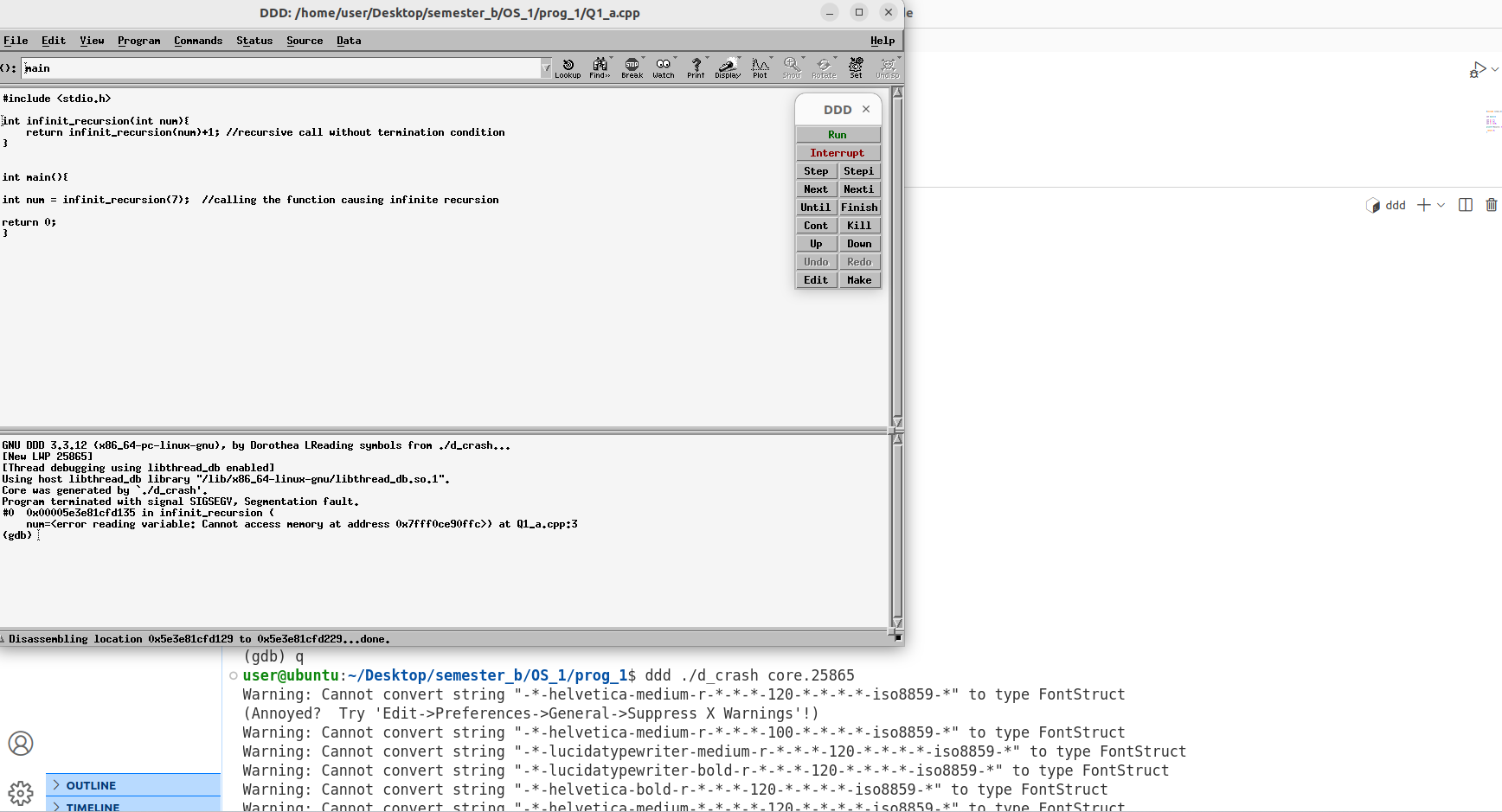


A computer screen with a white screen

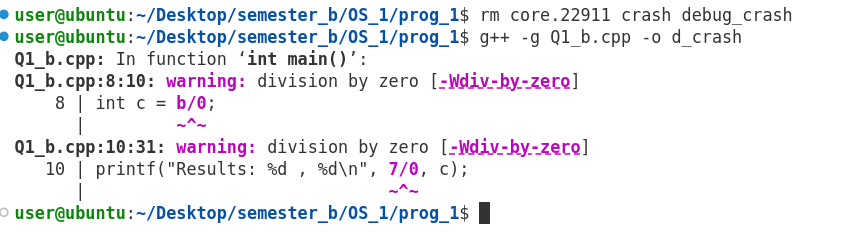
Description automatically generated

ובפקודת where אנחנו נכנסים לרקורסיה אינסופית שנראת ככה 

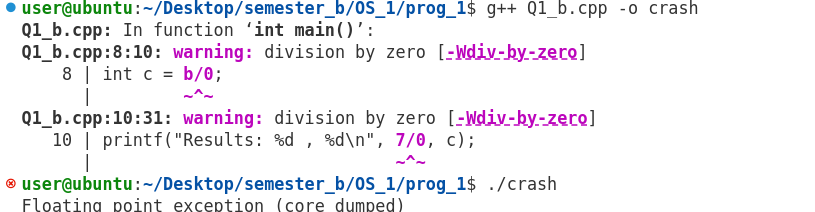
נריץ את השני (debug\_crash) –



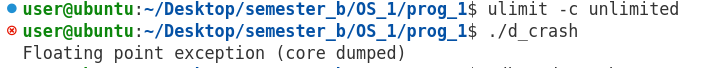
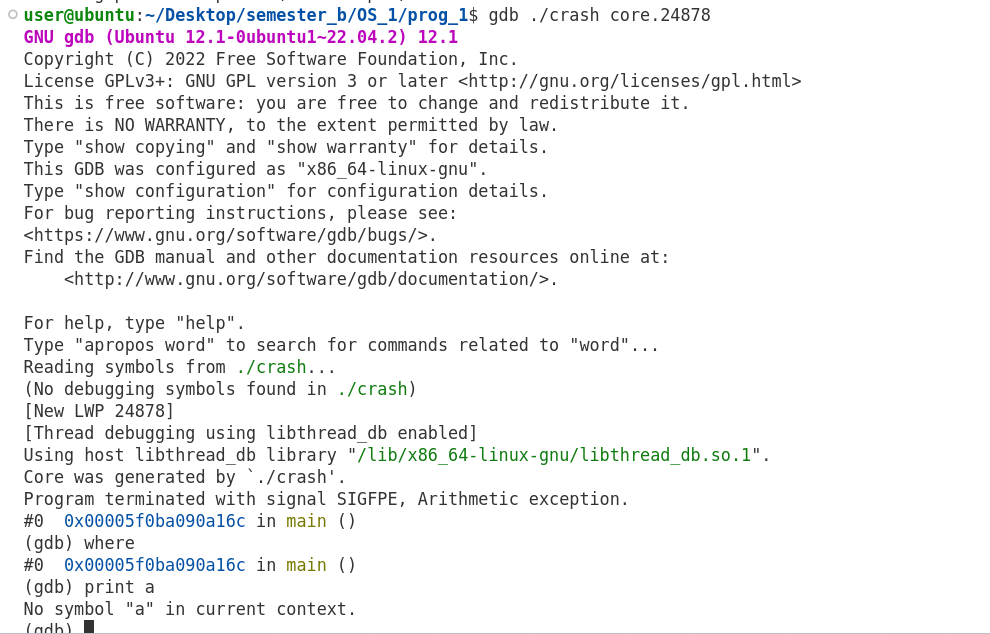
**מקרה ב' ->חלוקה ב-0.**

פתיחת הcore- באמצעות הdebugger- עם debug info 

פתיחת הcore- באמצעות הdebugger- ללא debug info



פתיחת הcore- בעזרת debugger גרפי

נמצא איפה הנפילה באמצעות where:

A close-up of a computer screen

Description automatically generatedA screenshot of a computer program

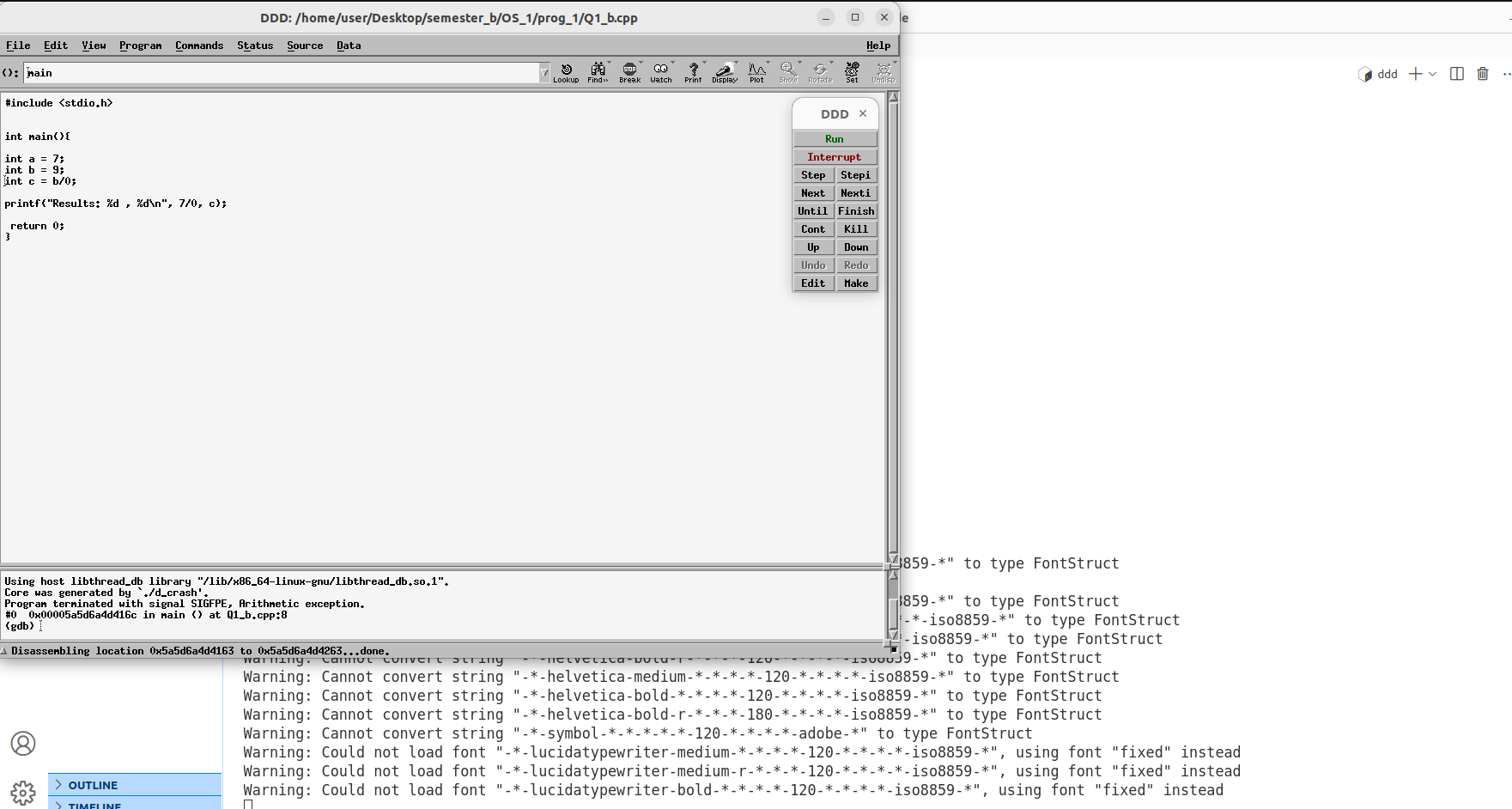
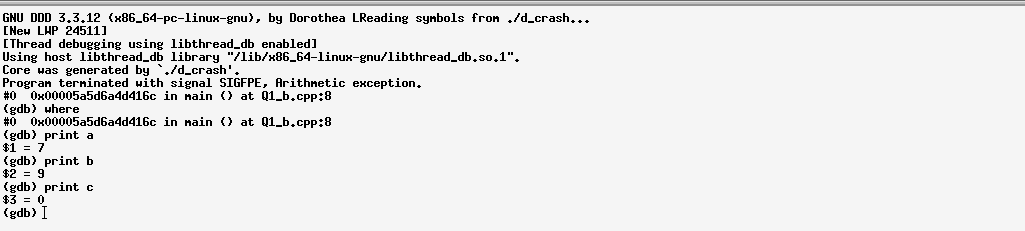
Description automatically generated

נמצא את ערכי הפרמטרים באמצעות print: A close-up of a computer screen

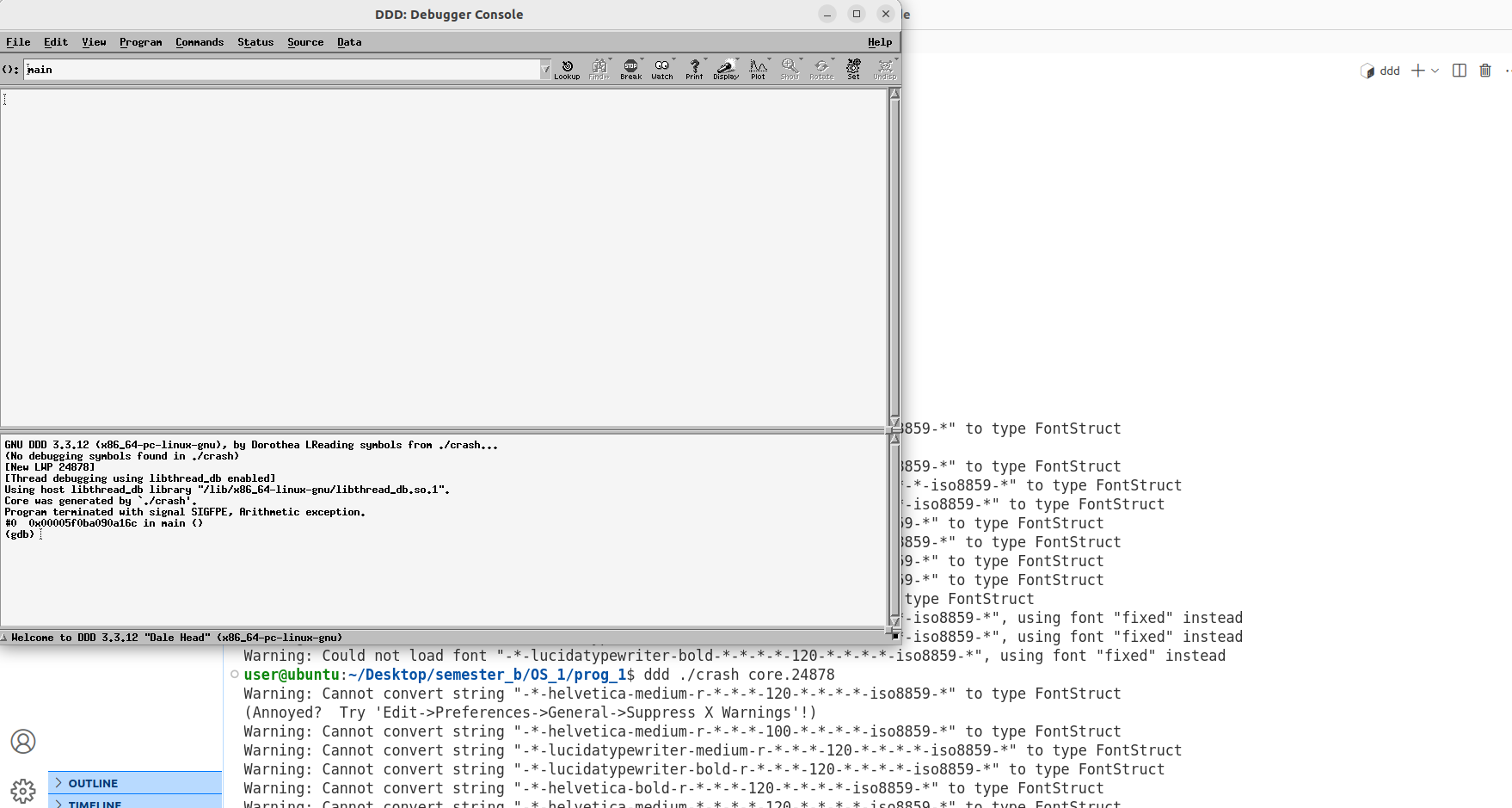
Description automatically generated

פתיחת הcore- בעזרת debugger טקסטואלי

פתיחת d\_crash שקומפל עם -g



A screenshot of a computer

Description automatically generatedפתיחת crash שקומפל ללא flags  


**מקרה ג' -> שימוש בזיכרון לא מוגדר.**

פתיחת הcore- באמצעות הdebugger- עם debug info



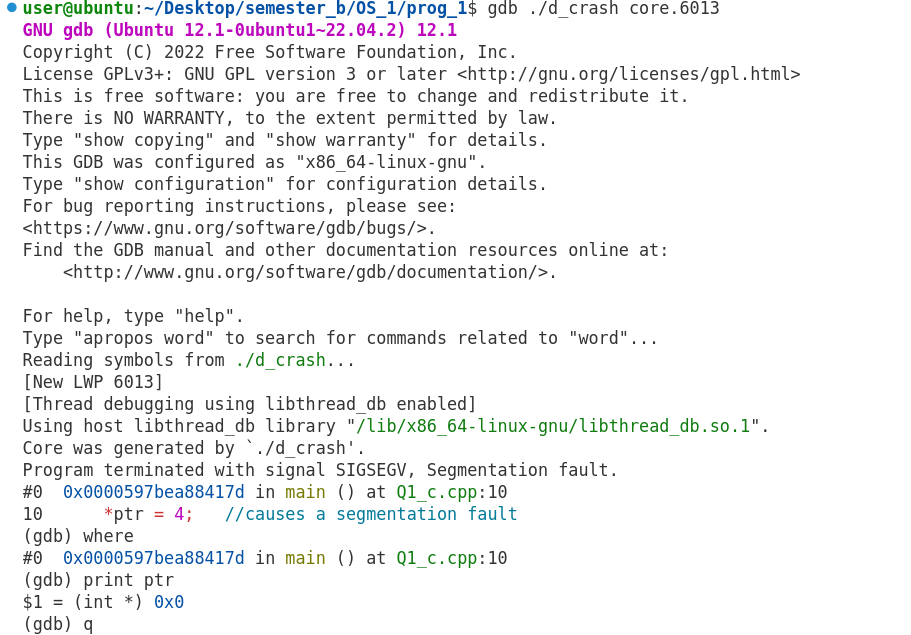
פתיחת הcore- באמצעות הdebugger- ללא debug info





פתיחת הcore- בעזרת debugger גרפי



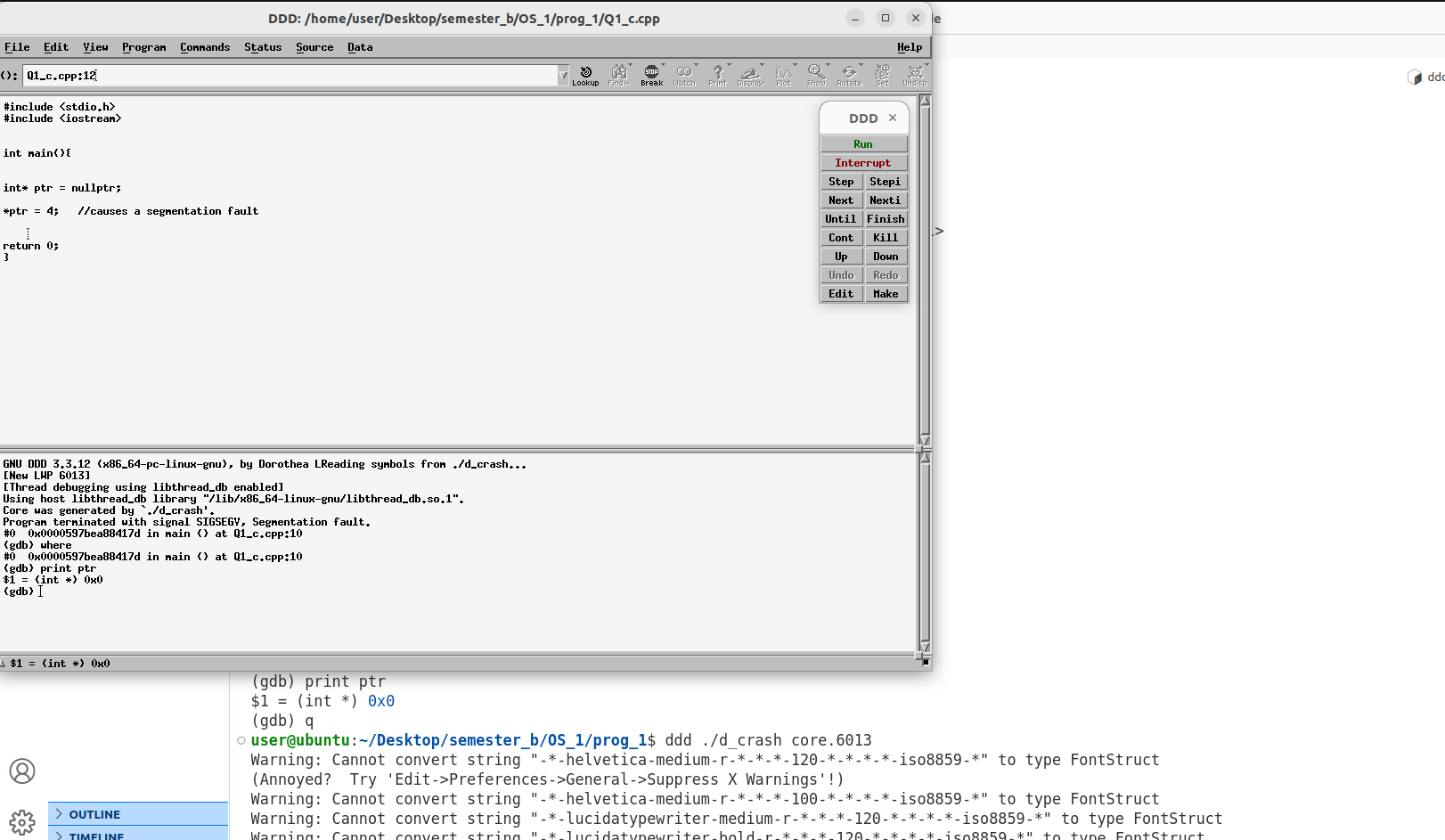
עבור ההרצה השנייה -   


פתיחת הcore- בעזרת debugger טקסטואלי

A screenshot of a computer

Description automatically generated

עבור ההרצה השנייה –



תרגיל 2 :

* יש לטעון את התיקיה ע"י הזנת cd Q2 לטרמינל
* על מנת לקמפל את הקוד יש להזין בטרמינל את הפקודה : make all
* להרצת הקוד יש להזין לטרמינל את הפקודה : **./poisson** **x y** כאשר **x** הוא ה- 𝛌 כמו 3.5 ו- **y** הוא ה- **k** , לדוגמא 2.
* על מנת למחוק את הקובץ הנוצר יש להריץ : make clean

דוגמת קימפול, הרצה ומחיקה: תמונה שמכילה טקסט, צילום מסך, גופן

התיאור נוצר באופן אוטומטי

תמונה שמכילה טקסט, תוכנה, תכונות מולטימדיה, צילום מסך

התיאור נוצר באופן אוטומטי

תמונה שמכילה טקסט, צילום מסך, גופן, תוכנה

התיאור נוצר באופן אוטומטינשים לב שבקוד אנחנו עושים #include math.h בשביל להשתמש ב expf, לכן נקמפל עם -lm.

תרגיל 3 :

* יש לטעון את התיקיה ע"י הזנת cd Q3 לטרמינל
* על מנת לקמפל את הקוד יש להזין בטרמינל את הפקודה : **make all**
* להרצת הקוד יש להזין לטרמינל את הפקודה : **./main**
* על מנת למחוק את הקובץ הנוצר יש להריץ : **make clean**

דוגמת קימפול, הרצה ומחיקה:

תמונה שמכילה טקסט, צילום מסך, תוכנה, תצוגה

התיאור נוצר באופן אוטומטי

תמונה שמכילה טקסט, צילום מסך, תוכנה, תכונות מולטימדיה

התיאור נוצר באופן אוטומטי

תמונה שמכילה טקסט, צילום מסך, גופן, תוכנה

התיאור נוצר באופן אוטומטי

תרגיל 4:

\*\*לצורך הדגמת ה-flags ותהליך הקומפילציה לפרטים, בדוגמת ההרצה לא השתמשנו ב-make, אלא בכל הפקודות המפורטות מטה.\*\*\*\*

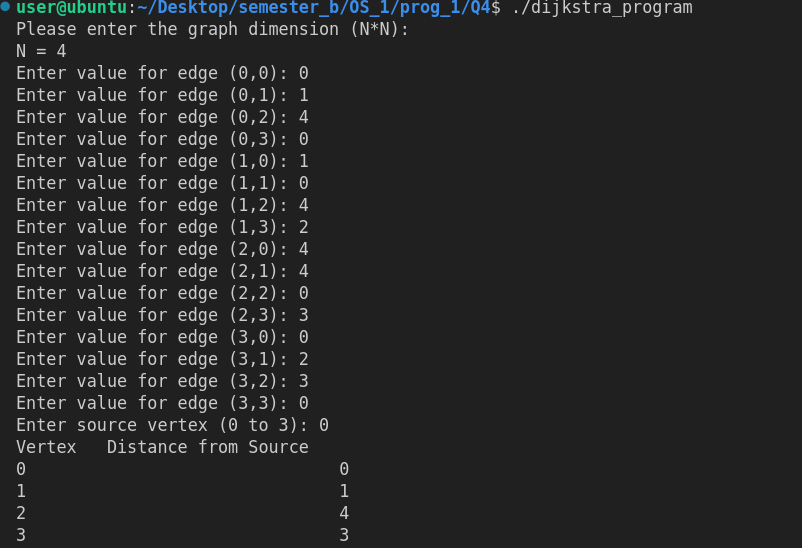
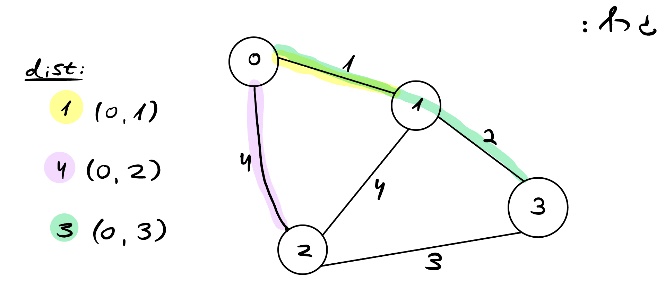
בהרצת התכנית תתבקשו להזין גרף, ונקודת התחלה כמפורט מטה.  
בנוסף, מצורפים צילומי מסך מהרצות מקרי הקצה.

**עבור הרצת התרגיל יש להשתמש ב-makefile כך:**  
**-** לכתוב בטרמינל : cd Q4  
**-** לאחר מכן עבור הרצת התוכנית עם gcov הזן לטרמינל: make coverage  
**-** עבר הרצת התוכנית ללא gcov הזן לטרמינל : make run  
**-** למחיקת הקבצים - יש להזין : make clean

הרצה לדוגמה –



הזנת הגרף –



A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

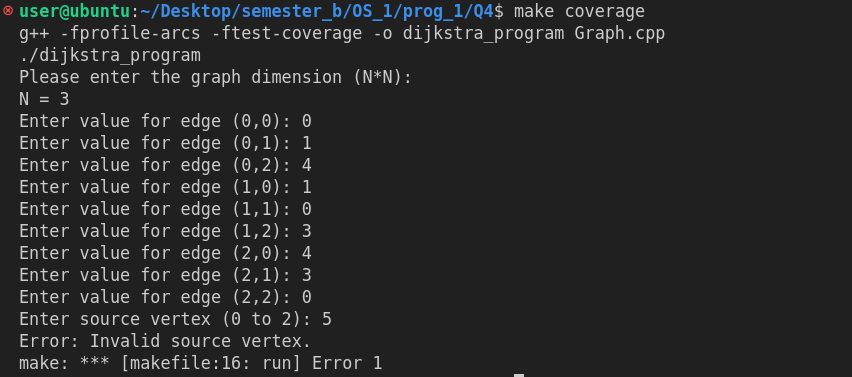
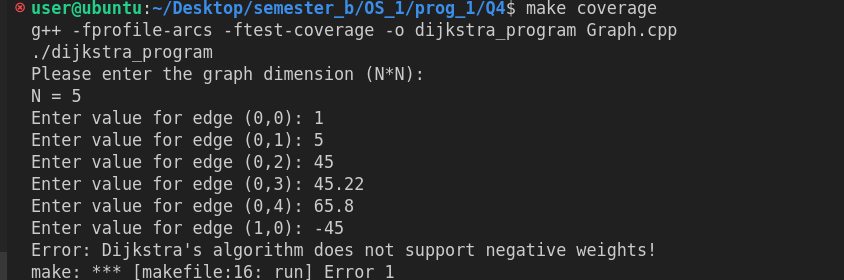
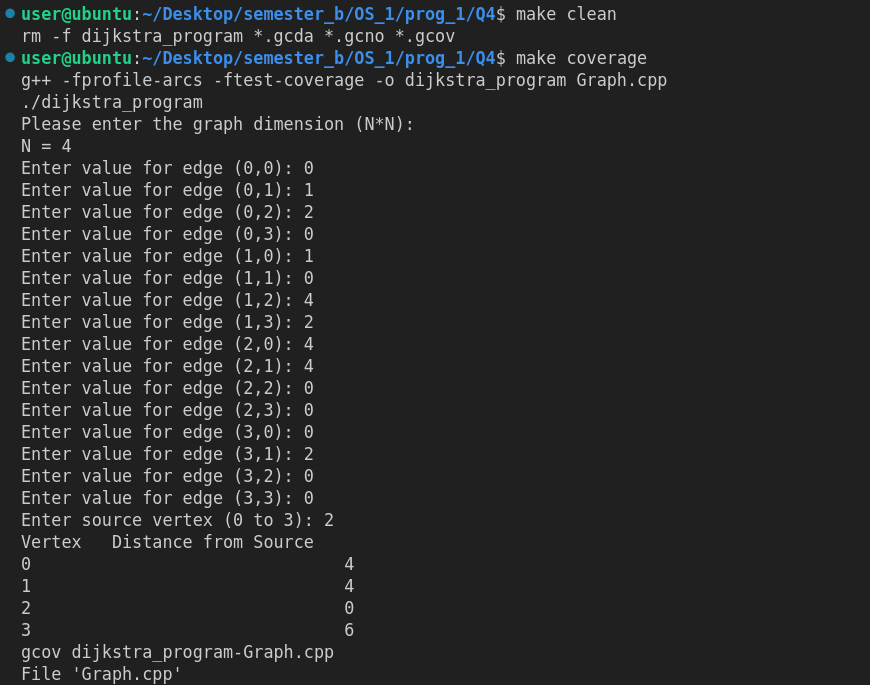
Description automatically generated

A screenshot of a computer

Description automatically generated

בהמשך לפקודות לעיל לאחר הפקודה האחרונה נוצרו קבצי ה-gcov (עם סיומת .gcov) אשר הם משמשים לתאר כמה חלקים מהקוד התבצעו במהלך הריצה של התוכנית, ומספקים תובנות לגבי כיסוי הקוד שנבדק באמצעות בדיקות.

להלן צילומי מסך מהרצות מקרי הקצה –

1. הזנת קודקוד לא קיים :  
   
2. הזנת משקל שלילי לקשת:  
   
3. אין מסלול בין קודקוד 2 לקודקוד 3 – הdistance- יהיה 0.

תרגיל 5:

**\*\***קבצי הפלט המלאים של gprof נמצאים בקוד המצורף תחת השם – “analysis\_WANTED**A**LGORITHM\_INPUT**S**IZE.txt”. \*\*

**הרצת התרגיל :**

להתאים למקרה הרצוי :

1. אלגוריתם א' -> סיבוכיות n בשלישית .

2. אלגוריתם ב' -> סיבוכיות n בשנייה .

3. אלגוריתם ג' -> סיבוכיות n כמתואר בנתוני השאלה .

- יש לטעון את התיקיה ע"י הזנת cd Q5 לטרמינל  
- להרצת אלגוריתם א/ ב/ ג עם יצירת פרופיל נתונים של -> gprof , יש להזין לטרמינל את הפקודה (להזין את ה-seed וה-size הרצויים ):   
make profile\_a seed=123 size=1000  
עבור אלגוריתם ב/ ג יש להחליף ל- profile\_b, profile\_c בהתאמה.

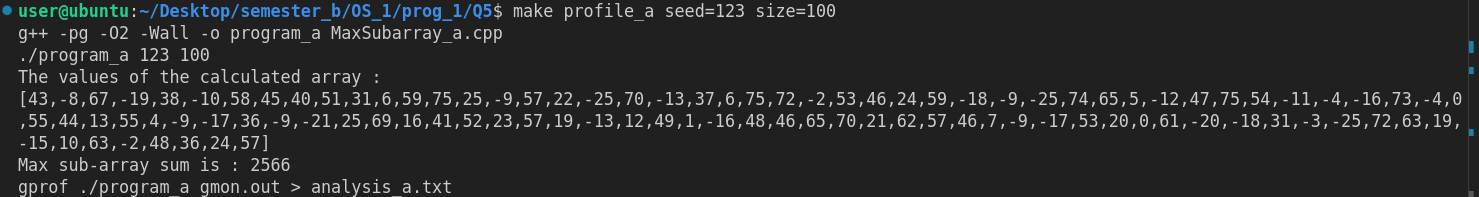
-להרצת התכנית ללא יצירת פרופיל נתונים יש להזין לטרמינל : make run\_a seed=123 size=1000

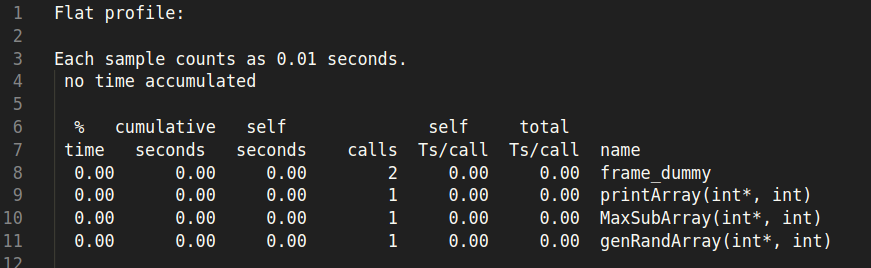
(run\_a/ run\_b/ run\_c).

- למחיקת הקבצים יש להזין לטרמינל את הפקודה : make clean

1. הרצת אלגוריתם 1 –

**קלט בגודל 100**

הרצה- ****

הדגמת זמן הרצת האלגוריתם לעומת זמן יצירת המספרים האקראיים (בקבצי הקוד קוראים לקובץ מטה analysis\_a\_100.txt) –  


ניתן לראות כי זמן ריצת האלגוריתם (והשהייה בכל פונקציה) מזערי, ולכן לא נוכל להסיק מסקנות בהרצה זו.

**קלט בגודל 1000**

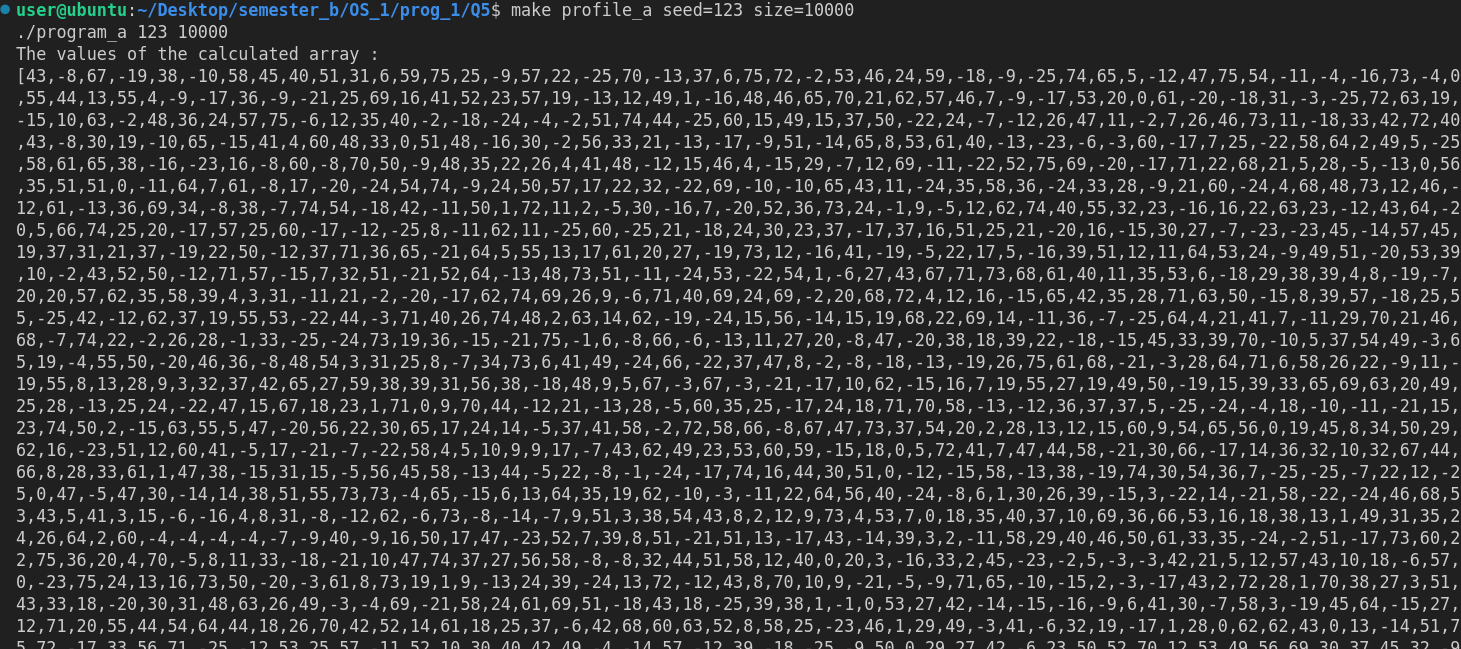
הרצה- A screenshot of a computer screen

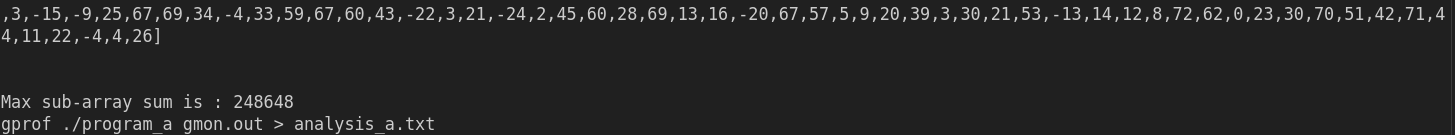
Description automatically generated

הדגמת זמן הרצת האלגוריתם לעומת זמן יצירת המספרים האקראיים (בקבצי הקוד קוראים לקובץ מטה analysis\_a\_1000.txt) –  
A screenshot of a computer screen

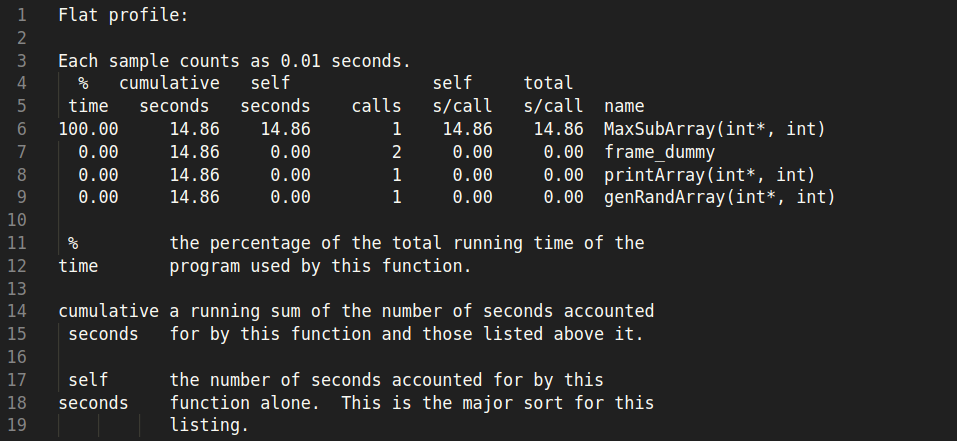
Description automatically generated

ניתן לראות כי זמן הריצה של יצירת המספרים האקראיים בפונקציה (getRandGen) הוא מזערי, בעוד ש-40ms הוקצו לקריאת הפונקציה MaxSubArray, שבבירור גדול יותר.

**קלט בגודל 10000** הרצה-

****

(זו רק חלק מההדפסה לא ניתן לראות כאן את כל המערך) .

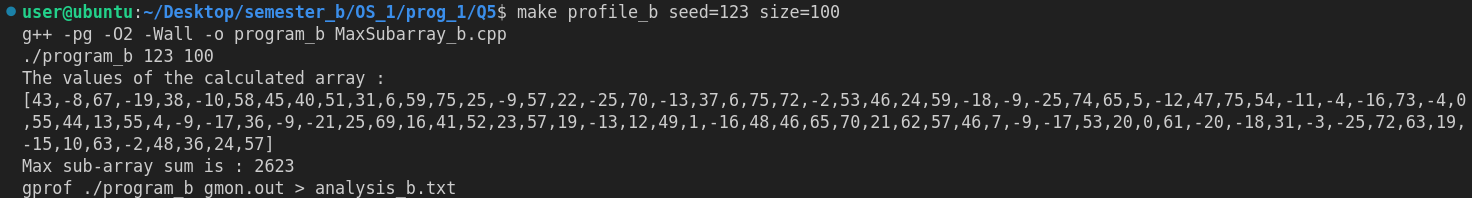
דגמת זמן הרצת האלגוריתם לעומת זמן יצירת המספרים האקראיים (בקבצי הקוד קוראים לקובץ מטה analysis\_a\_10000.txt) –  


הפונקציה genRandArray שאחראית ליצירת המספרים האקראיים, בעלת זמן ריצה מזערי זמן הביצוע שלה הוא 0.00 שניות. לעומת זאת, הפונקציה MaxSubArray, שאחראית על ביצוע האלגוריתם, צורכת את כל זמן הריצה של התוכנית: 14.86 שניות, המהוות 100% מזמן הריצה הכולל.  
מה שמדגיש שיצירת הנתונים היא שלב מהיר מאוד, בעוד שהחישובים של האלגוריתם עצמו הם שדורשים את רוב המשאבים וזמן הריצה של התוכנית.

1. הרצת אלגוריתם 2 –

**קלט בגודל 100**

הרצה-

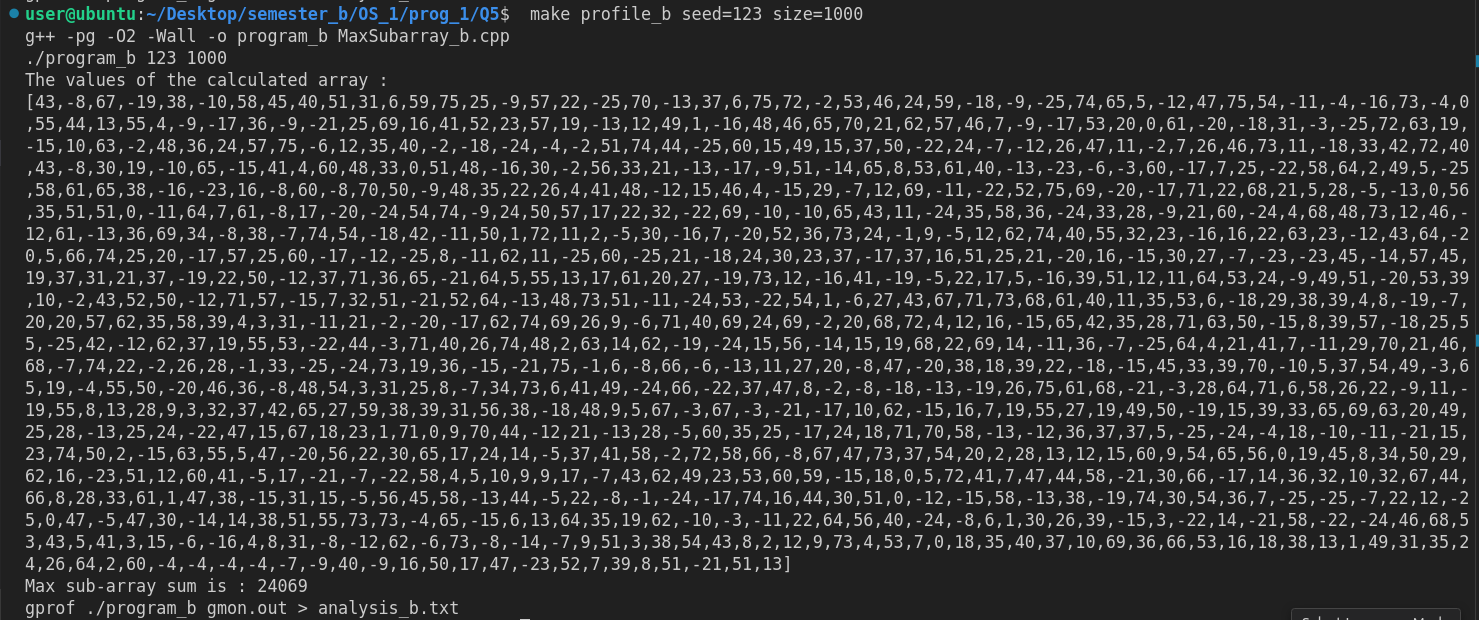
****

הדגמת זמן הרצת האלגוריתם לעומת זמן יצירת המספרים האקראיים (בקבצי הקוד קוראים לקובץ מטה analysis\_b\_100.txt) –  
A screenshot of a computer

Description automatically generated

ניתן לראות כי זמן ריצת האלגוריתם (והשהייה בכל פונקציה) מזערי, ולכן לא נוכל להסיק מסקנות בהרצה זו.

**קלט בגודל 10,000**

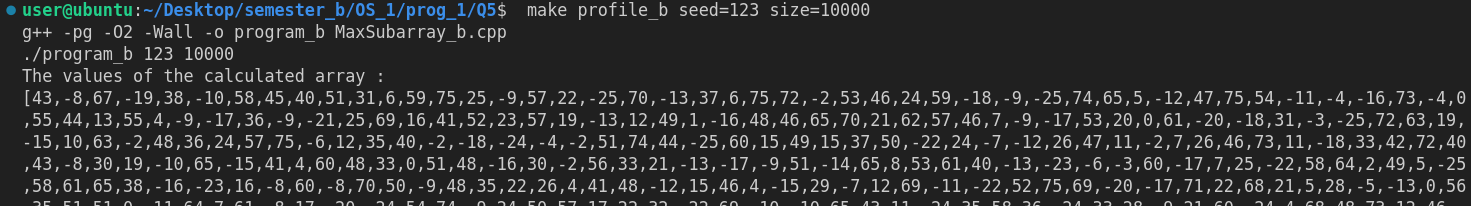
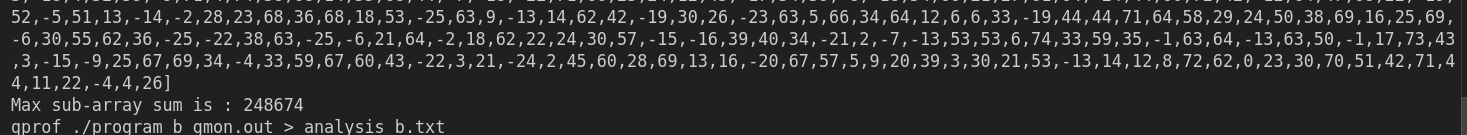
הרצה-

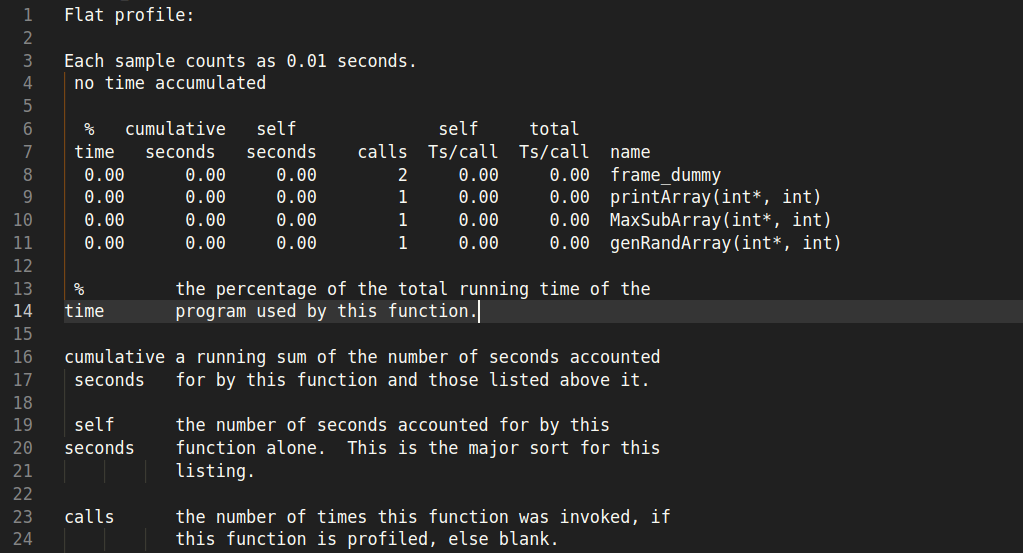
הדגמת זמן הרצת האלגוריתם לעומת זמן יצירת המספרים האקראיים (בקבצי הקוד קוראים לקובץ מטה analysis\_a\_1000.txt) –  
A screenshot of a computer

Description automatically generated

ניתן לראות כי זמן ריצת האלגוריתם (והשהייה בכל פונקציה) מזערי, ולכן לא נוכל להסיק מסקנות בהרצה זו.

**קלט בגודל 10000**

****הרצה (זו רק חלק מההדפסה לא ניתן לראות כאן את כל המערך) –

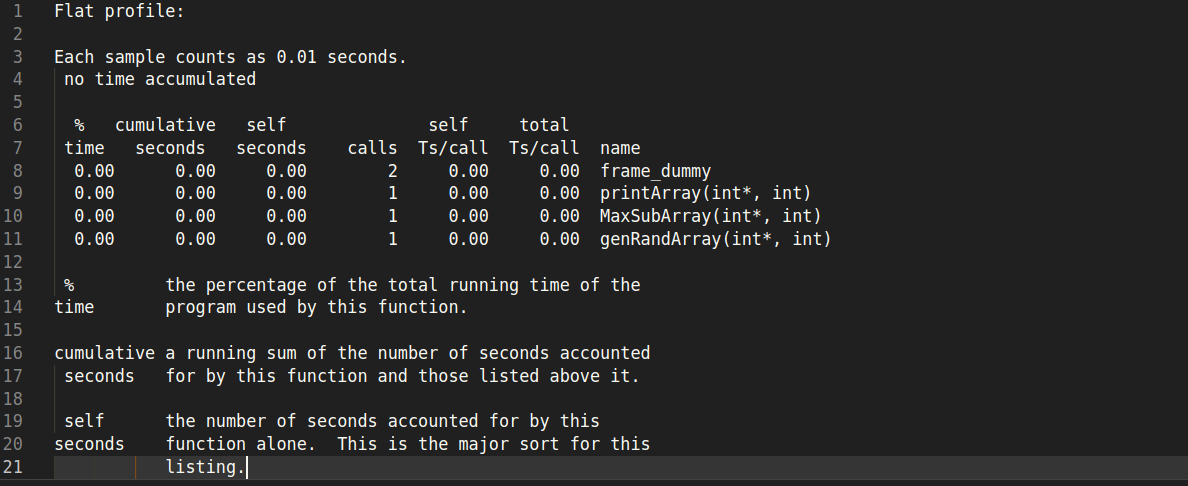
הדגמת זמן הרצת האלגוריתם לעומת זמן יצירת המספרים האקראיים (בקבצי הקוד קוראים לקובץ מטה analysis\_b\_10000.txt) –  


1. הרצת אלגוריתם 3 –

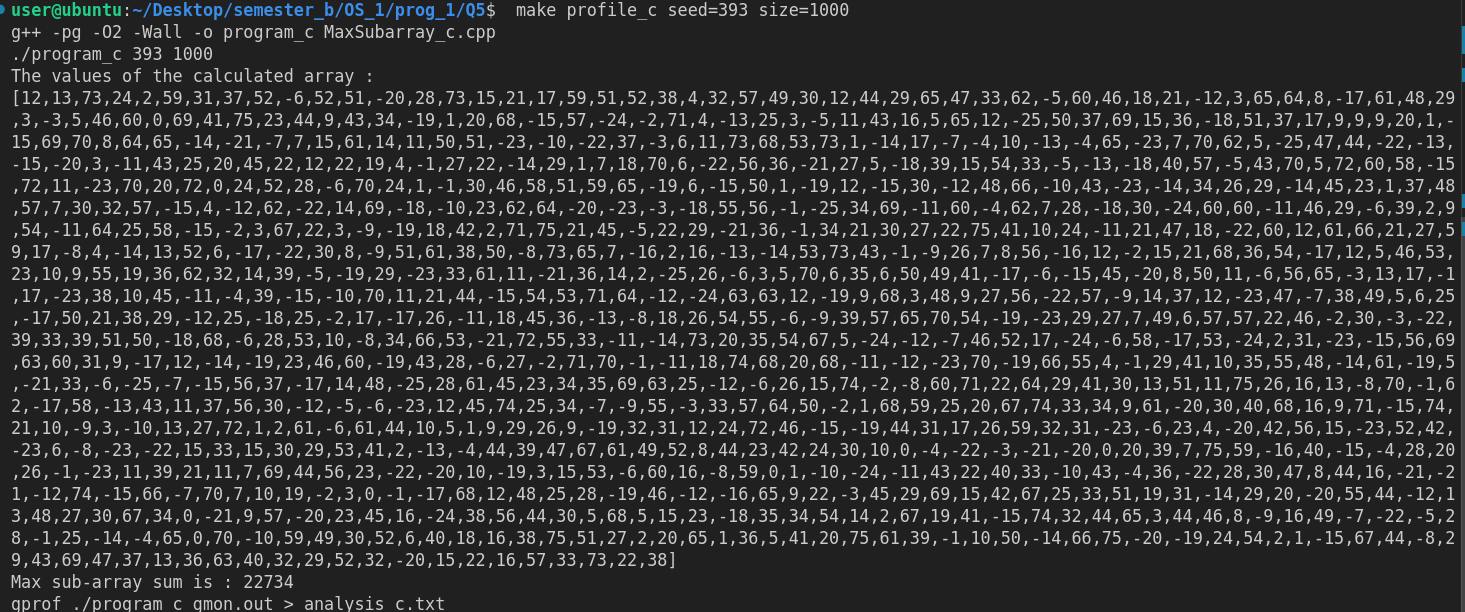
**קלט בגודל 100**

**A black screen with white text

Description automatically generated**

הדגמת זמן הרצת האלגוריתם לעומת זמן יצירת המספרים האקראיים (בקבצי הקוד קוראים לקובץ מטה analysis\_c\_100.txt) –  


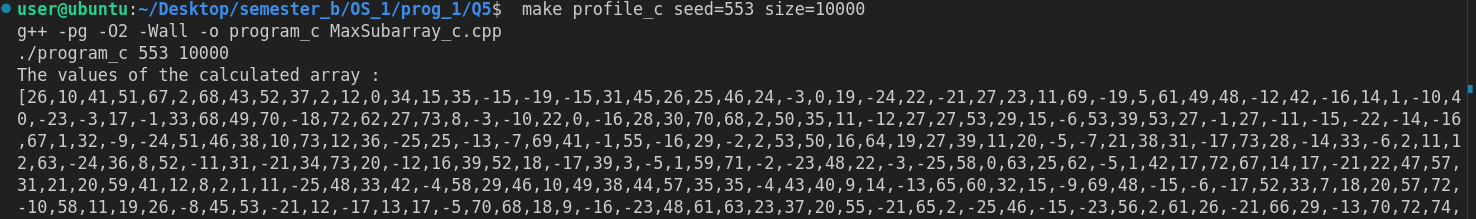
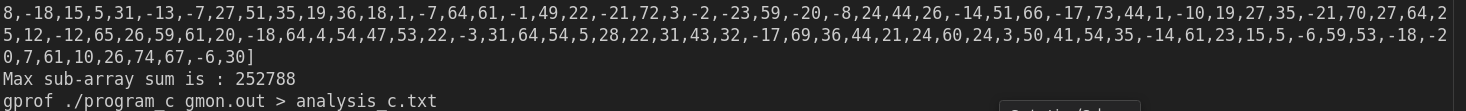
**קלט בגודל 1000**

****

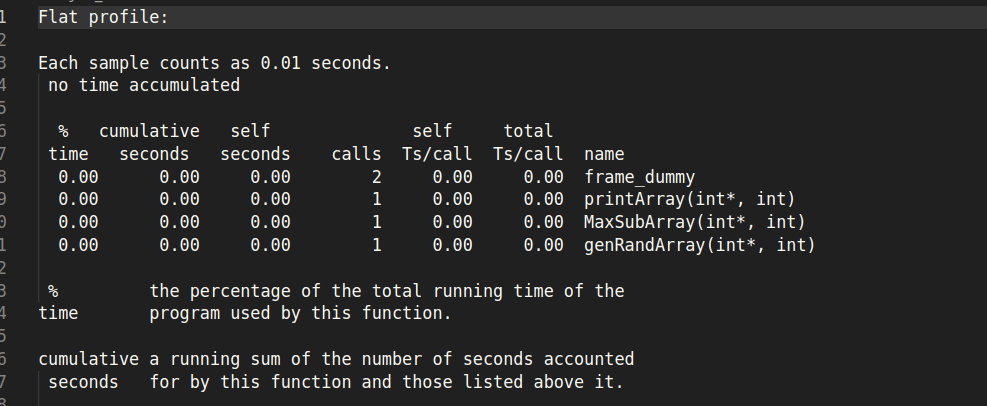
הדגמת זמן הרצת האלגוריתם לעומת זמן יצירת המספרים האקראיים (בקבצי הקוד קוראים לקובץ מטה analysis\_c\_1000.txt) –

**A screenshot of a computer screen

Description automatically generated**

**קלט בגודל 10000**

הדגמת זמן הרצת האלגוריתם לעומת זמן יצירת המספרים האקראיים (בקבצי הקוד קוראים לקובץ מטה analysis\_c\_10000.txt) –

****

לסיכום, מהפרופיל ניתן לראות:

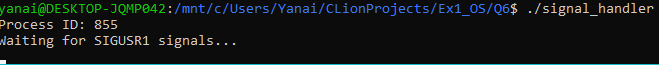
MaxSubArrayו-genRandArray הן פונקציות שנקראו במהלך התכנית אך לא צברו זמן ריצה מדיד. הפונקציות הללו רצות מהר מאוד (פחות מ-0.01 שניות), כך שאין כמעט זמן ריצה נצבר.

מאחר שהפונקציות לא צברו זמן ריצה מדיד, אין דרך להשוות ביניהן מבחינת השפעתן על הביצועים הכלליים, שכן הן כולן רצות ביעילות גבוהה מאוד בתנאים הנוכחיים (בניגוד למקרים באלגוריתם א' המפורטים לעיל).

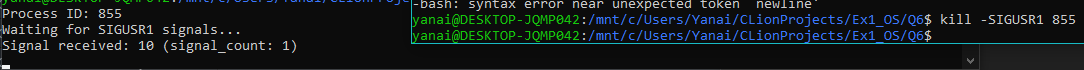
.

תרגיל 6 :

* יש לטעון את התיקיה ע"י הזנת cd Q6 לטרמינל
* על מנת לקמפל את הקוד יש להזין בטרמינל את הפקודה : **make all**
* להרצת הקוד יש להזין לטרמינל את הפקודה : **./signal\_handler**
* עלינו לפתוח טרמינל נוסף ודרכו לשלוח את הסיגנל כך : kill -SIGUSR1 <PID>
* כאשר <PID> הוא הid של התהליך המופיע לנו בהרצת הקוד.
* על מנת למחוק את הקובץ הנוצר יש להריץ : **make clean**

**דוגמא להרצת הקוד:**

**נשלח סיגנל ראשון:**

****

**נשלח עוד 3:**

**תמונה שמכילה טקסט, צילום מסך, תוכנה

התיאור נוצר באופן אוטומטי**

**נראה שהתכנית ספרה את הסיגנלים, כאשר היא מנהלת כל סיגנל תוך כדי ניהול הקודם, ולבסוף הוסיפה i++ ויצאה.**