**ממ"ן 13**

1. בדומה לערימה בינראית, העץ של הערימה הD-ית מלא לחלוטין בכל רמותיו פרט אולי לאחרונה, המלאה משמאל ועד לנקודה מסויימת. לפיכך, נוכל לייצג אותו כמערך בדרך דומה לערימה בינראית, כך שהשורש מאוחסן בתא הראשון, ולאחריו האיברים של הרמה הבאה בעץ, משמאל לימן, עד הרמה האחרונה, בה כל העלים הקיימים מאוחסנים גם הם משמאל לימין.  
   על מנת להפוך את המערך חזרה לערימה D-ית נבצע פעולה הפוכה, ונשתמש בערך בתא הראשון כשורש, בערכים בתא השני עד התא D+1 כבניו, וכן הלאה, עד שנגיע לאיברים האחרונים, שהם העלים, ואותם נשבץ משמאל לימין כילדיהם של הרמה הלפני האחרונה, עד תום.  
   הכללת החישובים הנדרשים לפעולות על הערימה הd-ית על סמך ערימה בינארית (לפי פרק 6.1 בספר) תתבצע כדלהלן (הצבת d=2 תוכיח את נכונות החישוב לפי החישובים לערימה בינארית):
   1. מציאת אינדקס האב של האיבר ה-i :
   2. מציאת האינדקס של הבן הn-י (n=1,2,3…) של אינדקס i :
   3. האינדקס הראשון של הרמה ה-i-ית (כאשר השורש הוא רמה 0) :
   4. האינדקס הראשון של העלים :
   5. גובה הערימה : )ראה סעיף ב'(
2. באופן דומה לפתרון בעיה 6.1-2 עבור חישוב גובה ערימה בינארית בת n איברים, נבצע חישוב דומה עבור ערימה D-ית.  
   יהיה h גובה הערימה. מאחר והעץ המייצג את הערימה הוא עץ D-י מלא לחלוטין עד אולי הרמה h-1, ולכל איבר בו יש D בנים, וברמה האחרונה יהיו k עלים. לפיכך, כמות האיברים בעץ תהיה שווה ל-  
    .  
   מכך נובע שהגובה, כלומר, אורך המסלול הפשוט הארוך ביותר מהשורש לעלה כלשהו יהיה -

כלומר, הגובה של ערימה d-ית בת n איברים הוא .

**המטלה התכנותית – שפת python**  
כעת, כהקדמה לסעיפים ג' – ו', נסביר על המימוש הפנימי של הערימה הd-ית והמחלקות הנוספות בפתרון שלנו–

הערימה הd-ית מיוצגת על ידי המחלקה DHeap, שיורשת מהמחלקה list, ומתאותחלת על ידי מערך של איברים והערך של d לערימה זו. בהתאם, מחלקת האב מאפשרת לקבל את כמות האיברים במערך (גודל הערימה) ואת היכולת לגשת ולקבוע ערכים לפי האינדקס שלהם. בנוסף, המחלקה עצמה מייצאת את התכונות heap\_size (גודל הערימה המשתנה), array\_length (גודל המערך ההתחלתי), וheight (גובה הערימה, חישובו מפורט בסעיף ב'). המחלקה מממשת פעולות פנימיות נפוצות בערימות, למשל SWAP בין ערכים, או הסרת הערך האחרון במערך.  
כמו כן, המחלקה מייצאת חישובים נפוצים, כמו שפירטנו בסעיף א'. לבסוף, מימשנו פעולות להדפסה למסך – בייצוג של רשימה (רשימה של רשימות המייצגות כל רמה בעץ), ובאילוסטרציה בתור עץ.  
בנוסף למחלקה של הערימה הd-ית, מימשנו מחלקה נוספת בשם GeneralAlgorithms שבה מימשנו אלגוריתמים נוספים הדרושים לפונקציונליות התכנית, למשל – BUILD-MAX-HEAP ו-MAX-HEAPIFY. בנספח א' למסמך צורפו כל המימושים בפסאודו-קוד לשגרות השונות שנדרשו לפתרון התרגיל.

מאחר ש MAX-HEAPIFY מרכזי לפונקציונליות של חלק גדול משאר הקוד, ננתח ראשית את הסיבוכיות שלו כדי להקל על הסעיפים הבאים-

לפי פרק 6.2 והפסאודו-קוד בנספח א' ניתן לראות שזמן הריצה של הפעולות הנדרשות לשגרה MAX-HEAPIFY הוא בתחילת השגרה על מנת לקבוע את היחס בין השורש של תת העץ הנבחר וכל אחד מ-d בניו, ולאחר מכן זמן הריצה של MAX-HEAPIFY על כל תת-עץ המתחיל בכל אחד מ-d בניו של השורש. נוכל להראות כי זמן הריצה יהיה תלוי בגובה הערימה הd-ית, שהוא, כפי שהדגמנו בסעיף ב', , מכיוון שהשגרה תרוץ רקורסיבית במקרה הכי גרוע על אורכו של אחד מ-d תתי העצים של השורש, אך ברמה האסימפטותית זה שקול לסיבוכיות של :

גודלו של כל אחד מתתי-העצים האלו, במקרה הגרוע ביותר (תתי עצים של שורש הערימה) הוא qn, כאשר q<1 הוא נתח כלשהו של האיברים בערימה (מאחר וגודלה של כל הערימה הוא n, וזהו רק תת-עץ, כלומר חלק שלה). לפיכך, ניתן לתאר את זמן הריצה של השגרה על ידי נוסחאת הנסיגה הכללית-

*לפי מקרה 2 של נוסאחת האב (משפט 4.1) –*

*כלומר, זמן הריצה של השגרה הוא בסיבוכיות האסימפטותית שלה הוא .*

1. בפתרון מימשנו את פעולה EXTRACT-MAX(A) על ערימת מקסימום d-ית (השגרה *extract\_max*(heap)).

הסבר על האלגוריתם (ראו פסאודו-קוד בנספח א'):

ש

סיבוכיות זמן ריצה:

האלגוריתם עושה ראשית מספר פעולות בעלות זמן ריצה קבוע (השוואות, הגדרת משתנים, השמת ערכים) שזמן הריצה שלהם הוא בסיבוכיות של ,  
ולבסוף קורא לשגרה MAX-HEAPIFY שרצה על כל גובה הערימה (השגרה MAX-HEAPIFY רצה רקורסיבית על כל העץ החל מהשורש החדש לאחר ההוצאה) בסיבוכיות של *. כלומר, סיבוכיות זמן הריצה של השגרה כולה היא הסיבוכיות של הפעולה הכי כבדה (*MAX-HEAPIFY*) - .*

1. בפתרון מימשנו את פעולה INSERT(A, v) על ערימת מקסימום d-ית (השגרה *insert*(heap,value)).

הסבר על האלגוריתם (ראו פסאודו-קוד בנספח א'):

ש

סיבוכיות זמן ריצה:

האלגוריתם עושה ראשית מספר פעולות בעלות זמן ריצה קבוע (הגדרת משתנים, השמת ערכים) שזמן הריצה שלהם הוא בסיבוכיות של ,  
ולבסוף קורא לשגרה INCREASE-KEY שרצה במקרה הכי גרוע לאורך כל הגובה העץ (מהעלה לשורש) בסיבוכיות של *. כלומר, סיבוכיות זמן הריצה של השגרה כולה היא הסיבוכיות של הפעולה הכי כבדה (*INCREASE-KEY*) - .*

1. בפתרון מימשנו את פעולה INCREASE-KEY(A,i,k) על ערימת מקסימום d-ית (השגרה *increase\_key*(heap,index\_to\_increase,new\_value)).

הסבר על האלגוריתם (ראו פסאודו-קוד בנספח א'):

ש

סיבוכיות זמן ריצה:

האלגוריתם עושה ראשית מספר פעולות בעלות זמן ריצה קבוע (השוואות, הגדרת משתנים, השמת ערכים) שזמן הריצה שלהם הוא בסיבוכיות של ,  
ולבסוף מפעפעת את הערך החדש למקומו הנכון בפעולות בעלות זמן ריצה קבוע, שבמקרה הכי גרוע עוברות את כל גובה העץ מלמטה למעלה עד השורש, בסיבוכיות של *. כלומר, סיבוכיות זמן הריצה של השגרה כולה היא - .*

1. בפתרון מימשנו את פעולה EXTRACT(A,i) על ערימת מקסימום d-ית (השגרה *extract*(heap,index\_to\_remove))

הסבר על האלגוריתם (ראו פסאודו-קוד בנספח א'):

ש

סיבוכיות זמן ריצה:

האלגוריתם עושה ראשית מספר פעולות בעלות זמן ריצה קבוע (השוואות, הגדרת משתנים, השמת ערכים) שזמן הריצה שלהם הוא בסיבוכיות של ,   
ולבסוף קורא לשגרה MAX-HEAPIFY שרצה, במקרה הכי גרוע (מחיקת השורש), על כל גובה הערימה (השגרה MAX-HEAPIFY רצה רקורסיבית על כל העץ החל מהשורש החדש לאחר המחיקה) בסיבוכיות של *. כלומר, סיבוכיות זמן הריצה של השגרה כולה היא הסיבוכיות של הפעולה הכי כבדה (*MAX-HEAPIFY*) - .*

**נספח א' – מימושי הפעולות בפתרון**

|  |  |  |  |
| --- | --- | --- | --- |
| השגרה | סיבוכיות זמן ריצה | המימוש | הערות |
| BUILD-MAX-HEAP | (calls MAX-HEAPIFY for each node of the heap that isn’t a leaf) | BUILD-MAX-HEAP(A):  For i from A.get\_first\_leaf\_index downto 0:  MAX-HEAPIFY(A, i) | בדומה למימוש המוצג בפרק 6.3, המימוש של חישוב העזר מוגדר בסעיף א' |
| MAX-HEAPIFY | (recursively goes over the sub-heap of largest child node out of d children, which in the worst case is the entire height of the heap) | MAX-HEAPIFY(A, i):  Largest\_index = i  For child from A.nth\_child\_index(i, 1) to A.nth\_child\_index(i, d):  If child < A.HEAP\_SIZE and A[child] > A[Largest\_index]:  Largest\_index = child  If Largest\_index != i:  SWAP(A, i, Largest\_index)  MAX-HEAPIFY(A, Largest\_index) | בדומה למימוש המוצג בפרק 6.2 |
| EXTRACT-MAX | (Const. time + MAX-HEAPIFY) | EXTRACT-MAX(A):  If A.HEAP\_SIZE < 1:  Error “heap underflow”  If A.HEAP\_SIZE == 1:  Return POP-LAST(A)  Max\_index = 0  SWAP(max\_index, A.HEAP\_SIZE - 1)  Popped\_node = POP-LAST(A)  MAX-HEAPIFY(A, 0)  Return Popped\_node | בדומה למימוש המוצג בפרק 6.5 |
| EXTRACT | (Const. time + MAX-HEAPIFY) | EXTRACT (A, i):  If A.HEAP\_SIZE < i:  Error “heap overflow”  If i < 0 or A.HEAP\_SIZE < 1:  Error “heap underflow”  If A.HEAP\_SIZE == 1:  Return POP-LAST(A)  SWAP(i, A.HEAP\_SIZE - 1)  Popped\_node = POP-LAST(A)  MAX-HEAPIFY(A, i)  Return Popped\_node |  |
| POP-LAST | (Const. time) | POP-LAST(A):  Last\_index = A.HEAP\_SIZE - 1  Popped\_node = A[last\_index]  A.HEAP\_SIZE = A.HEAP\_SIZE -1  Return Popped\_node | פונקציית עזר |
| INSERT | (Const. time + INCREASE-KEY) | INSERT(A, v):  A.HEAP\_SIZE += 1  A[A.HEAP\_SIZE] = v  INCREASE-KEY(A, A.HEAP\_SIZE, v) | בדומה למימוש המוצג בפרק 6.5 |
| INCREASE-KEY | (Swaps the value to his right place, which in the worst case is the entire height of the heap) | INCREASE-KEY(A, i, v):  If A. HEAP\_SIZE < i:  Error "Heap overflow.”  If i < 0:  Error "Heap underflow."  If A[i] > v:  Return # New key is smaller than current key  else:  A[i] = v  While i > 0:  Parent\_index = A.get\_parent\_index(i)  If A[i] > A[Parent\_index]:  A.swap(i, Parent\_index)  i = Parent\_index  Else:  Return | בדומה למימוש המוצג בפרק 6.5 |

**נספח ב' – דוגמא לשימוש בממשק המשתמש**

מצורף פלט לדוגמא של הרצת התכנית ושימוש בממשק. הקלטים וסדר הפעולות של הדוגמא הנ"ל כלולים, בין היתר, בקובץ הבדיקות של התכנה.

בכללי TODO:

* להוסיף את הפלט של ההרצה לדוגמא תחת נספח ב' – TODO שלי
* TODO סעיפים ג-ו - להסביר את הרעיון הכללי בעברית – טל
* לתקף את הפסאודו קוד לפי הגרסא הסופית של הקוד - שלי
* לוודא שהמסמך הנלווה כתוב באופן שבו הוא יכול לשמש את המנחה לבדיקה ללא צורך בהרצה (כך שכביכול ההרצה היא רק לבדיקה שמה שכתוב בקובץ המלווה הוא אכן נכון) – טל + שלי