211768379 סטודנט א' : טל דרוז'ינין, 211371596 2טודנט ב' : שלי גולצמן, 211371596

ממ"ן 13

א. 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 הb-ית על סמך ערימה בינארית (לפי פרק 6.1 בספר) תתבצע כדלהלן (הצבת d=2 תוכיח את נכונות החישוב לפי החישובים לערימה בינארית):

- $\left\lfloor \frac{i-1}{d} \right\rfloor$: i-מציאת אינדקס האב של האיבר ה.a.
- ((di)+n) : i מציאת האינדקס של הבן הח-י (n=1,2,3...) של אינדקס של הבן .b
- $rac{d^{i}-1}{d-1}$: (0 אשר השורש הוא רמה ה-i-ית (כאשר השורש הוא רמה -c.
 - $\left| \frac{n+1}{d} \right|$: האינדקס הראשון של העלים .d
 - ב. ב. [$\log_d n$]: גובה הערימה .e
- ב. באופן דומה לפתרון בעיה 6.1-2 עבור חישוב גובה ערימה בינארית בת n איברים, נבצע חישוב דומה ב. באופן דומה D-ית.

,h-1 אולי הרמה. מאחר והעץ המייצג את הערימה הוא עץ D-י מלא לחלוטין עד אולי הרמה h-יהיה h והיה h גובה הערימה. מאחר והעץ המייצג את הערימה איבר בו יש D בנים, וברמה האחרונה יהיו k עלים. לפיכך, כמות האיברים בעץ תהיה שווה ל $n=d^{h-1}+k$

מכך נובע שהגובה, כלומר, אורך המסלול הפשוט הארוך ביותר מהשורש לעלה כלשהו יהיה

$$n = d^{h-1} + k \to h = \lfloor \log_d n \rfloor$$

 $\log_d n$ ו איברים הוא n כלומר, הגובה של ערימה -d כלומר, הגובה של ערימה

211768379 סטודנט א' : טל דרוז'ינין, 211371596 2טודנט ב' : שלי גולצמן, 211371596

המטלה התכנותית – שפת python

כעת, כהקדמה לסעיפים ג' − ו', נסביר על המימוש הפנימי של הערימה הb-ית והמחלקות הנוספות בפתרון שלנו–

הערימה הb-ית מיוצגת על ידי המחלקה DHeap, שיורשת מהמחלקה list, ומתאותחלת על ידי מערך של איברים הערימה h 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), heighti (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 של רשימות המייצגות כל רמה בעץ), ובאילוסטרציה בתור עץ. בנוסף למחלקה של הערימה הb-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 לפי פרק $d*\Theta(1)$ הוא $d*\Theta(1)$ בתחילת השגרה על מנת לקבוע את היחס בין השורש של תת העץ הנבחר וכל אחד מ-b בניו, הוא $d*\Theta(1)$ בתחילת השגרה על מנת לקבוע את היחס בין השתחיל בכל אחד מ-b בניו של השורש. נוכל להראות כי MAX-HEAPIFY מכיוון שהשגרה תרוץ זמן הריצה יהיה תלוי בגובה הערימה הd-ית, שהוא, כפי שהדגמנו בסעיף ב', $\log_d n$, מכיוון שהשגרה תרוץ רקורסיבית במקרה הכי גרוע על אורכו של אחד מ-d תתי העצים של השורש, אך ברמה האסימפטותית זה שקול לסיבוכיות של $\Theta(\log(n))$:

גודלו של כל אחד מתתי-העצים האלו, במקרה הגרוע ביותר (תתי עצים של שורש הערימה) הוא qn, כאשר q<1, 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-

$$T(n) \le T(qn) + \Theta(1)$$

– (4.1 של נוסאחת האב (משפט 4.1)

$$a = 1, b = \frac{1}{q}, f(n) = \Theta(1) \rightarrow$$

$$f(n) = \Theta\left(n^{\log_{\frac{1}{q}}1}\right) = \Theta(n^0) = \Theta(1) \rightarrow$$

$$T(n) = \Theta(n^0 * \lg(n)) = \Theta(\lg(n))$$

 $\Theta(\lg(n))$ בסיבוכיות האסימפטותית שלה הוא $\log_d n$ כלומר, זמן הריצה של השגרה הוא

211768379 סטודנט א' : טל דרוז'ינין, 211371596 סטודנט ב' : שלי גולצמן,

ית (השגרה בפתרון מימשנו את פעולה (EXTRACT-MAX(A) על ערימת מקסימום בפתרון (השגרה ($extract_max$ (heap)

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

האלגוריתם יגדיר את האינדקס של התא עם הערך המקסימלי (במקרה שלנו הערימה היא ערימת מקסימום, ולכן האיבר המקסימלי הוא השורש).

האלגוריתם יקרא לשגרה EXTRACT עם האינדקס של השורש כפרמטר וישמור את הפלט למשתנה. לבסוף נחזיר את ערך המשתנה עם הפלט של EXTRACT על האיבר המקסימלי.

סיבוכיות זמן ריצה:

האלגוריתם עושה ראשית מספר פעולות בעלות זמן ריצה קבוע (הגדרת משתנים, השמת ערכים) שזמן הריצה שלהם הוא בסיבוכיות של $\Theta(1)$,

ולבסוף קורא לשגרה EXTRACT שבעצמה עושה מספר פעולות בעלות זמן ריצה קבוע ואז קוראת לשגרה על כל EXTRACT שרצה על כל גובה הערימה $\lfloor \log_d n \rfloor$ (השגרה MAX-HEAPIFY שרצה על כל גובה הערימה $\Theta(\lg(n))$ (בטיבוכיות של החל מהשורש החדש לאחר ההוצאה ועד לעלים) בסיבוכיות של $\Theta(\lg(n))$. כלומר, סיבוכיות זמן הריצה של השגרה כולה היא הסיבוכיות של הפעולה הכי כבדה $\Theta(\lg(n))$.

ד. בפתרון מימשנו את פעולה (INSERT(A, v) על ערימת מקסימום h-ית (השגרה (insert(heap,value)). הסבר על האלגוריתם (ראו פסאודו-קוד בנספח א'):

האלגוריתם יכין את הערימה לקבלת איבר חדש על ידי הגדלת המשתנה של גודל הערימה ב-1, בכך נוצר "תא אחרון" חדש במערך המייצג.

לאחר מכן, נגדיר את ערכו של של התא האחרון החדש הזה לערך שקיבלנו כקלט לשגרה. לבסוף, נקרא לשגרה INCREASE-KEY על מנת לפעפע את האיבר החדש למקומו הנכון.

סיבוכיות זמן ריצה:

האלגוריתם עושה ראשית מספר פעולות בעלות זמן ריצה קבוע (הגדרת משתנים, השמת ערכים) שזמן הריצה שלהם הוא בסיבוכיות של $\Theta(1)$,

211768379 סטודנט א' : טל דרוז'ינין, 211371596 סטודנט ב' : שלי גולצמן, 211371596

ה. בפתרון מימשנו את פעולה (INCREASE-KEY(A,i,k על ערימת מקסימום יית (השגרה

.(increase_key(heap,index_to_increase,new_value)

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

- האלגוריתם תחילה בודק מקרי קצה

אם האינדקס שמנסים להגדיל גדול מגודל הערימה, הרי זהו מצב של overflow ולכן נתריע על שגיאה; אם האינדקס לא תקין או שהערימה ריקה, אין אפשרות לעשות הגדלת איבר, ולכן נתריע על שגיאה; אם הערך באינדקס שאותו מנסים להגדיל גדול מהערך החדש – זהו מקרה ריק ולכן נסיים את השגרה. אם הכל כשורה –

כל עוד לא הגענו לאינדקס השורש (כלומר, העץ מסודר והגענו לתנאי העצירה), נבצע בלולאה: ניצור משתנה חדש עם ערך האינדקס של האבא של האינדקס הנוכחי, ונשווה בין הערכים שלהם. אם הבן יותר גדול מהאבא- ערימת המקסימום נשברה ולכן נעביר את הבן מעל לאבא, נגדיר את האינדקס הנוכחי להיות האינדקס של האבא ונמשיך בלולאה.

אחרת – הערימה מסודרת כראוי ונוכל להפסיק בריצה.

סיבוכיות זמן ריצה:

האלגוריתם עושה ראשית מספר פעולות בעלות זמן ריצה קבוע (השוואות, הגדרת משתנים, השמת ערכים) שזמן הריצה שלהם הוא בסיבוכיות של $\Theta(1)$,

ולבסוף מפעפעת את הערך החדש למקומו הנכון בפעולות בעלות זמן ריצה קבוע, שבמקרה הכי גרוע עוברות את כל גובה העץ $[\log_d n]$ מלמטה למעלה עד השורש, בסיבוכיות של $[\log_d n]$. כלומר, סיבוכיות זמן הריצה של השגרה כולה היא $\Theta(\lg(n))$.

וית (השגרה-d ערימת מקסימום EXTRACT(A,i) בפתרון מימשנו את פעולה

(extract(heap,index_to_remove)

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

- האלגוריתם תחילה בודק מקרי קצה

אם האינדקס שמנסים להוציא גדול מגודל הערימה, הרי זהו מצב של overflow ולכן נתריע על שגיאה; אם האינדקס לא תקין או שהערימה ריקה, אין אפשרות לעשות הוצאת איבר, ולכן נתריע על שגיאה; אם בערימה יש רק איבר אחד, הרי הוא בהכרח האיבר שמנסים להוציא (אחרת היינו מקבלים שגיאה קודם), ולכן נוציא אותו ונחסוך פעולות מיותרות נוספות.

אם הכל כשורה –

האלגוריתם יחליף בין האיבר שאנו מנסים להוציא לבין האיבר האחרון בערימה (כדי שיהיה יותר נוח להוציא אותו), ויקרא לשגרת-עזר שמוציאה את האיבר האחרון מהערימה (בודקת את ערכו ואז מקטינה את המשתנה של גודל הערימה ב-1). את ערך האיבר שהוצא נשמור במשתנה חדש. מאחר והעברנו את ערך העלה האחרון לאיבר כלשהו, כלומר ככל הנראה שברנו את ערימת המקסימום - נקרא לשגרה ערך העלה האינדקס של האיבר הנשלף כפרמטר על מנת לסדר את תת העץ שלו לערימת מקסימום. לבסוף נחזיר את המשתנה עם הערך שהוצא.

סיבוכיות זמן ריצה:

האלגוריתם עושה ראשית מספר פעולות בעלות זמן ריצה קבוע (השוואות, הגדרת משתנים, השמת ערכים) שזמן הריצה שלהם הוא בסיבוכיות של $\Theta(1)$,

ולבסוף קורא לשגרה MAX-HEAPIFY שרצה, במקרה הכי גרוע (מחיקת השורש), על כל גובה הערימה ולבסוף קורא לשגרה MAX-HEAPIFY רצה רקורסיבית על כל העץ החל מהשורש החדש לאחר המחיקה) $\lfloor \log_d n \rfloor$ בסיבוכיות של $\Theta(\lg(n))$. כלומר, סיבוכיות זמן הריצה של השגרה כולה היא הסיבוכיות של הפעולה הכי כבדה $\Theta(\lg(n))$. (MAX-HEAPIFY) - (CETה

211768379 סטודנט א' : טל דרוז'ינין, 211371596 סטודנט ב' : שלי גולצמן,

נספח א' – מימושי הפעולות בפתרון

SWAP(A, i, j): temp = A[i] A[i] = A[j] A[j] = temp A[i] A[i] = temp A[i] = temp A[i] A[i] = temp A[i] A[i] = temp A[i] = temp A[i] A[i] = temp A[i]	הערות	המימוש	סיבוכיות זמן ריצה	השגרה
temp = A[j] A[i] = A[i] A[i]				
אמא (lioga nl) A[i] = A[i] A[j] = temp	-	• • • •		
אמאר (log_d nl) בדומה למימוש (log_d nl) בדומה למימוש (log_d nl) (log_d nl) (angest_index = i for child from A.nth_child_index(i, 1) to A.nth_child_index(i		,	, ,	
אמאר א ([log_d n]] בדומה למימוש ([log_d n]] (and x = 1 to child index = 1 to child ind				
Largest_index = i For child from A.nth_child_index(i, 1) to A.nth_child_index(i, d): For child from A.nth_child_index(i, d): For child cannot be used to do ut of d children, which in the worst case is the entire height of the heap height o		01 F		
For child from A.nth_child_index(i, 1) to A.nth_child_index(i, d):		MAX-HEAPIFY(A, i):		MAX-HEAPIFY
וf child < A.HEAP_SIZE and A[child] > A[Largest_index]:	המוצג בפרק 6.2	Largest_index = i		
Largest_index = child If Largest_index = child Out of d children, which in the worst case is the entire height of the heap) If (n + 1)/d Rowring = child from A.get_first_leaf_index() downto 0: MAX-HEAPIFY(A, i) Por i from A.get_first_leaf_index() downto 0: MAX-HEAPIFY(A, i) POP-LAST(A): Last_index = A.HEAP_SIZE - 1 Popped_node = A[last_index] A.HEAP_SIZE = A.HEAP_SIZE - 1 Return Popped_node POP-LAST(A, i): If A.HEAP_SIZE = A.HEAP_SIZE - 1 Error "heap overflow" If A.HEAP_SIZE = A.HEAP_SIZE = 1: Return POP-LAST(A) SWAP(i, 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_DPI = Contact index A.HEAP_SIZE = 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 POPPed_node = EXTRACT(A, Max_index) Return Popped_node EXTRACT-MAX HEAPIFY() Popped_node = EXTRACT(A, Max_index) Return Popped_node Popped_node = EXTRACT(A, Max_index) Return Popped_node RETTRACT-MAX HEAPIFY() Return Popped_node EXTRACT-MAX HEAPIFY() RETTRACT-MAX HEAPIFY() RETTRACT-MAX HEAPIFY() RETTRACT-MAX HEAPIFY()				
וו fl Largest_index != i:		If child < A.HEAP_SIZE and A[child] > A[Largest_index]:		
אייני אובר אייני איני אייני א		·	_	
אמארHEAPIFY(A, Largest_index) MAX-HEAPIFY(A, Largest_index) BUILD-MAX-HEAP(A): (In + 1)/d המוצג בפרק (Const. time + max-HEAP(A)) (In + 1)/d המוצג בפרק (Const. time + MAX-HEAPIFY) בו SWAP(I, A. HEAP_SIZE - 1) Popped_node EXTRACT (A, i): If A. HEAP_SIZE = 1: Error "heap overflow" If A. HEAP_SIZE = 1: Return POPP-LAST(A) SWAP(I, A. HEAP_SIZE - 1) Popped_node EXTRACT (A, i): EXTRACT (A, i): If A. HEAP_SIZE = 1: Return POP-LAST(A) SWAP(I, A. HEAP_SIZE - 1) Popped_node EXTRACT (A, i): Return Popped_node EXTRACT (A, i): EXTRACT (A, i): 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 EXTRACT (A, i): EXTRACT (A, i): Return Popped_node		~ =	·	
אונע הפרק ((((n + 1)/d) height of the heap) BUILD-MAX-HEAP(A):				
Build-Max-Heap(a):		MAX-HEAPIFY(A, Largest_index)		
For i from A.get_first_leaf_index() downto 0:				
אמאר המימוש של החישוב העזר (calls MAX-HEAPIFY (A, i) $= O(n*lg(n))$ (calls MAX-HEAPIFY for each node of the neap that isn't a leaf) $= O(n*lg(n))$ (calls MAX-HEAPIFY for each node of the heap that isn't a leaf) $= O(n*lg(n))$ (Const. time) $= O(n$				BUILD-MAX-HEAP
ארבור ביי מישוב העזר (calls MAX-HEAPIFY for each node of the heap that isn't a leaf) POP-LAST(A): Last_index = A.HEAP_SIZE - 1 Popped_node = A[last_index] A.HEAP_SIZE = A.HEAP_SIZE - 1 Return Popped_node EXTRACT (A, i): If A.HEAP_SIZE < i: EYTRACT-MAX 6.5 א בפרק (Const. time) EXTRACT (A): Error "heap overflow" If A.HEAP_SIZE < 1: Error "heap underflow" If A.HEAP_SIZE < 1: Return POP-LAST(A) SWAP(i, A.HEAP_SIZE < 1) Return POP-LAST(A) SWAP(i, A.HEAP_SIZE < 1) Return Popped_node EXTRACT (A): Return POP-LAST(A) SWAP(i, A.HEAP_SIZE = 1): Return Popped_node EXTRACT (A): Return Popped_node EXTRACT-MAX (B): Return Popped_node EXTRACT-MAX (Const. time + MAX-HEAPIFY)	-			
אינור בסעיף א' פונקציית עזר poped_node = A.HEAP_SIZE - 1 Popped_node = A[last_index] A.HEAP_SIZE = A.HEAP_SIZE - 1 Return Popped_node EXTRACT (A, i): If A.HEAP_SIZE < i: EYRACT-MAX 6.5 מקרה פרטי של if i < 0 or A.HEAP_SIZE < 1: Return POP-LAST(A) SWAP(i, A.HEAP_SIZE < 1) Return Popped_node EXTRACT (A) EXTRACT (A) EXTRACT (A): If A.HEAP_SIZE < i: Error "heap overflow" If 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 EXTRACT (A): (Const. time + MAX-HEAPIFY(A, i) Return Popped_node EXTRACT (A): (B(1) + [log_d n]) (Const. time + Indicated to the heap that is n't a leaf) (B(1) + [log_d n]) (Const. time + Indicated to the heap that is n't a leaf) (B(1) + [log_d n]) (Const. time + Indicated to the heap that is n't a leaf) (Const. time + MAX-HEAPIFY) EXTRACT (A): (B(1) + [log_d n]) (Const. time + MAX-HEAPIFY) EXTRACT-MAX (Const. time + MAX-HEAPIFY) EXTRACT-MAX (Const. time + MAX-HEAPIFY)		MAX-HEAPIFY(A, I)		
ארביר (פונקציית עזר (מוקציית עזר אומר (מוקציית עזר עזר (מוקציית עזר (מוקצית עזר (מוקצית עזר (מוקצית עזר (מוקצית עזר (מוקציית עזר (מוקצית עזר (מוקצית עזר (מוקצי			, ,	
POP-LAST(A): Last_index = A.HEAP_SIZE - 1 Popped_node = A[last_index] A.HEAP_SIZE = A.HEAP_SIZE - 1 Return Popped_node EXTRACT (A, i): If A.HEAP_SIZE < i: EYRACT-MAX 6.5 א המימוש של ול י כ י סיי א המימוש של ול י כ י סיי א המימוש של ול י של ול י של ול ול י של	וואוז בסען א			
Last_index = A.HEAP_SIZE - 1 Popped_node = A[last_index] A.HEAP_SIZE = A.HEAP_SIZE - 1 Return Popped_node 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 EXTRACT-MAX(A): MAX_index = 0 Popped_node = EXTRACT(A, Max_index) Return Popped_node EXTRACT-MAX (Const. time) ((O(1) + [log_d n]) ((Const. time) EXTRACT ((O(1) + [log_d n]) ((O(1) + [log_d	פונקעיים עזר	POP-I ΔST(Δ)·	·	POP-LAST
Popped_node = A[last_index] A.HEAP_SIZE = A.HEAP_SIZE -1 Return Popped_node EXTRACT (A, i): If A.HEAP_SIZE < i: EXTRACT-MAX 6.5 המוצג בפרק 1: Error "heap overflow" If 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 EXTRACT-MAX(A): Max_i index = 0 Popped_node = EXTRACT(A, Max_index) Return Popped_node EXTRACT-MAX $(\Theta(1) + \lfloor \log_d n \rfloor)$ $= O(lg(n))$ $= O(lg(n$	פוניוןב וניעוו	` '		TOT LAST
אוד באר		-	(construint)	
Return Popped_node				
בארה כללי של EXTRACT (A, i): EXTRACT (A, i): ($\Theta(1) + \lfloor \log_d n \rfloor$) EXTRACT EXTRACT-MAX (Const. time + MAX-HEAPIFY) (Const. time + MAX-HEAPIFY) Error "heap overflow" (Const. time + MAX-HEAPIFY) 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 EXTRACT-MAX(A): ($\Theta(1) + \lfloor \log_d n \rfloor$) EXTRACT-MAX Heapify (Const. time + MAX-Heapify) Amx_index = 0 Popped_node EXTRACT(A, Max_index) Return Popped_node EXTRACT(A, Max_index) Return Popped_node EXTRACT(A, Max_index) Return Popped_node EXTRACT(A, Max_index) EXTRACT-MAX(A): (Const. time + MAX-HEAPIFY) EXTRACT-MAX(A): (Const. ti				
ווא של אוד (Const. time + MAX-HEAPISIZE < i: $= O(lg(n))$ (Const. time + MAX-HEAPISIZE < 1: $= O(lg(n))$ (Const. time + MAX-HEAPISIZE < 1: $= O(lg(n))$ (Const. time + MAX-HEAPISIZE < 1: $= O(lg(n))$ (Const. time + MAX-HEAPIFY) If $l = l = l$ (Const. time + MAX-HEAPIFY) If $l = l = l$ (Const. time + MAX-HEAPIFY) If $l = l = l$ (Const. time + MAX-HEAPIFY) If $l = l = l$ (Const. time + MAX-HEAPIFY) If $l = l = l$ (Const. time + MAX-HEAPIFY) If $l = l = l$ (Const. time + MAX-HEAPIFY) If $l = l = l$ (Const. time + MAX-HEAPIFY) If $l = l = l$ (Const. time + MAX-HEAPIFY) If $l = l = l$ (Const. time + MAX-HEAPIFY) If $l = l = l$ (Const. time + MAX-HEAPIFY)		•• =		
EXTRACT-MAX 6.5 בפרק "heap overflow" (Const. time + MAX-HEAPIFY) (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 ($\Theta(1) + \lfloor \log_d n \rfloor$) EXTRACT-MAX $\Theta(1)$ $\Theta(1)$ EXTRACT-MAX $\Theta(1)$	· ·	EXTRACT (A, i):		EXTRACT
6.5 המוצג בפרק הוצג ווf 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 EXTRACT-MAX(A): Max_index = 0 Popped_node = EXTRACT(A, Max_index) index = 0 Popped_node = EXTRACT(A, Max_index) Return Popped_node EXTRACT-MAX (O(1) + [log_d n]) = O(lg(n)) (Const. time + MAX-HEAPIFY)		-	1017	
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 EXTRACT-MAX(A): Max_index = 0 Popped_node = EXTRACT(A, Max_index) Amin Amin Amin Amin Amin Amin Amin A		·	· •	
ון 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 EXTRACT-MAX(A): Max_index = 0 Popped_node = EXTRACT(A, Max_index) Return Popped_node EXTRACT-MAX(A): Max_index = 0 Popped_node = EXTRACT(A, Max_index) Return Popped_node Return Popped_node EXTRACT-MAX EXTRACT-MAX EXTRACT-MAX HEAPIFY)	המוצג בפרק 6.5		HEAPIFY)	
Return POP-LAST(A) SWAP(i, A.HEAP_SIZE - 1) Popped_node = POP-LAST(A) MAX-HEAPIFY(A, i) Return Popped_node		·		
$SWAP(i, A.HEAP_SIZE - 1)$		_		
Popped_node = POP-LAST(A) MAX-HEAPIFY(A, i) Return Popped_node		` ,		
$MAX-HEAPIFY(A, i)$ $Return Popped_node$ $MAX-HEAPIFY(A, i)$ $Return Popped_node$ $MAX-HEAPIFY(A, i)$ $Return Popped_node$ $MAX-HEAPIFY(A, i)$ $May_index = 0$ $Max_index = $				
Return Popped_node				
מקרה פרטי של EXTRACT-MAX(A):		, · · ,		
לשורש EXTRACT $Max_index = 0$ $= O(lg(n))$ (Const. time + MAX-harponal popular) $Return\ Popped_node$ $EXTRACT(A, Max_index)$ $Return\ Popped_node$ $Return\ Popped_node$		Return Popped_node		
לשורש EXTRACT $Max_index = 0$ $= O(lg(n))$ (Const. time + MAX-harponal popular) $Return\ Popped_node$ $EXTRACT(A, Max_index)$ $Return\ Popped_node$ $EXTRACT(A, Max_index)$ $HEAPIFY)$	מקרה פרטי של	FXTRACT-MAX(A):	$(\Theta(1) + \log n)$	FXTRACT-MAX
ראיבר Popped_node = EXTRACT(A, Max_index) (Const. time + MAX- המקסימלי Return Popped_node בערימת	•	· ·		27110 (01 141) (1
המקסימלי Return Popped_node HEAPIFY) בערימת				
בערימת	,		'	
מקסימום)	· ·		, ,	
	מקסימום)			

211768379 סטודנט א' : טל דרוז'ינין, 211371596 סטודנט ב' : שלי גולצמן,

בדומה למימוש המוצג בפרק 6.5	Error "Heap overflow." If i < 0 or A.HEAP_SIZE < 1: 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): SWAP(i, Parent_index) i = Parent_index	$(\Theta(1) + [\log_d n])$ = $O(lg(n))$ (Swaps the value to his right place, which in the worst case is the entire height of the heap)	INCREASE-KEY
בדומה למימוש	INSERT(A, v):	$(\Theta(1) + \lfloor \log_d n \rfloor)$	INSERT
המוצג בפרק 6.5	A.HEAP_SIZE += 1	= O(lg(n))	
	A[A.HEAP_SIZE] = v	(Const. time +	
	INCREASE-KEY(A, A.HEAP_SIZE, v)	INCREASE-KEY)	

211768379 סטודנט א' : טל דרוז'ינין, 211371596 2טטודנט ב' : שלי גולצמן, 21

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.

211768379 סטודנט א' : טל דרוז'ינין, 211371596 סטודנט ב' : שלי גולצמן, 211371596

בכללי TODO:

- להוסיף את הפלט של ההרצה לדוגמא תחת נספח ב' TODO שלי
- שלי TODO לתקף את הפסאודו קוד לפי הגרסא הסופית של הקוד
- 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