סטודנט א' : טל דרוז'ינין, 211768379
סטודנט 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 לימין, 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):

a. מציאת אינדקס האב של האיבר ה-i : $\left\lfloor\frac{i-1}{d}\right\rfloor$
b. מציאת האינדקס של הבן הn-י (n=1,2,3…) של אינדקס i : $((di)+n)$
c. האינדקס הראשון של הרמה ה-i-ית (כאשר השורש הוא רמה 0) : $\frac{d^i-1}{d-1}$
d. האינדקס הראשון של העלים : $\left\lfloor\frac{n+1}{d}\right\rfloor$
e. גובה הערימה : $\lfloor\log_d n\rfloor$ (ראה סעיף ב')

ב. 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- $n = d^{h-1} + k$.
מכך נובע שהגובה, כלומר, אורך המסלול הפשוט הארוך ביותר מהשורש לעלה כלשהו יהיה -
$$n = d^{h-1} + k \rightarrow h = \lfloor\log_d n\rfloor$$
כלומר, הגובה של ערימה d-ית בת n איברים הוא $\lfloor\log_d n\rfloor$.

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

<u>המטלה התכנותית – שפת python</u>

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 $d * \Theta(1)$ 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', $\lfloor \log_d n \rfloor$, מכיוון שהשגרה תרוץ רקורסיבית במקרה הכי גרוע על אורכו של אחד מ-d תתי העצים של השורש, אך ברמה האסימפטותית זה שקול לסיבוכיות של $\Theta(\lg(n))$:

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-

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

לפי מקרה 2 של נוסאחת האב (משפט 4.1) –

$$a = 1, b = \frac{1}{q}, f(n) = \Theta(1) \rightarrow$$
$$f(n) = \Theta\left(n^{\log_{\frac{1}{q}} 1}\right) = \Theta(n^0) = \Theta(1) \rightarrow$$
$$T(n) = \Theta(n^0 * \lg(n)) = \Theta(\lg(n))$$

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

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

ג. בפתרון מימשנו את פעולה **EXTRACT-MAX(A)** על ערימת מקסימום d-ית (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 MAX-HEAPIFY שרצה על כל גובה הערימה $\lfloor \log_d n \rfloor$ (השגרה MAX-HEAPIFY רצה רקורסיבית על כל העץ 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 (MAX-HEAPIFY) - $\Theta(\lg(n))$.

ד. בפתרון מימשנו את פעולה **INSERT(A, v)** על ערימת מקסימום d-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 $\Theta(1)$,
ולבסוף קורא לשגרה INCREASE-KEY שרצה במקרה הכי גרוע לאורך כל הגובה העץ $\lfloor \log_d n \rfloor$ (מהעלה לשורש) בסיבוכיות של $\Theta(\lg(n))$. כלומר, סיבוכיות זמן הריצה של השגרה כולה היא הסיבוכיות של הפעולה הכי כבדה (INCREASE-KEY) - $\Theta(\lg(n))$.

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

ה. 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'):
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 $\lfloor \log_d n \rfloor$ מלמטה למעלה עד השורש, בסיבוכיות של $\Theta(\lg(n))$. כלומר, סיבוכיות זמן הריצה של השגרה כולה היא - $\Theta(\lg(n))$.

ו. 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'):
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 MAX-HEAPIFY עם 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 $\lfloor \log_d n \rfloor$ (השגרה MAX-HEAPIFY רצה רקורסיבית על כל העץ החל מהשורש החדש לאחר המחיקה) בסיבוכיות של $\Theta(\lg(n))$. כלומר, סיבוכיות זמן הריצה של השגרה כולה היא הסיבוכיות של הפעולה הכי כבדה (MAX-HEAPIFY) - $\Theta(\lg(n))$.

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

נספח א' – מימושי הפעולות בפתרון

השגרה	סיבוכיות זמן ריצה	המימוש	הערות
SWAP	$\Theta(1)$ (Const. time)	SWAP(A, i, j): temp = A[i] A[i] = A[j] A[j] = temp	פונקציית עזר

MAX-HEAPIFY	$(\lfloor \log_d n \rfloor)$
	$= O(lg(n))$
	(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

BUILD-MAX-HEAP	$(\lfloor (n+1)/d \rfloor$ $* \log_d(n))$ $= O(n * lg(n))$ (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'
POP-LAST	$\Theta(1)$ (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

EXTRACT	$(\Theta(1) + \lfloor \log_d n \rfloor)$
	$= O(lg(n))$
	(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
			מקרה כללי של המימוש של EXTRACT-MAX המוצג בפרק 6.5

EXTRACT-MAX	$(\Theta(1) + \lfloor \log_d n \rfloor)$ $= O(lg(n))$ (Const. time + MAX-HEAPIFY)	EXTRACT-MAX(A): Max_index = 0 Popped_node = EXTRACT(A, Max_index) Return Popped_node	מקרה פרטי של EXTRACT לשורש (האיבר המקסימלי בערימת מקסימום)

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

בדומה למימוש המוצג בפרק 6.5	INCREASE-KEY(A, i, v):
	If A. HEAP_SIZE < i:
	   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
	      Else:
	         Return
		$(\Theta(1) + \lfloor \log_d n \rfloor)$
		$= O(lg(n))$
		(Swaps the value to his right place, which in the worst case is the entire height of the heap)
			INCREASE-KEY

בדומה למימוש המוצג בפרק 6.5	INSERT(A, v): A.HEAP_SIZE += 1 A[A.HEAP_SIZE] = v INCREASE-KEY(A, A.HEAP_SIZE, v)	$(\Theta(1) + \lfloor \log_d n \rfloor)$ $= O(lg(n))$ (Const. time + INCREASE-KEY)	INSERT

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

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.

```
python main.py
> Insert D for d-heap:3
> Input path to file with heap list (default=input.txt):
Reading heap from path input.txt...
Got list (size 22): [3, 9, 2, 11, 14, 5, -5, 7, 15, -1, 0, -99999, 6, 10, 20,
-12, 1, 17, 4, 13, 16, -2000]
Converting into Max Heap...
Heap is ready!
--------------
                                                         20
                  17                    ||                    15                    ||                    11
      14           5           16      ||      7            2            -1      ||      0           -99999          6
 10    9   -12 ||  1    3    4  || 13   -5  -2000

Possible actions on heap:
    1. Extract maximum value
    2. Insert value
    3. Increase key
    4. Remove key
    5. Load different heap
    6. Exit
> Enter action: 1
    Extracted max node with value 20 from heap.
                                                         17
                  16                    ||                    15                    ||                    11
      14           5           13      ||      7            2            -1      ||      0           -99999          6
 10    9   -12 ||  1    3    4  ||-2000 -5

Possible actions on heap:
    1. Extract maximum value
    2. Insert value
    3. Increase key
    4. Remove key
    5. Load different heap
    6. Exit
> Enter action: 2
    >Input value to insert: 11
                                                         17
                  16                    ||                    15                    ||                    11
      14           5           13      ||      7            2            -1      ||      0           -99999          6
 10    9   -12 ||  1    3    4  ||-2000 -5   11

Possible actions on heap:
    1. Extract maximum value
    2. Insert value
    3. Increase key
    4. Remove key
    5. Load different heap
    6. Exit
> Enter action: 2
    >Input value to insert: 30
                                                         30
                  16                    ||                    17                    ||                    11
      14           5           13      ||      15           2            -1      ||      0           -99999          6
 10    9   -12 ||  1    3    4  ||-2000 -5   11  ||  7

Possible actions on heap:
```

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

```
    1. Extract maximum value
    2. Insert value
    3. Increase key
    4. Remove key
    5. Load different heap
    6. Exit
> Enter action: 3
    >Input index to increase: 0
    >Input value: 1
                                                             30
                   16                     ||                   17                     ||                   11
      14            5              13      ||      15            2              -1      ||      0           -99999           6
 10    9   -12 || 1    3    4  ||-2000 -5   11  ||  7

Possible actions on heap:
    1. Extract maximum value
    2. Insert value
    3. Increase key
    4. Remove key
    5. Load different heap
    6. Exit
> Enter action: 3
    >Input index to increase: 2
    >Input value: 31
                                                             31
                   16                     ||                   30                     ||                   11
      14            5              13      ||      15            2              -1      ||      0           -99999           6
 10    9   -12 || 1    3    4  ||-2000 -5   11  ||  7

Possible actions on heap:
    1. Extract maximum value
    2. Insert value
    3. Increase key
    4. Remove key
    5. Load different heap
    6. Exit
> Enter action: 4
    >Input index to remove: 10
    Extracted node #10: 0.
                                                             31
                   16                     ||                   30                     ||                   11
      14            5              13      ||      15            2              -1      ||      7           -99999           6
 10    9   -12 || 1    3    4  ||-2000 -5   11

Possible actions on heap:
    1. Extract maximum value
    2. Insert value
    3. Increase key
    4. Remove key
    5. Load different heap
    6. Exit
> Enter action: 5
> Insert D for d-heap:4
> Input path to file with heap list (default=input.txt): input2.txt
Reading heap from path input2.txt...
Got list (size 7): [1, 2, 3, 31, 3, 4, 5]
Converting into Max Heap...
Heap is ready!
--------------

                                    31
        5          ||        3          ||       1          ||        3
  4    2

Possible actions on heap:
```

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

```
    1. Extract maximum value
    2. Insert value
    3. Increase key
    4. Remove key
    5. Load different heap
    6. Exit
> Enter action: 6
Exiting...
```