מבני נתונים – תרגיל בית מעשי מספר 2

פרטי המגישות:

מגישה 1:

שם משתמש: koslowsky

(בעברית - איילה קוסלובסקי)

שם המגישה: איילה קוסלובסקי

תעודת זהות: 207618323

מגישה 2:

שם משתמש: ronishamai

(בעברית – רוני שמאי)

שם המגישה: רוני שמאי

תעודת זהות: 3190936909

תיעוד חיצוני ופירוט סיבוכיות זמן-ריצה של הקוד:

בקוד שלנו, יש מחלקה פנימית אחת: **HeapNode**.

המחלקה HeapNode:

- השדות במחלקה HeapNode:

HeapNode info
int key
HeapNode next
HeapNode prev
HeapNode parent
HeapNode child
int rank
boolean mark

- הבנאי של המחלקה: בנאי המקבל ערך אחד בלבד (int key).

- פירוט המתודות במחלקה HeapNode:

המתודה	הסבר	סיבוכיות זמן ריצה
int getKey()	מחזירה את השדה key, את המפתח, של הצומת	סיבוכיות זמן קבועה – O(1), מחזירים מצביע
void setKey(int key)	מעדכנת את השדה key של הצומת	
HeapNode getInfo()	מחזירה את השדה info של הצומת	
void setInfo(HeapNode node)	מעדכנת את השדה info של הצומת	
void setNext(HeapNode node)	מעדכנת את המצביע לצומת next של הצומת, ל HeapNode שמקבלת	
HeapNode getNext()	מחזירה את השדה next של הצומת, כלומר מחזירה את המצביע ל HeapNode שהינו הצומת שאחריו ברשימה המקושרת	
void setPrev(HeapNode node)	מעדכנת את המצביע לprev של הצומת, ל HeapNode שמקבלת	
HeapNode getPrev()	מחזירה את השדה prev של הצומת, כלומר מחזירה את המצביע ל HeapNode שהינו הצומת שלפניו ברשימה המקושרת	

void setParent(IAVLNode node)	מעדכנת את המצביע להורה של הצומת, ל HeapNode שמקבלת	
HeapNode getParent()	מחזירה את השדה parent של הצומת, כלומר מחזירה את המצביע ל HeapNode שהינו ההורה של הצומת; אם הוא שורש בערימה מחזיר null	
boolean isMark()	מחזירה true אם הצומת מסומן, כלומר בוצע cut עם אחד מילדיו	
void setMark(boolean b)	מעדכנת את השדה mark של הצומת	
int setRank(int k)	מחזירה את הדרגה של הצומת, כלומר כמות הילדים של הצומת	
int getRank()	מחזירה את הדרגה של הצומת	

המחלקה FibonacciHeap:

- השדות במחלקה FibonacciHeap:
  - שדות שאינם סטטיים:

private int size
private HeapNode firstRoot
private HeapNode minNode
private int num of trees
private int numOfMarkedNodes

  - שדות סטטיים:

private static int countCuts
private static int countLinks

*השדות הסטטיים מאותחלים ל 0.

- בנאי המחלקה FibonacciHeap:
  - בנאי ריק, המאתחל את שדות העץ באופן הבא: firstRoot ו- minNode יצביעו ל null, וגודל הערימה, כמות העצים וכמות הצמתים המסומנים יהיו אפס.

- מתודות המחלקה FibonacciHeap:

*נציין כי כלל פונק' העזר ופירוטן מופיעות תחת ההסבר על הפונק' שקוראת להן.

המתודה	הסבר + מתודות העזר בהן השתמשנו, ופירוט אודותן	סיבוכיות זמן הריצה
boolean isEmpty()	מחזירה True אם העץ ריק (אם גודל העץ הוא אפס); אחרת, מחזירה False	סיבוכיות זמן קבועה – **O(1)**, ניגשים לשדה size של הערימה ומחזירים ערך בוליאני
int insert(int k)	ראשית ניצור צומת חדש עם הkey שקיבלנו. כעת אם הערימה ריקה, ה-next וה-prev שלו מצביעים לעצמו, כי זוהי ערימה מקושרת דו כיוונית בעלת איבר אחד. נגדיר אותו להיות האיבר המינימלי בערימה. אחרת, בעזרת **insertAfter** נכניס את הצומת לרשימת השורשים אחרי האיבר הכי "ימני" ברשימה, כך שהאיבר החדש שהכנסנו יהיה האיבר הכי "שמאלי" ברשימה. נעדכן את כל השדות של המחלקה בהתאם.	יצירת צומת חדש לוקחת זמן קבוע O(1). הכנסה לרשימה ריקה או לא ריקה בעזרת קריאה ל-insertAfter לוקחת O(1), וגם שינוי של השדות. לכן סה"כ סיבוכיות זמן קבועה- **O(1)**
	void insertAfter(HeapNode A, HeapNode B) משנה ארבעה מצביעים כך שהצומת B תופיע אחרי הצומת A ברשימה המקושרת.	שינוי כמות קבועה של מצביעים בסיבוכיות זמן קבועה, לכן: **O(1)**

int deleteMin(int k)	אם הערימה ריקה, לא נעשה כלום. אחרת, אם לצומת שאנו רוצים למחוק יש ילדים נקרא ל- **deleteNodeWithChildren** ואם לצומת אין ילדים נקרא ל- **deleteNodeWithoutChildren**. אם המערך אינו ריק לאחר המחיקה נקרא ל- **consolidate** על מנת להפוך את הערימה לערימה "כמעט בינומית".	consolidate עולה O(n), כאשר שאר המתודות שנקראות הן בסיבוכיות זמן ריצה נמוכה יותר אסימפטוטית. שאר הפעולות בסיבוכיות זמן ריצה קבועה. לכן הסיבוכיות היא - **O(n)**
	void deleteNodeWithChildren(HeapNode node) אם הצומת שאנו רוצים למחוק היא הילד היחיד של ההורה נגדיר עבור האבא את הילד שלו להיות null. אחרת, נעבור על כל הילדים של הצומת ונגדיר את ההורה שלהם להיות null. אם אין לצומת שמחקנו "אחים", נהפוך את רשימת השורשים להיות רשימת הילדים. אחרת נחבר את הילדים לרשימת השורשים ע"י שינוי מספר קבוע של מצביעים. נעדכן את השדות בהתאם.	סיבוכיות זמן הריצה של כל הפעולות היא קבועה, למעט עדכון מצביע ההורה של כל אחד מהילדים; לצומת מדרגה k יש k ילדים, והוכחנו בהרצאה שדרגות הן O(logn). לכן סה"כ הסיבוכיות היא- **O(log(n)).**
	void deleteNodeWithoutChildren(HeapNode node) אם הצומת שאנו מוחקים הוא הצומת היחיד בערימה, נהפוך אותה לערימה ריקה. אחרת, נשנה שני מצביעים, של הצומת הקודם והצומת העוקב ברשימה המקושרת, ונוציא את הצומת מהרשימה. נעדכן את השדות בהתאם.	שינוי מספר קבוע של מצביעים ושדות לכן סיבוכיות זמן קבועה- **O(1)**
	HeapNode link(HeapNode x, HeapNode y) הפונקציה מחברת בין שני עצים בעלי אותה דרגה כך שהשורש של עץ אחד הופך להיות הילד של העץ השני. נעשה זאת באופן הבא- אם x גדול מ-y נחליף ביניהם. נמחק את y מרשימת השורשים ונחבר אותו לרשימת הילדים של x. נעדכן את השדות בהתאם.	שינוי מספר קבוע של מצביעים ושדות לכן סיבוכיות זמן קבועה- **O(1)**
	HeapNode[ ] toBuckets() נכניס כל שורש בערימה למערך דרגות, בהתאם לדרגה שלו. אם הערימה ריקה נחזיר מערך ריק. אחרת, נעבור על השורשים ונכניס את העץ מדרגה i למקום ה-i במערך. אם במקום אליו הכנסנו במערך קיים כבר עץ נבצע ביניהם חיבור בעזרת הפונקציה **link** ונכניס את העץ המחובר למקום הנכון במערך בהתאם לדרגתו החדשה. לבסוף נחזיר את המערך.	בערימת פיבונצ'י בעלת n צמתים יש לכל היותר n עצים (כל עץ הוא צומת בודד), במקרה הגרוע נעבור על כולם. קריאה לlink לוקחת O(1), כך גם שאר הפעולות במתודה, ולכן הסיבוכיות היא לכל היותר- **O(n)**
	HeapNode fromBuckets(HeapNode[ ] b) נרצה לחבר את כל העצים שקיבלנו במערך B על ידי **toBuckets** לכדי ערימה אחת שכל השורשים שלה מחוברים ע"י ערימה מקושרת דו כיוונית. נעבור בלולאה על המערך ונחבר כל שורש מהמערך לרשימת השורשים של הערימה בעזרת	ראינו בהרצאה שדרגות הן O(logn), לכן יש O(log(n)) תאים במערך B – כמס' האיטרציות הגדול ביותר שנבצע. בכל איטרציה אנו מבצעים פעולות שלוקחות O(1) – שינוי מצביעים וקריאה ל insertAfter ולכן סה"כ הסיבוכיות היא-

	insertAfter. תוך כדי הלולאה נבצע השוואות על מנת למצוא את האיבר המינימלי מבין כל השורשים שהוא גם האיבר המינימלי בערימה כולה.	**O(log(n))**
	void consolidate() נקרא לפונקציה **toBuckets** אשר מחזירה מערך של עצים בו יש לכל היותר עץ אחד מכל דרגה, ולאחר מכן נקרא לפונקציה **fromBuckets** על המערך שקיבלנו.	toBuckets לוקחת O(n) ו-fromBuckets לוקחת O(log(n)) לכן סה"כ הסיבוכיות היא- **O(n)**
HeapNode findMin()	אם הערימה ריקה נחזיר null; אחרת, נחזיר את השדה minNode של הערימה.	נחזיר מצביע, לכן סיבוכיות זמן קבועה- **O(1)**
void meld()	נחבר שתי ערימות פיבונצ'י לערימה אחת. נשנה את המצביעים של האיבר הראשון והאחרון בכל אחת מהערימות כך שנקבל רשימה מקושרת אחת שמכילה את השורשים של שתי הערימות. נשנה את השדות בהתאם.	שינוי מספר קבוע של מצביעים ושינוי שדות לוקח זמן קבוע- **O(1)**
Int size()	נחזיר את השדה size של הערימה – מספר הצמתים בערימה.	מחזירים מצביע בסיבוכיות זמן קבועה, לכן: **O(1)**
Int[] countersRep()	אם הערימה ריקה נחזיר מערך ריק. אחרת, נעבור בלולאה על רשימת השורשים, נמצא את הדרגה המקסימלית – ונעדכן אותה להיות גודל המערך. עבור כל שורש מדרגה k נגדיל את המקום ה-k במערך ב-1 ולבסוף נחזיר את המערך.	יש לכל היותר O(n) שורשים בערימה, ונעבור על כולם – פעמיים (פעם אחת למציאת הדרגה המקסימלית, בפעם השנייה לעדכון המערך). כל שאר הפעולות בסיבוכיות זמן ריצה קבועה, ולכן הסיבוכיות היא- **O(n)**
void delete(HeapNode x)	נבצע **decreaseKey** על המפתח של הצומת אותו נרצה למחוק, כך שהוא יהפוך למינימום בערימה על ידי הפחתת הערך המקסימלי של int ממנו. לאחר מכן, נבצע **deleteMin**.	decreaseKey עולה O(log(n)). deleteMin עולה O(n) ולכן סה"כ הסיבוכיות היא- **O(n)**
Void decreaseKey(HeapNode x, int delta)	נקטין את המפתח של הצומת בערך מסוים שמקבלים. אם הצומת הוא שורש בערימה, נבדוק אם הוא כעת המינימום בערימה ונעדכן את השדה minNode בהתאם. אחרת, אם הערך החדש קטן מהערך של המפתח של האבא שלו, נבצע **cascadingCuts** על הצומת, נחתוך אותו מהאבא ונהפוך אותו לשורש בערימה.	הסיבוכיות של cascadingCuts היא O(log(n)). כל שאר הפעולות לוקחות זמן קבוע O(1) לכן סה"כ הסיבוכיות היא- **O(log(n))**
	void cut(HeapNode child, HeapNode parent) נחתוך את child מההורה שלו. נהפוך אותו להיות שורש (לא מסומן, מופיע ברשימת השורשים, ואבא שלו null). נשנה כמות סופית של מצביעים ונשתמש ב-insertAfter כך שchild יהפוך להיות ברשימת השורשים. נעדכן את השדות בהתאם.	שינוי כמות סופית של מצביעים לוקחת O(1) וגם קריאה ל-insertAfter. לכן סה"כ סיבוכיות זמן קבועה- **O(1)**

	Void cascadingCuts(HeapNode node) אם node הוא null לא נעשה כלום. אחרת, נבצע **cut** על הצומת ועל אבא שלו. אם האבא אינו מסומן נהפוך אותו למסומן ונסיים. אם הוא מסומן נפעיל גם עליו את הפונקציה ברקורסיה (נחתוך גם אותו מאבא שלו).	Cut לוקחת O(1). לכל היותר נבצע את הפונקציה ברקורסיה עד שנגיע לשורש, ולכן לכל היותר נבצע אותה O(log(n)) פעמים. לכן הסיבוכיות היא- **O(log(n))**
Int potential()	נחזיר את השדה numOfTrees ועוד 2 כפול השדה numOfMarkedNodes.	אנו ניגשים לשדות במחלקה ומבצעים פעולה אריתמטית פשוטה, לכן סיבוכיות זמן קבועה- **O(1)**
int totalLinks()	מחזירה את השדה countLinks של המחלקה.	**O(1)**
int totalCuts()	מחזירה את השדה countCuts של המחלקה.	**O(1)**
int[] kMin(FibonacciHeap H, int k)	נרצה להחזיר את k האיברים הכי קטנים בערימה. ניצור מערך ריק וערימה נוספת ריקה heap. נוסיף לערימה heap את האיבר המינימלי ב-H. נגדיר ב-info של הצומת שהכנסנו מצביע לצומת בעל אותו מפתח בערימה המקורית H. נעבור בלולאה עד שנגיע ל- k. בכל פעם נשמור את השדה info של הצומת בצומת חדש, נכניס את המפתח של הצומת למערך ואז נמחק את הצומת מהערימה heap בעזרת **deleteMin**. כעת נעבור על הילדים של הצומת בערימה המקורית H ונוסיף אותו לערימה heap. נשמור את האיבר המינימלי כעת בערימה ונבצע שוב את הלולאה. לבסוף נחזיר את המערך.	נבצע בדיוק k איטרציות. בכל איטרציה אנו מוסיפים את כל הילדים של הצומת שסימנו כ-min בתחילת האיטרציה. לכל צומת בערימה יש לכל היותר Olog(k) ילדים ולכן סה"כ הסיבוכיות היא- **O(klog(k))**

חלק ניסויי (תיאורטי) - שאלה 1

סעיף א'

זמן הריצה האסימפטוטי של סדרת הפעולות כפונקציה של m היא O(m).

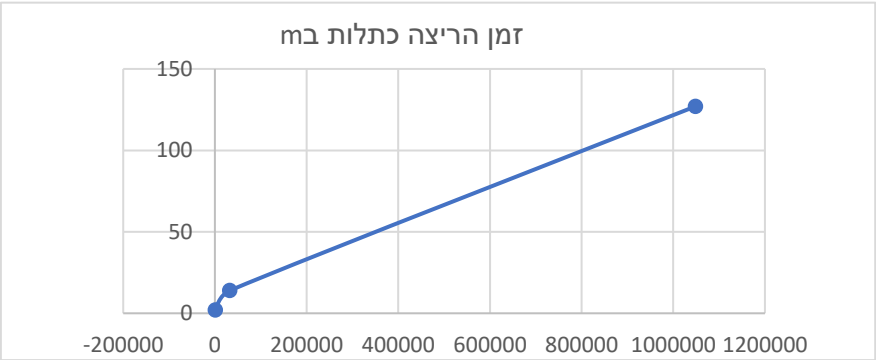
- תחילה, מבצעים m+1 פעולות **insert**, כל אחת בסיבוכיות זמן ריצה $O(1)$ – בסה"כ $O(m)$.
- לאחר מכן, מבצעים פעולת **delete-min** לצומת בעלת המפתח 1, סיבוכיות זמן הריצה:
$O(log(m) + numOfTrees) = O(\log(\mathrm{m}+1) + \mathrm{m} + 1) = O(m)$
(כמות העצים לפני ביצוע הפעולה הייתה ככמות הצמתים שהכנסנו לערימה, והיא m+1, ומדובר במקרה הגרוע ביותר – כל העצים בערימה היו צמתים יחידים).
- לבסוף, מבצעים $\log(m)$ פעולות **decrease-key**; במהלך כל פעולה כזו, הן מעדכנים את ערך המפתח של הצומת לערך אחר בסיבוכיות זמן ריצה קבועה, והן קוראים לפונקציה consolidate, במהלכה מתבצע חיתוך יחיד וללא חזרות רקורסיביות: מוחקים בן של השורש בכל פעם (פירוט על כך בסעיף ג' מטה). לכן כל קריאה כנ"ל מבין $\log(m)$ הקריאות ל consolidate הינה בסיבוכיות זמן ריצה קבועה – שינוי מס' קבוע של מצביעים. בסה"כ: $O(\log(m))$.

כלומר, **סיבוכיות זמן הריצה הכוללת של סדרת הפעולות כפונקציה של m היא O(m).**

סעיף ב'

m	Run-Time (ms)	totalLinks	totalCuts	Potential
2^{10}	2	1023	10	29
2^{15}	14	32767	15	44
2^{20}	127	1048575	20	59
2^{25}	3306	33554431	25	74

(תואם לתשובה מסעיף א' וראו הגרף מטה)



סעיף ג'

תחילה, מבצעים $m+1$ פעולות insert, יוצרים $m+1$ עצים בעלי צומת יחיד בערימה.

לאחר מכן, מבצעים פעולת delete-min לצומת בעל המפתח 1-. מפני שכל צומת הוכנס כעץ בעל צומת בודד, נבצע:

- מחיקה של הצומת המינימלי מרשימת השורשים ע"י שינוי מספר קבוע של מצביעים.
- פעולת consolidate, כאשר בראשיתה יש ברשותנו m צמתים מדרגה 0 כל אחד.
בסה"כ, במסגרת ה consolidate נבצע (בסדר שונה):
  - פעולת link בין עצים מדרגה 0 לכל זוג איברים מתוך m העצים הנתונים מדרגה 0, שכן כולם מדרגה 0 בהתחלה – לכדי סה"כ $\frac{m}{2}$ עצים מדרגה 1 ($\frac{m}{2}$ פעולות *link*)

- פעולת link בין עצים מדרגה 1 לכל זוג איברים מתוך $\frac{m}{2}$ העצים מדרגה 1 – לכדי $\frac{m}{2} \times \frac{1}{2} = \frac{m}{4}$ עצים מדרגה 2 ($\frac{m}{4}$ פעולות *link*)
- פעולת link בין עצים מדרגה 2 לכל זוג איברים מתוך $\frac{m}{4}$ העצים מדרגה 2 – לכדי $\frac{m}{4} \times \frac{1}{2} = \frac{m}{8}$ עצים מדרגה 3 ($\frac{m}{8}$ פעולות *link*)

כך הלאה, עד שלבסוף נבצע:

- פעולת link לכל זוג עצים מתוך $\frac{m}{2^{logm-1}}$ העצים מדרגה $logm - 1$, לכדי $\frac{m}{2^{logm-1}} \times \frac{1}{2} = \frac{m}{2^{logm}} = 1$ עצים מדרגה $logm$ (עץ יחיד, מדרגה זו) ($\frac{m}{2^{logm}} = 1$ פעולות *link*)

לכן, בסה"כ נבצע:

$$\frac{m}{2} + \frac{m}{4} + \frac{m}{8} + \cdots + 1 = m\left(\frac{1}{2^1} + \frac{1}{2^2} + \cdots + \frac{1}{2^{logm}}\right) = m\sum_{i=1}^{logm}\frac{1}{2^i} = m \times \frac{\frac{1}{2}\left(\left(\frac{1}{2}\right)^{logm} - 1\right)}{\frac{1}{2} - 1} =$$

$$= -m \times \left(\left(\frac{1}{2}\right)^{logm} - 1\right) = -m\left(\frac{1}{2^{logm}} - 1\right) = -\left(\frac{m}{m} - m\right) = -(1 - m) = \boldsymbol{m - 1}$$

פעולות link.

<u>בשלב הבא, מבצעים $\log(m)$ פעולות decrease-key:</u>

נשים לב:

- מתחילים את סדרת הפעולות לאחר פעולת delete-min, בה קיבלנו ערימה בינומית, המורכבת מעץ אחד ובו m צמתים, בה המפתח המינימלי הוא 0. דרגתו היא $\log(m)$, וכל אחד מהבנים שלו הוא עץ בינומי ששורשו הוא חזקה שלמה של 2, הקטנה ממש מ m. העץ הנ"ל מכיל את כל המספרים החל מ 0 ועד m-1.
- כל המספרים להם נבצע decrease-key הינם סכומים של חזקות של 2. לדוגמא, עבור i = p, נבצע decrease-key למפתח: $\sum_{i=p}^{logm-1} 2^p + 1$. לכן ככל ש i יהיה קטן יותר (או לחלופין ככל שנתקדם איטרציות) כך נבצע decrease-key לצומת בעל ערך גדול יותר.
- לכל מספר שנבצע לו decrease-key, הערך שנשים בו במקום יהיה חזקה שלמה של 2. עבור i = p, הערך שיעודכן עבור המספר לאחר הפעולה יהיה -2^p.

נטען כי בכל פעולת decrease-key נבצע חיתוך אחד לכל היותר:

- באיטרציה הראשונה: משנים את המפתח של הצומת 1 (לכל m) שמהאמור מעלה נמצא תמיד בנק' זמן זו בערימה כבן ישיר של הצומת 0 (הצומת המינימלי ברשימה), להיות בעל ערך קטן יותר ממנו (שלילי) – בניגוד לכלל הערימה.
לכן, נבצע cascadingCuts:
הצומת 0 הוא שורש ולכן לא מסומן; לכן, נבצע cut בין 0 לבין הצומת הנ"ל - שהופך להיות שורש לא מסומן. גם 0 נשאר לא מסומן, מפני ששורשים הם לא מסומנים. כמו כן, הצומת המינימלי מעודכן בהתאם להיות $-m$ (עד סיום סדרת הפעולות לאור האמור מעלה).
- באיטרציה השנייה: משנים את המפתח של הצומת בעל הערך $2^{logm-1} + 1$, לערך שלילי כלשהו (גדול מהערך המינימלי הנוכחי בערימה) – נסמן את צומת זה ב x.
בבירור כלל הערימה מופר: 2^{logm-1} הוא שורש של עץ בינומי, ובפרט הבן הגדול ביותר של שורש העץ ששורשו הוא 0. לכן יש לו $logm - 1$ בנים, והם כל הערכים $2^{logm-1} + 2^t$ כאשר

$0 \le t \le logm - 2$. לכן, ההורה של *x* הוא 2^{logm-1}, וההורה שלו הוא 0 - שניהם גדולים ממנו.
נבצע cascadingCuts:
הצומת x לא מסומן; נבצע cut בינו לבין ההורה שלו - 2^{logm-1}. x הופך להיות שורש לא מסומן, ו 2^{logm-1} הופך להיות מסומן.

- באיטרציה השלישית: משנים את המפתח של הצומת בעל הערך $2^{logm-1} + 2^{logm-2} + 1$, לערך שלילי כלשהו (גדול מהערך המינימלי הנוכחי בערימה) – נסמן את צומת זה ב x.
בבירור כלל הערימה מופר: $2^{logm-1} + 2^{logm-2}$ הוא שורש של עץ בינומי, ובפרט הבן הגדול ביותר של שורש העץ ששורשו הוא 2^{logm-1}. לכן יש לו $logm - 2$ בנים, והם כל הערכים $2^{logm-2} + 2^t$ כאשר $0 \le t \le logm - 3$. לכן, שרשרת הוריו של *x*, מהקרוב-לרחוק היא: $2^{logm-2} + 2^{logm-1}$, 2^{logm-1}, 0.
נבצע cascadingCuts:
הצומת x לא מסומן; נבצע cut בינו לבין ההורה שלו - $2^{logm-2} + 2^{logm-1}$. x הופך להיות שורש לא מסומן, ו $2^{logm-2} + 2^{logm-1}$ הופך להיות מסומן, וסיימנו.

...

- כך הלאה – בכל איטרציה, מבצעים decrease-key לבן הקטן ביותר של שורש של עץ בינומי, לערך שלילי כלשהו המפר את כלל הערימה; מבצעים חיתוך בינו לבין ההורה שלו, מסמנים את ההורה שלו – ומסיימים את הפעולה. הצומת עליו בצענו את הפעולה decrease-key הופך להיות שורש נוסף ברשימת השורשים המגדירים את הערימה.

בכל איטרציה כזו:

1) בצענו חיתוך אחד ויחיד.
2) "צברנו" עוד עץ לרשימת העצים – הצומת שעליו בצענו decrease-key.
3) למעט בסיום האיטרציה הראשונה (שבה ההורה של הצומת שחתכנו הוא שורש), "צברנו" צומת מסומן חדש – ההורה של הצומת אותו חתכנו.

*נציין כי בהכרח בכל פעם מסמנים צומת "חדש" – ההורה תמיד "יורד" דרגה אחת למטה, בכל איטרציה.

קיבלנו בסה"כ $\log(m)$ **חיתוכים**, כמספר פעולות ה decrease-key.

כמו כן, קיבלנו בסה"כ $\log(m) + 1$ **עצים** בערימה; $\log(m)$ עצים ששורשיהם הינם כל אחד מהצמתים שבצענו עבורם decrease-key, ועץ נוסף הכולל את יתר $m - \log(m)$ האיברים הנותרים בערימה. כלל הצמתים בערימה אינם מסומנים.

מספר **הצמתים המסומנים** בסה"כ יהיה – $\boldsymbol{logm - 1}$

<u>לסיכום</u>:

מספר פעולות link במהלך סדרת הפעולות: ***m-1***

מספר פעולות cut במהלך סדרת הפעולות: log(m)

הפוטנציאל של המבנה בסוף הסדרה:

$$numOfTrees + 2 \times marks = log(m) + 1 + 2(logm - 1) = \boldsymbol{3logm - 1}$$

<u>סעיף ד'</u>

אם בשורה 3 נבצע decreaseKey על המפתחות $m - 2^i$:

- מס' ה link - **נשאר זהה**: **m-1**, כי delete-min זוהי הפעולה היחידה שמבצעת link -בסדרת הפעולות הנ"ל, והיא קדמה לשורה 3, השינויים התקיימו אחריה.
- מס' ה cut:

באיטרציה הראשונה – נבצע decrease-key לצומת המינימלי ברשימה, ל 0 – ונשים בו את הערך m-1-. כלל הערימה לא מופר ולמעט עדכון ערכו לא נבצע דבר נוסף, מלבד עדכון הצומת המינימלי בערימה.

בכל אחת מיתר $logm - 1$ הפעמים שנבצע בהן decrease-key: הערך של הצומת שיתקבל יהיה גדול יותר מזה של ההורה שלו – ולכן, כלל הערימה לא מופר ולמעט עדכון ערכו לא נבצע דבר נוסף.

נסביר:

- לאחר האיטרציה הראשונה, שורש העץ הוא בעל הערך m-1-. בכל decrease-key נוסף שנעשה: i קטן ← 2^i קטן ← $m - 2^i - (m-1) = 1 - 2^i$ גדל. כלומר, בכל איטרציה אנחנו משנים את ערך המפתח הרלוונטי לאותה פעולת *decrease-key* לערך גדול יותר מזה ששינינו בפעולת ה *decrease-key* הקודמת.
- בנוסף, בכל פעם מבצעים *decrease-key* לצומת שהיא הצומת שהוא בן ישיר של הצומת שאותו עדכנו באיטרציה הקודמת; זאת לאור העובדה שלאחר פעולת ה *delete-min* במקרה הנ"ל נוצר עץ בינומי תקין מדרגה *logm*. לעץ יש *logm* ילדים, ששורשיהם הם $2^0, \dots, 2^{logm-1}$. באיטרציה השנייה – מבצעים *decrease-key* ל 2^{logm-1} – הבן של מי שהיה 0 במקור, ושינינו למספר שלילי באיטרציה הראשונה. באיטרציה השלישית – מבצעים *decrease-key* ל $2^{logm-1} + 2^{logm-2}$ שהינו הבן הגדול ביותר של 2^{logm-1} מי ששינינו באיטרציה הקודמת, וכך הלאה – בכל איטרציה מבצעים *decrease-key* לצומת שהינו הבן הגדול (לפני פעולת ההפחתה) של הצומת ששינינו קודם לכן.

לכן: לא נבצע *cut* בכלל (לא נכנסים ל *cascading-cuts* כי כלל הערימה לעולם לא מופר). **בסה"כ: 0 פעולות *cut*.**

הפוטנציאל:

בנוסף, מהאמור לעיל עולה שנשאר עם עץ אחד בערימה – לא בצענו שינויים בערימה כלל מלבד שינוי ערכי מפתחות שלא הפרו את כלל הערימה, וגם שמספר הצמתים המסומנים הוא 0. **בסה"כ: הפוטנציאל הוא 1, כמספר העצים בערימה לאחר הפעולות.**

לסיכום:

מספר פעולות link במהלך סדרת הפעולות: *m-1*

מספר פעולות cut במהלך סדרת הפעולות: 0

הפוטנציאל של המבנה בסוף הסדרה: 1

סעיף ה'

אם נמחק את שורה 2:

ראשית, נציין ש**לא נבצע פעולות link בכלל** מכיוון שהן insert והן decreae-key לא מבצעות link. כשנתחיל לבצע את פעולות ה decrease-key, הערימה כוללת m+1 עצים, כל אחד בעל צומת יחיד. לכן, כל פעולת decrease-key **שנבצע לא תפר את כלל הערימה – לא נבצע חיתוכים כלל, לא נסמן צמתים, ולא נשנה את מספר העצים.**

לסיכום:

מספר פעולות link במהלך סדרת הפעולות: 0

מספר פעולות cut במהלך סדרת הפעולות: 0

הפוטנציאל של המבנה בסוף הסדרה: כמספר העצים בערימה לאחר סיום סדרת הפעולות, m+1.

סעיף ו'

אם נוסיף שורה נוספת decreaseKey(m-1,m+1):

- מספר ה link:
נשאר זהה, כי delete-min זוהי הפעולה היחידה שמבצעת link -בסדרת הפעולות הנ"ל, והיא קדמה לשורה 3, השינויים התקיימו אחריה. כלומר - $\boldsymbol{m-1}$ **פעולות link**.
- מספר ה cut:
המפתח הנ"ל נמצא בעומק *logm* בעץ – ובכל פעם מפר את כלל הערימה עם ההורה שלו; לכן בנוסף ל *logm* החיתוכים שבצענו, נבצע עוד *logm-1* חיתוכים נוספים (כ"אורך המסלול" מהרמה התחתונה בעץ כלפי מעלה לשורש). בסה"כ - $\boldsymbol{2logm-1}$ **חיתוכים**.
- הפוטנציאל:
  - כל אחד מההורים הקדמונים שלו הם כל אחד מהצמתים שסימנו במהלך *logm* פעולות ה *decrease-key* קודם לכן; לכן, כל אחד מהם שהיה מסומן ונחתך מההורה שלו – הפך להיות שורש, וכבר לא יהיה מסומן. מכאן, שהצמתים היחידים שהיו מסומנים עד כה – יהפכו ללא מסומנים, ולא יתווספו צמתים מסומנים חדשים. בסך הכל 0 צמתים מסומנים.
  - מספר העצים שקיבלנו בסעיף ג' יגדל ב *logm – 1*, כלומר נקבל שכמות העצים בסה"כ:
$$logm + 1 + logm - 1 = 2logm$$
לכן, הפוטנציאל יהיה –
$$numOfTrees + 2 \times marks = 2logm + 0 = \boldsymbol{2logm}$$

לסיכום:

מספר פעולות link במהלך סדרת הפעולות: *m-1*

מספר פעולות cut במהלך סדרת הפעולות: 2logm-1

הפוטנציאל של המבנה בסוף הסדרה: כמספר העצים בערימה לאחר סיום סדרת הפעולות, 2logm.

לסיכום סעיפים ג-ו:

case	totalLinks	totalCuts	Potential	decreaseKey max cost
(c) original	m-1	logm	3logm - 1	(skip)
(d) decKey(m-2^i)	m-1	0	1	(skip)
(e) remove line #2	0	0	m+1	(skip)
(f) added line #4	m-1	2logm-1	2logm	logm-1

חלק ניסויי (תיאורטי) – שאלה 2

סעיף א'

m	Run-Time (ms)	totalLinks	totalCuts	Potential
728	2	723	0	6
6560	8	6555	0	6
59048	36	59040	0	9
531440	201	531431	0	10
4782968	1480	4782955	0	14

סעיף ב'

זמן הריצה האסימפטוטי של סדרת הפעולות כפונקציה של m הוא: $\boldsymbol{O(mlogm)}$

נסביר:

- תחילה, מבצעים m+1 פעולות **insert**, כל אחת בסיבוכיות זמן ריצה $O(1)$ – בסה"כ $O(m)$.
- לאחר מכן, מבצעים $\frac{3m}{4}$ פעולות **delete-min** סיבוכיות זמן הריצה:

עבור הפעולה הראשונה, נמחק את העץ בעל הצומת היחיד 0, סיבוכיות זמן הריצה:

$$O(log(m) + numOfTrees) = O(\log(\mathrm{m}+1) + \mathrm{m} + 1) = O(m)$$

מפני שכמות העצים לפני ביצוע הפעולה הייתה ככמות הצמתים שהכנסנו לערימה, והיא m+1,
ומדובר במקרה הגרוע ביותר – כל העצים בערימה היו צמתים יחידים.
לאחר פעולה זו – קיבלנו ערימה בינומית תקינה, כלומר לכל היותר O(logm) עצים בה.
נותר לבצע $\frac{3m}{4} - 1$ פעולות delete-min. כל אחת בעלות:

$$O(log(m) + numOfTrees) = O(logm)$$

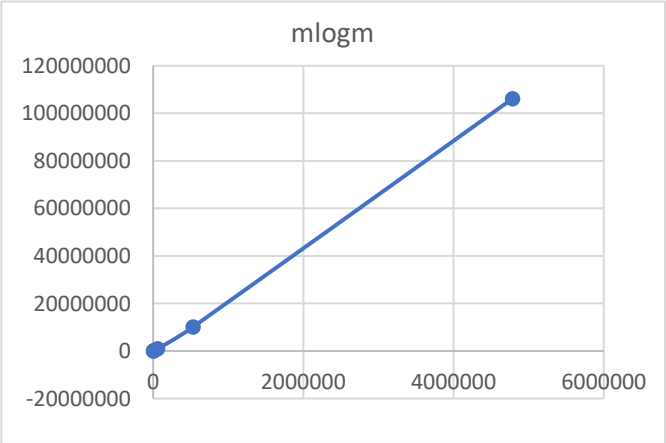
אבל, מפני שבכל מחיקת מינימום מס' הצמתים קטן ב 1, נקבל:

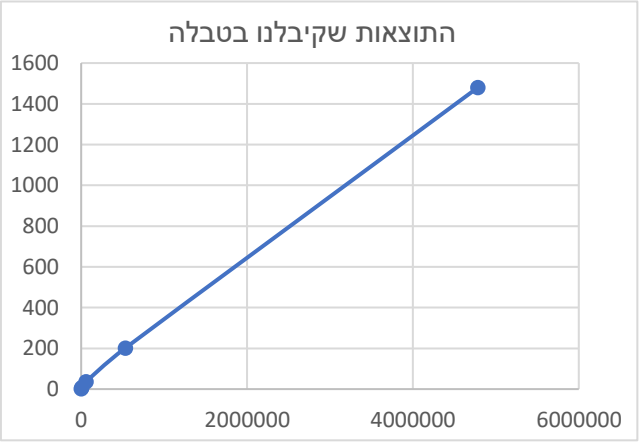
$$\log(m) + \log(m-1) + \cdots + \log\left(m - \frac{3m}{4}\right) = \log\left(m \times (m-1) \times \ldots \times \left(\frac{m}{4}\right)\right) \leq \log(m!) = mlogm$$

לכן, סיבוכיות זמן הריצה הכוללת היא:

$$O(m) + O(m) + O(mlogm) = O(mlogm)$$

אכן, כפי שניתן לראות בגרפים הסיבוכיות תואמת את התוצאות שקיבלנו בטבלה.





סעיף ג'

מספר פעולות ה link שבוצעו במהלך סדרת הפעולות:

בכל פעם שאנו מוחקים איבר מהערימה, אנו עוברים על כל השורשים ועבור שני עצים בעלי דרגה זהה נבצע link כך שנקבל ערימה "כמעט בינומית" שלא מכילה שני עצים בעלי אותה דרגה.

כאשר אנו מצבעים deleteMin בפעם הראשונה ישנם m עצים בערימה מדרגה 0. כל שני עצים נחבר לעץ מדרגה 1 - סה"כ m/2 פעולות link. כעת כל שני עצים מדרגה 1 נחבר לעץ מדרגה 2, לכן סה"כ m/4 פעולות link. נמשיך כך עד שנגיע לעץ אחד מדרגה log(m).

כלומר סה"כ:

$$\frac{m}{2}+\frac{m}{4}+\ldots+\frac{m}{2^{\log_2 m}}=\sum_{i=1}^{\log_2 m} m\cdot\frac{1}{2^i}=m\cdot\sum_{i=1}^{\log_2 m}\frac{1}{2}^i=m\cdot\frac{1}{2}\cdot\frac{1-\frac{1}{2}^{\log_2 m}}{\frac{1}{2}}=m-m\cdot\frac{1}{m}=O(m)$$

זה אכן תואם את התוצאות שקיבלנו בטבלה.

מספר פעולות ה cut שבוצעו במהלך סדרת הפעולות:

בתוכנית שלנו cut מתבצע רק על ידי קריאה מdecreaseKey. כלומר רק אם בוצע decreaseKey לצומת כלשהו בערימה, כך שלאחר השינוי המפתח של הצומת יהיה קטן מהמפתח של האבא שלו. במצב זה נרצה לחתוך את הצומת מהאבא שלו ולהוסיף אותו לרשימת השורשים כדי לשמור את מבנה תקין בעץ. נשים לב שבתוכנית הנוכחית לא ביצענו כלל את הפעולה decreaseKey, שכן רק הכנסנו והוצאנו איברים מהערימה ולכן לא בוצעו כלל פעולות cut.

כלומר עבור כל m בוצעו סה"כ בתוכנית **0 פעולות cut**.

זה אכן תואם את תוצאות הטבלה.

הפוטנציאל של המבנה בסוף הסדרה:

הפוטנציאל כפי שהגדרנו אותו בכיתה שווה לכמות העצים בערימה, פלוס 2 כפול כמות הצמתים המסומנים.

$$\Phi = numOfTrees + 2\cdot numOfMarkedNodes$$

$\underline{:numOfTrees}$

לאחר ביצוע ההכנסות לערימה היו בה m עצים מדרגה 0. נבצע $deleteMin$ עבור האיבר הראשון, ובמהלך המחיקה נבצע $link$ לעצים כפי שהסברנו קודם. בכל מחיקה נסיר את הצומת מהערימה ונאחד את העצים עד לקבלת עץ אחד לכל היותר מכל דרגה. כפי שהראנו בהרצאה, עבור ערימה עם n צמתים נקבל עץ עם לכל היותר $log(n)$ עצים.

לכן, בכל שלב כמות העצים הינה לכל היותר $O(log(n))$ כאשר n – כמות הצמתים הנוכחית בערימה. לפני המחיקה הראשונה ישנם m צמתים בערימה. לאחר כל מחיקה כמות הצמתים יורדת ב-1 עד שאנו מוחקים $\frac{3m}{4}$ איברים ונשארים עם $\frac{m}{4}$ איברים בערימה. לכן כמות העצים לאחר כל המחיקות היא:

$$O(log\ \frac{m}{4})$$

$\underline{:numOfMarkedNodes}$

לאחר ההכנסות לערימה, כל הצמתים בערימה הם שורשים ולכן אין אף צומת מסומן שכן שורש אינו מסומן לעולם. במהלך התוכנית אנו לא מבצעים כלל פעולות חיתוך כפי שהסברנו קודם ולכן בפרט אף צומת לא הופך במהלך התוכנית למסומן. לכן כמות הצמתים המסומנים בסוף התוכנית היא 0.

לכן סה"כ הפוטנציאל בסוף הריצה הוא:

$$\Phi = O(log\ \frac{m}{4}) + 2 \cdot 0 = O(log\ \frac{m}{4})$$

אכן, ניתן לראות שהסיבוכיות תואמת את התוצאות שקיבלנו בטבלה: