מבני נתונים מטלה מעשית 2

[redacted]

שאלה 1:

סעיף א':

חישוב זמן הריצה האסימפטוטי: תחילה מבוצעות m הכנסות לערימה, כל הכנסה מתבצעת בסיבוכיות של O(1) כפי שראינו בפירוט הפונקציות.
לאחר מכן, מתבצע מחיקת האיבר המינימלי. כיוון שבפונקציית deleteMin() מתבצע מעבר על כל השורשים, כלומר m שורשים, וחיבורם. לכן הסיבוכיות של חלק זה היא גם כן O(m).
לבסוף, מתבצע decreaseKey, כאשר על פי הניתוח שלנו כל פעולה מתבצעת בזמן amortized של O(1) או ב-worst case ב- O(logm). בכל מקרה סיבוכיות של חלק זה קטנה ממש מ-O(m).
לכן זמן הריצה האסימפטוטי צריך להיות: O(m+m+logm) = O(m).

סעיף ב':

m	Run-Time(ms)	totalLinks	totalCuts	Potential
2^{10}	6	1023	10	29
2^{15}	10	32767	15	44
2^{20}	44	1048575	20	59
2^{25}	7936	33554431	25	74

סעיף ג':

1. **מספר פעולות link: m-1.**
הסבר: כפי שתואר בסעיף א', בשלב של מחיקת המינימום מתבצע מעבר על הצמתים וצירופם לעצים בינומים. תחילה, כל שני צמתים יהפכו לעץ מדרגה 1 ואז כל העצים מדרגה 1 (יהיו $\frac{m}{2}$ עצים כאלו) יתחברו לעצים מדרגה 2 וכך הלאה עד שנקבל עץ יחיד מדרגה log(m). לכן מספר החיבורים הוא:

$$total\ number\ links\ =\ \sum_{i=1}^{\log_2 m}\frac{m}{2^i}\ = m\ \sum_{i=1}^{\log_2 m}\frac{1}{2^i} = m\frac{\frac{1}{2}*\left(\frac{1}{2}^{\log(m)}-1\right)}{\frac{1}{2}-1} = m\frac{\frac{1}{2}*\left(\frac{1}{m}-1\right)}{-\frac{1}{2}} = -m\left(\frac{1}{m}-1\right) = m-1$$

2. **מספר פעולות cut: log(m).**
הסבר: נבצע log(m) פעמים הפחתת מפתח ל- log(m) צמתים שונים. נשים לב כי בכל פעם שנפחית מפתח נצטרך לבצע cut כי המפתח שלו יהפוך להיות קטן מהמפתח של אביו, שכן במצב התחלתי העץ הבינומי מסודר ככה שהצמתים עליהם נבצע הפחתת מפתח הם עלים של אבות שונים. זאת משום שהעץ הבינומי לאחר מחיקת המינימום מסודר כך שכל העלים הם

מפתחות אי- זוגיים ואנו מפחיתים רק עלים אי-זוגיים. בנוסף בכל פעם בלולאה בשלב 3, בכל פעם שניגש למפתח כדי למחוק אותו נהיה תחת אב (הכוונה היא לאב ישיר) אחר בוודאות. האב יהיה שונה בוודאות כי לכל שני עלים אי זוגיים (v1-הצומת האי-זוגית הקטנה יותר, v2- הצומת האי זוגית הקטנה היותר), קיים לפחות צומת זוגית אחת שהמפתח שלה קטן מהמפתח של v2 וגדול מהמפתח של v1, שנמצאת בעץ. צומת זו צריכה להיות האב של v2 (זה נכון כי בעת יצירת העץ הבינומי ב ()deleteMin ניקח תחילה כל שני צמתים עוקבים:
(m-2, m-1),…, (2,3) , (0,1) ונחבר אותם לעץ בדרגה 1 כאשר הצומת הזוגית היא הקטנה יותר ולכן היא האב. לכן לכל עלה אי-זוגי אב זוגי ומפתחו של האב הוא 1+המפתח של העלה) ולא יכולה להיות האב של v1 כי היא גדולה ממנו. לכן לא נבצע שני חיתוכים מאותו האב ועל כן סך הכל יתבצעו מספר חיתוכים כמספר הפחתות המפתחות- log(m) חיתוכים.

3. **הפוטנציאל: 3log(m)-1.**

הסבר: הפוטנציאל מחושב על ידי: $potential = numOfTrees + 2 * numOfMarked$
כפי שתואר בג'.2 מספר העצים יהיה log(m)+1. בנוסף, כיוון שבוצעו log(m) חיתוכים צריכים להיות log(m) צמתים מסומנים, אך כיוון שהשורש לעולם אינו מסומן (תמיד נבצע חיתוך של הצומת שהמפתח המקורי שלה היה 1 וצומת זו היא בן אחד ישיר של השורש של העץ הבינומי המקורי). לכן יהיו $log(m) - 1$ צמתים מסומנים. כלומר חישוב הפוטנציאל הוא:
$$potential = \log(m) + 1 + 2(\log(m) - 1) = 3\log(m) - 1$$

<u>סעיף ד':</u>

1. **מספר פעולות link: $\boldsymbol{m-1}$.**

הסבר: בדומה להסבר בסעיף ג'.1.

2. **מספר פעולות cut: 0**

הסבר: הפעם נבצע decreaseKey על מפתחות שונים. בפעם הראשונה נפחית את מפתח 0 להיות (m+1)–, הוא יישאר המפתח הקטן בערימה ולכן יישאר השורש – אין צורך בחיתוכים.
לאחר מכן נעבור למפתח $m-2^{\log(m)-1}$ שהוא הבן של השורש ונפחית ממנו (m+1)– ונקבל מפתח בעל ערך: $m - 2^{logm-1} - (m+1) = m - \frac{1}{2}m - m - 1 = -\left(\frac{1}{2}m + 1\right)$.
מפתח זה יהיה עדיין גדול מהשורש (השורש הוא (m+1)–). כך נבצע בדומה log(m) פעמים, בכל פעם ניגש לבן של אב שכבר הוקטן, לכן לא יופר כלל הערמה, ולכן לא יהיה צורך בחיתוכים.
הסבר למה ניגש תמיד לבן של אב שכבר הקטנו: כפי שנאמר נתחיל ב-0, לאחר מכן ניגש לצמתים: $\frac{1}{2}m, \frac{3}{4}m, \ldots, m-2$. בשלב יצירת העץ (ב ()deleteMin) הצומת $\frac{1}{2}m$ תהיה בתת עץ בינומי שונה מהשורש לפני החיבור האחרון, זאת על פי סדר הפעולות של הפונקציה toBuckets, לכן בחיבור האחרון הצומת $\frac{1}{2}m$ תהפוך לבן הישיר של הצומת 0. כך באופן רקורסיבי עבור כל הצמתים אותם נפחית בסעיף זה.

3. **הפוטנציאל: 1.**

הסבר: כפי שהסובר בסעיף ד'.2 לא יבוצעו חיתוכים, המשמעות היא שיהיה עץ יחיד בסוף הפעולות ולכן הפוטנציאל יהיה שווה פשוט ל-1:
$$potential = numOfTrees + 2 * numOfMarked = 1 + 2 * 0 = 1$$

<u>סעיף ה':</u>

1. **מספר פעולות link: 0.**

הסבר: linking מתבצע אך ורק כאשר יש קריאה לפונקציה ()deleteMin, לכן כיוון שאין קריאה לפונקציה זו, לא מתבצעים lonking.

2. **מספר פעולות cut: 0**

הסבר: כיוון שיש לנו m+1 עצים בדרגה 0 בערימה, בכל פעם שנבצע decreaseKey על

איזשהי צומת, לא נצטרך לעשות cut כי צומת זו היא למעשה כבר שורש ללא בנים ולכן תנאי הערימה יתקיים ללא תלות בגודל הצומת.

3. **הפוטנציאל: m+1.**
הסבר: יהיו m+1 עצים ו-0 חיתוכים לכן חישוב הפוטנציאל:
$potential = numOfTrees + 2 * numOfMarked = m + 1 + 2 * 0 = m + 1$

סעיף ו':

1. **מספר פעולות link: m-1.**
הסבר: בדומה לסעיף ג'.1.
2. **מספר פעולות cut: 2log(m)-1.**
הסבר: תחילה, בדומה לסעיף ג'.2 יתבצעו log(m) חיתוכים.
לאחר מכן, נבצע decreaseKey על הצומת m-2.
כעת יש לשים לב: הצומת m-2 וכל האבות הקדומים שלה (חוץ מהשורש) יהיו מסומנים בשלב זה. זאת משום שנבצע חיתוכים לאבות של עלים אי-זוגיים, האבות של עלים אלו הם הצמתים: $\frac{1}{2}m, \frac{1}{4}m, \dots, \frac{1}{2(logm)}$ משום שכפי שהמפתח של אב של עלה אי-זוגי יהיה המפתח של העלה פחות 1. זה נכון כי בעת יצירת העץ הבינומי ב deleteMin() ניקח תחילה כל שני צמתים עוקבים: (m-2, m-1),...,(2,3) , (0,1) ונחבר אותם לעץ בדרגה 1 כאשר הצומת הזוגית היא הקטנה יותר ולכן היא האב. לכן לכל עלה אי-זוגי אב זוגי ומפתחו הוא 1+המפתח של העלה.
כפי שהוסבר בסעיף ד'.2 הצמתים $0, \frac{1}{2}m, \frac{3}{4}m, \dots, m-2$ מסודרים כך שהצומת 0 אב של $\frac{1}{2}m$ שהיא האב של $\frac{3}{4}m$ וכך הלאה.
כלומר על המסלול מ-m-2 עד לשורש (לא כולל השורש) כל הצמתים יהיו מסומנים. לכן בזמן decreaseKey על m-2 נצטרך לבצע עוד log(m)-1 חיתוכים (חיתוך על כל צומת מסומנת).
סך הכל התבצעו $\log(m) + \log(m) - 1 = 2\log(m) - 1$ חיתוכים.
במקרה זה העלות היקרה ביותר של decreaseKey היא העלות של הפעולה האחרונה על צומת m-2, כפי שראינו עלות של פעולה זו היא $\log(m) - 1$ (כי העלות נקבעת לפי מספר החיתוכים).
3. **הפוטנציאל: 2log(m).**
הסבר: כפי שראינו בסעיף ו'.2 התבצעו $2\log(m) - 1$ חיתוכים ולכן בסוף התהליך מספר העצים יהיה: $2\log(m) - 1 + 1 = 2\log(m)$. כיוון שעל פי סעיף ו'.2 מתבצע חיתוך על כל הצמתים המסומנים, לא יהיו צמתים מסומנים בסוף התהליך.
לכן חישוב הפוטנציאל:
$potential = numOfTrees + 2 * numOfMarked = 2\log(m) + 2 * 0 = 2\log(m)$

טבלת סיכום:

Case	toatlLinks	TotalCuts	Potential	decreaseKey max cost
(c) original	m-1	log(m)	3log(m)-1	-
(d) decKey(m-2^i)	m-1	0	1	-
(e) remove line #2	0	0	m+1	-
(f) added line #4	m-1	2log(m) -1	2log(m)	log(m) -1

שאלה 2:

סעיף א':

i	m	Run-Time(ms)	totalLinks	totalCuts	Potential
6	**728**	14	723	0	6
8	**6560**	16	6555	0	6
10	**59048**	37	59040	0	9
12	**531440**	206	531431	0	10
14	**4782968**	2196	4782955	0	14

תחילה ניתן תיאור לתוכנית שתפעיל את הפעולות בשאלה זו:
תחילה נכניס לערימה m+1 איברים: (0,1,....,m), כיוון שנתחיל בהכנסת 0 ואז בכל פעם נכניס איבר שגדול ב-1, נקבל למעשה ערימה שאיבר הראשון בה הוא עץ בגדול אחד עם האיבר m שמצביע לעץ בגודל אחד עם איבר m-1 וכך הלאה עד לאפס. בעת מחיקת המינימום בפעם הראשונה, כלומר מחיקת 0 יתבצעו לינקים לאיחוד העצים לעצים בינומים גדולים יותר. כיוון שיישארו m איברים יהיו לנו log(m) עצים. על פי מה שנלמד בהרצאה דרגת העצים תיקבע באופן יחיד על ידי הייצוג הבינארי של m.

נשים לב לעובדה חשובה בשלב זה: אם יש עץ בדרגת i ועץ בדרגת j כך ש: j>i אז איברי העץ j גדולים מאיברי עץ i. זאת משום שבעתם החיבור נתחיל באיבר הגדול ביותר ונחבר אותו לאיברים הקטנים ממנו. בדרך זו העץ j יבנה לפני שיבנה העץ i ולכן בעץ j יהיו איברים גדולים יותר.

לאחר מכן מתבצעות עוד $\frac{3}{4}m - 1$ מחיקות של מינימום. לפי תיאור בניין הערימה לאחר מחיקת המינימום הראשון ניתן לראות כי קודם ימחקו איברי העץ הקטן ביותר ורק אז ימחקו איברי העץ עם הדרגה מעליו, כך עד שנגיע לעץ בעל הדרגה הגדולה ביותר.

ניזכר כעת כי לשורש של עץ בדרגה k מחוברים k-1 עצים, הראשון בדרגה הk-1 המחובר לאח בדרגת k-2 כך עד לעץ בדרגת 1. בנוסף נשים לב כי גם עבור תתי עצים נכון להגיד שאיברי תת עץ בדרגה נמוכה יותר יהיו קטנים יותר (זה נכון כי הבנייה רקורסיבית).
לכן כאשר נתחיל ונמחק את המינימום השני, הוא יהיה בשורש של העץ הקטן ביותר בערימה ואנו נקבל k-1 עצים חדשים לערימה. כיוון שk היה העץ בעל הדרגה הכי קטנה לפני המחיקה אז לא יהיו לנו לינקים חדשים לעשות, כי אין עוד תתי עצים בדרגות זהות לחיבור בערימה.
באופן זהה נמשיך למחוק את המינימום, כאשר בכל פעם נקבל תתי עצים חדשים שאין לנו דרך לבצע עליהם לינק כי אין עצים בדרגתם בערימה.
על פי ההסבר שניתן לעיל נבצע לינקים אך ורק במחיקת המינימום הראשון.

סעיף ב':

על פי התיאור שניתן לפני סעיף זה, תחילה נבצע m+1 פעולות הכנסה בסיבוכיות O(1) לכל הכנסה, לכן סה"כ O(m).
לאחר מכן נבצע לינקים (חיבור איברים לעצים), חסם עליון על מספר הלינקים הוא m על פי WC סיבוכיות של deleteMin().
לאחר מכן נבצע עוד $\frac{3}{4}m-1$ מחיקות מינימום כאשר לא נבצע לינקים באף אחת מהן ולכן הסיבוכיות של כל אחת חסומה על ידי $O(logm)$ זאת כי בכל מקרה, גם אם לא בוצעו לינקים נצטרך לעבור על כל השורשים. נחסום את מספר השורשים על ידי $O(logm)$ כי יש עד m איברים ומספר העצים של m איברים נקבע על ידי הייצוג הבינארי (על פי מה שהראנו העצים הם עצים בינומים), כיוון שיש עד $logm+1$ ביטים בייצוג אז חסם עליון על מספר העצי הוא $logm+1$. לכן סיבוכיות חלק זה היא: $O(mlogm)$.
סך הכל זמן הריצה האסימפטוטי הוא $O(m+m+mlogm)=O(mlogm)$.

סעיף ג':

1. **מספר פעולות cut:**
פעולות cut יכולות להתבצע אך ורק בעת ביצוע מחיקה של איבר שהוא לא מינימלי או הפחתת ערך מפתח. כיוון שפעולות אלו לא נעשות בסדרת הפעולות של שאלה זו אז $totalCut=0$.
2. **מספר פעולות Links:**
לפי התיאור שלעיל מתבצעות פעולות link רק במחיקת המינימום הראשון ולכן נטען כי מספר ה-link בסדרת הפעולות מוגדרת באופן יחיד על ידי הייצוג הבינארי.
נייצג את m בבינארי ונכתוב לעצמנו את דרגות הביטים הדלוקים.
נטען כי: $totalLinks=(2^i-1)+(2^j-1)+\cdots$, כאשר כל מחובר מייצג לנו עץ בינארי אחד בערימה לאחר המחיקה הראשונה, והחזקה של 2 היא הדרגה של הביט הדולק שמייצג את דרגת העץ.
מספר זה מתאר באופן תיאורטי את מספר הלינקים כי עבור עץ בדרגה k (הביט ה-k דולק) כלשהי יש 2^k איברים וביניהם 2^k-1 'קשתות' – כלומר 2^k-1 לינקים.
3. **הפוטנציאל:**
נטען כי: $Potential=m-totalLinks+1$.
כיוון שאין צמתים מסומנים אז הפוטנציאל שווה למספר העצים בערימה בסוף התהליך.
כיוון שיש $\frac{1}{4}m$ איברים (לאחר מחיקת $\frac{3}{4}m$ איברים) אז מספר העצים הוא מספר הביטים הדולקים במספר $\frac{1}{4}m$ (כי כל העצים נשארים בינומים). כעת נשים לב כי $totalLinks=m-numOfBits$, כלומר מספר הלינקים שווה ל-m פחות מספר הביטים הדולקים בייצוג הבינארי, כי עבור כל ביט דולק נחבר את 2 בחזקת הדרגה ונפחית באחד. חיבור כל הביטים (2 בחזקת הדרגה) הוא m וחיסור של 1 כמספר הביטים הדולקים יוביל אותנו ל: $m-numOfBits$.
כיוון שבמספר $\frac{1}{4}m$ יש ביט דולק אחד פחות מב-m (m תמיד מתחלק ב-4) אז נקבל כי מספר הביטים הדולקים ב $\frac{1}{4}m$ הוא בדיוק:
$$Potential=m-totalLinks+1$$

פירוט הפונקציות במחלקה:

`public FibonacciHeap()`

בנאי לערימת פיבונאצ׳י ריקה. סיבוכיות : O(1).

`Public boolean isEmpty()`

מחזירה האם הערימה ריקה בכך שבודקת שהmin שווה לnull. סיבוכיות : O(1).

`Public HeapNode insert(int key)`

יוצרת צומת חדש עם המפתח key ואז פועלת לפי המקרים הבאים:
-אם הערימה ריקה, הצומת החדש יהיה המינימום והראשון וגם יצביע לעצמו בצורה מעגלית.
- אחרת, תדאג שהצומת החדש ייכנס מצד שמאל בכך שתהפוך אותו לראשון ותסדר את המצביעים כך שהראשון לפני ההכנסה יהיה העוקב של הצומת החדש, והאחרון (הprev של הראשון לפני ההכנסה) יהיה הקודם של הצומת החדש.
תדאג לעדכן את המינימום אם צריך (אם key קטן מהמינימום לפני ההכנסה) ולהגדיל את מספר העצים ואת גודל העץ ב1. סיבוכיות : הכנסה עצלה (שינוי מצביעים קבועים) - O(1) .

`public void deleteMin()`

מוחקת את הצומת אליו מצביע min ואז מבצעת איחודים אם נדרש, פועלת בצורה הבאה:
בודקת תחילה מקרי קצה – אם הערימה ריקה, פשוט לא תבצע כלום ואם הערימה מורכבת מצומת אחד בלבד אז תהפוך אותה לערימה ריקה ותעצור ללא ביצוע פעולות נוספות.
כעת תבדוק האם לצומת המינימלי יש ילדים:
- אם כן והצומת המינימלי הוא השורש היחיד אז היא תהפוך את הילד שלו להיות הfirst החדש.
- אם כן והצומת המינימלי אינו השורש היחיד אז היא תהפוך את הילדים שלו להיות אחים של אחים שלו (שורשים בעצמם) בכך שתגרום להם להצביע אחד על השני ולהפסיק להצביע לצומת המינימלי. בנוסף נפחית את מספר העצים ב1 (השורש המינימלי) ונגדיל אותו במספר הילדים שהיו לשורש המינימלי.
- אם לא והגענו למקרה זה אז בהכרח הצומת המינימלי אינו השורש היחיד ולכן נדאג לכך שאחיו הסמוכים לא יצביעו עליו, אלא זה על זה ובנוסף נוריד את מספר העצים ב1.
נקטין את גודל העץ ב1 ואז נקרא לפונקציה toBuckets אשר תבצע איחודים של עצים מאותה דרגה.
סיבוכיות : לפני הקריאה לtoBuckets , הפונקציה מבצעת בדיקת תנאים בודדים, שינוי מצביעים ותיחזוק שדות ולכן החלק הזה עולה O(1) ולכן סיבוכיות הפונקציה תהיה כשל הפונקציה toBuckets, O(log(size)) באמורטייזד ו O(size) במקרה הגרוע.

`private void toBuckets()`

יוצרת מערך דליים כך שהאינדקס i במערך הינו מקום לעץ עם rank = i.
עוברת בלולאה על כל העצים משמאל לימין (מתחילה מfirst) כך ששומרת מצביע לשורש עליו עוברת בשם y ובתחילת כל איטרציה גם שומרת מצביע לשורש הבא בשם copyOfyNext.
כעת היא מכניסה את העצים למערך, במקום הrank של השורש שלהם, כך שאם כבר המקום במערך תפוס אז היא מחברת בין העצים בעזרת הפונקציה linkTrees(x1,x2,x1.rank) כאשר העצים המועברים הם העץ שישב במערך והעץ שעברנו עליו בלולאה. כך היא תמשיך עד אשר המקום במערך בדרגה של העץ שעברנו עליו (אולי לאחר חיבורים), יהיה פנוי ואז היא תכניסו לשם. בסוף כל איטרציה היא תחליף את y להיות הצומת ששמרנו בcopyOfyNext .
כעת היא תכניס את העצים למערך חדש בשם finalTrees שבו לא יהיו תאים ריקים והוא יהווה את רשימת העצים הסופית לאחר כל האיחודים האפשריים.
במעבר על העצים הסופיים, היא תחבר ביניהם (כאחים) כך שהסדר בסוף יהיה לפי דרגות העצים כך שעץ הקטן ביותר יישב במקום השמאלי ביותר, ובנוסף תמצא את הmin החדש ותהפוך את השורשים ללא מסומנים (תתחזק גם את השדה numOfMark).
סיבוכיות : הפונקציה עוברת בלולאה על כמות העצים בערימה ומכניסה אותם למערך, כך שבמקרה הגרוע, יהיו size עצים מדרגה 0 תחילה, כל שני צמתים (עצים מדרגה 0) יהפכו לעץ מדרגה 1 ואז כל העצים מדרגה 1 (יהיו $\frac{size}{2}$ עצים כאלו) יתחברו לעצים מדרגה 2 וכך הלאה עד שנקבל עץ יחיד מדרגה log(size). לכן מספר החיבורים הוא:

$$total\ number\ links\ =\ \sum_{i=1}^{\log_2 n}\frac{\mathrm{n}}{2^i}\ = n*\ \sum_{i=1}^{\log_2 \mathrm{n}}\frac{1}{2^i} = \mathrm{n}*\frac{\frac{1}{2}*\left(\frac{1}{2}^{\log(n)}-1\right)}{\frac{1}{2}-1} = n*\frac{\frac{1}{2}*\left(\frac{1}{n}-1\right)}{-\frac{1}{2}} = -n\left(\frac{1}{n}-1\right) = n-1 = O(n)$$

כאשר n = size.

לאחר מכן הפונקציה עוברת בלולאה על מספר העצים הסופיים (יש לכל היותר אחד מכל דרגה – logn): מוצאת מי המינימלי החדש, הופכת ללא מסומנים ומעדכנת להם בזמן O(1) את המצביעים. לכן סה״כ הסיבוכיות היא O(size +log(size)) = O(size) במקרה הגרוע, וסיבוכיות O(log(size)) באמורטייז.

`private HeapNode linkTrees(HeapNode x1, HeapNode x2, int rank)`

הפונקציה מקבלת 2 עצים מדרגה זהה ובודקת מי מהשורשים קטן יותר (יהיה השורש של העץ לאחר החיבור, מכלל הערימה) ושולחת לlinkTreesHelper כך שהארגומנט הראשון שתעביר יהיה העץ ששורשו קטן יותר. סיבוכיות : כסיבוכיות הפונקציה, linkTreesHelper, O(1).

`private HeapNode linkTreesHelper(HeapNode x1, HeapNode x2, int rank)`

מחברת בין שני עצים x1,x2 כך שx1 יהיה השורש החדש וx2 ייתלה עליו בכך שיהפוך לילד של x1 וגם לאח של ילדיו של x1 בעזרת טיפול במצביעים prev, next. מטפלת במקרי קצה בהם הדרגה הינה 0 או 1 ולכן נדרש טיפול אחר במצביעים prev, next של השורשים.

הפונקציה גם דואגת להוריד את מספר העצים ב1 (לפני החיבור לעומת אחרי החיבור) ולהעלות את מספר החיבורים ב1.

סיבוכיות : O(1) מכיוון שבוצעו רק פעולות בודדות וקבועות (בדיקת תנאים ושינוי מצביעים).

`public HeapNode findMin()`

מחזירה את הערך השמור בשדה min. סיבוכיות : O(1).

`public void meld (FibonacciHeap heap2)`

אם הערימה המועברת כארגומנט ריקה, אז לא נעשה כלום.

תדאג לסכום את מספר העצים ומספר העצים המסומנים של 2 הערימות ואת מספר הצמתים גם כן.

בודקת מי מאיברי המינימום של 2 הערימות קטן יותר – הוא יהיה איבר המינימום החדש.

לאחר מכן פשוט מחברת בין הערימות על ידי חיבור של האחרונים והראשונים בין הערימות (בצורה מעגלית כך שערימה 2 תשב מצד ימין לערימה שלנו).

סיבוכיות :O(1) מכיוון שבוצעו רק פעולות בודדות וקבועות (בדיקת תנאים, חיבור שדות ושינוי מצביעים).

`public int size()`

פונקציה שמחזירה את מספר האיברים (מפתחות) על ידי קריאה למצביע size. סיבוכיות O(1).

`public int[] countersRep()`

אם הערימה ריקה, תחזיר מערך ריק.

אחרת, מחזירה מערך מונים של מספר העצים בעלי דרגה מסוימת, כך שבאינדקס i יש את מספר העצים עם דרגה i. כיוון שהדרגה הכי גדולה שיכולה להיות לעץ בערימה הוא log על בסיס 2 של מספר הצמתים בעץ נייצר מערך באורך log(size).

הפונקציה עוברת על כל השורשים בערימה, כלומר numOfTree. כיוון שבערימת פיבונאצי אם יש מספר צמתים ששווה ל- size, יכולים להיות size עצים בדרגה 0 אז החסם על סיבוכיות הפונקציה הוא: O(size).

`public void delete(HeapNode x)`

מפחיתה את המפתח של הצומת x באינסוף ע״י קריאה לפונקציה decreaseKey(x,∞), בכך המפתח הופך להיות המינימלי בערימה ואז היא מוחקת אותו בעזרת קריאה לdeleteMin.

סיבוכיות : חיבור הסיבוכיות של decreaseKey O(1) ושל deleteMin O(logn) ולכן סה״כ O(logn) כאשר n הוא מספר הצמתים בערימה.

`public void meldAndCut(HeapNode node)`

פונקציה זו מבצעת חיתוך של תת עץ ומוסיפה אותו לתחילת רשימת השורשים. כיוון שפונקציה זו מבצעת מספר קבוע של פעולות סטנדרטיות שעולות O(1), אז ניתן להסיק כי סיבוכיות הפונקציה היא O(1).

```
public void decreaseKey(HeapNode x, int delta)
```

פונקציה זו מקטינה את המפתח של צומת x הניתנת לה. אם X הוא שורש אין הערימה נשארת תקינה ואין מה לתקן, כמו כן, אם לאחר הפחתת המפתח של x, x עדיין גדול מאביו אז אין מה לתקן בערימה.
אך אם לאחר ההפחתה המפתח דל x קטן מאביו נבצע cut ו- meld (על ידי קריאה לפונקציה meldAndCut אשר מתבצעת בסיבוכיות O(1). נבצע זאת עד שנגיע לצומת הקדומה ל-x ואינה מסומנת, כאשר נפגוש צומת כזו פשוט נסמנה ונעצור את הלולאה.
בכל מקרה נבדוק האם x הפך להיות קטן מהמינימום בערימה, אם כן נשנה את המצביע למינימום.
ראינו בהרצאה שזמן ה- amortized של פונקציה זו הוא O(1), העלות האמתית היא מספר ה-cuts שנעשו אשר במקרה הגרוע ביותר הוא O(log(size)) כיוון שזה הגובה הגדול ביותר האפשרי לעץ בערימה, אם כל הצמתים לאורך העץ היו מסומני אז נבצע O(log(size)) פעולות cut ו meld .

```
public int potential()
```

מחזירה את הפוטנציאל של הערימה, כלומר את מספר העצים ועוד פעמיים מספר הצמתים המסומנים.
כיוון שיש מצביעים ששומרים ומתחזקים את מספר העצים ואת מספר הצמתים המסומנים לפונקציה זו סיבוכיות O(1).

```
public static int totalLinks()
```

מחזירה את הערך השמור בשדה הסטטי totalLinks. סיבוכיות : O(1).

```
public static int totalCuts()
```

מחזירה את הערך השמור בשדה הסטטי numOfCut. סיבוכיות : O(1).

```
public static int[] kMin(FibonacciHeap H, int k)
```

פונקציה שמקבלת ערימה שהיא עץ בינומי יחיד ומחזירה מערך ובו k האיברים הקטנים ביותר בה.
נבנה מערך בגודל k בו נאכסן את האיברים המינימליים שמצאנו.
נתחיל במינימום בעץ ונכניס אותו למערך במקום הראשון. לאחר מכן נכניס אותו ואת כל ילדיו לערימה חדשה. נשתמש במצביע pointer שנמצא בכל צומת כדי לאכסן מצביע למיקום של כל צומת בערימה המקורית. נמחוק את המינימום מהערימה הזמנית ונחפש מינימום חדש בה. כיוון שהעוקב של המינימום בערימה המקורית חייב להיות ילד שלו, בוודאות נמצא אותו בערימה הזמנית.
נכניס את העוקב למערך ונעבור לילדיו בעזרת המצביע pointer (נכניס אותם למערך הזמני, נמחק את המינימום ושוב נחפש עוקב), אם אין לו ילדים נבצע חיפוש של העוקב לו בין הצמתים שנמצאים כבר ברשימה הזמנית. נחזור על הפעולה k פעמים, עד למציאת k האיברים הקטנים ביותר.

נשים לב כי בשלב ה-i של הלולאה בפונקציה הסיבוכיות של ההכנסה למערך הזמני היא עד O(deg(H)) כי הדרגה של כל צומת בפונקציה חסומה על ידי deg(H), כלומר לכל צומת עד deg(H) בנים, סיבוכיות המחיקה היא במקרה הגרוע ביותר $O(\log(kdeg(h))$ כי גודל המערך הזמני הוא קטן-שווה ל:$kdeg(h)$. נשים לב כי: $deg(h) = \log size(H)$ וכי: $\log size(H) > \log k \rightarrow size(H) > k$.
לכן סיבוכיות פעולת המחיקה היא: $O(\log\left(kdeg(h)\right) = O(\log(k) + \log(degH)) \leq$
$O(\log size + \log degH) = O(degH + \log degH) = O(degH)$
לכן בכל שלב בלולאה הסיבוכיות חסומה על ידי O(deg(H)) ולכן בסך הכל, סיבוכיות הפונקציה היא O(k*deg(H)).

```
public HeapNode getFirst()
```

מחזירה את הערך השמור בשדה first. סיבוכיות : O(1).