סטודנט 1: עלי שוואהנה 326683885

סטודנט 2: מוחמד מנסור 324293331

# **החלק הניסויי + מסמך התיעודים**

## **החלק הניסויי**

### **שאלה 1**

ניסוי ראשון

| מספר סידורי i | זמן ריצה (מילישניות) | גודל הערמה בסיום | מספר חיבורים | מספר חיתוכים | מספר עצים בסיום |
|---|---|---|---|---|---|
| 1 | 0.811435 | 6559 | 6550 | 0 | 9 |
| 2 | 1.740834 | 19681 | 19674 | 0 | 7 |
| 3 | 4.103343 | 59047 | 59037 | 0 | 10 |
| 4 | 9.633056 | 177145 | 177133 | 0 | 12 |
| 5 | 38.64512 | 531439 | 531427 | 0 | 12 |

ניסוי שני

| מספר סידורי i | זמן ריצה (מילישניות) | גודל הערמה בסיום | מספר חיבורים | מספר חיתוכים | מספר עצים בסיום |
|---|---|---|---|---|---|
| 1 | 2.952708 | 3280 | 40999.7 | 37724.7 | 5 |
| 2 | 6.391047 | 9841 | 138702 | 128868 | 7 |
| 3 | 20.35383 | 29524 | 461496.9 | 431980.9 | 8 |
| 4 | 63.54728 | 88573 | 1530114.1 | 1441553.1 | 12 |
| 5 | 237.9027 | 265720 | 4989522.4 | 4723811.4 | 9 |

ניסוי שלישי

| מספר סידורי i | זמן ריצה (מילישניות) | גודל הערמה בסיום | מספר חיבורים | מספר חיתוכים | מספר עצים בסיום |
|---|---|---|---|---|---|
| 1 | 1.4481354 | 31 | 6550 | 6549.4 | 30.4 |
| 2 | 3.2735354 | 31 | 19674 | 19672.9 | 29.9 |
| 3 | 8.2388436 | 31 | 59037 | 59035.6 | 29.6 |
| 4 | 21.541833 | 31 | 177133 | 177132 | 30 |
| 5 | 92.290835 | 31 | 531427 | 531425.8 | 29.8 |

## **שאלה 2**

**ניסוי 1 - ניתוח זמן ריצה:** ביצוע $n$ ההכנסות עולה $O(n)$ זמן שהרי כל הכנסה עולה $O(1)$, ומחיקת המינימום תגרום לביצוע *consolidating* על $n$ צמתים בודדים, אשר יבצע $O(n)$ חיבורים, שהרי בהתחלה מספר תתי העצים הוא $n$ וכל חיבור מקטין את מספר תתי העצים ב- 1. ונצטרך גם לחפש את המינימום החדש בעלות $O(n)$ (זה הוא המקרה הגרוע של ה-*delete min*). ולכן מחיקת המינימום תעלה $O(n)$. ולכן סך עלות הניסוי היא $O(n)$.

**ניסוי 2 - ניתוח זמן ריצה:** ביצוע $n$ ההכנסות עולה $O(n)$ זמן שהרי כל הכנסה עולה $O(1)$. בנוסף, נבצע $\frac{n}{2}$ פעמים מחיקה למינימום. המחיקה הראשונה תעלה $O(n)$ כפי שהראינו בניתוח של הניסוי הראשון. לאחר מכן, בעץ יהיו $O(\log n)$ עצים (ראינו בשיעור) והדרגה של כל עץ היא לכל היותר $O(\log n)$ (ראינו בשיעור), ולכן מחיקת המינימום תגרום לביצוע consolidate על לכל היותר $O(2\log n) = O(\log n)$ תתי עצים, ונצטרך גם לחפש את המינימום בעלות $O(\log n)$. ולכן כל מחיקה כזאת תעלה $O(2\log n) = O(\log n)$. ולכן בסה"כ המחיקות יעלו $O\left(n + \left(\frac{n}{2} - 1\right)\log n\right) = O(n\log n)$, ולכן סך עלות הניסוי היא $O(n\log n)$.

**ניסוי 3 - ניתוח זמן ריצה:** ביצוע $n$ ההכנסות עולה $O(n)$ זמן שהרי כל הכנסה עולה $O(1)$, ומחיקת המינימום תעלה $O(n)$ כפי שהראינו בניתוח של הניסוי הראשון. אחרי ההכנסות ומחיקת המינימום, כל הצמתים בעץ אינם מסומנים. אנחנו מעוניינים למחוק $n-32$ פעמים את הצומת המקסימלי בעץ, כלומר $O(n)$ מחיקות לצומת המקסימלי. נשים לב כי לאורך כל תהליך המחיקות, יסומנו בעץ לכל היותר $O(n)$ צמתים, ולכן יתבצעו לכל היותר $O(n)$ ניתוקים (זו היא בפרט גם הכמות המקסימלית, אסמפטוטית, של צמתים שיכולים להפוך למסומנים), ולכן יחד עם הניתוקים של הצמתים שנמחק, יתבצעו לכל היותר $O(n)$ ניתוקים, שהרי בכדי שיתבצע ניתוק צריך שיהיה בעץ לפחות צומת אחד שהוא כבר מסומן, ולכן עלות הניתוקים לאורך כל סדרת המחיקות היא $O(n)$. ולכן, מכיוון ש- *delete* (על צומת שאינו מינימלי), מלבד ביצוע הניתוקים מבצעת בכל הרצה רק מספר סופי של פעולות שלוקחות זמן קבוע אזי סדרת פעולות ה- *delete* תעלה $O(n)$ ללא עלות הניתוקים, ולכן העלות הכוללת של המחיקות היא $O(2n) = O(n)$ . ולכן העלות הכוללת של הניסוי היא $O(n)$.

## <u>שאלה 3</u>

**הניסוי הראשון - סדר ההכנסה לא משפיע על המדידות**, שהרי כאשר מבצעים מחיקה לאיבר המינימלי, תתבצע סדרה של חיבורים שתניב ערימה בעלת אותו המבנה, עבור כל סדר הכנסה של המפתחות. ולכן לכל סדר הכנסה יתבצע אותו מספר של חיבורים ונקבל את אותה כמות של תתי עצים. מספר הניתוקים יהיה 0 תמיד שהרי לא התבצעו פעולות המצריכות ניתוקים (מחקנו איבר שאין לו ילדים ואין לו הורה). כמובן שגודל הערימה לא מושפע מסדר ההכנסה.

**הניסוי השני – סדר ההכנסה משפיע על מספר החיבורים ומספר הניתוקים.** ראשית מספר החיבורים יכול להשתנות. למשל אחרי מחיקת המינימום בפעם הראשונה, נוצרת כמות מסוימת של תתי עצים, אך כאשר מוחקים את המינימום הבא, זה אכן משנה באיזה תת עץ הוא נמצא, שהרי זה יקבע את מספר החיבורים שיתבצעו ב- consolidate שתתבצע אחרי המחיקה השנייה, וכך הלאה. סדר ההכנסה משפיע גם על מספר הניתוקים שהרי בכל מחיקה זה אכן משנה באיזה תת עץ נמצא המינימום שנרצה למחוק, כיוון שמספר הבנים שלו יכול להיות שונה. מספר העצים בסיום התהליך אינו מושפע מסדר ההכנסה, וכך גם גודל הערמה.

**הניסוי השלישי – סדר ההכנסה משפיע על מספר הניתוקים ומספר העצים בסיום.** לכל סדר הכנסה, לאחר שמכניסים את האיברים ומוחקים את המינימום מקבלים ערימה בעלת אותו מבנה (עד כדי סדר המפתחות בצמתים), ולכן מספר החיבורים זהה עבור כל סדר הכנסה (לא מתבצעים חיבורים מלבד אלה שמחיקת המינימום גורמת). אלא שכאשר מתחילים לבצע מחיקות לאיבר המקסימלי בכל פעם, חלק מהמחיקות יגרמו לכך שצומת מסוים יהפוך למסומן. מיקומו בעץ של האיבר (שמושפע מסדר ההכנסה) שנרצה למחוק, קובע איזה איברים יהפכו למסומנים, וכך משפיע גם על מספר הניתוקים שיתבצעו לאורך המחיקות הבאות (שהרי מתבצע כאשר נמחק איבר מסוים, אם ההורה מסומן ננתק אותו (את ההורה) מההורה שלו, וכך הלאה). נשים לב כי ניתוקים אלו משפיעים גם על מספר העצים, ולכן גם הוא תלוי בסדר ההכנסה.

## **שאלה 4**

(מספר החיתוכים) – (מספר החיבורים) + (מספר העצים בסיום) = (גודל הערמה)

כלומר, גודל הערמה בסיום שווה לסכום מספר החיבורים ומספר העצים שנמצאים בערימה בסיום פחות מספר החיתוכים.

**נכונות:** בכל אחד מהניסויים, אנחנו מאתחלים ערמה ריקה, ומתחילים להוסיף לה איברים. כל איבר שמוסף לרשימה, מוסף כתת עץ חדש, ולכן בכל פעולת insert מספר תתי העצים גדל ב- 1. מצד שני, כל חיבור שמתבצע, מחבר שני תתי עצים ולכן מקטין את מספר תתי העצים ב- 1. בנוסף, כל חיתוך שמתבצע מגדיל את מספר תתי העצים בעץ ב- 1.

ולכן, כאשר נסכום את מספר תתי העצים שנמצאים בערמה ומספר החיבורים שבוצעו (שמפצה על הירידה במספר תתי העצים כיוון שכל חיבור הקטין את מספר תתי העצים ב-1), ונפחית מהסכום את מספר הניתוקים שנעשו (שמקזז את העליה במספר תתי העצים כיוון שכל ניתוק הוסיף תת עץ), נקבל את מספר תתי העצים שהיה בעץ אם לא היינו מבצעים חיבורים וניתוקים (כל צומת היה צומת בודד), שזהו בדיוק מספר הצמתים שנמצאים בעץ כרגע. ולכן צד ימין שווה לצד שמאל.

# מסמך התיעודים

## מחלקה ראשית: `FibonacciHeap`

תיאור: מחלקה שמייצגת ערימות פיבונאצ'י. כלומר כל אובייקט מטיפוס `FibonacciHeap` הינו ערימת פיבונאצ'י.

### שדות:

`min` – שדה מטיפוס HeapNode שמצביע על הצומת המינימלי בערימה.
`size` – שדה מטיפוס int שמכיל את מספר הצמתים שנמצאות בערימה.
`totalLinks` – שדה מטיפוס int שמכיל את סך החיבורים (links) שבוצעו.
`totalCuts` – שדה מטיפוס int שמכיל את סך הניתוקים (cuts) שבוצעו.
`numtrees` – שדה מטיפוס int שמכיל את מספר תתי העצים שנמצאים בערימה.

### מטודות:

`FibonacciHeap()`

מטודת בנאי, שמייצרת ערימה חדשה ששדה ה- min שלה מצביע על null. והשדות size, totalLinks ו- totalCuts מאותחלים ל-0 .

סיבוכיות: הפונקציה מבצעת השמות לתוך מספר סופי של שדות ולכן רצה בזמן קבוע O(1) במצב הגרוע וגם באמורטייזיד.

`addtorootlist(HeapNode node)`

הפונקציה מקבלת מצביע לשורש של תת עץ, ומוסיפה את השורש הזה לשרשרת השורשים שמכילה את השורשים של כל תתי העצים הנמצאים בערימה.

סיבוכיות: הפונקציה מבצעת בדיקה אחת שרצה בזמן קבוע ומבצעת מספר סופי של עדכוני שדות ומצביעים, אשר גם הם מבוצעים בזמן קבוע. ולכן הפונקציה מסיבוכיות O(1) במקרה הגרוע ובפרט גם אמורטייזיד.

`insert(int key, String info)`

פונקציה שמקבלת מפתח `key` וגם מידע `info`, מייצרת צומת חדש שהמפתח ששמור בו הוא `key` והמידע ששמור בו הוא `info`, ומוסיפה את הצומת הזה לעץ כתת עץ חדש בעל צומת אחד (שורש) ע״י קריאה לפונקציה `addtorootlist` . ומעדכנת את השדות הרלוונטים בהתאם.

סיבוכיות: הפונקציה מייצרת צומת חדש בזמן קבוע ומבצעת מספר סופי של בדיקות ועדכוני שדות ומצביעים, אשר מתבצעות בזמן קבוע. ובמקרה הגרוע, הפונקציה קוראת לפונקציה `addtorootlist` אשר רצה בזמן קבוע גם היא. ולכן סיבוכיות הזמן של הפונקציה היא O(1) במקרה הגרוע ובפרט גם אמורטייזיד.

findMin()

הפונקציה מחזירה מצביע לצומת בעל המפתח המינימלי בעץ.

סיבוכיות: הפונקציה אך ורק ניגשת לצומת ומחזירה מצביע אליו, ולכן היא מסיבוכיות קבועה O(1) במקרה הגרוע ובפרט גם אמורטייזיד.

find_min_in_sequence(HeapNode node)

הפונקציה מקבלת מצביע לשורש של תת עץ שנמצא בשרשרת של שורשים של תתי עצים שנמצאים בערימה. הפונקציה סורקת את כל השורשים שנמצאים בשרשרת ומוצאת את השורש בעל המפתח הקטן ביותר (הרי בפרט זה הוא גם הצומת המינימלי בערימה שמורכבת אך ורק מתתי העצים שנמצאים בשרשרת).

סיבוכיות: הפונקציה סורקת את כל השורשים שנמצאים בשרשרת, ובכל שורש מבצעת עבודה שלוקחת זמן קבוע ולכן הסיבוכיות שלה לינארית במספר השורשים שנמצאים בשרשרת השורשים, ומכיוון שמספר הצמתים שנמצאים בשרשרת הוא לכל היותר $O(n)$ אזי הסיבוכיות היא $O(n)$ במקרה הגרוע.

update_deleted_node_childs()

הפונקציה מעדכנת את כל הבנים של הצומת המינימלי כך ששדות ה- parent שלהם יצביעו על null. הפונקציה עושה זאת ע״י מעבר על כל הבנים של הצומת המינימלי.

סיבוכיות: ראשית, הפונקציה מבצעת בדיקה שלוקחת זמן קבוע ומספר סופי של השמות ועדכוני שדות אשר גם הם לוקחים זמן קבוע. לאחר מכן, הפונקציה עוברת בלולאה על הבנים של הצומת המינימלי בעץ, ובכל איטרציה מבצעת עבודה שלוקחת זמן קבוע. ולכן סיבוכיות הפונקציה היא לינארית במספר הבנים של הצומת המינימלי, ומכיוון שמספר הבנים של הצומת המינימלי הוא לכל היותר $O(\log n)$ (הוכחנו זאת בכיתה עבור כל שורש בערמה) אזי זמן הריצה הוא $O(\log n)$.

link(HeapNode root1, HeapNode root2)

הפונקציה מקבלת שני שורשים של תתי עצים, ששניהם מאותה דרגה (השורשים), ומבצעת להם link כפי שלמדנו בשיעור. כלומר, את השורש בעל המפתח הגדול ביותר היא הופכת לבן של השורש השני, וכך מייצרת תת עץ חדש מדרגה גדולה ב- 1 מהדרגה של כל אחד מתתי העצים שהועברו כקלט.

סיבוכיות: הפונקציה מבצעת מספר סופי של בדיקות שכל אחת מהן לוקחת זמן קבוע, ומבצעת גם מספר סופי של עדכוני שדות ומצביעים אשר גם הם מתבצעים בזמן קבוע, וכך גם הפעולות האריתמטיות שהפונקציה מבצעת. ולכן סיבוכיות זמן הריצה של הפונקציה היא O(1) במקרה הגרוע ובפרט גם אמורטייזיד.

```
totalLinks()
```

הפונקציה מחזירה את מספר החיבורים links שבוצעו על הערימה.

סיבוכיות: הפונקציה אך ורק ניגשת לשדה ומחזירה את ערכו, ולכן היא מסיבוכיות קבועה O(1) במקרה הגרוע ובפרט גם אמורטייזיד.

```
totalCuts()
```

הפונקציה מחזירה את מספר הניתוקים cuts שבוצעו על הערימה.

סיבוכיות: הפונקציה אך ורק ניגשת לשדה ומחזירה את ערכו, ולכן היא מסיבוכיות קבועה O(1) במקרה הגרוע ובפרט גם אמורטייזיד.

```
meld(FibonacciHeap heap2)
```

הפונקציה מקבלת ערימת פיבונאצי נוספת וממזגת אותה עם הערימה הנוכחית, ע״י חיבור רשימת השורשים של הערימה השנייה לזו של הערימה הנוכחית. הפונקציה גם מעדכנת את השדות הרלוונטים בהתאם, לרבות המצביע למינימום.

סיבוכיות: הפונקציה מבצעת מספר סופי של בדיקות שכל אחת מהן לוקחת זמן קבוע, ובמקרה הגרוע היא מבצעת גם מספר סופי של עדכוני שדות ומצביעים שגם הם מתבצעים בזמן קבוע. וכך גם הפעולות האריתמטיות שהפונקציה מבצעת. ולכן הפונקציה מסיבוכיות O(1) במצב הגרוע ובפרט גם באמורטייזיד.

```
size()
```

הפונקציה מחזירה את הערך השמור בשדה size של הערימה, כלומר מספר הצמתים שנמצאים בערימה.

סיבוכיות: הפונקציה אך ורק ניגשת לשדה ומחזירה את ערכו, ולכן היא מסיבוכיות קבועה O(1) במקרה הגרוע ובפרט גם אמורטייזיד.

```
numTrees()
```

הפונקציה מחזירה את הערך השמור בשדה numtrees של הערימה, כלומר מספר תתי העצים שנמצאים בערימה.

סיבוכיות: הפונקציה אך ורק ניגשת לשדה ומחזירה את ערכו, ולכן היא מסיבוכיות קבועה O(1) במקרה הגרוע ובפרט גם אמורטייזיד.

**לצורך ניתוחי עלויות אמורטייזיד של הפונקציות למטה, נסמן:**

$$\mathrm{T_0} - \text{number of trees before}$$

$$\mathrm{T_1} - \text{number of trees after}$$

**ונגדיר את פונקציית הפוטנציאל:**

$$\Phi = 2 * \text{numOfTrees} + 3 * \text{numOfMarked}$$

## consolidate()

הפונקציה מבצעת consolidating ו- successive linking (במידת הצורך) כפי שראינו בשיעור. כלומר, היא מאתחלת מערך שהתאים שלו משמשים כ- pockets ועוברת על רשימת השורשים של הערימה, ושמה כל שורש בתא שהאינדקס שלו שווה לדרגת השורש. אם בתא זה יש כבר שורש, היא מבצעת link לשני השורשים ומעבירה את תת העץ החדש לתא הבא, וממשיכה באותה הצורה (מה שיכול לגרום ל- successive linking). בסוף, הפונקציה מחברת את כל השורשים שנמצאים במערך כלומר מייצרת מהם רשימת שורשים חדשה.

סיבוכיות: הפונקציה מאתחלת מערך בגודל $O(\log n)$ בעלות $O(\log n)$. וכל מעבר על המערך יעלה לנו $O(\log n)$ לרבות המעבר האחרון שבו מחברים את כל השורשים לרשימת שורשים אחת. הפונקציה עוברת על כל תתי העצים שיש בשרשרת השורשים (יש לכל היותר n כאלה), והיא מבצעת לכל היותר $O(n)$ חיבורים, שהרי כל חיבור מקטין את מספר תתי העצים ב- 1 כאשר מספרם המקסימלי האפשרי הוא $n$ והמינימלי האפשרי הוא 1. שאר הפעולות שהפונקציה מבצעת הן מסיבוכיות קבועה ולכן אינן משפיעות על הסיבוכיות הכוללת. מכאן, הפונקציה מסיבוכיות $O(n)$ במקרה הגרוע.

סיבוכיות אמורטייזיד:

נסמן:

$$\mathrm{L_i} = \text{number of links when processing tree i} \;, \qquad \mathrm{L} = \text{total number of links}$$

$$\mathrm{k} = \text{rank of deleted root}$$

מתקיים:

$$\text{actual time} \le \sum_i L_i + 1 = L + T_0 + k - 1 \le 2(T_0 + \log n)$$

שהרי כל חיבור מקטין את מספר העצים בערמה ב- 1 ולכן לא ייתכן שעשינו יותר מ- $\mathrm{T_0}$ חיבורים, ולכן $L \le T_0$ ובנוסף ראינו בשיעור כי הדרגה של כל שורש של תת עץ בערימה היא $O(\log n)$ ולכן $k = O(\log n)$. נשים לב כי מתקיים $T_1 = O(\log n)$ כפי שראינו בשיעור. ולכן נקבל:

$$\text{amortized cost} = \text{actual cost} + \Phi_{\text{after}} - \Phi_{\text{before}} \le 2(T_0 + \log n) + 2(T_1 - T_0)$$
$$= 2T_1 + 2\log n = c * \log n = O(\log n)$$

**ולכן עלות אמורטייזיד של הפונקציה היא $O(\log n)$.**

`deleteMin()`

הפונקציה מוחקת את הצומת המינימלי מהערימה, מחפשת את המינימום החדש ומעדכנת את המצביע למינימום להצביע עליו. בנוסף, הפונקציה מבצעת consolidating שמתבצעת כפי שהסברנו למעלה. הפונקציה גם מעדכנת את שדות ה- parent של הבנים של הצומת שנמחק, כך שיצביעו ל- null.

סיבוכיות: הפונקציה מחפשת את המינימום החדש בערימה ע״י סריקת כל הבנים של המינימום הישן וגם כל השורשים של שאר תתי העצים שנמצאים בערימה (מבצעת זאת ע״י קריאות ל- `find_min_in_sequence` אשר רצה בזמן לינארי באורך רשימת השורשים). נשים לב שבמקרה הגרוע הסריקה הזו תעלה לנו $O(n)$ שהרי מספר השורשים שנצטרך לסרוק הוא $O(n)$. בנוסף הפונקציה תבצע `consolidate` אשר יעלה לנו $O(n)$ במקרה הגרוע. בנוסף, עדכון שדות ה- parent של הבנים של הצומת שנמחק יעלה לנו $O(\log n)$ במקרה הגרוע. שאר הפעולות שהפונקציה מבצעת לוקחות זמן קבוע. ולכן סיבוכיות הפונקציה היא $O(n)$ במקרה הגרוע.

סיבוכיות אמורטייזיד: העלות האמיתית היא עלות מציאת המינימום החדש כלומר סריקת כל השורשים של $\mathrm{T}_0$ תתי העצים, וגם ביצוע מספר סופי של פעולות שרצות בזמן קבוע, וגם ביצוע `consolidate`. נשים לב שביצוע `consolidate` כולל אתחול מערך בגודל $O(\log n)$ ומעבר עליו מספר קבוע של פעמים, ובנוסף, כולל גם ביצוע לכל היותר $\mathrm{T}_0$ חיבורים (שהרי כל חיבור מקטין את מספר העצים בערמה ב- 1 ולכן לא ייתכן שעשינו יותר מ- $\mathrm{T}_0$ חיבורים). ולכן העלות של `consolidate` היא $O(\mathrm{T}_0 + \log n)$. נשים לב כי מתקיים $T_1 = O(\log n)$ כפי שראינו בשיעור.

$$\text{amortized cost} = \text{actual cost} + \Phi_{\text{after}} - \Phi_{\text{before}} = (2T_0 + 2\log n) + 2(T_1 - T_0)$$
$$= 2\log n + 2T_1 \leq c * \log n = O(\log n)$$

ולכן לפי הניתוח למעלה ובהסתמך על מה שראינו בשיעור נקבל כי עלות אמורטייזיד של הפונקציה היא $O(\log n)$.

`cut(HeapNode x)`

הפונקציה מקבלת מצביע לצומת x שנמצא בעץ, ומנתקת אותו מאביו, ומסמנת את האבא אם הוא לא מסומן. אם האבא אכן מסומן, מתבצעת קריאה רקורסיבית לפונקציה על האבא, וכך הלאה. הפונקציה מוסיפה לרשימת השורשים את הצומת שניתקנו אותו, ומעדכנת את השדות של הצומת שניתקנו בהתאם, ומעדכנת גם את השדה numtrees, ו- totalCuts של העץ.

סיבוכיות: הפונקציה מבצעת מספר סופי של בדיקות ועדכוני שדות ומצביעים. כל אחת מפעולות אלו מתבצעת בזמן קבוע ולכן כולן מתבצעות בזמן קבוע. במקרה הגרוע, הפונקציה תקרא לעצמה (קריאה רקורסיבית) על ההורה של הצומת הנוכחי, ותתבצע סדרה של קריאות רקורסיביות על כל הצמתים שנמצאים על המסלול מהצומת x ועד לשורש. כלומר סדרה של קריאות שאורכה הוא לכל היותר כאורך המסלול מ- x ועד לשורש. אורך המסלול חסום ע״י $O(n)$ שהרי הגובה המקסימלי של ערמת פיבונאצ'י הוא $O(n)$. ולכן, ומכיוון שבכל קריאה רקורסיבית תתבצע עבודה שלוקחת זמן קבוע, אזי הפונקציה מסיבוכיות $O(n)$ במקרה הגרוע. ניתוח סיבוכיות אמורטייזיד זהה לחלוטין לניתוח עלות אמורטייזיד שביענו עבור `decreaseKey` למטה. ולכן עלות אמורטייזיד של הפונקציה היא $O(1)$.

`decreaseKey(HeapNode x, int diff)`

הפונקציה מקבלת מצביע x לצומת בערימה, ומקבלת מספר שלם `diff`, היא מפחיתה את המפתח של הצומת x ב- `diff`. כאשר הפונקציה מבצעת את השינוי הזה, יכול להיות שכלל הערמה הופר. אם הכלל אכן הופר, הפונקציה תבצע cut לצומת באמצעות קריאה לפונקציה `cut`, מה שיכול גם לגרום ל- cascading cut.

סיבוכיות: הפונקציה מבצעת מספר סופי של בדיקות ופעולות אריתמטיות, אשר מתבצעות בזמן קבוע. בנוסף, הפונקציה קוראת ל- `cut` במידת הצורך. נשים לב כי סיבוכיות `cut` במקרה הגרוע היא $O(n)$ כפי שהראינו למעלה. ולכן הפונקציה מסיבוכיות $O(n)$ במקרה הגרוע.

סיבוכיות אמורטייזיד: הפונקציה מבצעת קריאה לפונקציה `cut` שיכולה לגרום לשרשרת של ניתוקים, כלומר לשרשרת של קריאות רקורסיביות לפונקציה `cut` אשר כל אחת מהן מבצעת עבודה שלוקחת זמן קבוע. נסמן ב- $c$ את מספר הקריאות הרקורסיביות הללו (כלומר מספר הניתוקים). נשים לב ש- $c$ ניתוקים יכולים לגרום ל- לכל היותר $c-1$ צמתים מסומנים להפוך ללא מסומנים ועוד צומת אחד לא מסומן להפוך למסומן (כפי שראינו בשיעור). בנוסף, נשים לב ששרשרת כזו תגרום להגדלת מספר תתי העצים שבערמה ב- $c$ לכל היותר, שהרי כל ניתוק מייצר תת עץ חדש. שאר הפעולות שהפונקציה מבצעת עולות זמן קבוע. מתקיים:

$$\text{amortized cost} = \text{actual cost} + \Phi_{\text{after}} - \Phi_{\text{before}} \leq c + 2(T_1 - T_0) + 3(c-1)$$
$$\leq c + 2c + 3(2-c) = 6 = O(1)$$

ולכן לפי הניתוח למעלה ובהסתמך על מה שראינו בשיעור נקבל כי עלות אמורטייזיד של הפונקציה היא $O(1)$.

`delete(HeapNode x)`

הפונקציה מקבלת מצביע לצומת x שנמצא בערימה, ומוחקת אותו מהערימה. אם x הוא הצומת המינימלי, הפונקציה עושה זאת ע״ קריאה לפונקציה deletemin. אחרת, הפונקציה מוחקת את הצומת ע״י ניתוק הצומת מאביו באמצעות קריאה ל- cut, ומחברת את הבנים שלו לרשימת השורשים של הערמה.

סיבוכיות: אם הצומת הוא הצומת המינימלי אז ניתוח הסיבוכיות זהה לחלוטין לניתוח של deletemin, כלומר $O(n)$ במקרה הגרוע. אחרת, הפונקציה מבצעת מספר סופי של פעולות אריתמטיות, בדיקות ועדכוני שדות, כלומר פעולות אשר כל אחת מהן לוקחת זמן קבוע ולכן כולן יחד עולות זמן קבוע. בנוסף, במקרה הגרוע הפונקציה קוראת לפונקציה cut אשר עולה לכל היותר $O(n)$ זמן. ולכן הפונקציה מסיבוכיות $O(n)$ במקרה הגרוע.

ניתוח אמורטייזיד: אם הצומת שנרצה למחוק הוא הצומת המינימלי, הפונקציה תבצע אך ורק קריאה לפונקציה deletemin ותסיים, ולכן זמן הריצה שלה יהיה זהה לחלוטין לזמן הריצה של deletemin. אחרת, הפונקציה תבצע מספר סופי של עדכוני שדות ויצירת צמתים חדשים, פעולות אשר רצות בזמן קבוע כל אחת. בנוסף, הפונקציה מבצעת מספר סופי של בדיקות אשר מתבצעות בזמן קבוע כל אחת. במקרה הגרוע, הפונקציה קוראת גם לפונקציה cut, אשר הראינו קודם כי זמן הריצה שלה הוא $O(1)$ אמורטייזיד. מכאן נובע כי עבור כל פעולת `delete` מתקיים: $\text{amortized}(\text{op}) \leq \mathrm{O}(\max\{1, \log n\}) = O(\log n)$ ולכן העלות אמורטייזיד היא $O(\log n)$.

## **מחלקה פנימית:** HeapNode

תיאור: מחלקה שמייצגת צומת בערימת פיבונאצ'י. כלומר כל אובייקט מטיפוס HeapNode הינו צומת בערימת פיבונאצ'י.

**שדות:**

key – שדה מטיפוס int שמכיל את המפתח השמור בצומת.
info – שדה מטיפוס String שמכיל את המידע השמור בצומת.
child – שדה מטיפוס HeapNode שמצביע על אחד הבנים של הצומת.
next – שדה מטיפוס HeapNode שמצביע על האח הבא של הצומת.
prev – שדה מטיפוס HeapNode שמצביע על האח הקודם של הצומת.
parent – שדה מטיפוס HeapNode שמצביע על ההורה של צומת זה בערימה.
rank – שדה מטיפוס int שמכיל את דרגת הצומת, כלומר מספר הבנים שיש לו.
mark – שדה מטיפוס boolean ששומר האם הצומת מסומן (marked) או לא.

**מטודות:**

```
HeapNode(int key, String info)
```

מטודת בנאי, שמקבלת מפתח key, ומקבלת מידע info, ומייצרת צומת חדש שמכיל את המידע והפתח שהועברו כקלט, המצביעים parent ו- child מצביעים ל- null, והמצביעים next ו- prev מצביעים לצומת החדש עצמו. הדרגה של הצומת החדש היא 0, והוא אינו מסומן.

סיבוכיות: הפונקציה מבצעת השמות לתוך מספר סופי של שדות ולכן רצה בזמן קבוע O(1) במצב הגרוע וגם באמורטייזיד.