שם מגיש: עומר דוד דה-סבן

תעודת זהות: 315635441

שם משתמש: $omerdavidd$

(רני הוד אישר לי להגיש לבד)

סיבוכיות: פשוט וקל

אני אתחיל בלציין שמעבר לרשימה זו, קיים נספח בסוף הקובץ (מתחיל בעמוד 6 וממשיך עד עמוד 14, אז כן – זה כנראה מעמיק יותר מהרצוי) שמכיל ניתוח מעמיק יותר של כלל הפונקציות והסיבוכיות שלהן. כמו כן, אתחיל מלציין את סיבוכיות המחלקות מהמחלקה הפנימית ביותר ($HeapItem$) ועד המחלקה החיצונית ביותר ($BinomialHeap$). בנוסף, בכל מתודה בה קיים פרמטר, לא אציין זאת אבל ישנה דרישת קדם ברירת מחדל שהקלט שלה יהיה מהמחלקה המתאימה לפרמטר המצוין. לבסוף, אציין על כל מתודה האם היא מתודה סטטית (ניתנת לקריאה על ידי המחלקה עצמה, ואף רצוי שלא לקרוא לה על ידי מופע של המחלקה), מתודת מופע (ניתנת לקריאה על ידי מופע של המחלקה בלבד), או בנאי:

מחלקת $HeapItem$:

למחלקה זו לא התווספו שדות.

כמו כן, למחלקה זו לא היו מתודות בקובץ השלד, ולכן קיימות בה רק מתודות שהתווספו אליה, נציין כעת את הסיבוכיות שלהן:

<u>מתודות שהתווספו לקובץ השלד ולא היו בו מראש:</u>

1. $HeapItem()$: בנאי; אין דרישות קדם; סיבוכיות של $O(1)$.
2. $HeapItem(int\ key, String\ info)$: בנאי; אין דרישות קדם; סיבוכיות של $O(1)$.

מחלקת $HeapNode$:

שדות שהתווספו למחלקה זו:

1) $last$: מצביע על הצאצא עם הדרגה הגבוהה ביותר מבין הצאצאים של $this$.
2) min: מצביע על הצאצא עם המפתח המינימלי ביותר מבין הצאצאים של $this$.

גם למחלקה זו לא היו מתודות בקובץ השלד, ולכן גם בה קיימות רק מתודות שהתווספו אליה. נציין כעת את הסיבוכיות שלהן:

<u>מתודות שהתווספו לקובץ השלד ולא היו קיימות בו מראש:</u>

1. $HeapNode()$: בנאי; אין דרישות קדם; סיבוכיות של $O(1)$.
2. $HeapNode(HeapItem\ item)$: בנאי; **קיימת דרישת קדם** לפיה $item.key > 0$; סיבוכיות של $O(1)$.

3. $smallerKey(HeapNode\ node1, HeapNode\ node2)$: מתודה סטטית ; אין דרישות קדם ; סיבוכיות של $O(1)$.
4. $largerKey(HeapNode\ node1, HeapNode\ node2)$: מתודה סטטית ; אין דרישות קדם ; סיבוכיות של $O(1)$.
5. $linkNodes(HeapNode\ node)$: מתודת מופע ; **קיימות דרישות קדם** לפיהן גם $this$ וגם $node$ הם לא $null$; סיבוכיות של $O(1)$.
6. $postLinkFix()$: מתודת מופע ; אין דרישות קדם ; סיבוכיות של $O(1)$.
7. $smallerRank(HeapNode\ node1, HeapNode\ node2)$: מתודה סטטית ; אין דרישות קדם ; סיבוכיות של $O(1)$.
8. $intoMelded(HeapNode[]\ melded, int\ melded\_\max\ \_index)$: מתודת מופע ; **קיימת דרישת קדם** לפיה $melded\_\max\ \_index \geq 0$; סיבוכיות של $O(1)$.
9. $shiftUp()$: מתודת מופע ; אין דרישות קדם ; אם נסמן ב – d את גובה הצומת $this$ (כלומר כמות הצמתים שיש לעבור מהצומת $this$ עד שמגיעים לשורש העץ הבינומי), אזי סיבוכיות של $O(d)$. אם נסמן ב - n_{this} את כמות הצמתים בעץ הבינומי, אזי סיבוכיות במקרה הגרוע של $O(\log_2(n_{this}))$.

<u>***מחלקת $BinomialHeap$:***</u>

שדות שהתווספו למחלקה זו :

1) $first$: מצביע על הצומת שמייצגת את השורש של העץ הבינומי מהדרגה הנמוכה ביותר בערמה הבינומית (ללא מבט על תתי עצים).
2) $trees\_num$: כמות העצים הבינומיים בערמה הבינומית (לא כולל תתי עצים).

ניתוח מתודות (נתחיל דווקא ממתודות שלא היו קודם בקובץ השלד) :

<u>מתודות שהתווספו לקובץ השלד ולא היו קיימות בו מראש :</u>

1. $BinomialHeap()$: בנאי ; אין דרישות קדם ; סיבוכיות של $O(1)$.
2. $heapFromChildren(HeapNode\ node)$: מתודה סטטית ; אין דרישות קדם ; סיבוכיות של $O(node.rank + 1)$.
3. $heapWithoutATree(HeapNode\ node)$: מתודת מופע ; אין דרישות קדם ; סיבוכיות של $O(this.trees\_num\ + 1)$.
4. $setSelf(BinomialHeap\ heap)$: מתודת מופע ; אין דרישות קדם ; סיבוכיות של $O(1)$.

<u>מתודות שהיו מראש בקובץ השלד (נתחיל פה דווקא מהמתודה $empty()$ ומהמתודה $meld(BinomialHeap\ heap2)$ כי הן משפיעות על חלק מהאחרות) :</u>

5. $empty()$: מתודת מופע ; **קיימת דרישת קדם** לפיה $this$ היא לא $null$; סיבוכיות של $O(1)$.
6. $meld(BinomialHeap\ heap2)$: מתודת מופע ; אין דרישות קדם ; סיבוכיות במקרה הגרוע של $O(this.trees\_num + heap2.tress\_num + 1)$.

7. $insert(int\ key, String\ info)$: מתודת מופע ; **קיימת דרישת קדם** לפיה $key > 0$; סיבוכיות במקרה הגרוע ביותר של:
$O(this.trees\_num + 1 + 1) = O(this.trees\_num + 1)$.
8. $deleteMin()$: מתודת מופע ; לא קיימות דרישות קדם ; סיבוכיות במקרה הגרוע ביותר של:
$$O\big((this.trees\_num - 1) + (this.min.rank) + 1\big) =$$
$$= O(this.trees\_num + this.min.rank).$$
9. $findMin()$: מתודת מופע ; אין דרישות קדם ; סיבוכיות של $O(1)$.
10. $decreaseKey(HeapItem\ item, int\ diff)$: מתודת מופע ; **קיימת דרישת קדם** לפיה $0 < diff < item.key$; סיבוכיות של $O(1)$.
11. $delete(HeapItem\ item)$: מתודת מופע ; **קיימת דרישת קדם** לפיה $this$ אינה ערמה בינומית ריקה, ומעבר לכך - $item$ קיים בתוך $this$; סיבוכיות במקרה הגרוע של:
$$O\big((this.trees\_num - 1) + (item.node.rank) + 1\big) =$$
$$= O(this.trees\_num + item.node.rank).$$
12. $size()$: מתודת מופע ; אין דרישות קדם ; סיבוכיות של $O(1)$.
13. $numTrees()$: מתודת מופע ; אין דרישות קדם ; סיבוכיות של $O(1)$.

חלק ניסויי / תיאורטי

1. בכל אחד מהניסויים, ולכל $i \in [1,6] \cap \mathbb{N}$, חזרתי על כל ניסוי 100 פעמים. בטבלאות רשומים הממוצעים של 100 ההרצות השונות הללו. שימו לב שבניסויים בהם בוצעו מחיקות, העמודה 'מספר החיבורים הכולל', כולל גם את מספר החיבורים לצורך הכנסת האיברים וגם את מספר החיבורים לצורך מחיקת האיברים. כמו כן, בניסויים שבהם בוצעו מחיקות, העמודה 'סכום דרגות הצמתים שנמחקו' מתארת את סכום דרגות הצמתים כפי שהיו לפני תהליך פעפוע הצמתים למעלה (אשר בוצע לפי הצורך).

בניסוי הראשון, לכל $i \in [1,6] \cap \mathbb{N}$, הכנסנו לערמה בינומית ריקה את האיברים של הקבוצה $\left[1,\ 3^{i+5} - 1\right] \cap \mathbb{N}$ בסדר עולה (כלומר הכנסנו את 1 ראשון ואת $3^{i+5} - 1$ אחרון). להלן טבלה המתארת את הניסוי הראשון:

מספר סידורי i	זמן ריצה (מילישניות)	מספר החיבורים הכולל	מספר העצים בסיום
$i = 1$	0.072797	723	5
$i = 2$	0.191766	2,182	4
$i = 3$	0.578129	6,555	5
$i = 4$	1.667553	19,675	7
$i = 5$	5.271124	59,040	8
$i = 6$	16.047503	177,134	12

בניסוי השני, לכל $i \in [1,6] \cap \mathbb{N}$, הכנסנו לערמה בינומית ריקה את האיברים של הקבוצה $[1,\ 3^{i+5}-1] \cap \mathbb{N}$ בסדר אקראי, ואז מחקנו את האיבר המינימלי $\frac{3^{i+5}-1}{2}$ פעמים. להלן טבלה המתארת את הניסוי השני:

מספר סידורי i	זמן ריצה (מילישניות)	מספר החיבורים הכולל	מספר העצים בסיום	סכום דרגות הצמתים שמחקנו
$i=1$	0.205591	72,300	5	2,912
$i=2$	0.680915	218,200	4	10,409
$i=3$	2.291568	655,500	5	36,536
$i=4$	8.850851	1,967,500	7	125,373
$i=5$	34.247709	5,904,000	8	421,370
$i=6$	122.738230	17,713,400	12	1,410,391

בניסוי השלישי, לכל $i \in [1,6] \cap \mathbb{N}$, הכנסנו לערמה בינומית ריקה את האיברים של הקבוצה $[1,\ 3^{i-5}-1] \cap \mathbb{N}$ בסדר יורד (כלומר הכנסנו את $3^{i+5}-1$ ראשון ואת 1 אחרון), ואז מחקנו את האיבר המינימלי עד שנותרו רק 2^5-1 איברים בערמה. להלן טבלה המתארת את הניסוי השלישי:

מספר סידורי i	זמן ריצה (מילישניות)	מספר החיבורים הכולל	מספר העצים בסיום	סכום דרגות הצמתים שמחקנו
$i=1$	0.141243	723	5	697
$i=2$	0.422373	2,182	5	2,156
$i=3$	1.394433	6,555	5	6,529
$i=4$	4.377695	19,675	5	19,649
$i=5$	13.991457	59,040	5	59,014
$i=6$	43.112536	177,134	5	177,108

2. יהא $i \in [1,6] \cap \mathbb{N}$. נגדיר $n_i := 3^{i+5}-1$ (מספר הצמתים שנכניס לערמה בכל אחד מהניסויים עבור i זה).

עבור הניסוי הראשון:
מתבצעות n_i פעולות הכנסה לערמה בינומית, כך שבכל פעם מכניסים איבר מקסימלי חדש לערמה (כלומר איבר שהמפתח שלו גדול מהמפתח של כל אחד משאר האיברים בערמה). ראינו כבר כי סיבוכיות המתודה $insert(int\ key, String\ info)$ היא $O(\log_2(n_{this})+1)$, כפי שהוסבר בנספח. מכאן נסיק כי סיבוכיות כל הכנסה כזו היא לכל היותר $O(\log_2(n_i)+1)$. לכן סיבוכיות זמן הריצה עבור ניסוי זה היא לכל היותר: $n_i \cdot O(\log_2(n_i)+1) = O(n_i \cdot \log_2(n_i)+1)$. כמו כן, נשים לב כי:
$n_i =$ מספר העצים בסיום עבור i זה + מספר החיבורים הכולל עבור i זה.

עבור הניסוי השני:
מתבצעות n_i פעולות הכנסה לערמה בינומית, כך שבכל פעם מכניסים איבר אקראי לערמה. ראינו כבר כי סיבוכיות המתודה

$insert(int\ key, String\ info)$ היא $O(\log_2(n_{this}))$, כפי שהוסבר בנספח. . מכאן נסיק כי סיבוכיות כל הכנסה כזו היא לכל היותר $O(\log_2(n_i))$. לכן סיבוכיות זמן הריצה עבור ההכנסות בניסוי זה היא לכל היותר $n_i \cdot O(\log_2(n_i)) = O(n_i \cdot \log_2(n_i))$. לאחר הכנסות אלו, מבוצעות $\frac{n_i}{2}$ מחיקות של האיבר המינימלי בערמה. ראינו כבר כי סיבוכיות המתודה $deleteMin()$ היא $O(\lceil\log_2(n_{this})\rceil + 1)$, כפי שהוסבר בנספח. מכאן נסיק כי סיבוכיות כל מחיקה כזו היא לכל היותר $O(\lceil\log_2(n_i)\rceil + 1)$. לכן סיבוכיות זמן הריצה עבור המחיקות בניסוי זה היא לכל היותר: $\frac{n_i}{2} \cdot O(\lceil\log_2(n_i)\rceil + 1) = O(n_i \cdot \lceil\log_2(n_i)\rceil + 1)$, ומכאן סיבוכיות זמן הריצה עבור ניסוי זה היא לכל היותר:

$$O(n_i \cdot \log_2(n_i) + 1) + O(n_i \cdot \lceil\log_2(n_i)\rceil + 1) =$$
$$= O(n_i \cdot \lceil\log_2(n_i)\rceil + 1)$$

ראיתי אצל אחרים שהם לא הכלילו את מספר החיבורים שנוצרו כתוצאה מהמחיקות בעמודה של 'מספר החיבורים הכולל' אצלם, וכשזה היה כך, התקיים $\frac{n_i}{2}$ = מספר העצים בסיום עבור i זה + סכום דרגות הצמתים שנמחקו עבור i זה – מספר החיבורים הכולל עבור i זה. מודה שעם התוצאות שלי לא הצלחתי למצוא קשר ישיר, כנראה שלא הייתי צריך להכליל את החיבורים שנוצרו כתוצאה מהמחיקות בעמודה של 'מספר החיבורים הכולל' אצלי, אבל אז זה מביס את המטרה לפיה עמודה זו אמורה לייצג את מספר כלל החיבורים שנעשו בניסוי זה...

עבור הניסוי השלישי:

מתבצעות n_i פעולות הכנסה לערמה בינומית, כך שבכל פעם מכניסים איבר מינימלי חדש לערמה (כלומר איבר שהמפתח שלו קטן מהמפתח של כל אחד משאר האיברים בערמה). ראינו כבר כי סיבוכיות המתודה $insert(int\ key, String\ info)$ היא $O(\log_2(n_{this}) + 1)$, כפי שהוסבר בנספח. מכאן נסיק כי סיבוכיות כל הכנסה כזו היא לכל היותר $O(\log_2(n_i) + 1)$. לכן סיבוכיות זמן הריצה עבור ההכנסות בניסוי זה היא לכל היותר $n_i \cdot O(\log_2(n_i)) = O(n_i \cdot \log_2(n_i))$. לאחר הכנסות אלו, מבוצעות $n_i - (2^5 - 1) = n_i - 2^5 + 1$ מחיקות של האיבר המינימלי בערמה. ראינו כבר כי סיבוכיות המתודה $deleteMin()$ היא $O(\lceil\log_2(n_{this})\rceil + 1)$, כפי שהוסבר בנספח. מכאן נסיק כי סיבוכיות כל מחיקה כזו היא לכל היותר $O(\lceil\log_2(n_i)\rceil + 1)$. לכן סיבוכיות זמן הריצה עבור המחיקות בניסוי זה היא לכל היותר:

$$(n_i - 2^5 + 1) \cdot O(\lceil\log_2(n_i) + 1) =$$
$$= n_i \cdot O(\lceil\log_2(n_i)\rceil + 1) - 2^5 \cdot O(\lceil\log_2(n_i)\rceil + 1) + O(\lceil\log_2(n_i)\rceil + 1) =$$
$$= O(n_i \cdot \lceil\log_2(n_i)\rceil + 1) + O(\lceil\log_2(n_i)\rceil + 1) =$$
$$= O(n_i \cdot \lceil\log_2(n_i)\rceil + 1)$$

ומכאן סיבוכיות זמן הריצה עבור ניסוי זה היא לכל היותר:

$$O(n_i \cdot \log_2(n_i) + 1) + O(n_i \cdot \lceil\log_2(n_i)\rceil + 1) =$$

$= O(n_i \cdot \lceil \log_2(n_i) \rceil + 1)$

כמו כן, נשים לב לדמיון הרב בין ניסוי זה לבין הניסוי הראשון: האיברים מוכנסים בסדר קבוע וממוין, ולא באקראי. בניסוי הראשון תמיד קיבלנו שהמינימום הוא השורש של העץ בדרגה הגבוהה ביותר בערמה, כאן הוא תמיד השורש של העץ בדרגה הנמוכה ביותר בערמה.

כשנביט על ניסוי זה, המינימום הוא השורש של העץ בדרגה הנמוכה ביותר בערמה, והילדים שלו הם שורשים של עצים בדרגה נמוכה אף יותר – לכן כשנמחוק אותו לא נצטרך לאחד עצים בכלל. אז לא מתווספות פעולות חיבור.

כמו כן, אנו נותרים תמיד עם $31 = 1 - 2^5$ צמתים בערמה, וסה"כ 5 עצים.

נשים לב כי מתקיים $31 = 5 +$ סכום דרגות הצמתים שנמחקו עבור i זה – מספר החיבורים הכולל עבור i זה, ובנוסף, בדיוק כפי שהיה בניסוי הראשון:

$n_i =$ מספר העצים בסיום עבור i זה + מספר החיבורים הכולל עבור i זה.

נספח – סיבוכיות: ניתוח מעמיק

אני אתחיל בלומר שלכל פונקציה קיים פירוט בקובץ ה - *java*. שלי. פירוט זה הינו מעמיק (אך לא מדי, לדעתי – בול במידה, אולי קצת מעבר לחלק מהאנשים), ונוצר במחשבה שאדם אחר יוכל לתחזק את הקוד שלי במידת הצורך (לא שאני חושב שזה יקרה פה – אני מנסה ליצור הרגלים עבור עבודה עתידית בתור מתכנת, בתקווה שאמצא כמה שיותר מהר). פירוט זה הינו לא רק בצורה של *Doc Comment* מפורט שמסביר על כל פרמטר, על ערכי החזרה וכו', אלא גם בצורה של *comments* בתור הקוד עצמו, כל אחת ממוקמת בחלק הרלוונטי לה. אם משהו ממה שאני רושם בקובץ זה אינו ברור – ניתוח הסיבוכיות מופיע בקובץ ה - *py*. שלי בצורה של סיכום סיבוכיות כל מתודה כבר ב - *Doc Comment* שלה.

נעיר מראש שסיבוכיות קריאה לערכו של שדה של אובייקט ממחלקה X כלשהי (ללא חשיבות למי זו המחלקה X עצמה) בתוך מתודות ופונקציות היא $O(1)$ כל עוד השדה הרלוונטי הינו בנראות *public* (בה מצויים כל השדות של כל המחלקות במימוש שלי), ואחרת סיבוכיות קריאה זו היא כסיבוכיות מתודת ה - *getter* של שדה זה (נטו להעשרה – לא רלוונטי למימוש שלי). כך, למשל, בתוך המחלקה *BinomialHeap*, בתוך כל מתודה של *BinomialHeap*, ועבור כל אובייקט *heap* מסוג *BinomialHeap*, הקריאות *heap.last*, *heap.last.child*, *heap.last.child.item* וגם *heap.last.child.item.key* הן בסיבוכיות של $O(1)$. באופן דומה כך גם כל הקריאות לכל שאר השדות של כל אחת משלוש המחלקות הממומשות בקובץ השלד (*BinomialHeap*, *HeapNode* ו - *HeapItem*).

ננתח את סיבוכיות כלל הפונקציות שבקובץ, מלבד אלו שרק מבצעות קריאה לשדות מחלקה – שכן, כאמור, הן בסיבוכיות של $O(1)$. כמו כן, נציין על כל מתודה האם היא בנאי ריק או בנאי לא ריק של מחלקה מסוימת, או האם היא מתודה סטטית (מתודה שניתן לקרוא לה בעזרת המחלקה ללא מופע של המחלקה, ואפילו רצוי לא לקרוא לה על ידי מופע), או האם היא מתודת מופע (מתודה שניתן לקרוא לה רק על ידי מופע של המחלקה בשם *this*). יחד עם זאת נציין את נראות המתודות (במימוש שלי יש רק מתודות בנראות *public* או בנראות *private*). בנוסף, כפי שציינתי בתחילת הקובץ, בכל מתודה בה קיים פרמטר, לא אציין זאת אבל ישנה דרישת קדם ברירת מחדל שהקלט שלה יהיה מהמחלקה המתאימה לפרמטר המצוין.

נתחיל עם מחלקת ***HeapItem***, אליה לא הוספתי שדות. נשים לב שבמחלקה זו לא היו מתודות נתונות בקובץ השלד, ולכן נותר רק לנתח את סיבוכיות המתודות שהוספתי לקובץ השלד ולא היו

קיימות בו מראש, וכמובן – להסביר את תפקידן. נעיר רק שבמחלקה זו כל פעם שמופיע $this$, מדובר באובייקט בשם $this$ מהמחלקה $HeapItem$:

- המתודה $HeapItem()$ הינה בנאי ריק של המחלקה, והיא בנראות $public$. למתודה זו לא קיימות דרישות קדם. מתודה זו מאתחלת את ערך השדה $this.key$ להיות 1, וכפי שציינתי בקובץ השלד, יכולתי לא לכתוב את שאר המתודה, אך לשם בהירות הקוד הוספתי את מימושה כפי ש - $Java$ הייתה מממשת אותה באופן ברירת מחדל: אתחול השדות $this.node$ ו - $this.info$ להיות $null$. סיבוכיות של $O(1)$.
- המתודה $HeapItem(int\ key, String\ info)$ הינה בנאי לא ריק של המחלקה, והיא בנראות $public$. למתודה זה **קיימת דרישת קדם** לפיה $key > 0$. המתודה מאתחלת את השדות $this.key$ ו - $this.info$ להיות הפרמטרים key ו - $info$ בהתאמה. כפי שציינתי בקובץ השלד, יכולתי לוותר על המשך המתודה כי $Java$ הייתה מבצעת זאת באופן ברירת מחדל, אך לשם בהירות הוספתי את השורה לפיה מאתחלים את השדה $this.node$ להיות $null$. סיבוכיות של $O(1)$.

כעת נעבור למחלקה ***HeapNode*** (כן, המחלקה $HeapItem$ כה קצרה ונגמרת בבנאים בלבד), אליה הוספתי את השדות $last$ ו - min. עבור אובייקט $node$ כלשהו מסוג $HeapNode$, השדות $last$ ו – min הינם מצביעים לאובייקטים מסוג $HeapNode$ שמקיימים שהשדה $parent$ שלהם מצביע על $node$, וכן $last$ הוא האובייקט עבורו השדה $rank$ הינו הגדול ביותר, ו - min הינו האובייקט עבורו השדה $item.key$ (עבור השדה $item$ שלו) הוא הקטן ביותר, מבין כל האובייקטים מסוג $HeapNode$ שמקיימים שהשדה $parent$ שלהם מצביע על $node$. מעכשיו נאמר שאובייקט מסוג $HeapNode$ הינו "צומת", ועבור צומת $node$, כל צומת שמקיימת שהשדה $parent$ שלה מצביע על $node$ תיקרא "ילד של $node$". כל ילדים אלה, הילדים של ילדים אלה, ילדיהם, וילדיהם שלהם (וכך הלאה באופן אינדוקטיבי) שאינם $null$ יקראו "הצאצאים של $node$". יחד עם זאת, לכל צומת $node$, נגדיר את "הדרגה של $node$" להיות ערך השדה $node.rank$, ואת "המפתח של $node$" להיות ערך השדה $node.item.key$. תחת הגדרות אלה ניתן לומר שעבור צומת $node$, השדה $last$ מצביע על הילד של $node$ עם הדרגה הגדולה ביותר, וכן השדה min מצביע על הילד של $node$ עם המפתח הקטן ביותר.

קל לשים לב שלכל צומת $node$, השדה $node.rank$ מציין את כמות הילדים של $node$.

במחלקה זו לא היו מתודות נתונות בקובץ השלד, ולכן נותר רק לנתח את סיבוכיות המתודות שהוספתי לקובץ השלד ולא היו קיימות בו מראש, וכמובן – להסביר את תפקידן (נתחיל דווקא במתודה HeapNode(HeapItem item) כי המתודה $HeapNode()$ משתמשת בה). נעיר רק שבמחלקה זו כל פעם שמופיע $this$, מדובר באובייקט בשם $this$ מהמחלקה $HeapNode$, כלומר $this$ הינו צומת:

- המתודה $HeapNode(HeapItem\ item)$ הינה בנאי לא ריק של המחלקה $HeapNode$, והיא בנראות $public$. למתודה זו לא **קיימת דרישת קדם**, לפיה $item.key > 0$. מתודה זו מאתחלת את השדה $this.item$ להיות הפרמטר $item$, ואת השדה $this.item.node$ היא מעדכנת להיות $this$. כפי שציינתי בקובץ השלד, יכולתי שלא למלא את שאר המתודה כי $Java$ הייתה מבצעת זאת באופן ברירת מחדל, אך הוספתי את שאר המתודה לשם בהירות הקוד: אתחול השדות $this.child$, $this.next$, $this.parent$, $this.last$ ו - $this.min$ להיות $null$, ואת השדה $this.rank$ להיות 0. סיבוכיות של $O(1)$.

- המתודה $HeapNode()$ הינה בנאי ריק של המחלקה $HeapNode$, והיא בנראות $public$. למתודה זו לא קיימות דרישות קדם. מתודה זו מבצעת את הקריאה הבאה: $this(new\ HeapItem())$ בלבד. סיבוכיות של $O(1)$.
- המתודה $smallerKey(HeapNode\ node1, HeapNode\ node2)$ היא מתודה סטטית, והיא בנראות $private$. למתודה זו אין דרישות קדם. מתודה זו בודקת לאיזה משני הצמתים הנתונים יש מפתח קטן יותר (ומחזירה את צומת זה), ובמקרה שבו שני הצמתים הם $null$ היא גם מחזירה $null$. סיבוכיות של $O(1)$.
- המתודה $largerKey(HeapNode\ node1, HeapNode\ node2)$ היא מתודה סטטית, והיא בנראות $private$. למתודה זו אין דרישות קדם. מתודה זו מבצעת את הקריאה $HeapNode\ smaller = smallerKey(node1, node2)$, ואז בודקת האם $smaller$ הוא $null$ (אם כן היא מחזירה $null$), אחרת היא מחזירה את הצומת (מבין $node1$ ו - $node2$) שאיננו $smaller$. סיבוכיות של $O(1)$.
- המתודה $linkNodes(HeapNode\ node)$ היא מתודת מופע, והיא בנראות $private$. למתודה זו **קיימת דרישות קדם** לפיהן גם $this \neq null$ וגם $node \neq null$. מתודה זו מבצעת את הקריאות $HeapNode\ root\_node = smallerKey(this, node)$ וגם $HeapNode\ child\_node = largerKey(this, node)$, ואז, על ידי עדכוני שדות, מקשרת בין $root\_node$ ו - $child\_node$ כך שיתקיים:
  $root\_node.last = child\_node$ וגם $child\_node.parent = root\_node$, וכמובן גורמת לכך שכל שאר השדות של $root\_node$ יעודכנו בהתאם לכך שיש לו ילד חדש. סיבוכיות של $O(1)$.
- המתודה $postLinkFix()$ היא מתודה מופע, והיא בנראות $private$. למתודה זו אין דרישות קדם. מתודה זו, במידה ומתקיים $this \neq null$ וגם $this.last \neq null$, מעדכנת את השדה $this.last.next$ להיות $null$. נשים לב שאם $this = null$ או $this.last = null$, השדה $this.last.next$ לא קיים ולכן אין צורך לעדכן אותו. מתודה זו תיקרא רק אחרי המתודה $linkNodes(HeapNode\ node)$ (ובמימוש שלי, במתודה $meld(BinomialHeap\ heap2)$, עליה נדבר בהמשך, היא תיקרא רק אחרי קידום הצמתים $node\_this$ ו - $node\_heap2$, שיוצגו בהמשך, בהסבר על מתודה זו, בהתאם לצורך). סיבוכיות של $O(1)$.
- המתודה $smallerRank(HeapNode\ node1, HeapNode\ node2)$ היא מתודה סטטית, והיא בנראות $private$. למתודה זו אין דרישות קדם. מתודה זו בודקת לאיזה משני הצמתים יש דרגה נמוכה יותר (ומחזירה את צומת זה), ובמקרה שבו שני הצמתים הם $null$ היא גם מחזירה $null$. סיבוכיות של $O(1)$.
- המתודה $intoMelded(HeapNode[]\ melded, int\ mleded\_max\_index)$ היא מתודת מופע, והיא בנראות $private$. למתודה זו **קיימת דרישת קדם** לפיה $melded\_max\_index \geq 0$. מתודה זו מעדכנת את השדה $this.parent$ להיות $null$, ואז מעדכנת את האיבר באינדקס $melded\_\max\ \_index$ במערך $melded$ להיות $this$. במידה ומתקיים $melded\_\max\ \_index > 0$, סימן שהיה ב - $melded$ איבר שאינו $null$ (ראו הסבר של המתודה $meld(BinomialHeap\ heap2)$), ולכן היא מעדכנת את השדה $melded[melded\_max\_index - 1].next$ להיות $this$. סיבוכיות של $O(1)$.
- המתודה $shiftUp()$ היא מתודת מופע, והיא בנראות $private$. למתודה זו אין דרישות קדם. במידה ומתקיים $this.parent \neq null$, נסיק ש - $this$ הוא לא שורש של עץ בינומי, ובנוסף, במקרה זה, המתודה תאתחל $HeapNode\ current = this$ וגם $HeapNode\ parent = current.parent$. כעת, בתוך תנאי זה (שאנחנו לא בשורש של עץ בינומי), המתודה תעבור בעזרת לולאה על כל הצמתים בדרך בין $this$ לבין השורש של העץ הבינומי בו $this$ נמצא, ואם היא תגלה שקיים צומת בדרך זו עם מפתח שגדול

מהמפתח של $this$, היא תחליף בין השדות $item$ שלהם (וכמובן לכל $item$ שכזה היא תעדכן גם את השדה $item.node$). כך המתודה בעצם תיצור "הזזה" של הצמתים בעץ, כך שכל הצמתים בין $this$ לבין צומת מעליו בעץ הבינומי שבו $this$ נמצא, שהמפתחות שלהם קטנים מהמפתח של $this$, ישמרו על הסדר שלהם, אך ירדו למטה כפי שהוסבר בשיעור. אם נסמן ב – d את מספר הצמתים בדרך בין $this$ לבין השורש של העץ הבינומי שבו $this$ נמצא, שהמפתחות שלהם קטנים מהמפתח של $this$, אזי סיבוכיות של $d \cdot O(1) = O(d)$. במקרה הגרוע ביותר, $this$ יהיה הצומת בעומק הגדול ביותר בעץ הבינומי, והמפתח שלו יהיה המפתח המינימלי מבין כל המפתחות בדרך מ - $this$ אל שורש העץ הבינומי שבו $this$ נמצא. כלומר, אם נסמן ב – n_{this} את מספר הצמתים בעץ הבינומי שבו $this$ נמצא, אזי נקבל שישנם $\log_2(n_{this})$ צמתים בדרך בין $this$ לבין שורש העץ הבינומי שבו $this$ נמצא, ולכן סיבוכיות המתודה במקרה הגרוע ביותר תהיה $\log_2(n_{this}) \cdot O(1) = O(\log_2(n_{this}))$.

כעת נעבור למחלקה $\boldsymbol{BinomialHeap}$, אליה הוספתי את השדות $first$ ו - $trees\_num$. עבור אובייקט $heap$ מסוג $BinomialHeap$, השדה $first$ הינו מצביע לעץ הבינומי בעל הדרגה הנמוכה ביותר ב - $heap$, והשדה $trees\_num$ מייצג את כמות העצים הבינומיים שיש ב - $heap$. נגדיר אובייקט $heap$ מסוג $BinomialHeap$ להיות "ערמה", וכמו כן, נאמר שכל צומת החל מ - $heap.first$ ועד $heap.last$, כך שמגיעים אליהם על ידי קריאות $next$. מצומת בשרשרת שמתחילה ב - $heap.first$, היא "שורש של עץ בינומי בערמה", ולעיתים גם נתייחס אליהן בתור העצים הבינומיים בערמה.

נתחיל בלציין שעבור מתודות המופע $findMin()$, $size()$, $empty()$ ו - $numTrees()$, שהיו קיימות מראש בקובץ השלד, וכל מה שהן מבצעות זה קריאות לשדות המתאימים, ולכל היותר בדיקה לפיה השדה הרלוונטי הוא לא $null$, הסיבוכיות היא $O(1)$.

נמשיך בלנתח את המתודות שלא היו מראש בקובץ השלד, ושהוספתי אותן אליו בעצמי. נעיר רק שבמחלקה זו כל פעם שמופיע $this$, מדובר באובייקט בשם $this$ מהמחלקה $BinomialHeap$, כלומר $this$ הוא ערמה:

- המתודה $BinomialHeap()$ היא בנאי של המחלקה $BinomialHeap$, והיא בנראות $public$. למתודה זו אין דרישות קדם. כפי שציינתי גם בקובץ השלד, יכולתי להשאיר את מתודה זו ריקה מאחר ו - $Java$ הייתה מבצעת באופן ברירת מחדל את הבנייה המתוארת בה, אך הוספתי את השורות למען בהירות הקוד. המתודה מאתחלת את השדות $this.size$ ו - $this.trees\_num$ להיות 0, ואת השדות $this.last$, $this.min$ ו - $this.first$ להיות $null$. סיבוכיות של $O(1)$.
- המתודה $heapFromChildren(HeapNode\ node)$ היא מתודה סטטית, והיא בנראות $private$. למתודה זו אין דרישות קדם. מתודה זו מייצרת ומחזירה ערמה מהעצים הבינומיים שהשורשים שלהם הם הילדים של $node$. המתודה עוברת על כל הצמתים האלה ומעדכנת את השדה $parent$ שלהם להיות $null$. מאחר וישנם $node.rank$ ילדים של $node$, סיבוכיות המתודה היא $O(node.rank + 1)$. נשים לב כי אם נסמן ב - n_{node} את כמות הצאצאים של $node$ פלוס 1 (עבור $node$ עצמו), אזי יתקיים:
  $node.rank = \log_2(n_{node})$, ולכן ניתן לומר שסיבוכיות המתודה היא $O(\log_2(n_{node}) + 1)$.
- המתודה $heapWithoutATree(HeapNode\ node)$ היא מתודת מופע, והיא בנראות $private$. למתודה זו אין דרישות קדם. מתודה זו מייצרת ומחזירה ערמה מהעצים הבינומיים שנמצאים ב - $this$, מלבד העץ הבינומי ששורשו $node$. במידה ו – $node$ הוא

$null$, המתודה תחזיר את $this$, ובמידה ומתקיים $this.empty()$ או שמתקיים ($this.trees\_num = 1$ וגם $this.first = node$), המתודה תחזיר $new\ BinomialHeap()$. בשאר המקרים, המתודה תיצור ערמה חדשה בשם $result$, ותעדכן את השדות $result.size = this.size - \lceil 2^{node.rank} \rceil + 1$ וגם: $result.trees\_num = this.trees\_num - 1$ כמתואר, ואז תעבור על כל העצים הבינומיים שנמצאים ב - $this$, אם שורשם הוא $node$ היא תפסח עליהם, אחרת היא תבצע קישור בין הצמתים כך שישמר הסדר שהיה ב - $this$ (מלבד הקישור ל - $node$), ותבצע עדכונים לשאר השדות של $result$ בהתאם לצורך, ולבסוף גם תחזיר את $result$. סיבוכיות של $O(this.trees\_num + 1)$. אם נסמן ב - n_{this} את כמות הצמתים בכל הערמה (תוצאת כמות הצמתים בכל העצים הבינומיים), אזי: $this.trees\ num \leq \lceil \log_2(n_{this}) \rceil$, ולכן סיבוכיות המתודה היא $O(\lceil \log_2(n_{this}) \rceil)$.

- המתודה $setSelf(BinomialHeap\ heap)$ היא מתודת מופע, והיא בנראות $private$. למתודה זו אין דרישות קדם. המתודה מעדכנת את כל השדות של $this$ שיהיו זהים לשדות של $heap$. סיבוכיות של $O(1)$.

ונסיים בלנתח את המתודות שהיו קיימות מראש בקובץ השלד (מלבד אלו שכבר ניתחנו מראש), ונתחיל דווקא מהמתודה $meld(BinomialHeap\ heap2)$:

- המתודה $meld(BinomialHeap\ heap2)$ היא מתודת מופע, והיא בנראות $public$. למתודה זו אין דרישות קדם. אם $this.empty()$ וגם $\neg heap2.empty()$, המתודה מעדכנת את השדות של $this$ כך שיהיו זהים לשדות של $heap2$. אחרת, אם $\neg heap2.empty()$, המתודה מאתחלת $HeapNode\ node\_this = this.first$, $HeapNode\ node\_heap2 = heap2.first$, $HeapNode\ carry = null$, $int\ len = this.trees\_num + heap2.trees\_num + 1$, $HeapNode[]\ melded = new\ HeapNode[len]$, $int\ melded\_max\_index = 0$, וכל עוד מתקיים: $(node\_this \neq null\ \vee\ node\_heap2 \neq null)$, המתודה בודקת את המקרים הבאים (להבנה מעמיקה יותר, ראו נספח על המקרה הזה שמתחיל בעמוד 14):
  1. האם $node\_this \neq null$ וגם $node\_heap2 \neq null$:
     a. האם $carry \neq null$:
        א. האם $node\_this.rank = node\_heap.rank$:
           I. האם $node\_this.rank = carry.rank$, אם כן אז המתודה מבצעת את הקריאה: $carry.intoMelded(melded, melded\_max\_index)$, מעלה את הערך של $melded\_max\_index$ ב $- 1$, מעדכנת את $carry$ להיות התוצאה של הקריאה: $node\_this.linkNodes(node\_heap2)$, מקדמת את $node\_this$ ואת $node\_heap2$ להיות הצמתים שעליהם מצביעים השדות $next$ שלהם, ולבסוף מבצעת את הקריאה: $carry.postLinkFix()$.
           II. כאן, $node\_this.rank \neq carry.rank$. המתודה מבצעת את הקריאה: $carry.intoMelded(melded, melded\_max\_index)$, מעלה את הערך של $melded\_max\_index$ ב $- 1$, ולבסוף מעדכנת את $carry$ להיות $null$.
        ב. כאן $node\_this.rank \neq node\_heap2.rank$.

I. האם $HeapNode.smallerRank(node\_this, node\_heap2).rank$ שונה מ - $carry.rank$, אם כן אזי ההבדל בין הדרגה של $node\_this$ ו - $node\_heap2$ הנוכחיים לעומת קודמיהם הוא לפחות 2, שכן $carry$ נוצר רק על ידי המתודה $linkNodes(HeapNode\ node)$, ולכן דרגתו תמיד גבוהה ב $1 -$ מהדרגה של הצמתים שמאיחודן הוא נוצר. לכן, אם הדרגה הנמוכה מבין $node\_this.rank$ ו - $node\_heap2.rank$ אינה שווה לדרגה של $carry$, וכן $carry \neq null$, אזי הדרגה המינימלית הזו גדולה מהדרגה של carry. כאן המתודה מבצעת את הקריאה:
$carry.intoMelded(melded, melded\_max\_index)$, מעלה את הערך של $melded\_max\_index$ ב $1 -$, ולבסוף מעדכנת את $carry$ להיות $null$.

II. האם $HeapNode.smallerRank(node\_this, node\_heap2)$ שווה ל - $node\_this$, אם כן המתודה מעדכנת את $carry$ להיות התוצאה של הקריאה $\mathrm{carry.linkNodes(node\_this)}$, מעדכנת את $node\_this$ להיות הצומת עליו מצביע השדה $node\_this.next$, ולבסוף מבצעת את הקריאה:
$carry.postLinkFix()$.

III. כאן, $HeapNode.smallerRank(node\_this, node\_heap2)$ שווה ל - $node\_heap2$. במקרה זה המתודה מעדכנת את $carry$ להיות התוצאה של הקריאה $\mathrm{carry.linkNodes(node\_heap2)}$, מעדכנת את $node\_heap2$ להיות הצומת עליו מצביע השדה $node\_heap2.next$, ולבסוף מבצעת את הקריאה:
$carry.postLinkFix()$.

b. כאן $carry = null$:

א. האם $node\_this.rank = node\_heap2.rank$, אם כן המתודה מעדכנת את $carry$ להיות התוצאה של הקריאה:
$node\_this.linkNodes(node\_heap2)$, מעדכנת את $node\_this$ ואת $node\_heap2$ להיות הצמתים שעליהם מצביעים השדות $next$ שלהם, ולבסוף מבצעת את הקריאה $carry.postLInkFix()$.

ב. כאן $node\_this.rank \neq node\_heap2.rank$:

I. האם $HeapNode.smallerRank(node\_this, node\_heap2)$ שווה ל - $node\_this$, אם כן המתודה מבצעת את הקריאה:
$node\_this.intoMelded(melded, melded\_max\_index)$
ומעדכנת את $node\_this$ להיות הצומת עליו מבציע השדה $node\_this.next$.

II. כאן $HeapNode.smallerRank(node\_this, node\_heap2)$ שווה ל - $node\_heap2$. במקרה זה המתודה מבצעת את הקריאה:
$node\_heap2.intoMelded(melded, melded\_max\_index)$
ומעדכנת את $node\_heap2$ להיות הצומת עליו מצביע השדה $node\_heap2.next$.

2. האם $carry \neq null$:

a. האם $node\_this \neq null$ וגם $node\_this.rank = carry.rank$, אם כן המתודה מעדכנת את $carry$ להיות התוצאה של הקריאה:

$carry.linkNodes(node\_this)$, מעדכנת את $node\_this$ להיות הצומת עליו מצביע השדה $node\_this.next$, ולבסוף מבצעת את הקריאה: $carry.postLinkFix()$.

b. האם $node\_heap2 \neq null$ וגם $node\_heap2.rank = carry.rank$, אם כן המתודה מעדכנת את $carry$ להיות התוצאה של הקריאה: $carry.linkNodes(node\_heap2)$, מעדכנת את $node\_heap2$ להיות הצומת עליו מצביע השדה $node\_heap2.next$, ולבסוף מבצעת את הקריאה: $carry.postLinkFix()$.

c. כאן, אם $(node\_this \neq null \vee node\_heap2 \neq null)$ אזי $carry.rank$ הוא המינימלי מבין הדרגות של הצמתים שאינם $null$, ואחרת – $carry$ הוא היחיד שיש לו דרגה (כמובן שלא יכול להיות שיתקיים $node\_this = null$ וגם $node\_heap2 = null$, כי אז נצא מן הלולאה הזו, ולכן מדובר בחלק שלפחות אחד מהם הוא לא $null$, ולא בחלק של $carry$ הוא לא $null$, אבל הכנסתי כדי לפתח את כיוון המחשבה עד הסוף). במקרה זה המתודה מבצעת את הקריאה: $carry.intoMelded(melded, melded\_max\_index)$, מעלה את הערך של $melded\_max\_index$ ב -1, ולבסוף מעדכנת את $carry$ להיות $null$.

3. האם $node\_this \neq null$ המתודה מבצעת את הקריאה: $node\_this.intoMelded(melded, melded\_max\_index)$, מעלה את הערך של $melded\_\max\ \_index$ ב -1, ולבסוף מעדכנת את $node\_this$ להיות הצומת שעליו מצביע השדה $node\_this.next$.
4. כאן $node\_heap2 \neq null$ המתודה מבצעת את הקריאה:
5. $node\_heap2.intoMelded(melded, melded\_max\_index)$, מעלה את הערך של $melded\_\max\ \_index$ ב -1, ולבסוף מעדכנת את $node\_heap2$ להיות הצומת שעליו מצביע השדה $node\_heap2.next$.

בסוף הלולאה הזו, במידה ו - $carry \neq null$, המתודה מבצעת את הקריאה $carry.intoMelded(melded, mleded\_max\_index)$, ומעלה את הערך של $melded\_\max\ \_index$ ב -1.

לאחר מכן, המתודה מוסיפה ל - $this.size$ את $heap2.size$, מעדכנת את $this.last$ להיות $melded[melded\_max\_index - 1]$, מעדכנת את $this.min$ להיות תוצאת הקריאה $HeapNode.smallerKey(this.min, heap2.\min)$, מעדכנת את $this.first$ להיות $melded[0]$, מעדכנת את $this.trees\_num$ להיות $melded\_\max\ \_index$, ומעדכנת את השדה $this.last.next$ להיות $null$.

במידה ו - $heap2.empty()$, המתודה לא מבצעת כלום, שכן אין צמתים ב - $heap2$ שעלינו להוסיף ל - $this$.

נשים לב כי המקרה הגרוע הוא באמת כאשר אנחנו נכנסים לכל החלק של הפירוט הארוך והפרך, כלומר כאשר $this$ ו - $heap2$ אינן ריקות, ומעבר לכך – המקרה הגרוע ביותר יהיה כאשר אין שני עצים (אחד ב - $this$ ואחד ב - $heap2$) מאותה דרגה. במקרה זה יהיה לנו $this.trees\_num + heap2.trees\_num$ חזרות בלולאה, כל אחת בסיבוכיות של $O(1)$, לכן במקרה הגרוע ביותר סיבוכיות המתודה תהיה:

$O(this.trees\_num + heap2.trees\_num + 1)$. באופן שקול, אם נסמן ב - n_{this} את מספר הצמתים בערמה $this$, נסמן ב - n_{heap2} את מספר הצמתים בערמה $heap2$, ונסמן ב – n את $\max(n_{this}, n_{heap2})$. נשים לב כי מספר הכניסות ללולאה במקרה זה

יהיה לכל היותר $2 \cdot \lceil \log_2(n) \rceil$, כל אחת בסיבוכיות של $O(1)$, לכן סיבוכיות המתודה במקרה הגרוע ביותר היא $O(\lceil \log_2(n) \rceil + 1)$.

- המתודה $insert(int\ key, String\ info)$ היא מתודת מופע, והיא בנראות $public$. למתודה זו **קיימת דרישת קדם** לפיה $key > 0$. במידה ו- $this.empty()$, המתודה מייצרת צומת חדש $node$ ומעדכנת את השדות $this.last$, $this.min$ ו- $this.first$ שיצביע על $node$, ואת השדות $this.size$ ו- $this.trees\_num$ להיות 1. המתודה שומרת את $node.item$ בתור המשתנה $item$. אחרת ($this$ היא לא ערמה ריקה), המתודה יוצרת ערמה חדשה בשם $heap2$, קוראת לעצמה בעזרת $heap2$ בעזרת אותם הפרמטרים, שומרת את תוצאת קריאה זו בתור המשתנה $item$, ומבצעת את הקריאה $this.meld(heap2)$. לבסוף המתודה מחזירה את $item$. מאחר ומתקיים $heap2.trees\_num = 1$, נסיק כי סיבוכיות המתודה היא: $O(this.trees\_num + 1 + 1) = O(this.trees\_num + 1)$. אם נסמן ב $- n_{this}$ את מספר הצמתים בעץ (כולל השורש), נשים לב כי $this.trees\_num = \log_2(n_{this})$, ולכן נסיק כי סיבוכיות המתודה היא $O(\log_2(n_{this}) + 1)$.
- המתודה $deleteMin()$ היא מתודת מופע, והיא בנראות $public$. למתודה זו לא קיימות דרישות קדם. במידה ו- $this \neq null$ וגם $\neg this.empty()$, אם $this.size = 1$, נסיק כי ב - $this$ קיים רק צומת אחד, ולכן בהכרח צומת זה הוא $this.min$. לכן כשנרצה למחוק את המינימום של $this$, עלינו בעצם להפוך את $this$ לערמה ריקה. לכן המתודה מבצעת את הקריאה $this.setSelf(new\ BinomialHeap())$. אחרת (כאשר מתקיים $this.size \geq 2$), המתודה מבצעת את הקריאות:
$BinomialHeap\ heap\_from\_children = heapFromChildren(this.min)$,
וגם:
$$BinomialHeap\ heap\_without\_\min\ \_tree = \\ = this.heapWithoutATree(this.min)$$
לאחר מכן המתודה מייצרת ערמה בשם new_this. במידה ו- $heap\_from\_children$ לא קיימת (כלומר מקיימת $heap\_from\_children = null$) או ריקה, אזי יתקיים $new\_this = heap\_without\_\min\ \_tree$. באופן סימטרי גם כאשר $heap\_without\_\min\ \_tree$ לא קיימת (כלומר מקיימת $heap\_without\_\min\ \_tree = null$) או ריקה. במידה ושתי הערמות הללו קיימות ולא ריקות, המתודה מיישמת $new\_this = heap\_without\_\min\ \_tree$ ומבצעת את הקריאה $new\_this.meld(heap\_from\_children)$. במקרה הגרוע ביותר יהיו בערמה לפחות 2 עצים בינומיים, והשורש של העץ הגדול ביותר בערמה יהיה $this.min$. לכן, אם נסמן ב $- n_{this}$ את כמות הצמתים שיש בערמה כולה, נסיק כי מתקיים:
$this.trees\_num = O(\lceil \log_2(n_{this}) \rceil) \wedge this.min.rank = O(\lceil \log_2(n_{this}) \rceil)$
ולכן יצירת שתי הערמות $heap\_from\_children$ ו- $heap\_without\_\min\ \_tree$ היא בסיבוכיות של $O(\lceil \log_2(n_{this}) \rceil + 1)$. ולכן בפרט, סיבוכיות המתודה במקרה הגרוע ביותר היא $O(\lceil \log_2(n_{this}) \rceil + 1)$.
- המתודה $decreaseKey(HeapItem\ item, int\ diff)$ היא מתודת מופע, והיא בנראות $public$. למתודה זו **יש דרישת קדם** לפיה $0 < diff < item.key$. מתודה זו מעדכנת את $item.key$ להיות $item.key - diff$, ולאחר מכן מבצעת את הקריאה $item.node.shiftUp()$. סיבוכיות של $O(1)$.
- המתודה $delete(HeapItem\ item)$ היא מתודת מופע, והיא בנראות $private$. אם $this.min \neq item.node$, המתודה מבצעת את הקריאה $this.decreaseKey(item, item.key - this.min.item.key)$ ואז מעדכנת את

השדה $this.min$ להצביע על $item.node$. לבסוף, בין אם המתודה נכנסה למקרה זה שתואר ובין אם לאו, המתודה מבצעת את הקריאה $this.deleteMin()$. לכן, אם נסמן ב – n את כמות הצמתים שיש בערמה כולה, נסיק כי סיבוכיות המתודה במקרה הגרוע ביותר היא $O(\lceil\log_2(n)\rceil + 1)$.

נספח – מה עושים כששתי הערמות לא ריקות במתודה $meld(BinomialHeap\ heap2)$?

במקרה שבו שתי הערמות לא ריקות עלינו לאחד אותן. כשאנחנו מאחדים את הערמות, אנו עוברים על כל העצים הבינומיים בכל אחת משתי הערמות ($this$ ו - $heap2$), החל מהעצים בדרגה הנמוכה ביותר בכל ערמה, ועד העצים בדרגה הגבוהה ביותר בכל ערמה, ובודקים האם צריך לאחד אותם. איחודם יעשה על ידי האלגוריתם הבא:

1. נאתחל רשימה של העצים הבינומיים שיהיו בערמה $this$ לאחר איחודה עם הערמה $heap2$. לצורך הקלה בהסבר האלגוריתם, לרשימה זו נקרא $melded$. שימו לב שבקוד רשימה זו מיוצגת על ידי מערך, ושכמות האיברים במערך זה שהם עצים בינומיים שנוצרו כתוצאה מאיחוד הערמה $this$ והערמה $heap2$ נקראת $melded\_\max\ \_index$ בקוד.

2. לצורך הקלה בהסבר האלגוריתם, לעץ הבינומי שעליו אנו מסתכלים כרגע בערמה $this$ נקרא $T_{this}.current$, ולעץ הבינומי שעליו אנו מסתכלים כרגע בערמה $heap2$ נקרא $T_{heap2}.current$. לעץ הבינומי שעליו הסתכלנו בערמה $this$ לפני $T_{this}.current$ נקרא $T_{this}.previous$, ולעץ הבינומי שעליו הסתכלנו בערמה $heap2$ לפני $T_{heap2}.current$ נקרא $T_{heap2}.previous$. שימו לב שבקוד $T_{this}.current$ נקרא $node\_this$, ו - $T_{heap2}.current$ נקרא $node\_heap2$.

3. לצורך הקלה בהסבר האלגוריתם, אם קיים עץ בינומי שאנו זוכרים מאיחוד שני עצים בינומיים T_1 ו - T_2 כלשהם, נקרא לעץ זה $T_{carry}.previous$, ואם אנו מאחדים שני עצים בינומיים T_1 ו - T_2 כלשהם כרגע, נקרא לעץ שהוא תוצאת האיחוד $T_{carry}.current$. שימו לב שבקוד אין הבחנה בין $T_{carry}.previous$ לבין $T_{carry}.current$, והם שניהם נקראים $carry$. בנוסף, שימו לב שמתקיים $T_{carry}.rank = T_1.rank + 1 = T_2.rank + 1$, בין אם מדובר ב - $T_{carry}.previous$ או ב- $T_{carry}.current$ (שכן מאחדים רק עצים שדרגתם זהה).

4. לצורך הקלה בהסבר האלגוריתם, במידה ו - $T_{this}.current = null$ או $T_{heap2}.current \neq null$, אך לא שניהם, לעץ הבינומי הקיים (האחד שאינו $null$), נקרא $node$. שימו לב שבקוד אין אבחנה זו, במקום זאת ישנם תנאים שבודקים מי הוא העץ שקיים (האחד שאינו $null$). במקרה זה נאמר כי $\min(T_{this}.current.rank, T_{heap2}.current.rank) = node.rank$, כי ל - $null$ אין דרגה ולכן הוא לא יכול לתרום למציאת המינימום.

5. אם מתקיימים התנאים הבאים:
   a. $T_{this}.current \neq null \land T_{heap2}.current \neq null$.
   b. $T_{this}.current.rank = T_{heap2}.current.rank$.
   c. $T_{carry}.previous$ קיים (ולכן מקיים $T_{carry}.previous \neq null$).

d. $T_{carry}.previous.rank = T_{this}.current.rank$ (ולכן גם מתקיים: $T_{carry}.previous.rank = T_{heap2}.current.rank$).

אזי נכניס את $T_{carry}.previous$ ל - $melded$ בתור האיבר הבא, ונזכור את $T_{carry}.current$, שהוא תוצאת האיחוד של $T_{this}.current$ ושל $T_{heap2}.current$.
שימו לב שבמקרה זה יתקיים ש - $T_{carry}.current.rank$ יהיה שווה ל: $\min(T_{this}.current.rank, T_{heap2}.current.rank) + 1$. לאחר מכן נחזור על התהליך עם $T_{this}.current.next$ ועם $T_{heap2}.current.next$.

6. אם מתקיימים התנאים הבאים:
   a. $T_{this}.current \neq null \ \wedge \ T_{heap2}.current \neq null$.
   b. $T_{this}.current.rank = T_{heap2}.current.rank$.
   c. $T_{carry}.previous$ קיים (ולכן מקיים $T_{carry}.previous \neq null$).
   d. $T_{carry}.previous.rank \neq T_{this}.current.rank$ (ולכן מתקיים גם: $T_{carry}.previous.rank \neq T_{heap2}.current.rank$).

אזי נסיק כי מתקיים:

$$\min(T_{this}.previous.rank, T_{heap2}.previous.rank) + 1 \neq$$
$$\neq T_{this}.current.rank \ \left(= T_{heap2}.current.rank\right)$$

מאחר ומתקיים $T_{this}.current.rank \geq T_{this}.previous.rank$, $T_{heap2}.current.rank \geq T_{heap2}.previous.rank$, וכן $T_{this}.current.rank = T_{heap2}.current.rank$, נסיק כי מתקיים:

$$T_{this}.current.rank = T_{heap2}.current.rank \geq$$
$$\geq \min(T_{this}.previous.rank, T_{heap2}.previous.rank) + 2 =$$
$$= T_{carry}.previous.rank + 1$$

ולכן נצטרך להכניס את $T_{carry}.previous$ ל - $melded$ בתור האיבר הבא, ולעדכן את $T_{carry}.previous$ להיות $null$ לאחר מכן (כך נמנע מלחזור על מקרה זה באופן אינסופי).
לאחר מכן נחזור על התהליך עם $T_{this}.current$ ועם $T_{heap2}.current$.

7. אם מתקיימים התנאים הבאים:
   a. $T_{this}.current \neq null \ \wedge \ T_{heap2}.current \neq null$.
   b. $T_{this}.current.rank \neq T_{heap2}.current.rank$.
   c. $T_{carry}.previous$ קיים (ולכן מקיים $T_{carry}.previous \neq null$).
   d. $T_{carry}.previous.rank$ **שונה מ**: $\min(T_{this}.current.rank, T_{heap2}.current.rank)$.

אזי נסיק כי מתקיים:

$$\min(T_{this}.previous.rank, T_{heap2}.previous.rank) + 1 \neq$$
$$\neq \min(T_{this}.current.rank, T_{heap}.current.rank)$$

מאחר ומתקיים $T_{this}.current.rank \geq T_{this}.previous.next$, $T_{heap2}.current.rank \geq T_{heap2}.previous.rank$, וכן (במקרה זה, ויבוא הסבר ישר אחרי הטענה הזו כמובן):

$$T_{this}.current.rank + T_{heap2}.current.rank >$$
$$> T_{this}.previous.rank + T_{heap2}.previous.rank$$

כי יצרנו את $T_{carry}.previous$, והוא לא $null$, לכן בחזרה הקודמת על התהליך איחדנו שני עצים, לכן או שביצענו את סעיף 5, או שביצענו את סעיף 8, או את סעיף 9, או את

סעיף 11. כלומר, הדרגה של לפחות אחד מהעצים $T_{this}.current$ או $T_{heap2}.current$ גדולה בלפחות 1 מהדרגה של העץ הקודם עליו הסתכלנו באותה הערמה. מכל זאת נסיק כי מתקיים:

$$T_{this}.current.rank + T_{heap2}.current.rank \geq$$
$$\geq 2 \cdot \min\left(T_{this}.current.rank, T_{heap2}.current.rank\right) \geq$$
$$\geq 2 \cdot \left(\min\left(T_{this}.previous.rank, T_{heap2}.previous.rank\right) + 1\right)$$

ולכן בפרט:

$$\min\left(T_{this}.current.rank, T_{heap2}.current.rank\right) \geq$$
$$\geq \min\left(T_{this}.previous.rank, T_{heap2}.previous.rank\right) + 1$$

אך מאחר וכבר ראינו כי:

$$\min\left(T_{this}.previous.rank, T_{heap2}.previous.rank\right) + 1 \neq$$
$$\neq \min\left(T_{this}.current.rank, T_{heap2}.current.rank\right)$$

נסיק כי:

$$\min\left(T_{this}.current.rank, T_{heap2}.current.rank\right) >$$
$$> \min\left(T_{this}.previous.rank, T_{heap2}.previous.rank\right) + 1$$

כלומר:

$$T_{carry}.previous.rank < \min\left(T_{this}.current.rank, T_{heap2}.current.rank\right)$$

ולכן נצטרך להכניס את $T_{carry}.previous$ ל - $melded$ בתור האיבר הבא, ולעדכן את $T_{carry}.previous$ להיות $null$ לאחר מכן (כך נמנע מלחזור על מקרה זה באופן אינסופי). לאחר מכן נחזור על התהליך עם $T_{this}.current$ ועם $T_{heap2}.current$.

8. אם מתקיימים התנאים הבאים:
   a. $T_{this}.current \neq null \ \wedge \ T_{heap2}.current \neq null$.
   b. $T_{this}.current.rank \neq T_{heap2}.current.rank$.
   c. $T_{carry}.previous$ קיים (ולכן מקיים $T_{carry}.previous \neq null$).
   d. $T_{carry}.previous.rank$ **שווה ל**: $\min\left(T_{this}.current.rank, T_{heap2}.current.rank\right)$.

אזי נזכור את $T_{carry}.current$ שהוא תוצאת האיחוד של $T_{carry}.previous$ עם העץ הבינומי עם הדרגה הנמוכה מבין $T_{this}.current$ ו - $T_{heap2}.current$. שימו לב שגם במקרה זה יתקיים ש - $T_{carry}.current.rank$ יהיה שווה ל: $\min\left(T_{this}.current.rank, T_{heap2}.current.rank\right) + 1$. לאחר מכן, במידה ו - $T_{this}.current$ היה העץ הבינומי עם הדרגה הנמוכה מבין $T_{this}.current$ ו - $T\_heap2.current$, נחזור על התהליך עם $T_{this}.current.next$ ו - $T_{heap2}.current$. אחרת, נחזור על התהליך עם $T_{this}.current$ ו - $T_{heap2}.current.next$.

9. אם מתקיימים התנאים הבאים:
   a. $T_{this}.current \neq null \ \wedge \ T_{heap2}.current \neq null$.
   b. $T_{this}.current.rank = T_{heap2}.current.rank$.
   c. $T_{carry}.previous$ **לא** קיים (ולכן מקיים $T_{carry}.previous = null$).

אזי נזכור את $T_{carry}.current$ שהוא תוצאת האיחוד של $T_{this}.current$ ושל $T_{heap2}.current$. שימו לב שגם במקרה זה יתקיים ש - $T_{carry}.current.rank$ יהיה

שווה ל - 1 + $\min(T_{this}.current.rank, T_{heap2}.current.rank)$. לאחר מכן נחזור על התהליך עם $T_{this}.current.next$ ועם $T_{heap2}.current.next$.

10. אם מתקיימים התנאים הבאים:
    a. $T_{this}.current \neq null \wedge T_{heap2}.current \neq null$.
    b. $T_{this}.current.rank \neq T_{heap2}.current.rank$.
    c. $T_{carry}.previous$ **לא** קיים (ולכן מקיים $T_{carry}.previous = null$).

אזי, במידה ו - $T_{this}.current$ הוא העץ הבינומי עם הדרגה הנמוכה מבין $T_{this}.current$ ו - $T_{heap2}.current$, נכניס את $T_{this}.current$ ל - $melded$ בתור האיבר הבא, ונחזור על התהליך עם $T_{this}.current.next$ ו - $T_{heap2}.current$. אחרת, נכניס את $T_{heap2}.current$ ל - $melded$ בתור האיבר הבא, ונחזור על התהליך עם $T_{this}.current$ ועם $T_{heap2}.current.next$.

11. אם מתקיימים התנאים הבאים:
    a. $T_{this}.current \neq null \vee T_{heap2}.current \neq null$.
    b. $\neg(T_{this}.current = null \wedge T_{heap2}.current = null)$.
    c. $T_{carry}.previous$ קיים (ולכן מקיים $T_{carry}.previous \neq null$).
    d. $T_{carry}.previous.rank = node.rank$.

אזי נזכור את $T_{carry}.current$ שהוא תוצאת האיחוד של $T_{carry}.previous$ ושל $node$.
שימו לב שגם במקרה זה $T_{carry}.current.rank$ שווה ל:
$\max(T_{this}.current.rank, T_{heap2}.current.rank) + 1$. לאחר מכן נחזור על התהליך עם $node.next$ ועם $null$.

12. אם מתקיימים התנאים הבאים:
    a. $T_{this}.current \neq null \vee T_{heap2}.current \neq null$.
    b. $\neg(T_{this}.current = null \wedge T_{heap2}.current = null)$.
    c. $T_{carry}.previous$ קיים (ולכן מקיים $T_{carry}.previous \neq null$).
    d. $T_{carry}.previous.rank \neq node.rank$.

אזי נסיק כי מתקיים:

$$\min(T_{this}.previous.rank, T_{heap2}.previous.rank) + 1 \neq node.rank \quad (= \\ = \min(T_{this}.current.rank, T_{heap2}.current.rank))$$

מאחר ומתקיים $node.rank \geq T_{this}.previous.rank$, וכן $node.rank \geq T_{heap2}.previous.rank$, וכן (במקרה זה, ויבוא הסבר ישר אחרי הטענה הזו כמובן):

$$2 \cdot node.rank \geq T_{this}.previous.rank + T_{heap2}.previous.rank$$

שכן, אם הגענו למצב שבו $T_{this}.current = null \vee T_{heap2}.current = null$, כך שלפחות אחד מהם אינו $null$, אזי בוודאות ש - $node.rank$ לפחות שווה ל:
$\max(T_{this}.previous.rank, T_{heap2}.previous.rank)$, וזה יקרה במקרה שבו $|T_{this}.previous.rank - T_{heap2}.previous.rank| > 0$ ושהשדה $next$ של העץ בעל הדרגה המינימלית מבין $T_{this}.previous$ ו - $T_{heap2}.previous$ היה $null$. לכן, אם קיים $T_{carry}.previous$, אזי בחזרה הקודמת על התהליך ביצענו איחוד של צמתים, ולכן אם $T_{carry}.previous.rank \neq node.rank$, נסיק כי לאחר איחוד הצמתים אנו כעת

מסתכלים על צומת בדרגה גדולה בלפחות 2 מהדרגה של הצומת הקודמת שעליה הסתכלנו, ולכן $T_{carry}.previous.rank < node.rank$.

לכן נצטרך להכניס את $T_{carry}.previous$ ל $- melded$ בתור האיבר הבא, ואז לעדכן את $T_{carry}.previous$ להיות $null$ (כך נמנע מלחזור על המקרה הזה באופן אינסופי). לאחר מכן נחזור על התהליך עם $node$ ועם $null$.

13. אם מתקיימים התנאים הבאים:
    a. $T_{this}.current \neq null \lor T_{heap2}.current \neq null$.
    b. $\neg(T_{this}.current = null \land T_{heap2}.current = null)$
    c. $T_{carry}.previous$ **לא** קיים (ולכן מקיים $T_{carry}.previous = null$).

    אזי נכניס את $node$ ל - $melded$ בתור האיבר הבא. לאחר מכן נחזור על התהליך עם $node.next$ ועם $null$.

14. אם מתקיימים התנאים הבאים:
    a. $T_{this}.current = null \land T_{heap2}.current = null$.
    b. $T_{carry}.previous$ קיים (ולכן מקיים $T_{carry}.prvious \neq null$).

    אזי נכניס את $T_{carry}.previous$ ל - $melded$ בתור האיבר הבא.

15. אחרי כל זה, נבצע את העדכונים הבאים:
    a. $this.size \leftarrow this.size + heap2.size$
    b. $this.last \leftarrow melded[melded.length - 1]$
    c. $this.min \leftarrow this.\min if\ this.min.item.key < heap2.min.item.key$
    $heap2.min\ otherwise$
    d. $this.first \leftarrow melded[0]$
    e. $this.trees\_num \leftarrow melded.length$
    f. $this.last.next \leftarrow null$

בסוף האלגוריתם הזה בעצם איחדנו את שתי הערמות $this$ ו - $heap2$, על ידי איחוד העצים הבינומיים בהן בעלי אותה דרגה, והשארת העצים הבינומיים האחרים בהן, בדיוק כפי שתואר בכיתה שיש לעשות (כאן פשוט יש טיפול בכל מקרה אפשרי)