עמרי יקיר 318867199
מאיה רייטן 209085711

מבני נתונים - פרויקט מספר 2- ערימת פיבונאצ'י

name: Omri Yakir, username: omriyakir, id: 318867199
name: Maya Raytan, username: mayaraytan, id: 209085711

חלק מעשי - תיעוד פונקציות

המחלקות:
FibonacciHeap:

- **private static int LINKS** - משתנה סטטי שסופר את כמות הלינקים הכוללת.
- **private static int CUTS** – משתנה סטטי שסופר את כמות החיתוכים הכוללת.
- **private HeapNode first** – מצביע לשורש העץ הראשון.
- **private HeapNode min** – מצביע לצומת המינימאלי בערימה.
- **private int trees** – מספר העצים בערימה.
- **private int marked** – מספר הצמתים המסומנים בערימה.
- **private int size** – מספר הצמתים בערימה.

HeapNode:

- **public int key** – מפתח הצומת.
- **private int rank** – דרגת הצומת.
- **private boolean mark** – האם הצומת מסומן.
- **private HeapNode child** – מצביע לבן הראשון של הצומת.
- **private HeapNode next** – מצביע לצומת הבא.
- **private HeapNode prev** – מצביע לצומת הקודם.
- **private HeapNode parent** – מצביע להורה של הצומת.
- **private HeapNode pointer** – מצביע לצומת אחר (משמש את הפונקציה kmin).

פעולה	תיאור	סיבוכיות זמן
Public Boolean isEmpty()	הפונקציה מחזירה אמת אם הערימה ריקה (ערך ה- size הוא 0) אחרת מחזירה שקר.	$O(1)$
Public HeapNode insert(int key)	הפונקציה מאתחלת צומת חדש x עם המפתח key. הפונקציה מבצעת קריאה לפונקציית העזר add_tree_to_first עם x, מעלה את מספר העצים של הערימה ב-1 ואת גודל הערימה ב-1. לסיום הפונקציה מחזירה את x.	$O(1)$
private void add_tree_to_first(HeapNode x)	הפונקציה מכניסה את x לראש הערימה. במידה והערימה ריקה (ערך הsize הוא 0) שדה min יצביע על הצומת החדשה, אחרת min יצביע על הצומת בעלת המפתח המינימלי מבין min והצומת החדשה.	$O(1)$
public void deleteMin()	אם הערימה שלנו ריקה- לא נבצע שינוי. אחרת- נוריד את המרקורים של כל הבנים של הצומת המינימאלי (במידה והם קיימים) ונעדכן את מספר המרקורים הכולל של הערימה. נוסיף את מספר הבנים של הצומת המינימאלי פחות אחד (פחות העץ של המינימום) למספר העצים הכולל של הערימה. נחלק למקרים:	עלות הסיבוכיות הכוללת נקבעת לפי: • מעבר על הבנים של צומת המינימום. מספר הבנים הוא כדרגת המינימום = $O(logn)$ • שינויי מצביעים בעלות קבועה.

	• גודל הערימה הוא 1: נעדכן את ערך ה-first,min ל-null. אחרת: • נחלק לעוד מקרים: ○ אם לאיבר המינימאלי שנמחק אין ילדים, נעדכן את first להיות האיבר הבא אחרי המינימאלי. נבצע חיבורים בין הקודם לעוקב של המינימאלי. ○ אם לאיבר הינימאלי יש ילדים: אם למינימום אין אחים, נעדכן את first להיות אחד הילדים של min. אחרת (למינימום יש אחים ויש ילדים), נחבר את הילדים של המינימום בין האח הקודם והעוקב של מינימום. כעת נוריד את גודל הערימה ב-1. ונקרא לפונקציית העזר consolidation (שמבצעת עדכון למינימום החדש בערימה בנוסף לאיחוד העצים).	• סיבוכיות consolidation = $amortized\ cost = O(logn)$, $wc = O(n)$ לכן סיבוכיות זמן הריצה היא: $amortized\ cost = O(logn)$ $wc = O(n)$
private void consolidation()	כל עוד הערימה לא ריקה, נאתחל מערך של צמתים בגודל *1.5*log(n)* (מערך של סלים). נעבור על העצים בערימה לפי הסדר שלהם. נכניס כל עץ למערך במיקום של הדרגה שלו. כל עוד המיקום של הדרגה שלו לא ריק (קיים שם עץ באותה הדרגה) נאחד ביניהם באמצעות פונקציית העזר *link* וננסה להכניסם לתא הבא (אם נדרש נבצע לינק נוסף) עד שנגיע לתא ריק בדרגתם החדשה. אחרי שהכנסנו את כל העצים, נאפס את ערך ה-*first* ונכניס את העצים לערימה לפי סדר הדרגות שלהם במערך הסלים. בהכנסה אנו נשתמש בפונקציית העזר *add_tree_to_start* שמעדכנת גם את ערך המינימום.	הפונקציה ממומשת על פי האלגוריתם שראינו בכיתה. חישבנו את זמן ה-amortized של הפעולה בעזרת פונקציית פוטנציאל שהיא מספר העצים בערימה. ראינו כי: $amortized\ cost = O(logn)$ ובנוסף ראינו: $wc = O(n)$
private HeapNode link(HeapNode x , HeapNode y)	הפונקציה מקבלת שני עצים בעלי דרגה זהה. הפונקציה מחברת את העצים כך שהעץ שהמפתח המינימאלי שלו קטן יותר יכיל את העץ השני בתור בן. בסוף החיבורים הפונקצייה תעדכן את דרגת העץ המשותף ב-1, תוריד את כמות העצים הכוללת בערימה ב-1, ותעלה את כמות הלינקים הכוללת בערימה ב-1. לסיום הפונקציה תחזיר את העץ המשותף.	$O(1)$ שינוי מצביעים ושדות בעלות קבועה.
public HeapNode findMin()	הפונקציה מחזירה את שדה ה-min של הערימה	$O(1)$ גישה לשדות בעלות קבועה.
public void meld (FibonacciHeap heap2)	הפונקציה מחברת את הערימה heap2 אל הערימה הקיימת. הפוקנציה מטפלת במקרים החריגים בהם הערימה ריקה או heap2 ריקה. הפונקציה עושה את החיבורים הדרושים לחיבור הערימות כך שמיקום איברי הערימה החדשה הם אחרי איברי הערימה הקיימת סיבוכיות O(1). הפונקציה מחליפה את מצביע הmin במידת הצורך O(1), ולבסוף מוסיפה את גודל heap2, את מספר העצים שלה ואת מספר הצמתים	$O(1)$ שינוי מצביעים ושדות בעלות קבועה.

	הממורקרים שבה לסכומים trees size וmarked של הערימה הקיימת בהתאמה.	
public int size()	הפונקציה מחזירה את ערך השדה size , מספר איברי הערימה.	$O(1)$
public int[] countersRep()	אם הערימה המקורית ריקה- הפונקציה תחזיר מערך ריק. אחרת: הפונקציה מאתחלת מערך באורך 1.5*logn כיוון שראינו בכיתה שבערימת פיבונאצ'י עם n צמתים, דרגת העץ המקסימלי תהייה לכל היותר 1.4404*logn. לאחר מכן הפונקציה עוברת על כל שורשי העצים של הערימה (מתחילה מ- first ומסיימת כשמגיעה שוב אל first) ומוסיפה 1 לאינדקס במערך ששווה ל-rank של השורש. הפונקציה תבדוק מה האורך האמיתי של המערך- כלומר מה האינדקס האחרון בו איבר ששונה מאפס ותחזיר את המערך ללא איברים מיותרים באמצעות פונקציות העזר get_real_length, copyArrayToSmallerLength	Wc: $O(n)$ במקרה הגרוע לערימה יש n בנים מדרגה 0 כל אחד. נצטרך לעבור על כולם.
private int get_real_length(int[] arr)	הפונקציה מחזירה את האינדקס של האיבר האחרון ברשימה ששונה מאפס.	$O(n)$. במקרה בו הרשימה מכילה רק אפסים יתבצע מעבר על הרשימה כולה.
private int[] copyArrayToSmallerLength (int[] arr, int len)	הפונקציה מקבלת רשימה ואורך ומחזירה רשימה חדשה שאיבריה באינדקסים שקטנים שווים לאורך שהוכנס.	$O(n)$. במקרה הגרוע הקלט len שווה לאורך רשימת הקלט.
public void delete(HeapNode x)	הפונקציה מוחקת את הצומת x מהרשימה: הפונקציה מבצעת decreaseKey(x,min.key-1) (סיבוכיות WC O(n) ו- O(1) אמורטייזד), ובכך מורידה את ערך המפתח של x להיות המינימלי בערימה. ואחרי מבצעת deleteMin (סיבוכיות WC O(n) ו- O(logn) אמורטייזד) ובכך מוחקת את x מהרשימה. ערכי trees , marked , size , min מתעדכנים בפונקציות decreaseKey וdeleteMin.	$wc: O(n+n) = O(n)$ $Amortize: O(1+logn) = O(logn)$
public void decreaseKey(HeapNode x, int delta)	הפונקציה תוריד delta מהמפתח של x. אם המפתח החדש של x קטן מהמפתח של המינימום של הערימה- נעדכן את המינימום של הערימה להיות x. כל ל-x יש אבא והמפתח שלו גדול משל x, נבצע חיתוכים בעזרת פונקציית העזר cascading_cut. נקדם את x ואת האבא של x להורים שלהם.	$wc: O(logn)$ במקרה הגרוע, כל ההורים במסלול מצומת x אל השורש של העץ מסומנים והמפתח שלהם גדול משלx לאחר השינוי. לכן נצטרך לעלות עד השורש ולבצע $O(logn)$ פעולות cut בעלות קבועה לכל אחת. $amortized: O(1)$ חישבנו את זמן ה-amortized של הפעולה בעזרת פונקציית פוטנציאל שהיא מספר העצים הצמתים המסומנים.
private void cascading_cut(HeapNode x ,HeapNode y)	הפונקציה מבצעת חיתוך באמצעות פונקציית העזר cut ומוסיפה את העץ שנחתך אל תחילת העצים (first). הפונקציה מעלה ב-1 את מספר העצים הכולל של הערימה. הפונקציה תמשיך בחיתוכים כל עוד האבא של הצומת שנחתך מסומן.	$wc: O(logn)$ במקרה הגרוע, כל ההורים במסלול מצומת x אל השורש של העץ מסומנים מראש. לכן נצטרך לעלות עד השורש ולבצע $O(logn)$ פעולות cut בעלות קבועה לכל אחת. $amortized: O(1)$ חישבנו את זמן ה-amortized של הפעולה בעזרת פונקציית פוטנציאל שהיא מספר העצים הצמתים המסומנים.

private void cut(HeapNode x,HeapNode y)	הפונקציה בודקת את x מסומן, אם כן תוריד ב-1 אם מספר הצמתים המסומנים בערימה. הפונקציה מגדירה את x כלא מסומן כיוון שהוא יהיה שורש של עץ חדש ושורשים אינם מסומנים. הפונקציה מורידה 1 מה-rank של y כיוון שהורדנו לו את הבן x. אם y אינו שורש ואינו מסומן- נסמנו. אם ל-y לא היו בנים נוספים מלבד x, נגדיר את y.child = null, אחרת נחבר בין הבנים של y כך שיעקפו את x שמחקנו. לסיום נעלה את מספר החיתוכים הכולל של הערימה ב-1.	$O(1)$ עלות קבועה כיוון שבפונקציה זו ניגשים לשדות ומבצעים שינויי מצביעים בלבד.
public int potential()	הפונקציה מחזירה את הפוטנציאל של הערימה: הסכום של ערך השדה trees ופעמיים ערך השדה marked.	$O(1)$ גישה לשדות בעלות קבועה
public static int totalLinks()	הפוקנציה מחזירה את LINKS , המשתנה הסטטי שסוכם את כל פעולות ה- link מהיווצרותה של המחלקה FibonacciHeap.	$O(1)$ גישה לשדות בעלות קבועה
public static int totalCuts()	הפוקנציה מחזירה את CUTS , המשתנה הסטטי שסוכם את כל פעולות ה- cut מהיווצרותה של המחלקה FibonacciHeap.	$O(1)$ גישה לשדות בעלות קבועה
public static int[] kMin(FibonacciHeap H, int k)	הפונקציה מאתחלת מערך באורך k בו יישמרו k המפתחות המינימאליים. הפונקציה מייצרת ערימת עזר ומכניסה אליה את האיבר המינימאלי בערימה המקורית. הפונקציה מאתחלת לולאת for מ-0 עד k-1 כולל, כך שעבור כל i תתבצע מחיקה של האיבר המינימאלי בערימת העזר והכנסת ערך המפתח שלו למערך הסופי. לאחר מכאן ילדיו של המפתח המינימאלי שמחקנו ייכנסו לערימת העזר.	ראינו בכיתה כי הדרגה המקסימלית של עץ בערימה בינומית היא *log(n)* עבור ערימה בעלת *n* צמתים. כלומר $O(logn)$. לכן $deg(H) = O(logn)$. עבור כל איטרציה מתקיים כי מספר העצים בערימה אחרי פעולת *delete min* הוא $O\big(deg(H)\big)$ ומספר האיברים שנוסיף לערימת העזר (בעלות קבועה להכנסה) הוא כמובן גם $O\big(deg(H)\big)$. לכן בפעולת *delete min* הבאה תעלה כמספר העצים בערימת העזר = $O\big(deg(H) + deg(H)\big) = O\big(deg(H)\big)$. סה"כ *k* איטרציות בעלות $O\big(deg(H)\big)$ לאיטרציה ולכן הסיבוכיות הכוללת היא: $O\big(k * deg(H)\big)$

חלק ניסויי/תיאורטי

<u>שאלה 1:</u>

א. זמן הריצה של סדרת הפעולות:

a. <u>עלות insert:</u>

נבצע m+1 הכנסות בעלות קבועה כל אחד. סה"כ $O(m)$

b. <u>עלות delete-min:</u>

לפי עלות פעולת ה-consolidation:

$O(t_0 + logm) = O(m + logm) = O(m)$ כאשר t_0 הוא מספר העצים לפני שרשור הבנים של צומת המינימום (0 בנים).

- נשים לב כי בעת ביצוע consolidation כחלק ממחיקת המינימום, נחבר כל מספר זוגי לאי זוגי העוקב לו כך שיתחברו לעץ בינומי מדרגה 1. כלומר כל צומת בעל מפתח אי זוגי הוא בן של צומת עם מפתח הזוגי הקודם לו.
- לאחר ביצוע consolidation בערימה יהיה עץ יחיד, עץ בינומיאלי מדרגה logm.

c. <u>עלות decrease-key:</u>

נבצע decrease-key על מפתחות אי זוגיים בלבד:

$\left(\forall i: i = \log_2 m, .., 1. \quad m - 2^i + 1\right)$ ולכן ההורה של כל מפתח שנבצע עליו decrease key יהיה בעל מפתח $m - 2^i$ והוא ייחודי כפי שהסברנו קודם.

בכל פעולת decrease-key נוריד את ערך המפתח עליו מתבצעת הפעולה מתחת לערך המינימאלי בעץ: $m - 2^i + 1 - (m+1) = -2^i < 0$ כאשר 0 הוא המפתח המינימאלי בעץ. לכן נקבל צומת שמפר את תנאי הערימה ונצטרך לבצע פעולת cut.

בפעולת ה-cut נמרקר את אב הצומת (למעט חיתוך האיבר שהמפתח שלו הוא 1 ואביו הוא 0 – שורש העץ). כפי שהסברנו קודם, ניגש אל אב זה בדיוק פעם אחת כיוון שלכל צומת שנוריד את ערך המפתח שלו יש אב ייחודי. המשמעות היא שלא נצטרך לסמן פעמיים צומת ולכן לא נבצע פעולת cascading-cuts.

לסיכום נקבל כי עלות decrease-key בודדת היא קבועה. וסך עלות סדרת הפעולות של decrease-key היא $O(\log_2 m)$.

זמן הריצה הכולל הוא: $O(m) + O(logm) + O(logm) = \boldsymbol{O(m)}$

ב.

m	$\text{Run} - \text{Time}(ms)$	$totalLinks$	$totalCuts$	$Potential$
2^{10}	0	1023	10	29
2^{15}	21	32767	15	44
2^{20}	123	1048575	20	59
2^{25}	31325	33554431	25	74

סיכום סעיפים ג'-ו':

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

ג.

a. <u>Link</u>

נתון: $m = 2^k$. לאחר מחיקת האיבר המינימאלי, נשארנו עם m עצים מדרגה 0.

במהלך ביצוע consolidation:

נבצע $\frac{m}{2}$ לינקים לקבלת עצים בדרגה 1.

נבצע $\frac{m}{4}$ לינקים לקבלת עצים בדרגה 2.

נבצע $\frac{m}{8}$ לינקים לקבלת עצים בדרגה 3.

.

.

.

נבצע $\frac{m}{2^i}$ לינקים לקבלת עצים בדרגה i.

סה"כ נבצע:

$$\#\boldsymbol{TotalLink} = \sum_{i=1}^{logm} \frac{m}{2^i} = \sum_{i=1}^{k} \frac{2^k}{2^i} = \sum_{i=1}^{k} 2^{k-i} = \sum_{j=0}^{k-1} 2^j = \frac{2^k - 2^0}{2-1} = 2^k - 1 = \boldsymbol{m-1}$$

b. <u>Cut</u>

בהמשך להסבר מסעיף א', עבור כל פעולת decrease-key נבצע פעולת cut בודדת. נבצע logm פעולות decrease-key, לכן: $\#\boldsymbol{Cut = logm}$

c. <u>Potential</u>

מתקיים: $\#Potential = \#trees + 2 * \#marks$

ראינו כי מספר פעולות ה-cut הוא $logm$ ולכן אנו מוסיפים $logm$ עצים לערימה שלנו שהכילה עץ בודד במקור. סך העצים הוא $logm + 1$.

לפני פעולות ה-decrease-key לא היו צמתים ממורקרים. כפי שהסברנו בסעיף א', נבצע $logm$ פעולות cut. ונמרקר $logm - 1$ צמתים שהם אבות הצמתים שהורדנו להם את ערך המפתח (מלבד צומת השורש).

$$\#\boldsymbol{Potential} = \#trees + 2 * \#marks = logm + 1 + 2 * (logm - 1) = \boldsymbol{3logm - 1}$$

ד. נבצע decrease-key על המפתחות $m - 2^i$:

a. <u>:Link</u>

פעולות הלינק מתבצעות בחלק מפעולת deleteMin. כיוון שלא שינינו פעולה זו, מספר הלינקים הכולל יהיה זהה לזה שבסעיף ג': $\#\boldsymbol{TotalLink} = \boldsymbol{m-1}$

- ראינו בכיתה כי ניתן לחלק עץ בינומי מדרגה k לשני עצים בינומיאליים מדרגה k-1 כל אחת כאשר העץ בעל השורש הגדול מבין שני העצים הוא בן של הצומת המינימאלי של הערימה המקורית.

  לפי נתוני השאלה, סדר הכנסת המפתחות ולאחר ביצוע delete-min קיבלנו שלכל תת עץ בינומיאלי בדרגה k באותו תת עץ בדרגה k-1 שהשורש שלו הוא בן של השורש של העץ המקורי מתקיים שהאיברים שלו גדולים מהאיברים שבתת העץ בדרגה k-1 שהשורש שלו הוא השורש המקורי. מסדר הכנסת המפתחות וביצוע delete-min נקבל כי עבור כל שני צמתים בעלי המפתחות $m - 2^j, m - 2^{j+1}$ כך ש- $1 \le j < j+1 \le logm$ מתקיים שהצומת בעל המפתח $m - 2^j$ הוא הבן של $m - 2^{j+1}$.

b. Cut

נשים לב כי הצמתים בעלי המפתחות $m-2^i$ עבור $i=logm,\dots,1$

עבור כל שני צמתים בעלי המפתחות $m-2^j, m-2^{j+1}$ כך ש-

$1\le j<j+1\le logm$ מתקיים שהצומת בעל המפתח $m-2^j$ הוא הבן של $m-2^{j+1}$ ופעולת ה-decrease-key נעשתה תחילה על המפתח $m-2^{j+1}$ ומיד לאחר מכן על המפתח $m-2^j$ לכן לא יופר כלל הערימה ולא נצטרך לבצע חיתוך כלל.

נקבל: $\#\boldsymbol{Cut}=\mathbf{0}$.

c. Potential

לא ביצענו אף פעולת cut ולכן:

i. לא ביצענו אף מירקור של צומת ולכן מספר הצמתים הממורקרים הוא 0.
ii. מספר העצים הכולל זהה למספר העצים לאחר פעולת delete-min והוא 1 (לאחר consolidation קיבלנו עץ בינומי יחיד).

$$\#\boldsymbol{Potential}=\#trees+2*\#marks=1+2*0=\mathbf{1}$$

ה. נמחק את שורה 2:

לא נבצע delete-min ולכן נישאר עם m+1 צמתים בודדים שהתקבלו לאחר הכנסתם.

a. :Link

פעולות הלינק מתבצעות כחלק מפעולת deleteMin. כיוון שלא ביצענו פעולה זו, מספר הלינקים הכולל הוא 0. $\#\boldsymbol{TotalLink}=\mathbf{0}$.

b. Cut

פעולת ה-decrease-key תוריד את ערך המפתחות עבור צמתים בודדים. לכן פעולה זו לא תפר את כלל הערימה לכן לא ידרשו פעולות cut. $\#\boldsymbol{Cut}=\mathbf{0}$

c. Potential

לא ביצענו אף פעולת cut ולכן:

i. לא ביצענו אף מירקור של צומת ולכן מספר הצמתים הממורקרים הוא 0.
ii. מספר העצים הכולל הוא כמספר הצמתים שהוכנסו כלומר m+1.

$$\#\boldsymbol{Potential}=\#trees+2*\#marks=m+1+2*0=\boldsymbol{m}+\mathbf{1}$$

ו. נוסיף שורה: "decrease-key(m-2,m+1)":

a. :Link

פעולות הלינק מתבצעות כחלק מפעולת deleteMin. הפעולה הנוספת לא תוסיף פעולות לינק נוספות ולכן מספר הלינקים הכולל זהה לזה שבסעיף ג':

$\#\boldsymbol{TotalLink}=\boldsymbol{m}-\mathbf{1}$.

b. Cut

בהמשך להסבר מסעיף ג', עבור כל פעולת decrease-key בשורה 3 נבצע פעולת cut בודדת. נבצע logm פעולות decrease-key בחלק זה. בסעיף זה התווספה פעולת decrease-key נוספת לצומת עם המפתח m-2. פעולה זו הפרה את כלל הערימה ולכן תתבצע פעולת cut על צומת זה. כפי שראינו בסעיף ג', לכל $i=logm-1,\dots,2$, הצומת $m-2^i$ ממורקרת. כפי שהראנו בסעיף ד', כל הצמתים הנ"ל הם האבות של הצומת בעל המפתח $m-2^1=m-2$. לכן נצטרך לבצע פעולת cascading-cuts לכל הצמתים הללו עד השורש. סך הכל נבצע עוד $logm-1$ פעולות cut. נשים לב שהורדת ערך המפתח m-2 גוררת את העלות היקרה ביותר של פעולת decreaseKey (logm-1)

לעומת הפעולות הקודמות בעלות קבועה. נוסיף $logm - 1$ עצים חדשים ונבטל את המרקור של $logm - 1$ הצמתים שביצענו עליהם cut.

$$\#\boldsymbol{Cut = logm + logm - 1 = 2logm - 1}$$

c. <u>Potential</u>

לפי ההסבר שלנו בתת הסעיף הקודם נוסיף $logm - 1$ עצים חדשים ונבטל את המרקור של $logm - 1$ הצמתים שביצענו עליהם cut.

סך העצים יהיה: $logm + 1 + logm - 1 = 2logm$

סך הצמתים הממורקרים יהיה 0.

$$\#\boldsymbol{Potential} = \#trees + 2 * \#marks = 2logm + 2 * 0 = \boldsymbol{2logm}$$

שאלה 2

א.

i	m	$\mathbf{Run-Time}(ms)$	$totalLinks$	$totalCuts$	$Potential$
6	728	13	723	0	6
8	6560	32	6555	0	6
10	59048	122	59040	0	9
12	531440	367	531431	0	10
14	4782968	6689	4782955	0	14

ב. זמן הריצה של סדרת הפעולות:
נשים לב כי במהלך ריצת הפעולות לא מתבצעות פעולות decrease-key ולכן לא "מופר" האיזון של העצים הבינומיאליים והערימה מתנהגת כמו ערימה בינומית עצלה. לכן, לאחר ביצוע consolidation נקבל ערימה בינומית תקינה. בפרט, עץ אחד בלבד יהיה בעל הדרגה המינימאלית בערימה.
כעת ננתח את זמן הריצה של הפעולות הבאות:

a. insert: הכנסת צומת בודד בעלות קבועה (הצומת נוסף כעץ השמאלי ביותר בערימה). לכן הכנסת m צמתים לערימה מתבצעת בעלות $O(m)$.
נשים לב כי בסיום ההכנסה יהיו m עצים מדרגה 0 שמסודרים בסדר יורד.

b. deleteMin:

i. deleteMin ראשון: תחילה נמחק את הצומת עם מפתח 0. נישאר עם m צמתים בודדים עליהם נבצע consolidation. נעבור על m עצים ונכניס אותם לסלסלת הלינקים לכן נקבל את סיבוכיות wc: $O(m)$.
הצמתים הוכנסו בהתחלה בסדר יורד ולכן הלינקים יתבצעו מהצומת הגדול ביותר אל הקטן ביותר. נקבל ערימה שמורכבת מעצים בינומיאליים.
עבור כל שני עצים בדרגות i,j כך ש: i<j מתקיים שמפתחות הצמתים בעץ i קטנים ממפתחות הצמתים בעץ j (*). נקבל שהצומת המינימאלי בערימה הכוללת הוא שורש העץ הבינומי בעל הדרגה המינימאלית בערימה.

ii. deleteMin הבאים: נמחק את הצומת המינימאלי בערימה. כפי שהסברנו צומת זה הוא השורש של העץ הבינומי בעל הדרגה המינימאלית בערימה.
בניו של הצומת המינימאלי שמחקנו הם עצים בינומיאליים בעלי דרגות שונות זה מזה כך שכל אחד מהם בעל דרגה קטנה מהעץ המקורי ובהכרח קטנה משאר העצים בערימה. לכן בכל מחיקת הצומת המינימאלי לא יהיו בערימה עצים בעלי דרגה זהה ולא יהיה צורך לבצע לינקים ביניהם. עלות פעולת ה-deleteMin תהיה מורכבת מעלות המעבר על העצים בערימה והכנסתם לסלסלות. בניית הסלסלות בעלות $O(logm)$ כמספר הבנים בערימה.
מ-(*) נקבל גם שעבור הבנים של השורש שמחקנו העץ בעל הדרגה המינימאלית הוא גם העץ שהאיברים בו הם המינימאליים בערימה.
לכן, בסוף כל deleteMin נקבל ערימה שעומדת בתנאי (*).
בחלק זה מבצעים $\frac{3m}{4}-1$ פעולות deleteMin בעלות $O(logm)$ כל אחת ולכן עלות פעולות אלו: $O(mlogm)$.

עלות הכוללת של deleteMin: $O(m+mlogm)=O(mlogm)$

עלות כוללת insert+deleteMin: $O(m+mlogm)=\boldsymbol{O(mlogm)}$

ג. **כמות פעולות link:**

כפי שהסברנו בסעיף ב' בפעולת deleteMin, הפעולה היחידה בה יתבצעו לינקים היא פעולת ה-deleteMin הראשונה.

- נוכיח טענת עזר:
  כדי לבנות עץ בינומיאלי מדרגה k מצמתים בודדים, נדרש לבצע $2^k - 1$ פעולות link.
  נוכיח באינדוקציה:
  <u>בסיס האינדוקציה:</u> k=0, העץ מכיל צומת בודד ולכן לא נדרש לבצע פעולות link. אכן $2^0 - 1 = 0$.
  <u>נניח נכונות על k-1 ונראה נכונות עבור k:</u> עץ בינומי מדרגה k מורכב מתת עץ בינומי בעל דרגה k-1 תלוי על עץ בינומי מדרגה k-1. עלות בניית כל עץ כזה לפני הלינק ביניהם היא $2^{k-1} - 1$ (מהנחת האינדוקציה). נוסיף 1 לחיבור העצים ביניהם ונקבל את סך הלינקים שנדרשו לבצע לבניית העץ מדרגה k: $2^{k-1} - 1 + 2^{k-1} - 1 + 1 = 2 * 2^{k-1} - 2 + 1 = 2^k - 1$ כדרוש.

כפי שהסברנו קודם, לאחר פעולת consolidation במחיקת האיבר המינימאלי הראשון נקבל ערימה בינומית בגודל m. כפי שראינו בביתה, נוכל לדעת אלו את דרגות תתי העצים בערימה הבינומית לפי הייצוג הבינארי של מספר הצמתים הכולל m. נסמן את הייצוג הבינארי של m: $m := m_0 m_1 m_2 \dots m_{logm}$. כאשר לכל $0 \le i \le logm$, $m_i \in \{0,1\}$. לכן מטענת העזר מספר הלינקים הכולל הוא: $\#\boldsymbol{links} = \sum_{m_i=1} \mathbf{2^i - 1}$.

ניתן לראות כי התוצאות מסעיף א' מתאימות לניתוח שביצענו.

כמות פעולות cut:

במהלך סדרת הפעולות לא מתבצעת אף פעולת decrease-key ולכן לא מתבצעת אף פעולת cut: $\#\boldsymbol{cuts} = \mathbf{0}$. ניתן לראות בתוצאות מסעיף א' כי התבצעו 0 פעולות cut עבור כל קלט.

כמות פעולות potential:

מתקיים: $\#Potential = \#trees + 2 * \#marked$.

כיוון שלא ביצענו אף פעולת cut, מספר ה-marked הוא 0. לכן:

$\#Potential = \#trees + 2 * 0 = \#trees$

במהלך סדרת הפעולות אנו מכניסים $m + 1$ צמתים ולאחר מכן מוחקים $\frac{3m}{4}$ צמתים. בסיום המחיקות נקבל ערימה שמורכבת מעצים בינומיים.

בערימה זו נשארו $m + 1 - \frac{3m}{4} = \frac{m}{4} + 1$ צמתים וכפי שהסברנו קודם מספר העצים הוא מספר האחדות בייצוג הבינארי של מספר הצמתים הכולל בערימה. נסמן את הייצוג הבינארי של $\frac{m}{4} + 1$:

$\frac{m}{4} + 1 := m_0 m_1 m_2 \dots m_{\log(\frac{m}{4}+1)}$. כאשר לכל $0 \le i \le \log(\frac{m}{4} + 1)$, $m_i \in \{0,1\}$

$$\#\boldsymbol{Potential} = \sum_{\boldsymbol{i=0}}^{\log(\frac{\boldsymbol{m}}{\boldsymbol{4}}+\mathbf{1})} \boldsymbol{m_i}$$

ניתן לראות כי התוצאות מסעיף א' מתאימות לניתוח שביצענו.