מחלקת FibonacciHeap :	
HeapNode min	שדה שיחזיק מצביע לאיבר בערמה בעל המפתח המינימלי ביותר.
HeapNode first	שדה שיחזיק מצביע לאיבר הראשון ברשימה המקושרת של שורשי העץ בערמה.
Int size	שדה שיחזיק את כמות האיברים הכוללת בערמה.
Int markedNodes	שדה שיחזיק את מספר הצמתים המסומנים בערמה.
Int totalLinks	שדה סטטי שיחזיק את המספר הכולל של פעולות link שנעשו מתחילת ריצת התוכנית
Int totalCuts	שדה סטטי שיחזיק את המספר הכולל של פעולות ה- cuts שנעשו מתחילת ריצת התוכנית.
Int numOfTrees	שדה שיחזיק את כמות האיברים ברשימה המקושרת של שורשי העצים בערמה, כלומר את כמות העצים בערמה.
getMarkedNodes()	מחזיר את השדה ששומר את מספר הצמתים המסומנים בערמה. סיבוכיות: O(1) – החזרת שדה קיים.
setMarkedNodes	מעדכן את השדה ששומר את מספר הצמתים המסומנים בערמה. סיבוכיות: O(1) – עדכון של שדה אחד.
getNumOfTrees()	מחזיר את השדה ששומר את כמות האיברים ברשימה המקושרת של שורשי העצים בערמה, כלומר את כמות העצים בערמה. סיבוכיות: O(1) – החזרת שדה קיים.
setNumOfTrees	מעדכן את השדה ששומר את כמות העצים בערמה. סיבוכיות: O(1) – עדכון של שדה אחד.
מתודות נדרשות	
חתימה	**אופן פעולה וסיבוכיות**
Public boolean isEmpty()	מחזירה true אם המצביע לmin הוא null (שקול לכך שהערמה ריקה) ; false אחרת. סיבוכיות: O(1) – מבצעת השוואה של שדה אחד.
Public HeapNode insert(int key)	המתודה מכניסה איבר לערמה עם המפתח הנתון. ראשית, היא יוצרת HeapNode חדש עם המפתח הנתון. מבצעת בדיקה האם הערמה ריקה, אם כן, פשוט מכניסה את האיבר החדש ומעדכנת את המצביע של המin וה- first להיות שווים זה לזה ולהצביע לאיבר החדש, וכמובן מעדכנת את השדה size וnumOfTrees להיות 1. אם הערמה אינה ריקה, היא מכניסה את האיבר החדש בתחילת הרשימה במקום ה- first, מעדכנת את הnext של האיבר החדש להיות ה-first הקודם ואת ה-prev שלו להיות הprev של ה-first הקודם ובצורה הדדית גם את המצביעים שלהם. במידה והצומת החדש קטן יותר מהמינימום הקיים בערמה נעדכן גם אותו בהתאם. וכמובן את גודל הערמה(size) ואת

	מספר העצים ברשימה המקושרת (numOfTrees) נגדיל ב-1. <u>סיבוכיות:</u> O(1)- מתבצעת קריאה למתודה isEmpty שעלותה O(1) ,שינוי המצביעים ברשימה המקושרת והשוואה לשדה min ושינויו במידת הצורך שעלותם גם כן O(1).
Public void deleteMin()	המתודה מוחקת את האיבר המינימלי בערמה באופן הבא: 1. ראשית בודקת האם הערמה ריקה, ואם כן, יוצאת מהפונקציה מבלי לשנות דבר. 2. בשלב הבא, בודקת האם המצביע של min ו first מצביעים לאותו איבר, במידה וכן, נבדוק אם לfirst יש ילד ונעדכן את המצביע first להיות הילד שלו, אם אין לו ילד נעדכן את המצביע first להיות האח הימני של המינימום. 3. נבדוק אם למינימום יש ילד, אם כן – נעבור על רשימת הילדים של השורש המינימלי, ונחברם לרשימת השורשים באותו מקום שאבא שלהם היה ברשימה. אם אין למינימום ילד, ננתק את המינימום מהרשימה. 4.לבסוף נעדכן את כלל המצביעים הנדרשים לאחר המחיקה. 5. נקרא לפונקציה Consolidate. <u>סיבוכיות:</u> O(n)- מעבר על רשימת הילדים של השורש המינימלי היא לכל היותר O(logn) שכן כפי שראינו בכיתה הדרגה של כל שורש בערמה חסומה על ידי O(logn) – כלומר המספר המקסימלי של הילדים שלו. **private void Consolidate()** הפונקציה יוצרת מערך "דליים" כפי שנלמד בכיתה- איברים מטיפוס HeapNode, באמצעות קריאה לפונקציה toBuckets ולאחר מכן מבצעת קריאה לפונקציה fromBuckets עם מערך הדליים שיצרנו. <u>סיבוכיות:</u> O(logn) - סכום הסיבוכיות של שתי הפונקציות הנקראות. amortized O(logn)= toBuckets+ fromBuckets=O(logn) **private HeapNode [] toBuckets()** המתודה מאתחלת מערך של איברי ערמה HeapNode בגודל log בבסיס "מספר הזהב" של מספר האיברים בערמה. לאחר מכן היא עוברת על הרשימה המקושרת של שורשי העץ, עבור כל שורש היא "מנתקת" אותו מהרשימה ומכניסה אותו לתא במערך לפי דרגתו. אם תא זה במערך לא ריק, כלומר כבר קיים עץ בדרגה זו, היא קוראת למתודת העזר link שתחבר את שני העצים (לפי השורש הקטן מביניהם), מרוקנת את תא זה במערך וממשיכה לבדוק האם התא הבא ריק או לא. ברגע שהיא מגיעה לתא ריק במערך, היא מכניסה אליו את השורש וממשיכה לשורש הבא ברשימה.

	<u>סיבוכיות:</u> (O(logn = amort – לפי הניתוח שראינו בכיתה אנחנו למעשה עוברים על כל הצמתים בעץ. **private void fromBuckets(HeapNode [] buckets)** המתודה מאתחלת את רשימת השורשים להיות רשימה ריקה ועוברת על המערך כדי ליצור את רשימת השורשים החדשה. <u>סיבוכיות:</u> (O(logn- לאחר פעולת ה- toBuckets לכל היותר נעבור על מספר העצים שזה logn. **private HeapNode link (HeapNode x, HeapNode y)** המתודה מחברת שני שורשים כך ש-y הוא הצומת "הנתלה" על x. ראשית היא בודקת אם המפתח של x גדול יותר מהמפתח של y, אם כן, היא מחליפה בין x לy. לאחר מכן בודקת האם ל x יש ילדים, אם אין- מוסיפה לy מאתחלת את המצביעים next ו – prev של y להיות הוא עצמו. אם כן יש לו ילדים- מאתחלת את המצביעים next ו- prev של y להיות הילדים של x . לבסוף מעדכנת את כך ש- y התווסף לרשימת הילדים של x ואת x להיות ההורה של y. מעלה בנוסף את הדרגה של x ב- 1. <u>סיבוכיות:</u> (O(1- המתודה מבצעת עדכוני מצביעים ושדות בלבד. **private void insertAfter(HeapNode x , HeapNode tmp)** המתודה מחברת בין x לtmp כך שיתחברו לרשימה המקושרת בסוף. כלומר x יהיה האיבר לפני האחרון וtmp האיבר האחרון, והוא גם יהיה מחובר לאיבר first שכן זו רשימה מעגלית. <u>סיבוכיות:</u> (O(1- המתודה מבצעת עדכוני מצביעים בלבד.
public HeapNode findMin()	המתודה מחזירה את האיבר בעל המפתח הקטן ביותר בערמה, על ידי החזרת השדה min. <u>סיבוכיות:</u> (O(1 – החזרת שדה קיים.
public void meld (FibonacciHeap heap2)	המתודה מקבלת ערמת פיבונאצ'י heap2 ומאחדת אותה עם הערמה הנוכחית באופן הבא – משרשרת את הערמה החדשה לסוף הרשימה המקושרת הנוכחית. במידה והערמה ריקה נעדכן את המצביעים של הערמה נוכחית להיות לפי המצביעים המתאימים בheap2.

	אחרת, נוסיף את heap2 לסוף הרשימה המקושרת, באופן כזה שהוא יהיה האיבר האחרון ברשימה המעגלית, נעדכן את המינימום לפי מידת הצורך. בכל מקרה, נגדיל את numOfTrees ו- size של הערמה הנוכחית בהתאם לפי הערכים של heap2. <u>סיבוכיות:</u> O(1) – מדובר במספר קבוע של עדכון מצביעים ושדות ולכן ריצת המתודה היא O(1) במקרה הגרוע.
public int size()	המתודה מחזירה את מספר האיברים הכולל בערמה על ידי החזרת השדה size. <u>סיבוכיות:</u> O(1)- החזרה של שדה קיים.
public int[] countersRep()	המתודה מחזירה מערך int כך שבמקום ה-i במערך יופיע מספר העצים מדרגה i שבערמה (אם הערמה ריקה יוחזר מערך ריק). נשמור מצביע זמני עבור המינימום של הערמה, ומצביע לדרגה של המינימום שבסוף נרצה שיהיה המצביע לדרגה המקסימלית. כעת, נעבור על האחים של המינימום ונעדכן את המצביע לדרגה המקסימלית בהתאם. נעדכן את גודל המערך להיות כגודל הדרגה המקסימלית+1. ונעבור על כל אחד מהאחים של המינימום בערמה ונקדם את הדרגה במקום המתאים במערך. <u>סיבוכיות:</u> לא נדרש.
public void delete(HeapNode x)	המתודה מקבלת מצביע לאיבר x שבערמה ומוחקת אותו מהערמה. נבצע קריאה למתודת העזר decreaseKey כך שנוריד את ערך המפתח של x באופן כזה שיהיה האיבר המינימלי בערמה ולאחר מכן נקרא למתודה deleteMin כך שתסיר את האיבר המינימלי שהוא כרגע x. <u>סיבוכיות:</u> O(n)- המתודה decreaseKey הינה מסיבוכיות O(logn) במקרה הגרוע. והמתודה deleteMin הינה O(n). סה"כ- במקרה הגרוע נקבל שזמן ריצת המתודה היא O(n).
public void decreaseKey(HeapNode x, int delta)	המתודה מקבלת מצביע לאיבר x ומספר שלם delta ומורידה את ערך המפתח של x בdelta. המתודה דואגת לשמור על האינווריאנטה של ערמת פיבונאצ'י. אם x שורש ולאחר השינוי המפתח שלו קטן מהמפתח של המינימום, אז המינימום יצביע על x. אם x אינו שורש, והמפתח שלו קטן משל אביו, מתבצעת קריאה למתודת העזר cascadingCuts אשר מנתקת את x ומעבירה אותו להיות השורש ומגלגלת את פעולת הניתוקים במידת הצורך –(פירוט בתיעוד המתודה). <u>סיבוכיות:</u>

	O(logn)- מתבצע מספר קבוע של שדות ומצביעים ב- O(1) וקריאה למתודת העזר cascadingCuts שהיא מסיבוכיות O(logn). **private void cut (HeapNode child, HeapNode parent)** המתודה מקבלת child מטיפוס HeapNode וparent מטיפוס HeapNode ומנתקת את child מה-parent שלו באופן הבא: 1. נגדיל את השדה totalCuts שכן מתבצע פה חיתוך. 2. נעדכן את ההורה של הילד שקיבלנו להיות null. ואם הילד היה הורה מסומן אז נמחק אותו מרשימת ההורים המסומנים. ונעדכן את הדרגה של ההורה שקיבלנו להיות פחות 1. 3. כעת נבדוק האם לchild שקיבלנו יש אחים נוספים, אם אין – אז פשוט ננתק בין האב לבן ללא בעיה, אחרת, נעדכן את הבן של האב להיות הבן הבא ברשימת האחים שלו. 4. לבסוף נשלח את הבן אותו ניתקנו לפונקציה insertFirst. <u>סיבוכיות:</u> O(1)- מתבצע שינוי לכל היותר במספר קבוע של מצביעים ושדות. **private void insertFirst(HeapNode x)** נקבל x מטיפוס HeapNode ונרצה להכניס אותו לראש רשימת העצים. נבצע זאת באמצעות עדכון של המצביעים המתאימים. <u>סיבוכיות:</u> O(1)- מבצעים שינוי במספר קבוע של מצביעים. **private void cascadingCuts(HeapNode child, HeapNode parent)** המתודה קוראת למתודת העזר Cut אשר מנתקת את child מ-parent ומעדכנת שדות. אם parent אינו שורש וגם אינו מסומן נסמן אותו על ידי עדכון שדה המופע parent. Marked ונגדיל את מספר הצמתים המסומנים בשדה ששמרנו. אם parent אינו שורש וגם היה מסומן קודם לכן נבצע קריאה רקורסיבית ל-cascadingCuts,כלומר נגלגל את הבעיה כלפי מעלה. <u>סיבוכיות:</u> O(logn)- הקריאה למתודה Cut הינה O(1) במקרה הגרוע. פרט לשינוי מספר קבוע של שדות, במקרה הגרוע מספר הקריאות הרקורסיביות יהיה כגובה העץ וראינו בהרצאה כי הוא חסום על ידי O(logn). סה"כ כל קריאה היא O(1) ויבוצעו לכל היותר O(logn) קריאות, לכן ריצת המתודה הינה O(logn) במקרה הגרוע.

public int potential()	המתודה מחשבת ומחזירה את הפוטנציאל של מבנה הנתונים לפי החישוב : numOfTrees + 2*MarkedNodes <u>סיבוכיות:</u> O(1)- מבצעת חישוב על ידי שימוש בשדות קיימים.
public static int totalLinks()	המתודה מחזירה את המספר הכולל של פעולות ה-link שנעשו מתחילת ריצת התוכנית על ידי החזרת השדה הסטטי totalLinks. <u>סיבוכיות:</u> O(1)- החזרת שדה קיים.
public static int totalCuts()	המתודה מחזירה את המספר הכולל של פעולות ה- cut שנעשו מתחילת ריצת התוכנית על ידי החזרת השדה הסטטי totalCuts. <u>סיבוכיות:</u> O(1)- החזרת שדה קיים.
public static int[] kMin(FibonacciHeap H, int k)	המתודה מקבלת עץ בינומי ומספר חיובי k ומחזירה מערך ממוין של k הצמתים הקטנים ביותר בעץ באופן הבא: תחילה מאתחלת מערך בגודל k של מספרים שלמים ,מאתחלת עץ פיבונאצ'י חדש ריק ושומרת באיבר נפרד את המינימום של העץ הבינומי H. לאחר מכן נתחיל להכניס איברים לעץ פיבונאצ'י החדש שיצרנו, כאשר בכל פעם נכניס את המינימום מהעץ ואת כל ילדיו(נעזר בפונקציה insertChildren), לאחר מכן נכניס למערך את המינימום הנוכחי מהעץ פיבונאצ'י ולבסוף נמחק את המינימום הזה מהעץ ונחזור על פעולה זו k פעמים. <u>סיבוכיות:</u> אתחול מערך חדש בגודל K יעלה לנו O(k) לאחר מכן נעבור בלולאה k פעמים על העץ הבינומי H ובכל פעם נכניס את המינימום והילדים שלו- דבר אשר יעלה לנו O(degH) לבסוף מחיקת המינימום תעלה לנו O(logk). סה"כ: $O(k) + O\big(k * (\deg(H) + logk)\big)$ $= O\big(k(logk + degH)\big)$ כנדרש. **private static void insertChildren (FibonacciHeap H,FibonacciHeap tmp, HeapNode cur,int[]arr)** המתודה מכניסה אל העץ פיבונאצ'י החדש שיצרנו את כל ילדיו של המינימום בעץ המקורי H באופן הבא: קודם כל נבדוק כי אכן למינימום יש ילדים ונכניס את הילד הראשון של המינימום באמצעות פונקציית העזר insertForKmin ולאחר מכן נרצה להכניס את אחיו של הילד במידה וקיימים, מאחר ומדובר ברשימה מעגלית

	נעבור על האחים של הילד באמצעות לולאה שמכניסה את כל אחיו של הילד מבלי להכניס שוב את הילד עצמו. <u>סיבוכיות:</u> אנחנו עוברים על כל הילדים של המינימום שלכל היותר במקרה הגרוע יהיו כדרגת העץ H (שזו למעשה הדרגה הכי גבוה של ילדים בעץ) ולכן סה"כ הסיבוכיות תהיה – O(deg(H)). **private static void insertForKmin(FibonacciHeap tmp,HeapNode node)** המתודה מבצעת הכנסה לתוך העץ פיבונאצ'י החדש שיצרנו כך שבכל הכנסה גם נשמור פוינטר לnode עצמו מהעץ הבינומי H כדי שנוכל לדעת כיצד להתקדם בעץ לאחר כל הכנסה לעץ הזמני שיצרנו. אז המתודה פשוט מכניסה באופן רגיל באמצעות שימוש במתודה insert ולאחר מכן מאתחלת את השדה node שיצרנו במיוחד עבור המתודה במחלקה HeapNode שבסך הכול שומר מצביע לאיבר המתאים ב H המקורי. <u>סיבוכיות:</u> O(1)- שימוש במתודה Insert שעלותה O(1) ועדכון שדה נוסף אחד בלבד.
מחלקת HeapNode:	
Int key	שדה שיחזיק את המפתח של האיבר
Int rank	שדה שיחזיק את דרגת האיבר = מספר הילדים
Boolean mark	True אמ"מ האיבר מסומן- מחקו לו בן אחד.
HeapNode child	שדה שמחזיק מצביע לאחד הילדים או null.
HeapNode parent	שדה שמחזיק מצביע לאב של האיבר או null אם הוא שורש.
HeapNode next	שדה שמחזיק מצביע לאיבר הבא ברשימה – אח של הצומת.
HeapNode prev	שדה שמחזיק מצביע לאיבר הקודם ברשימה- אח של הצומת.

HeapNode node	לשימוש רק במתודה kMin – שמירה של מצביע עבור הHeapNode עצמו כאשר אנחנו רוצים להכניס אותו לתוך ערמה חדשה אבל לשמור קישור לערמה הקודמת ממנה הוא נלקח, פירוט נוסף במתודה עצמה.
מתודות get וset עבור כל אחד מהשדות שצוינו מעלה	מחזירות או מאתחלות מחדש את השדות הכתובים מעלה. בסיבוכיות O(1).