<u>תיעוד חיצוני ופירוט סיבוכיות זמן-ריצה של הקוד:</u>

בקוד שלנו, יש מחלקה פנימית אחת: **HeapNode**.

<u>המחלקה HeapNode:</u>

- השדות במחלקה HeapNode:

HeapNode info
int key
HeapNode next
HeapNode prev
HeapNode parent
HeapNode child
int rank
boolean mark

- הבנאי של המחלקה: בנאי המקבל ערך אחד בלבד (int key).

- פירוט המתודות במחלקה HeapNode:

המתודה	הסבר	סיבוכיות זמן ריצה
int getKey()	מחזירה את השדה key, את המפתח, של הצומת	סיבוכיות זמן קבועה – O(1), מחזירים מצביע
void setKey(int key)	מעדכנת את השדה key של הצומת	
HeapNode getInfo()	מחזירה את השדה info של הצומת	
void setInfo(HeapNode node)	מעדכנת את השדה info של הצומת	
void setNext(HeapNode node)	מעדכנת את המצביע לצומת next של הצומת, ל HeapNode שמקבלת	
HeapNode getNext()	מחזירה את השדה next של הצומת, כלומר מחזירה את המצביע ל HeapNode שהינו הצומת שאחריו ברשימה המקושרת	
void setPrev(HeapNode node)	מעדכנת את המצביע לprev של הצומת, ל HeapNode שמקבלת	
HeapNode getPrev()	מחזירה את השדה prev של הצומת, כלומר מחזירה את המצביע ל HeapNode שהינו הצומת שלפניו ברשימה המקושרת	
void setParent(IAVLNode node)	מעדכנת את המצביע להורה של הצומת, ל HeapNode שמקבלת	
HeapNode getParent()	מחזירה את השדה parent של הצומת, כלומר מחזירה את המצביע ל HeapNode שהינו ההורה של הצומת; אם הוא שורש בערימה מחזיר null	
boolean isMark()	מחזירה true אם הצומת מסומן, כלומר בוצע cut עם אחד מילדיו	
void setMark(boolean b)	מעדכנת את השדה mark של הצומת	
int setRank(int k)	מחזירה את הדרגה של הצומת, כלומר כמות הילדים של הצומת	
int getRank()	מחזירה את הדרגה של הצומת	

<u>המחלקה FibonacciHeap:</u>

- השדות במחלקה FibonacciHeap:
  - <u>שדות שאינם סטטיים:</u>

private int size
private HeapNode firstRoot
private HeapNode minNode
private int num of trees
private int numOfMarkedNodes

  - <u>שדות סטטיים:</u>

private static int countCuts
private static int countLinks

*השדות הסטטיים מאותחלים ל 0.

- בנאי המחלקה FibonacciHeap:
  - בנאי ריק, המאתחל את שדות העץ באופן הבא: firstRoot ו- minNode יצביעו ל null, וגודל הערימה, כמות העצים וכמות הצמתים המסומנים יהיו אפס.

- מתודות המחלקה FibonacciHeap:

*נציין כי כלל פונק' העזר ופירוטן מופיעות תחת ההסבר על הפונק' שקוראת להן.

המתודה	הסבר + מתודות העזר בהן השתמשנו, ופירוט אודותן	סיבוכיות זמן הריצה
boolean isEmpty()	מחזירה True אם העץ ריק (אם גודל העץ הוא אפס); אחרת, מחזירה False	סיבוכיות זמן קבועה – **O(1)**, ניגשים לשדה size של הערימה ומחזירים ערך בוליאני
int insert(int k)	ראשית ניצור צומת חדש עם הkey שקיבלנו. כעת אם הערימה ריקה, ה-next וה-prev שלו מצביעים לעצמו, כי זוהי ערימה מקושרת דו כיוונית בעלת איבר אחד. נגדיר אותו להיות האיבר המינימלי בערימה. אחרת, בעזרת **insertAfter** נכניס את הצומת לרשימת השורשים אחרי האיבר הכי "ימני" ברשימה, כך שהאיבר החדש שהכנסנו יהיה האיבר הכי "שמאלי" ברשימה. נעדכן את כל השדות של המחלקה בהתאם.	יצירת צומת חדש לוקחת זמן קבוע O(1). הכנסה לרשימה ריקה או לא ריקה בעזרת קריאה ל-insertAfter לוקחת O(1), וגם שינוי של השדות. לכן סה"כ סיבוכיות זמן קבועה- **O(1)**
	void insertAfter(HeapNode A, HeapNode B) משנה ארבעה מצביעים כך שהצומת B תופיע אחרי הצומת A ברשימה המקושרת.	שינוי כמות קבועה של מצביעים בסיבוכיות זמן קבועה, לכן: **O(1)**
int deleteMin(int k)	אם הערימה ריקה, לא נעשה כלום. אחרת, אם לצומת שאנו רוצים למחוק יש ילדים נקרא ל- **deleteNodeWithChildren** ואם לצומת אין ילדים נקרא ל- **deleteNodeWithoutChildren**. אם המערך אינו ריק לאחר המחיקה נקרא ל- **consolidate** על מנת להפוך את הערימה לערימה "כמעט בינומית".	consolidate עולה O(n), כאשר שאר המתודות שנקראות בסיבוכיות זמן ריצה נמוכה יותר אסימפטוטית. שאר הפעולות בסיבוכיות זמן ריצה קבועה. לכן הסיבוכיות היא - **O(n)**
	void deleteNodeWithChildren(HeapNode node) אם הצומת שאנו רוצים למחוק היא הילד היחיד של ההורה נגדיר עבור האבא את הילד שלו להיות null. אחרת, נעבור על כל הילדים של הצומת ונגדיר את ההורה שלהם להיות null. אם אין לצומת שמחקנו "אחים", נהפוך את רשימת השורשים להיות רשימת הילדים. אחרת נחבר את הילדים לרשימת השורשים ע"י שינוי מספר קבוע של מצביעים. נעדכן את השדות בהתאם.	סיבוכיות זמן הריצה של כל הפעולות היא קבועה, למעט עדכון מצביע ההורה של כל אחד מהילדים; לצומת מדרגה k יש k ילדים, והוכחנו בהרצאה שדרגות הן O(logn). לכן סה"כ הסיבוכיות היא- **O(log(n)).**
	void deleteNodeWithoutChildren(HeapNode node)	שינוי מספר קבוע של מצביעים ושדות לכן סיבוכיות זמן קבועה- **O(1)**

	אם הצומת שאנו מוחקים הוא הצומת היחיד בערימה, נהפוך אותה לערימה ריקה. אחרת, נשנה שני מצביעים, של הצומת הקודם והצומת העוקב ברשימה המקושרת, ונוציא את הצומת מהרשימה. נעדכן את השדות בהתאם.	
	HeapNode link(HeapNode x, HeapNode y) הפונקציה מחברת בין שני עצים בעלי אותה דרגה כך שהשורש של עץ אחד הופך להיות הילד של העץ השני. נעשה זאת באופן הבא- אם x גדול מ-y נחליף ביניהם. נמחק את y מרשימת השורשים ונחבר אותו לרשימת הילדים של x. נעדכן את השדות בהתאם.	שינוי מספר קבוע של מצביעים ושדות לכן סיבוכיות זמן קבועה- **O(1)**
	HeapNode[ ] toBuckets() נכניס כל שורש בערימה למערך דרגות, בהתאם לדרגה שלו. אם הערימה ריקה נחזיר מערך ריק. אחרת, נעבור על השורשים ונכניס את העץ מדרגה i למקום ה-i במערך. אם במקום אליו הכנסנו במערך קיים כבר עץ נבצע ביניהם חיבור בעזרת הפונקציה **link** ונכניס את העץ המחובר למקום הנכון במערך בהתאם לדרגתו החדשה. לבסוף נחזיר את המערך.	בערימת פיבונצ'י בעלת n צמתים יש לכל היותר n עצים (כל עץ הוא צומת בודד), במקרה הגרוע נעבור על כולם. קריאה לlink לוקחת O(1), כך גם שאר הפעולות במתודה, ולכן הסיבוכיות היא לכל היותר- **O(n)**
	HeapNode fromBuckets(HeapNode[ ] b) נרצה לחבר את כל העצים שקיבלנו במערך B על ידי **toBuckets** לכדי ערימה אחת שכל השורשים שלה מחוברים ע"י ערימה מקושרת דו כיוונית. נעבור בלולאה על המערך ונחבר כל שורש מהמערך לרשימת השורשים של הערימה בעזרת **insertAfter.** תוך כדי הלולאה נבצע השוואות על מנת למצוא את האיבר המינימלי מבין כל השורשים שהוא גם האיבר המינימלי בערימה כולה.	ראינו בהרצאה שדרגות הן O(logn), לכן יש O(log(n)) תאים במערך B – כמס' האיטרציות הגדול ביותר שנבצע. בכל איטרציה אנו מבצעים פעולות שלוקחות O(1) – שינוי מצביעים וקריאה ל insertAfter ולכן סה"כ הסיבוכיות היא- **O(log(n))**
	void consolidate() נקרא לפונקציה **toBuckets** אשר מחזירה מערך של עצים, ולאחר מכן נקרא לפונקציה **fromBuckets** על המערך שקיבלנו.	toBuckets לוקחת O(n) ו-fromBuckets לוקחת O(log(n)) לכן סה"כ הסיבוכיות היא- **O(n)**
HeapNode findMin()	אם הערימה ריקה נחזיר null; אחרת, נחזיר את השדה minNode של הערימה.	נחזיר מצביע, לכן סיבוכיות זמן קבועה- **O(1)**
void meld()	נחבר שתי ערימות פיבונצ'י לערימה אחת. נשנה את המצביעים של האיבר הראשון והאחרון בכל אחת מהערימות כך שנקבל רשימה מקושרת אחת שמכילה את השורשים של שתי הערימות. נשנה את השדות בהתאם.	שינוי מספר קבוע של מצביעים ושינוי שדות לוקח זמן קבוע- **O(1)**
Int size()	נחזיר את השדה size של הערימה – מספר הצמתים בערימה.	מחזירים מצביע בסיבוכיות זמן קבועה, לכן: **O(1)**

Int[] countersRep()	אם הערימה ריקה נחזיר מערך ריק. אחרת, נעבור בלולאה על רשימת השורשים, נמצא את הדרגה המקסימלית – ונעדכן אותה להיות גודל המערך. עבור כל שורש מדרגה k נגדיל את המקום ה-k במערך ב-1 ולבסוף נחזיר את המערך.	יש לכל היותר O(n) שורשים בערימה, ונעבור על כולם – פעמיים (פעם אחת למציאת הדרגה המקסימלית, בפעם השנייה לעדכון המערך). כל שאר הפעולות בסיבוכיות זמן ריצה קבועה, ולכן הסיבוכיות היא- **O(n)**
void delete(HeapNode x)	נבצע **decreaseKey** על המפתח של הצומת אותו נרצה למחוק, כך שהוא יהפוך למינימום בערימה על ידי הפחתת הערך המקסימלי. לאחר מכן, נבצע **deleteMin**.	decreaseKey עולה O(n), וגם deleteMin ולכן סה"כ הסיבוכיות היא- **O(n)**
Void decreaseKey(HeapNode x, int delta)	נקטין את המפתח של הצומת בערך מסוים שמקבלים. אם הצומת הוא שורש בערימה, נבדוק אם הוא כעת המינימום בערימה ונעדכן את השדה minNode בהתאם. אחרת, אם הערך החדש קטן מהערך של המפתח של האבא, נבצע **cascadingCuts** על הצומת ונחתוך אותה מהאבא.	הסיבוכיות של cascadingCuts היא O(log(n)). כל שאר הפעולות לוקחות זמן קבוע O(1) לכן סה"כ הסיבוכיות היא- **O(log(n))**
	void cut(HeapNode child, HeapNode parent) נחתוך את child מההורה שלו. נהפוך אותו להיות שורש (לא מסומן, מופיע ברשימת השורשים, ואבא שלו null). נשנה כמות סופית של מצביעים ונשתמש ב-insertAfter כך שchild יהפוך להיות ברשימת השורשים. נעדכן את השדות בהתאם.	שינוי כמות סופית של מצביעים לוקחת O(1) וגם קריאה ל-insertAfter. לכן סה"כ סיבוכיות זמן קבועה- **O(1)**
	Void cascadingCuts(HeapNode node) אם node הוא null לא נעשה כלום. אחרת, נבצע **cut** על הצומת ועל אבא שלו. אם האבא אינו מסומן נהפוך אותו למסומן ונסיים. אם הוא מסומן נפעיל גם עליו את הפונקציה ברקורסיה (נחתוך גם אותו מאבא שלו).	Cut לוקחת O(1). לכל היותר נבצע את הפונקציה ברקורסיה עד שנגיע לשורש, ולכן לכל היותר נבצע אותה O(log(n)) פעמים. לכן הסיבוכיות היא- **O(log(n))**
Int potential()	נחזיר את השדה numOfTrees ועוד 2 כפול השדה numOfMarkedNodes.	אנו ניגשים לשדות במחלקה ומבצעים פעולה אריתמטית פשוטה, לכן סיבוכיות זמן קבועה- **O(1)**
int totalLinks()	מחזירה את השדה countLinks של המחלקה.	**O(1)**
int totalCuts()	מחזירה את השדה countCuts של המחלקה.	**O(1)**
int[] kMin(FibonacciHeap H, int k)	נרצה להחזיר את k האיברים הכי קטנים בערימה. ניצור מערך ריק וערימה נוספת ריקה heap. נוסיף לערימה heap את האיבר המינימלי ב-H. נגדיר ב-info של הצומת שהכנסנו מצביע לצומת בעל אותו מפתח בערימה המקורית H. נעבור בלולאה עד שנגיע ל- k. בכל פעם נשמור את השדה info של הצומת בצומת חדש, נכניס את המפתח של הצומת למערך ואז נמחק את הצומת מהערימה heap בעזרת **deleteMin**. כעת נעבור על הילדים של הצומת בערימה המקורית	נבצע בדיוק k איטרציות. בכל איטרציה אנו מוסיפים את כל הילדים של הצומת שסימנו כ-min בתחילת האיטרציה. לכל צומת בערימה יש לכל היותר Olog(k) ילדים ולכן סה"כ הסיבוכיות היא- **O(klog(k))**

	H ונוסיף אותו לערימה heap. נשמור את האיבר המינימלי כעת בערימה ונבצע שוב את הלולאה. לבסוף נחזיר את המערך.	