ממ"ן 16 – מבני נתונים ומבוא לאלגוריתמים

תיאור הבעיה – נתון מערך מספרים, גודל k ו-3 נקודות בדיקה. המטרה היא לסרוק את המערך, ובכל נקודת בדיקה להדפיס את k האיברים הקטנים שנסרקו. לאחר סריקת המערך, להחזיר את רשימת k האיברים הקטנים. זאת, תחת סיבוכיות זמן נתונה.

הגישה הכללית של התוכנית לפתרונה - יצרנו מבנה נתונים "תור קדימויות" שממומש בעיקרו באמצעות עץ אדום שחור. (ניתן לומר שזהו הרחבה של עץ א"ש). מבנה הנתונים מוגבל ל-k נתון. כשמגיע איבר נוסף, הוא מוכנס ונשמר בעץ רק אם הוא קטן מהאיבר הגדול ביותר בתור/בעץ, ומוחלף בו. אלגוריתם הפתרון מכניס את איברי המערך לתור קדימויות בזה אחר זה. כאשר מגיעים לנקודת בדיקה ניתן להדפיס בקלות באמצעות סדר תוכי את האיברים שבתור בסדר ממויין. כן, לאחר סריקת כל האיברים, התוכנית מכניסה (בסדר תוכי) את האיברים שבתור (שמורים בעץ א"ש) לתוך מערך, זאת, בסדר ממויין.

תיאור מבני הנתונים העיקריים שבהם התוכנית משתמשת – יצרנו מבנה נתונים "תור קדימויות" – זהו תור קדימויות (למעט הפעולה increaseKey) שמושתת בעיקרו על מבנה הנתונים עץ א"ש – שמוסבר כנדרש בספר. תור הקדימויות [מעתה בקצרה: התור] מוגבל לגודל של k איברים, עבור k שנתון לבנאי התור. אם הוכנס איבר נוסף מעל ל-k האיברים, נשמרים k האיברים הקטנים ביותר מבין האיברים שהיו קיימים בתור והאיבר החדש. כמו כן, שמורים כמשתנים של האובייקט 2 משתנים שמייצגים מהו מספר האיברים בתור ומהו האיבר המקסימלי שקיים בתור ברגע נתון. ע"פ בקשה, ניתן להדפיס את האיברים המוכלים בתור בסדר ממויין באמצעות סריקה תוכית של העץ. כמו כן באמצעות סריקה תוכית של העץ, ניתן להכניס את איברי התור לתוך רשימה (מערך).

תיאור כללי של הפונקציות המרכיבות את התוכנית – מוצג כתיעוד קוד לפני כל פעולה. (לא מוצג עבור המחלקה עץ-אדום-שחור ועבור חוליית-עץ-אדום-שחור).

הקשרים בין הפונקציות (מי קורא למי) – מכיוון שנוח להציג זאת ויזואלית, קשרים אלה מוצגים בקובץ וורד נפרד. כמו כן נציין:

1. השגרה ()getNumber אינה רלוונטית לפתרון. היא מוצגת כמחוללת המספרים האקראיים בשביל בדיקת קלטים שונים.
2. ע"מ ש-PriorityQueue יהיה תור קדימויות לפי ההגדרה של תור קדימויות בעמוד 115 בספר:
   א. נכתבה השגרה ()PriorityQueue.maximum אע"פ שהיא אינה רלוונטית למהלך האלגוריתם.

ב. צריך להוסיף את השגרה ()increaseKey, אך השגרה אינה רלוונטית לפתרון שלנו לכן במקרה זה הרשנו לעצמנו לפסוח על מימושה.
ג. השגרה ()extractMax צריכה להיות ציבורית. מסיבותינו אנו, העדפנו לסווג את השגרה הזו כפרטית.

תיאור שגרת הפתרון - lowestK(arr,k,n1,n2,n3) – נציין שהתיאור של שגרת הפתרון שמוצג להלן, מוצג באופן זהה (באנגלית) בתיעוד הקוד שלפני קוד השגרה. השגרה מקבלת גודל k, 3 נקודות בדיקה n_1, n_2, n_3 ומערך של ממשיים. נתון על הקלט: $n_1 < n_2 <$ $n_3 < arr.length$ וכן $n_1, n_2, n_3, arr.length$ הם שלמים- אי שליליים. מכאן: $3 \leq arr.length$. כעת, אם k=0 השגרה מחזירה מערך ממשיים בגודל 0. אחרת – השגרה מאתחלת תור חדש בגודל k: pq. השגרה מכניסה כל איבר של המערך ל-pq. אחרי הכנסת n1 איברים, אחרי הכנסת n2 איברים ואחרי הכנסת n3 איברים השגרה מדפיסה את k המספרים הקטנים ביותר עד כה. לאחר הכנסת כל איברי arr לתוך pq, השגרה מאתחלת רשימה חדשה (מערך) ומכניסה לרשימה את k או את n האיברים שנמצאים ב-pq. הכנסה זו מתבצעת כסריקה תוכית של pq.tree ולכן המספרים מוכנסים לרשימה בסדר ממויין עולה.

מדגם קלטים – הריני מפנה אותך לקובץ "מדגם קלטים".

ניתוח נכונות האלגוריתם –

- נוכיח את הפעולות מהתחתית למעלה כמוצג בנספח.
- המימוש של מבנה הנתונים הוא באמצעות עץ א"ש (שממומש בעצמו ע"י חוליות RBTNode). השגרות של עץ א"ש מוכחות בספר ולכן נתייחס לפעולות אלה (וכן לסיבוכיות המקום והזמן שלהן) כנכונות.

נתחיל בהוכחת טענות על השגרות של התור:

נגדיר את שמורת הלולאה הבאה:

לכל $1 \leq k \in \mathbb{N}$, לאחר בניית תור הקדימויות [ע"י הבנאי] המשתנים של תור הקדימויות מקיימים:

1. בסיום כל פעולה של תור הקדימויות, tree מצביע לעץ א"ש חוקי.
2. בסיום כל פעולה <u>ציבורית</u> [public] המשתנה numberOfElements הוא מספר האיברים בתור הקדימויות והוא אינו עולה על k.
3. בסיום כל פעולה <u>ציבורית</u> של תור הקדימויות, במידה וקיים איבר כלשהוא בתור, אזי currentMaxValue הוא ערך האיבר המקסימלי שקיים בתור. במידה ולא קיימים איברים בתור אזי currentMaxValue שווה לערך הממשי המינימלי.
4. בסיום כל שגרה ציבורית התור מכיל את k האיברים המינימליים שהוכנסו בו עד כה.

0. בנוסף, עבור חלק מהשגרות, נוכיח טענה נוספת שמתקיימת בסיום כל שגרה כנ"ל. [מובן שהטענה שנוכיח לכל שגרה כנ"ל היא לא חלק טריוויאלי משמורת הלולאה, אך נבקש להתייחס לטענה הספציפית הנ"ל של כל שגרה כנ"ל כמעין "**טענת 0**" שמתקיימת לצד שמורת הלולאה הנ"ל ואותה גם נוכיח בפני עצמה].

נוכיח באינדוקציה את שמורת הלולאה:

- אתחול: השגרה הציבורית הראשונה שתפעל עבור כל תור קדימויות היא הבנאי:
  [constructor] **PriorityQueue** – בשגרה מאותחל עץ א"ש, numberOfElements=0 **ו-** currentMaxValue מאותחל לערך הממשי המינימלי, המשתנה k מאותחל ל-k הטבעי החיובי הנתון. מאחר ויש 0 איברים בתור, שמורת הלולאה, על ארבע הטענות שהיא מכילה -נכונה.

- שימור: נניח שטענה 1 נכונה לפני כניסה לכל שגרה וטענות 2-4 נכונות לפני כל כניסה לשגרה ציבורית.
- סיום: נשים לב שאף שגרה ציבורית לא קוראת לשגרה ציבורית אחרת ובתוך כך נפנה להוכיח את הסיום עבור כל שגרה:

נשים לב – לאחר שנוכיח את שמורת הלולאה נוכל לומר שהתכונות המובאות בה מהוות תכונות ל מבנה הנתונים שלנו.

printkMin – טענת ה-0: השגרה מדפיסה את k האיברים הקטנים ביותר שהוכנסו לתור עד כה. <u>הוכחה:</u> 0. ע"פ טענה 4 בתחילת הריצה התור מכיל את k האיברים הקטנים ביותר שהוכנסו בו עד כה. ולכן בסריקה התוכית שמתבצעת בשגרה (מוכח בספר) השגרה תדפיס את כל איברי העץ – כלומר את k האיברים הקטנים ביותר שהוכנסו לתור עד כה. 1,2,3,4. השגרה לא שינתה שום מידע ולכן ע"פ שהטענות היו נכונות בתחילת הריצה, הן נכונות בסיומה.

delete – טענת ה-0: השגרה מוחקת את הצומת ב-tree שהיא מקבלת. <u>הוכחה</u>: 0. הפעולה מבצעת את הדרוש באמצעות פעולת מחיקה מעץ א"ש. [מוכח בספר]. 1. לאחר מחיקת הצומת, tree עדיין מצביע כעת א"ש חוקי, (כמוכח בספר). 2,3,4. זוהי אינה שגרה ציבורית.

extractMax – טענת ה-0: במידה וקיים איבר כלשהוא בתור, השגרה מוציאה את האיבר המקסימלי מהתור. <u>הוכחה</u>: 0. ע"פ נכונות טענה 2 בתחילת הריצה, התוכנית יודעת כמה איברים יש בתור. כך ניתן לבדוק בשורה 1 אם יש איברים בתור. לפי נכונות ()RBTree.maximum ונכונות PriorityQueue.delete(pos) שהוכחנו, בשורה 3 אכן מוצא הערך המקסימלי מ-tree. 1. השינויים שעלולים להתרחש על tree הם שינויים מהשגרות delete ו-maximum – ע"פ מה

שהראנו וע"פ הספר tree נותר עא"ש חוקי. 2. זוהי שגרה פרטית, אך נשים לב שאם טענה 2 מתקיימת בתחילת השגרה אזי: אם לא היו קיימים איברים בתור בהתחלה לא מבוצע שינוי. אם היו קיימים איברים בהתחלה אזי בשורה 3 נמחק צומת מ-tree, הלא הוא איבר בתור, ובשורה 5 --numberOfElements. כך הטענה נשארת נכונה. 3,4. זוהי שגרה פרטית.

insertValueToTree – טענת ה-0: השגרה מכניסה את הממשי שהיא מקבלת כצומת ל-tree. הוכחה: 0. ההוכחה טריוויאלית: הפעולה בונה צומת חדש לפי הערך z ומכניסה אותו ל-tree. 1. הפעולה מכניסה ל-tree צומת חוקי, כך שלאחר ההכנסה, tree עץ א"ש חוקי. 2. זוהי שגרה פרטית, אך נשים לב שאם טענה 2 מתקיימת בתחילת השגרה **וגם** $numberOfElements < k$ אזי: השגרה אכן מכניסה את הממשי שהיא מקבלת כצומת ל-tree, ולאחר מכן מעלה ב-1 את numberOfElements. מכאן: numberOfElements הוא מספר האיברים בתור (בעץ) וכן $numberOfElements \leq k$ וטענה 2 מתקיימת בסיום ריצת השגרה. 3,4. זוהי אינה פעולה ציבורית לכן תנאי זה הוא אמת.

insert – בשגרה זו לא ננסח טענת 0. נוכיח את שמורת הלולאה (באמצעות חלוקה למקרים): בשורה 1 נבדק האם יש 'מקום' בתור פנוי בתור. נגדיר את המקרים האפשריים השונים:

- מקרה 1: יש מקום פנוי בתור. *אזי ע"פ שהוכחנו את insertValueToTree, בשורה 55 מוכנס הערך z כהלכה ל-tree.*
- מקרה 2: אין מקום פנוי בתור: אזי בהכרח קיים לפחות איבר בתור וכך ע"פ טענה 3 התוכנית יודעת מהו הערך המקסימלי בתור; בשורה 9 נבדק האם z קטן מהערך המקסימלי בתור.
  - מקרה 2א': z הנתון גדול או שווה לערך המקסימלי בתור.
  - מקרה 2ב': z הנתון קטן ממש מהערך המקסימלי בתור. – *אזי לפי שהוכחנו את extractMax, בשורה 63 הערך המקסימלי מוצא מהתור ולפי שהוכחנו את insertValueToTree הערך z מוכנס לתור.*

נוכיח כעת את 4 הטענות בשמורת הלולאה:

1. במקרה 1 ובמקרה 2ב' מתבצעות פעולות insertValueToTree ו-extractMax. בכניסה לפעולות אלה טענה 1 מתקיימת, והוכחנו שאם הטענה מתקיימת בתחילת אותן שגרות, הטענה תישאר נכונה בסיום ריצת השגרות. מעבר ל-2 הקריאות לשגרות אלה, לא מבוצע כל שינוי על tree. כך חוקיותו של tree כעץ א"ש. במקרה 2א' דבר משמעותי לא קורה ולכן לפי הנחת האינדוקציה, בסיום הפעולה tree יישאר עץ א"ש חוקי.

2.

מקרה 1 – במקרה זה $numberOfElements < k$. כמו כן עד שורה 2 לא מתבצע שום שינוי משמעותי וכך ע"פ הנחת האינדוקציה טענה 2 נכונה לפני ביצוע insertValueToTree. הוכחנו מעלה שאם בכניסה לשגרת insertValueToTree טענה 2 נכונה וגם $numberOfElements < k$ אזי בסיומה טענה 2 תישאר נכונה. מעבר

לשגרה זו, לא מבוצע שינוי על numberOfElements ו/או משתנה מספר האיברים בתור. מכך: במקרה זה בסיום ריצת השגרה טענה 2 תישאר נכונה.

במקרה 2א' דבר משמעותי לא קורה ותנאי 2 נשאר נכון ע"פ הנחת האינדוקציה.

במקרה 2ב' – לפי תנאי 2 מתקיים בתחילת הפעולה $numberOfElements \leq k$. אבל במקרה זה $numberOfElements \geq k$ נסיק שמתקיים $numberOfElements = k$, ולפי טענה 2 זהו מספר האיברים בתור. במקרה זה מתבצעת הפעולה הפרטית insertValueToTree שם מתבצע numberOfElements++ וגם מתווסף איבר לתור. כלומר יש $k + 1 \geq 2$ איברים בתור. לאחר מכן מתבצעת הפעולה הציבורית extractMax שם נמחק איבר ומכיוון ש-$numberOfElements$ כעת גדול/שווה ל-2 (ובפרט הוא חיובי), ייתבצע --numberOfElements. לכן – בסוף הפעולה במקרה זה - $numberOfElements = k$. ואכן מה שקרה בפועל זה הכנסת איבר לתור (לעץ) והוצאת איבר מהתור; בתור אכן יש k איברים כערכו של numberOfElements=k. כך, גם במקרה זה ע"פ הנחת האינדוקציה טענה 2 תישאר נכונה בסיום ריצת השגרה.

3.

במקרה 1 – אם בתחילת הפעולה לא היו איברים בתור אזי ע"פ תנאי 3 currentMaxValue שווה לערך הממשי המינימלי ובהכרח z גדול מערך זה. וכך בשורה 5 ייתבצע $z \leftarrow currentMaxValue$. z הערך היחיד בתור ובפרט הערך המקסימלי. ולכן בתת-מקרה זה טענה 3 תיהיה נכונה בסיום השגרה. אבל, אם בתחילת הפעולה כן היו איברים בתור אזי ע"פ תנאי 3 currentMaxValue שווה לערך הגדול ביותר מביניהם. אם z גדול מערך זה אזי בשורה 5 הערך של z יוכנס ל-currentMaxValue, וזהו אכן הערך המקסימלי מבין הערכים בתור. אבל אם z קטן או שווה ל-currentMaxValue אזי דבר משמעותי לא ייקרה ואכן currentMaxValue יישאר להיות הערך המקסימלי מבין ערכי האיברים בתור. מכאן – טענה 3 תישאר בנכונותה בסיום השגרה.

במקרה 2א' – דבר משמעותי לא ייקרה וע"פ ההנחה, טענה 3 תתקיים בסוף השגרה.

במקרה 2ב' – בתחילה, בהכרח קיימים $k \geq 1$ איברים בתור. לאחר שבשורה 11 מוכנס איבר נוסף לתור יש k+1 איברים בעץ. אח"כ ייתבצע extractMax. נבדוק את extractMax: מספר האיברים הוא חיובי, כך בשורה 4 ב'הוצאת-מקס' currentMaxValue מתעדכן להיות המפתח של המקסימום מבין האיברים הקיימים בעץ: (ע"פ שגרות עא"ש), (ובהכרח קיימים $k \geq 1$ איברים בעץ כרגע). בחזרה ל-insert – כעת מסתיימת ריצת insert. אכן קיים לפחות איבר כלשהוא בתור (בעץ) ואכן currentMaxValue הוא המקסימום מבין האיברים הקיימים בעץ.

4.

במקרה 1 – בתחילת השגרה העץ מכיל מספר $0 \leq m < k$ כלשהוא של איברים. וכך הערך הנתון z, הלא הוא המספר ה-m+1 שמוכנס לתור, מוכנס לעץ בשורה 3. ואכן בסיום השגרה, התור מכיל $m + 1 \leq k$ איברים, שהם בפרט k האיברים הקטנים ביותר שהוכנסו לתור עד כה.

מקרה 2א' – בעץ יש k איברים, שע"פ הנחת האינדוקציה הם k האיברים הקטנים ביותר שהוכנסו לתור עד לפני הרצת insert. בהכנסת הערך z, במקרה זה הוא גדול או שווה לערך המקסימלי שנמצא בעץ (זאת ע"פ טענה 3) ולכן גם אחרי הכנסת k לתור (הכנסה לתור ולא לעץ), אם k גדול ממש מהמקסימום הנוכחי, הוא אינו נכלל בין k האיברים הקטנים ביותר שהוכנסו לעץ עד כה. אם k שווה למקסימום הנוכחי אזי k הערכים הקטנים ביותר שהוכנסו לעץ אכן נמצאים בעץ גם בלי הכנסה של z הנתון לעץ. – נובע – במקרה זה טענה 4 נשארת נכונה כבתחילה.

מקרה 2ב' – במקרה זה z אכן חלק מ-k האיברים הקטנים ביותר שהוכנסו לתור. (נבדק בשורה 9, ע"פ טענה 3). אכן z מוכנס לעץ בשורה 11 (הוכחנו שזה מה שמתבצע ב-insertValueToTree). כעת, האיבר המקסימלי בשמור בעץ הוא האיבר ה-k+1 הקטן ביותר שהוכנס לתור עד כה. נבצע את extractMax. הוכחנו מעלה שאכן השגרה מוציאה את האיבר המקסימלי מהעץ וכך גם מהתור. כעת, אכן נמצאים בעץ (בתור) k האיברים הקטנים ביותר שהוכנסו עד כה וטענה 4 מתקיימת.

returnAsListRecurtion – טענת ה-0: אם בקריאה הראשונה השגרה מקבלת בקלט את השורש של tree, מספר i=0 ומערך ריק של ממשיים בגודל numberOfElements: list, אזי השגרה ממלאת את איברי list בערכי האיברים בעץ בסדר עולה. <u>הוכחה:</u>

0. [את טענה 0 ניתן גם להוכיח באינדוקציה]; לפי טענה 1, tree הוא עא"ש חוקי ובפרט עץ חיפוש בינארי חוקי. השגרה שלפנינו היא סריקה תוכית (נכונותה מוכחת בעמוד 214) עם שינוי כזה, שבמקום הדפסת האיברים, האיברים מוכנסים ל-list במקום ה-i, ובכל הכנסה כזו, מקודם i ב-1.i המעודכן מוחזר לשגרה הקוראת, וכך השגרה הקוראת משתמשת ב-i המעודכן. כעת, באותו אופן שסריקה תוכית של הדפסות מדפיסה את איברי העץ בסדר עולה, הסריקה התוכית בשגרה שלנו מכניסה את האיברים בסדר עולה. נשים לב, לא תיתכן גלישה מ-list ו-list אכן ייתמלא במלואו מכיוון שלפי טענה 2, numberOfElements הוא מספר האיברים בתור (בעץ) ואכן list הוא בגודל numberOfElements. הסריקה התוכית עוברת על כל איבר בעץ פעם אחת בדיוק. לכן, ההכנסות ל-list יהיו בדיוק כל איברי העץ – שזה גם גודל list.

1. הפניה היחידה ל-tree היא בהוצאת המפתח של צומת ספציפי ב-tree. טריוויאלי שאכן tree מצביע לעא"ש חוקי בסיום השגרה.

2,3,4. זוהי שגרה פרטית.

returnAsList – טענת ה-0: הפעולה מחזירה רשימה (כמערך) שמכיל את כל ערכי איברי tree בסדר עולה. <u>הוכחה</u>:

0. השגרה יוצרת מערך list בגודל numberOfElements. השגרה קוראת לשגרת העזר הרקורסיבית returnAsListRecurtion עם הפרמטרים: השורש של tree, מספר 0 ומערך ריק של ממשיים בגודל numberOfElements. הוכחנו מעלה שתחת התנאים הללו השגרה הרקורסיבית ממלאת את איברי list בערכי האיברים בעץ בסדר עולה.
1. הפניה היחידה ל-tree בשגרה היא דרך שגרת העזר הרקורסיבית. לכן, עד תחילת השגרה הרקורסיבית בפעם הראשונה לפי הנחת האינדוקציה, tree הוא עא"ש חוקי. הוכחנו מעלה שהשגרה הרקורסיבית אכן שומרת על tree כעל עא"ש חוקי. לכן גם returnAsList שומרת על tree כעל עא"ש חוקי.
2. בתחילת השגרה numberOfElements הוא מספר האיברים בתור הקדימויות והוא אינו עולה על k. מכיוון שלא מתבצע כל שינוי על איברי התור ולא על numberOfElements, טענה 2 נשארת נכונה גם בסיום השגרה.
3. במידה וקיים איבר כלשהוא בתור, אזי currentMaxValue הוא הערך המקסימלי שקיים בתור. במידה ולא קיימים איברים בתור אזי currentMaxValue שווה לערך הממשי המינימלי. כאמור, לא התבצע כל שינוי על איברי העץ. גם לא התבצע שינוי על currentMaxValue ולכן לפי הנחת האינדוקציה הטענה נשארת נכונה בסוף הריצה.
4. בריצה השגרה, לא מובא כל שינוי על איברי העץ ועל העץ עצמו (מלבד ()pos.getKey בשגרת העזר הרקורסיבית שכמובן לא משנה את העץ או את איבריו). מכאן, ע"פ הנחת האינדוקציה, בסיום ריצת השגרה, טענה 4 נשארת אמת.

נוכיח כעת את טענות נכונות על השגרות באלגוריתם הפתרון:

kIsZero – <u>טענה</u>: השגרה מחזירה מערך ריק של ממשיים. <u>הוכחה:</u> נכון האופן טריוויאלי.

whileScannedIsSmallerThenCheckpoint – <u>טענה</u> – בינתן מערך arr של ממשיים, תור קדימויות pq שאליו הוכנסו scanned (נתון) האיברים הראשונים ב-arr (ומלבדם לא הוכנסו איברים ל-pq) ונקודת ביקורת גדולה ממש מ-scanned וקטנה/שווה לגודל arr, השגרה מכניסה ל-pq את האיברים ב-arr באינדקסים scanned (כולל) ועד checkpoint-1 (כולל), ומחזירה את מספר האיברים שנסרקו ל-pq עד כה. <u>הוכחה:</u> השגרה בהכרח תיכנס ללולאה לפחות פעם אחת מכיוון שבתחילה scanned<checkpoint. בכל חִזְרוּר של הלולאה מועלה scanned ב-1 [שורה 4], ולכן השגרה תרוץ עד אשר scanned יהיה שווה ל-checkpoint. בכל חִזְרוּר כזה, מוכנס האיבר במקום ה-scanned ל-pq. כך הראנו שאיברי arr באינדקסים scanned,…,checkpoint-1 [בהתייחס לערכו של scanned בהתחלה] מוכנסים ל-pq. כמו כן, הלולאה נעצרת כאשר scanned=checkpoint, scanned מוחזר, וזהו אכן מספר האיברים שהוכנסו ל-pq עד כה: (הרי נתון שהוכנסו scanned איברים לפני תחילת השגרה, במהלך

השגרה הוכנסו checkpoint – scanned איברים [בהתייחס ל-scanned ההתחלתי] וכך סה"כ הוכנסו עד כה ל-pq checkpoint איברים).

promoteCheckpoint – <u>טענה:</u> בהינתן $0 \leq n_1 < n_2 < n_3 < n$ ופרמטר checkpoint ששווה לאחד מהפרמטרים הנתונים האחרים, checkpoint מועלה מ-n1 ל-n2, מ-n2 ל-n3 או מ-n3 ל-n. <u>הוכחה:</u> מתקבל באופן טריוויאלי מהקוד.

scanTheArrey – <u>טענה</u> – בינתן מערך של מספרים ממשיים arr, פרמטרים שלמים n1,n2,n3 שמקיימים $0 \leq n_1 < n_2 < n_3 < arr.length$, מספר טבעי *k* ותור קדימויות ריק *pq* שאותחל עבור הגודל *k*, השגרה מדפיסה את *k* האיברים הקטנים ביותר בשלושת תחומי הערכים: *[0,n1], [0,n2], [0,n3]* וגם מכניסה ל-*pq* את הערכים שב-*arr*.

<u>הוכחה</u> –ראשית, נקרא לשגרה whileScannedIsSmallerThenCheckpoint בקיצור ws.

כעת, <u>ננסח שמורת לולאה</u>: לפני כל בדיקה של התנאי בשורה 4, הוכנסו ל-*pq* אך ורק *scanned* האיברים הראשונים מהמערך *arr* מאז הפעם האחרונה ש-*pq* היה ריק.

<u>אתחול</u>: לפני הכניסה הראשונה, *scanned=0*, *pq* הוא ריק ולכן באופן ריק ניתן להגיד שהוכנסו ל-*pq* 0 האיברים הראשונים מ-*arr* מאז הפעם האחרונה ש-*pq* היה ריק (הפעם האחרונה היא רגע זה).

<u>תחזוקה:</u> נניח שהטענה נכונה לפני החִזְרוּר ה-*i*, כאשר *i=1,2,3*.

<u>סיום:</u> מיד לאחר הכניסה ללולאה מופעל *ws* עם הקלט arr, תור קדימויות pq שאליו הוכנסו scanned האיברים הראשונים ב-arr (ע"פ ההנחה בתחזוקה) ונקודת ביקורת גדולה ממש מ-scanned וקטנה/שווה לגודל arr, (הלא היא checkpoint שקודמה – ע"פ מה שהוכחנו ב-promoteCheckpoint – לנקודת הביקורת הבאה). כך בשורה 4 scanned מתעדכן להיות מספר האיברים שהוכנסו ל-pq עד כה. (ע"פ מה שהוכחנו שמוחזר מ-ws). השגרה חוזרת שוב לבדיקת התנאי משורה 4 – ואכן בשלב זה נסרקו מ-*scanned arr* איברים. הוכחנו את שמורת הלולאה.

נחזור על קריאת הפסקה הבאה *3* פעמים:

כעת, בשורה 4 השגרה תיכנס ללולאה. בשמורת הלולאה האחרונה הצגנו את הקלטים שמוכנסים ל-*ws* בשורה 5. כך לפי הטענה שהוכחנו עבור *ws*, בשורה 5 יוכנסו ל-*pq checkpoint* האיברים הראשונים מ-*arr* כך ש-*checkpoint* שווה ל:[קריאה ראשונה: *n1*. קריאה שניה: *n2*. קריאה שלישית: *n3*.] ע"פ תכונת תור הקדימויות שבנינו, ב-*pq* נמצאים כעת *k* האיברים הקטנים ביותר מ-*checkpoint* האיברים הראשונים ב-*arr*. בשורה 7 (הוכחנו ש)יודפסו *k* האיברים הקטנים ביותר שהוכנסו ל-*pq* עד כה – הלא הם *k* האיברים המינימליים ב-*arr* באינדקסים *0,…,scanned-1*. בשורה 8 *checkpoint* מתקדם להיות [קריאה ראשונה: *n2*. קריאה ראשונה: *n3*. קריאה ראשונה: *n*.]

הגענו שוב לבדיקת התנאי בשורה 4. ע"פ שמורת הלולאה שהוכחנו, הוכנסו ל-*pq scanned=n* איברי *arr*. כלומר: מאז ש-*pq* היה ריק, הוכנסו כל האיברים מ-*arr* ל-*pq*. כך הוכחנו את הטענה כנדרש.

lowestK – זוהי שגרת הפתרון - נוכיח כי בהינתן הנתונים כמוסבר למעלה וגם כמוסבר בשאלה, השגרה מדפיסה ב-3 נקודות הבדיקה את k המספרים הקטנים בסדר עולה, עד לאותן נקודות וגם מחזירה רשימה של k המספרים הקטנים ביותר שניתנו ב-arr.

<u>הוכחה:</u> נניח את נכונות הקלט.

אם k=0, אזי בשורה 6, (ע"פ שהוכחנו את ()kIsZero) מוחזרת רשימת ממשיים בגודל 0. באופן ריק ניתן לומר שבכל אחת מנקודות הבדיקה, הודפסו k=0 האיברים הקטנים ביותר עד כה.

אם $k \geq 1$ אזי בשורה 7 נבנה תור הקדימויות pq (עם קלט חוקי: k טבעי). בשורה 8 מתבצעת קריאה ל-scanTheArray עם הקלטים arr, $0 \leq n_1 < n_2 < n_3 < arr.length$,k ותור הקדימויות הריק *pq* שאותחל עבור הגודל *k*. ע"פ מה שהוכחנו עבור *scanTheArrey*, השגרה מדפיסה את *k* האיברים הקטנים ביותר בשלושת תחומי הערכים: *[0,n1], [0,n2], [0,n3]* וגם מכניסה ל-*pq* את הערכים שב-*arr*. ע"פ תכונת תור הקדימויות, *pq* מכיל את *k* הערכים הקטנים ביותר שהוכנסו בו עד כה. אבל, הערכים היחידים שהוכנסו ל-*pq* בינתיים הם כל איברי *arr*. נובע: *pq* מכיל את *k* האיברים הקטנים ביותר מבין איברי *arr*. כעת, בשורה 9 ע"פ מה שהוכחנו עבור ()PriorityQueue.returnAsList, list תקבל את כל ערכי איברי pq.tree בסדר עולה. כלומר, את k הערכים המינימליים מהמערך arr. בשורה 10 השגרה מחזירה את הרשימה הזו. כך הוכחנו שהטענה נכונה. הוכחנו שהשגרה הזו פותרת את הבעיה הנתונה. כנדרש.

הוכחת סיבוכיות זמן וסיבוכיות מקום – נוכיח עבור כל שגרה במבנה הנתונים ובתוכנית הראשית את סיבוכיות הזמן והמקום שלה:

נניח שגודל המערך הנתון הוא n.

PriorityQueue *[constructor]* – בשגרה יש מספר קבוע של פעולות – זמן הריצה הוא $O(1)$. השגרה מקצה מקום קבוע של ערכים לכן סיבוכיות המקום: $O(1)$.

printkMin – ע"פ התכונה של תור הקדימויות שהוכחנו אשר אומרת שמספר האיברים המקסימלי בתור הוא k: התוכנית מבצעת סרקיה תוכית של tree – מכאן, במקרה הגרוע <u>סיבוכיות הזמן של השגרה היא $\Theta(k)$ כפי שנדרש בשאלה.</u> כמו כן, בסריקה התוכית של העץ, כל צומת קורא (רקורסיבית) לאחד מבניו. מספר השגרות המקסימלי שיכולות "להיות פתוחות" כלומר להתבצע רקורסיבית הוא כגובה העץ+1 (הצומת, בהתחלה השורש, קורא לאחד מבניו, וזה חוזר חלילה עד לעלה. העלה קורא לבניו הלא הם Nil). כל פעולה שנשארת פתוחה דורשת מקום קבוע בזיכרון. מכאן, סיבוכיות המקום היא לינארית בגובה

של tree. נזכור: ב-tree יש לא יותר מ-k צמתים וע"פ התכונה של תור הקדימויות שהוכחנו, tree הוא עא"ש חוקי. גובה של עא"ש הוא לוגריתמי בגודל העץ. מכאן: במקרה הגרוע, כאשר התור מלא, סיבוכיות המקום של השגרה היא $O(\lg k)$.

delete – מבצע פעולת מחיקה מעא"ש. לכן, ע"פ מה שמוכח בספר נסיק שבמקרה הגרוע, כאשר התור מלא ב-k איברים, סיבוכיות זמן $O(\lg k)$ סיבוכיות מקום: $O(1)$.

extractMax – בשורות 3-4 מתבצעות באופן בלתי תלוי (מבחינת כך שכל אחת מהן מתבצעת בנפרד) הפעולה *BRTree.maximum* פעמיים ו-*RBTree.delete* פעם אחת. כל פעולה כזו בסיבוכיות זמן של $O(k)$ כאשר התור מלא. שאר השורות בשגרה דורשות סיבוכיות זמן קבועה. מכאן: סיבוכיות הזמן של השגרה: $T(k) = 3O(\lg k) + c \cdot O(1) = O(\lg k)$. כל הפעולות בשגרה, (ובפרט הפעולות על עץ-אדום-שחור, כמוכח בספר) רצות בסיבוכיות מקום קבועה: *O(1)*.

insertValueToTree – השגרות RBTree.getNil(), RBTree.getBlack() רצות בסיבוכיות מקום ובסיבוכיות זמן קבועה. השגרה *RBTree.RBTInsert*, כמוכח בספר, רצה בסיבוכיות מקום קבועה ובסיבוכיות זמן לוגריתמית בגודל העץ. מכאן: סיבוכיות הזמן: $T_1(k) = c_1 \cdot O(1) + O(\lg k) = O(\lg k)$. סיבוכיות המקום: $T_2(k) = c_2 \cdot O(1) = O(1)$.

insert – מלבד השגרות הקבועות בסיבוכיות הזמן והמקום שלהן, מתבצעת בשורה 3 *insertValueToTree* [הראנו שסיבוכיות הזמן שלה: *O(lgk)* והמקום *O(1)*] או בשורות 11-12 השגרות *insertValueToTree* ו-*extractMax*. [*extractMax*, הראנו, שבעלת סיבוכיות זמן *O(lg k)* ומקום *O(1)*]. לכן – בהתאם לדרישה בשאלה, סיבוכיות הזמן היא: $\mathrm{T}_1 = c_1 O(1) + 2 \cdot O(\lg k) = O(\lg k)$. סיבוכיות המקום היא: $T_2 = c_3 \cdot O(1) = O(1)$.

returnAsListRecurtion – רמות הסיבוכיות של שגרה זו תוצגנה כמקשה אחת עם רמות הסיבוכיות של השגרה הבאה.

returnAsList – שורות 1 ו-3 דורשות סיבוכיות זמן קבועה כל אחת. שורה 2 היא הפעלת הרקורסיה של השגרה returnAsListRecurtion. זוהי ריצה תוכית על *tree*, שמלבד עצם הריצה מבצעת פעולות בסיבוכיות זמן קבועה. הראנו בנימוק הסיבוכיות של *printkMin()* שסריקה תוכית של *tree* היא בסיבוכיות זמן של *O(k)*. מכאן, סיבוכיות הזמן של שגרה זו היא: *O(k)*. ובכן, שורה 1 דורשת הקצאת מערך בגודל של *numberOfElements* שבמקרה הגרוע הוא *k*; שורה 1 דורשת מקום של *O(k)*. הראנו בנימוק הסיבוכיות של *printkMin()* שסיבוכיות המקום של סריקה תוכית של *tree* היא *O(lg k)*. לכן, שורה 2 דורשת סיבוכיות מקום כנ"ל. שורה 3 דורשת הקצאת מקום קבוע. מכאן: סיבוכיות המקום של שגרה זו היא $T(k) = O(k) + O(\lg k) + O(1) = O(k)$.

kIsZero – השגרה מקצה מערך בגודל 0 ומחזירה אותו. מכאן: <u>סיבוכיות הזמן וסיבוכיות המקום של השגרה היא O(1).</u>

whileScannedIsSmallerThenCheckpoint –במקרה המקסימלי השגרה תרוץ על כל האיברים ב-arr ועבור כל אחד מהם, תכניס את הערך ל-pq. הוכחנו ששגרת ההכנסה היא בסיבוכיות זמן *O(lg k)*. כך – <u>סיבוכיות הזמן היא</u> $O(n \cdot \lg k)$. כמו כן, ריצה על האיברים דורשת מקום קבוע. הראנו שפעולת ההכנסה לתור דורשת גם מקום קבוע. מכאן, לכאורה סיבוכיות המקום של השגרה היא *O(n)*. אך נזכור ש-*pq* שומר לא יותר מ-*k* ערכים. לכן: סיבוכיות המקום של השגרה היא $O(\min\{n,k\})$.

promoteCheckpoint – כל הפעולות בשגרה זו דורשות מקום וזמן קבוע. לכן <u>סיבוכיות המקום וסיבוכיות הזמן של השגרה היא *O(1)*.</u>

scanTheArray – השגרה רצה בשורות 1-3 ברמות סיבוכיות מקום וזמן קבועות. בשגרה רצה על הלולאה שבשורה הרביעית 4 פעמים: כאשר *scanned* שווה ל-0 ולאחר מכן עבור כל אחת מנקודות הבדיקה הנתונות. בשורה 6 מתבצעת *whileScannedIsSmallerThenCheckpoint* בסיבוכיות זמן *O(n*lg(k))* ובסיבוכיות מקום $O(\min\{n,k\})$ בהתאם למה שהראנו. בשורה 7 מתבצעת *printkMin()* בסיבוכיות זמן *O(k)* ומקום *O(lg(k))* (בהתאם למה שהראנו). בשורה 8 מתבצע *promoteCheckpoint* ברמות סיבוכיות זמן ומקום קבועות. נקבל ש<u>סיבוכיות הזמן של השגרה</u> היא: $T(n,k) = 4 \cdot O(n \cdot \lg k) + O(k) + c_1 \cdot O(1) = O(n \lg k)$. <u>סיבוכיות המקום של השגרה</u> היא: $T_2(n,k) = c_2 \cdot O(1) + O(\lg k) + O(\min\{n,k\}) = O(\lg k + \min\{n,k\})$.

lowestK – השגרה רצה על שורות 1-4 ברמות סיבוכיות מקום וזמן קבועות. בהתאם לרמות הסיבוכיות שהוכחנו עבור *kIsZero*, שורות 5-6 רצות ברמות סיבוכיות זמן ומקום קבועות. בשורה 7 מוקצה מאותחל תור קדימויות, לפי מה שהוכחנו, ברמות סיבוכיות זמן ומקום קבועות. בשורה 8 מתבצע *scanTheArray*, לפי מה שהראנו, בסיבוכיות זמן $O(n \lg k)$ ומקום *O(lg(k))*. בשורה 9 מתבצע *PriorityQueue.returnAsList* ברמות סיבוכיות זמן ומקום לינאריות ב-*k*. בשורה 10 מוחזרת הרשימה ברמות סיבוכיות זמן ומקום קבועות. בסה"כ נקבל:

<u>סיבוכיות הזמן:</u> $T_1(n,k) = c_1 \cdot O(1) + O(n \lg k) + O(k) = O(n \lg k)$.

<u>סיבוכיות המקום:</u> $T_2(n,k) = c_2 \cdot O(1) + O(\lg k) + O(k)$