<u>עבודה 2 מבני נתונים – חלק ג'</u>

1. א. ראשית, יצרנו שדה maxArrSize אשר שומר את גודל המערך של החוליה בעלת ה arrSize הגדול ביותר שהוכנסה לרשימה. זאת על מנת למנוע מעבר מיותר על כל המערך של חוליית "המינוס אינסוף", ולהתחיל רק מתא רלוונטי אשר עלול להוביל לחולייה שאינה חוליית "האינסוף".
שיטת החיפוש כללה מעבר על קומות מערך ה Next של חולייה מסוימת מלמעלה למטה (תחילה חוליית "המינוס אינסוף"), והשוואה של ה key אותו אנו מחפשים עם ה key של החוליה עליה מצביעה הקומה אותה אנו "בודקים". במידה וערך ה key של החוליה עליה מצביעה הקומה אותה אנו בודקים גדול או שווה מערך ה key אותו אנו מחפשים, נתקדם אל החוליה הנ"ל ונבצע המשך סריקה על מערך ה Next שלה מאותה הקומה (במידה והייתה חוליה רלוונטית בעלת מערך גדול יותר היינו מתקדמים אליה מלפני, ולכן אין טעם לעבור בכל פעם מחדש על כל המערך של כל חוליה אליה אנו מתקדמים). במידה והתקדמנו אל חוליה בעלת ה key אותו אנו מחפשים נחזיר אותה. במקרה האחרון, סיימנו לסרוק את כל קומות מערך ה Next של חוליה כלשהי אותה סרקנו ולא הצלחנו "להתקדם" (כל החוליות הבאות בעלות key גדול ממה שאנו מחפשים), החיפוש מסתיים. משמע, ה key אותו אנו מחפשים לא נמצא ברשימה.

ב. נביט בטבלא של גובה מערך החוליה כפונקציה של מספר החוליה:

מס' חוליה	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
גובה המערך	1	2	1	3	1	2	1	4	1	2	1	3	1	2	1	5	1	2	1	3

נרצה להוכיח כי בין כל שתי חוליות בעלות אותו גובה מערך נמצאת חוליה בעלת גובה מערך גדול ממש משלהן. הטענה שקולה להוכחה כי בין כל 2 מספרים המחלקים לכל היותר חזקה n כלשהי של 2, נמצא מספר המחלק לכל הפחות את החזקה ה n+1 (והוא 2^{n+1}):
יהי p מספר טבעי חיובי המחלק לכל היותר את החזקה ה n של 2, משמע $p = 2^n * t$. קל לראות כי המספר הבא שמחלק לכל היותר את החזקה ה n של 2 הוא $p' = 2^n * (t+1)$. ניתן להבחין כי $t > 2$, אחרת:
*אם $t = 0$ אז $p = 0$ (מחוץ לתחום שלנו)
*אם $t = 1$ אז $p' = 2^n * (1+1) = 2^{n+1}$ ואז p' אינו עומד בתנאי שהוא מחלק לכל היותר את החזקה ה n של 2.
*אם $t = 2$ אז $p = 2^n * 2 = 2^{n+1}$ ואז p אינו עומד בתנאי שהוא מחלק לכל היותר את החזקה ה n של 2.

ומכאן שקיים $m = 2^n * 2 = 2^{n+1}$ המקיים p<m<p'.

לפי ההוכחה שהוכחנו ניתן לראות כי בכל קומה של מערך אנו מבצעים לכל היותר צעד אחד ימינה. במידה ונבצע שני מעברים ימינה באותה הקומה

נקבל כי המעבר הראשון מתבצע מקומה מסוימת לחוליה שגובה המערך שלה כגובה הקומה עצמה (אחרת היינו עוברים אליו לפני כן), והמעבר השני גם הוא לחולייה בעלת אותו גובה מערך מאותה הסיבה, בסתירה לכך שבין כל שני חוליות בעלות גובה זהה, קיימת חוליה בעלת גובה מערך גדול יותר. כעת, ע״פ אלגוריתם החיפוש אנו מבצעים מעבר בודד לכל היותר בכל קומה, ויורדים לכל היותר maxArrSize קומות. בכל תא במערך אנו מבצעים מספר קבוע של פעולות ולכן זמן ריצת האלגוריתם חסום ע״י maxArrSize. ניתן לראות כי החוליה בעלת גובה מערך הכי גבוהה הינה החזקה הגדולה ביותר של 2 הקטנה\שווה ל n. נתון כי $x \leq \log(n)$ כש x+1 הוא גובה המערך ומכאן $maxArrSize \leq \log(n) - 1$. ניתן להתעלם מ 1- משום שהוא זניח ביחס ל $\log(n)$, ומכיוון שכפי שמצאנו זמן ריצת האלגוריתם חסום ע״י maxArrSize נקבל כי:

$$\boldsymbol{T(n) = O(\log(n))}$$

2. פונקציית ה insert מתקדמת באופן זהה לפונקציית החיפוש, לעבר המיקום בו עלינו להכניס את החוליה החדשה מכיוון ״המינוס אינסוף״ ומכיוון ״האינסוף״. בכל פעם לפני שהפונקציה ״יורדת קומה״ היא בודקת האם הגענו לקומה שקטנה או שווה לגובה המערך של החוליה אותה אנו מכניסים, ובמידה וכן מעדכנת את המצביעים של החוליה בה אנו נמצאים בקומה הנתונה לעבר החוליה המוכנסת ולהפך. באופן זהה לפונקציית החיפוש זמן הריצה של הפונקציה תלוי בשתי הלולאות שכל אחת מהן חסומה ע״י log(n) ובסה״כ קיבלנו כי זמן הריצה של הפונקציה חסום ע״י 2log(n), עד כדי קבוע ולכן:

$$\boldsymbol{T(n) = O(\log(n))}$$

3. א. ראשית, נקטין באחד את השדה אשר שומר את מספר החוליות ברשימה. לאחר מכן, נבדוק האם החוליה אותה אנו מסירים היא בעלת גובה מערך השווה לגובה המקסימאלי של המבנה שלנו. במידה וכן, נרוץ בעזרת לולאת *while* על מערך החוליה אותה אנו מסירים מהקומה האחרונה ונבדוק באיזו קומה הכי גבוהה המערך האחורי לא מצביע על "מינוס אינסוף" או המערך הקדמי לא מצביע על "אינסוף". ברגע שמצאנו הקומה בה אנו נמצאים היא ה *maxArrSize* החדש. לאחר מכן נעבור בעזרת לולאת *for* על כל תאי המערכים של החוליה אותה אנו מעוניינים להסיר. נבדוק בכל תא *i* על איזו חוליה מצביע מערך ה *Next* של החוליה אותה אנו מעוניינים להסיר, ונקשר את תא ה *i* של מערך ה *Prev* של החוליה הזאת לחוליה עליה מצביע מערך ה *Prev* בתא ה *i* של החוליה אותה אנו מסירים. באותה לולאת ה *for* נעשה בדיוק אותו הדבר באופן סימטרי לכיוון השני. כעת שום תא ברשימה שלנו אינו מצביע על החוליה אותה רצינו להסיר והיא תיעלם ב *Garbage Collector* של *Java*.
ב. הפונקציה שלנו בעלת שתי איטרציות נפרדות אשר קובעת את זמן הריצה של הפונקציה, משום ששאר הפעולות מחוץ ובתוך הלולאות נעשות בזמן קבוע. במידה ושתי הלולאות רצות בנפרד, זה קורה לכל היותר מספר פעמים כגודל המערך של החוליה אותה אנו מסירים, ועל פי השאלות הקודמות הגדול הנ"ל חסום ע"י *log(n)*. גם אם שתיהן רצות בנפרד את מקסימום הפעמים זה חסום ע"י *2log(n)*. ומכאן ש:

$$\boldsymbol{T(n) = \mathrm{O}(\log(n))}$$

4. 2 חוליות "האינסוף" ו"המינוס אינסוף" שומרות key = O(2) ושני מערכי מצביעים בגודל N+1 = O(2N+2). שאר השדות אשר שמורים במבנה לוקחים O(1) מקום. כל חולייה ברשימה שומרת key וישנן n חוליות = O(n) ושני מערכי מצביעים שגודלם הינו לכל היותר log(n), וישנן n חוליות = O(n*2log(n)). מכאן שאם S(N,n) הינה פונקציית המקום בזיכרון כתלות של N (אותו מקבלים כקלט בבניית המערך) ו – n (מספר החוליות במערך) נקבל כי:

$$\begin{aligned}\boldsymbol{S(N,n)} &= O(2) + O(2N+2) + O(n) + O(n * 2\log(n)) \\ &= \boldsymbol{O(N + n * \log(n))}\end{aligned}$$