פרטים טכניים:

1. התוכנית נכתבה בשפת C בסביבת עובדה Ubuntu עם מהדר GCC.
2. התוכנית תומכת בקריאת קלט מקובץ ע"י הוספת שם הקובץ בתור ארגומנט בהרצת התוכנית כך:

./MergeableHeap inputfile.txt

1. צורף קובץ ההרצה וקובץ ה-make file.
2. בתיקייה “Compilers files” צורפו הקבצים של הקומפיילר, במידה וצריך.
3. בתוכנית נעשה שימוש ברשימה מקושרת מעגלית.
4. בתחילת התוכנית יש לבחור את סוג הקבוצות: 1 - קבוצות ממויינות. 2 - רשימות לא ממויינות.

3 - רשימות לא ממויינות והרשימות השונות זרות.

האלגוריתם:

1. לאחר בחירת סוג הרשימה, התוכנית תציג למשתמש את כל הפונקציות האפשריות על מנת ליצור רשימה עם קבוצות מסוג A, B ו-C ולבצע פעולות נוספות.
2. עבור כל סוגי הקבוצות - הפונקציה “make heap” תיצור struct בשם heads עם מצביע בשם head מסוג struct node שיצביע על רשימה מקושרת כלשהי שניצור בהמשך בעזרת הפונקציה “insert”. בכל הפעלה של הפונקציה יוצר מצביע חדש לרשימה חדשה. יעילות הפונקציה היא o(1).
3. עבור כל סוגי הקבוצות – הפונקציה “print” תעבור על כל ערכי ה-key של כל צומת בכל הרשימות המקושרות שהוכנסו לתוכנית. המעבר יבתצע ע"י לולאת FOR שתעבור על כל מצביעי head של הרשימות המקושרות ולולאה מקוננת של while שתבצע את המעבר על כל ערכי הרשימה שכרגע המצביע עליה בלולאת ה-for. יעילות הפונקציה היא o(n) קבועה.

עבור קבוצות מסוג A:

1. נשתמש בפונקציה הכנסה ממויינת – “insert” (insert\_sorted), הפונקציה תקבל מהתשמש ערך key להכניס לתוך הרשימה לדוגמא – “insert 6” – אז הפונקציה תיצור צומת חדש (הקצאת מקום עבור struct node). במידה והמצביע של הרשימה הנוכחית (head+index) הוא NULL אז האיבר שיצרנו הוא האיבר הראשון שיוכנס ולכן נקשר את head לצומת החדש ונחבר את next ו-prev שלו לעצמו (רשימה מעגלית). אם הצומת שיצרנו הוא לא האיבר הראשון שנכניס לרשימה אז הפונקציה תרוץ על הרשימה עד שימצא המקום המתאים – הבדיקות בסדר הבא: הצומת החדש צריך להיות האיבר הראשון, הצומת החדש הוא סנדוויץ' בין 2 איברים רנדומליים ברשימה, אם לא נמצא מקום אז נוסיף לסוף הרשימה. במקרה הטוב נקבל יעילות o(1), במקרה הגרוע נקבל יעלות o(n) במקרה הממוצע נקבל יעילות o(logn).
2. בגלל שהרשימה ממויינת נוכל לקבל בעזרת הפונקציה “minimum” (sorted\_minimum) את המינימום של הרשימה הנוכחית ביעילות של o(1) קבועה כי האיבר הראשון הוא האיבר המינימלי. הפונקציה תדפיס את הערך key של צומת הראשון ברשימה.
3. הוצאת המינימום – “extract minimum” (extract\_min\_sorted) – גם תהיה פשוטה כי אנחנו יודעים שהמיקום שלה הוא בתחילת הרשימה וגם את הפעולה הזאת נבצע ביעילות o(1). הפונקציה תקשר את הלינקים מחדש בין האיבר אחרון ברשימה לאיבר השני ברשימה ותשחרר את הזיכרון שהוקצה עבור המינימום שהוצאנו.
4. מיזוג הרשימות – “union” (union\_sort\_list) – הפונקציה תקשר את כל הרשימות לרשימה אחת. היא תרוץ על כל הרשימות שיצרנו למעט הרשימה הראשונה ותכניס כל איבר לרשימה בעזרת הפונקציה insert\_sorted שתמיין את האיבר הנתון ברשימה הראשונה. לאחר שכל האיברים הוכנסו – הפונקציה תמחק את כל המקום שהוקצה לרשימות למעט הרשימה הראשונה. היעילות במקרה הטוב o(logn), במקרה הממוצע o(n) ובמקרה הגרוע o(n^2). קיימת אפשרות לבצע את הפונקציה ביעילות יותר טובה יותר (כי הרשימה מעגלית) אם נבדוק את הקצוות של הרשימה שאליה אנחנו רוצים להכניס את האיבר ונתחיל לחפש את המקום המתאים לפי הנתון הזה. הפונקציה לא נכתבה כך בעקבות מחסור בזמנים.

עבור קבוצות מסוג B – ללא מיון:

1. נשתמש בפונקצית הכנסה רגילה – “insert” - הפונקציה עובדת בדומה ל-“insert\_sorted” אך במקום לעבור על הרשימה ולחפש את המקום המתאים של הצומת החדש, הפונקציה תכניס אותו לסוף הרשימה הנוכחית. יעילות הפונקציה היא o(1) קבועה.
2. בגלל השרשימה לא ממויינת נצטרך לחפש את המינימום בעזרת הפונקציה “minimum”, הפונקציה תעבור על הרשימה הנוכחית ותציג את הערך שלו. יעלות במקרה הטוב o(1), במקרה הממוצע o(logn) ובמקרה הגרוע o(n).
3. הוצאת המינימום – “extract\_min” - תתבצע באופן דומה לסעיף הקודם. יתבצע חיפוש על המערך – עד שנגיע לערך הנמוך ביותר, כאשר נמצא את הערך הנמוך נמחק אותו מהרשימה בעזרת קישור מחדש של האיברים הצמודים למינימום. יעלות במקרה הטוב o(1), במקרה הממוצע o(logn) ובמקרה הגרוע o(n).
4. מיזור הרשימות – “union” (union \_list) – תקשר את כל הרשימות. נניח שיש לנו 3 רשימות A, B ו-C. אז נחבר את האיבור האחרון של A עם האיבר הראשון של B ונחבר את האיבר האחרון של B עם האיבר הראשון של C ולסיום נחבר את האיבר הראשון של A עם האיבר האחרון של C (יצירת המעגליות). היעילות היא בהתאם למספר הרשימות שיצרנו – k – ונקבל יעילות o(k-1).

עבור קבוצות מסוג C – ללא מיון וזרות:

1. נשתמש באותן הפונקציות שהשתמשנו בקבוצות B למעט שינוי שמתבצע עם הפונקציה insert.
2. ההבדל היחידי הוא שימוש בפונקציה שבודקת שאין רשימה שהיא לא הרשימה הנוכחית שמכילה את אותו האיבר. הבדיקה מתבצעת לפני הקריאה לפונקציה “insert\_sorted”.

הפונקציה “boolean check\_disjoint” – מקבלת ערך מספרי ובודקת אם הערך הזה קיים בכל K הרשימות (על כל רשימה עוברים n פעמים, כאשר n הוא מספר האיברים ברשימה k). בהתאם לערך המוחזר נדע אם להכניס את הערך החדש לרשימה הנוכחית.