מיונים:

מיון	תיאור מילולי	סיבוכיות
Selection Sort	• מעבר על כל האיברים ומציאת המקסימום, והשמתו בסוף המערך. • חזרה על פעולה זו רקורסיבית על המערך פחות החלק האחרון (שמוין).	$\Theta(n^2)$
Insertion Sort	• מתקדמים מימין לשמאל כאשר ימין הוא החלק הממויין ושמאל החלק המתמיין. • לוקחים את האיבר הימני ביותר בחלק השמאלי ומוצאים את מיקומו (הזמני)בחלק הימני. את חיפוש המיקום ניתן לעשות בחיפוש בינארי, ובמקום לפעפע את האיבר למקומו בחלק הימני, ניתן להזיז את כל אלו שצריך אחד שמאלה, ולשים אותו במקומו. • מתקדמים שמאלה עד שנותר רק החלק הימני – כל המערך ממויין.	$\Theta(n^2)$ WC $\Theta(n)$ BC - כאשר לא מתבצע חיפוש בינארי (אחרת יהיו O(nlogn) השוואות)
Counting Sort (לא תחת מודל ההשוואת)	• נתון מערך המכיל מספרים שלמים בטווח 1 עד k. בונים מערך C בגודל k ומערך B בגודל מערך הקלט, A. • עוברים על כל מערך A, ועבור כל ערך i (בין 1 ל-k) מוסיפים 1 ל-C[i]. • עוברים על מערך C מהתחלה וכל C[i] מעדכנים להיות C[i-1]+C[i] – כל מקום מכיל את המיקום האחרון של המספר i במערך הממויין. • עוברים על כל מערך A: לכל i שמופיע ב-A, בודקים ב-C[A[i]] מה המיקום האחרון שלו במערך הממויין, ואז שמים את i ב-B במקום הזה. מעדכנים את המיקום האחרון להיות פחות אחד.	$\Theta\max\{n, k\} = \Theta(n + k)$
Radix Sort (לא תחת מודל ההשוואת)	• מיון למספרים שלמים הנמצאים בטווח של עד k ספרות: • ממיינים ב-counting sort ($\Theta(n)$) לפי ספרת האחדות, לאחר מכן ספרת העשרות וכן הלאה עד הספרה ה-k – חשוב ללכת מהספרה הנמוכה לגבוהה.	$\Theta(k\times n)$
Merge Sort	• מחלקים את המערך ל-2, מפעילים על כל אחד מהחצאים את האלגוריתם בצורה רקורסיבית, ובסוף ממזגים ב-n את שני החצאים הממויינים. • בחלוקת המערכים נעצור כאשר נגיע לגודל קבוע כלשהו, שם יפסקו הקריאות הרקורסיביות. תתקבל נוסחת הנסיגה: $T(n) = 2\times T(n/2) + \Theta(n)$	O(nlogn)
Quick Sort	• נמצא את החציון m של מערך (ב-O(n), ונבצע partition על המערך (גם ב-O(n) פעמיים כך שנקבל את המערך מסודר לפי איברים גודלים, שווים וקטנים מ-m. • נקרא רקורסיבית על שני חלקי המערך. נוסחת הנסיגה דומה לזו של Merge sort. • גרסה כללית: נבחר pivot כלשהו במקום חציון. הערות: o חציון מסורבל אך מבטיח סיבוכיות O(nlogn). o בחירה שרירותית ואקראית יתנו WC ($O(n^2)$, וממוצע O(nlogn). עדיפה בחירה אקראית (הגרל pivot) כדי שלא יוכלו לצפות ולתכנן קלט גרוע כנגד.	O(nlogn)

אלגוריתם החמישיות – מציאת i-יון במערך	• מחלקים את המערך לחמישיות, ומוצאים את החציונים המדוייקים על כל חמישיה. • מבצעים קריאות רקורסיביות (ב- T(n/5)) על מערך החציונים שקיבלנו עד שנקבל חציון יחיד שהוא חציון מקורב של המערך, נסמנו m. • נשתמש בשגרת partition שב-O(n) מסדרת את המערך לאיברים גדולים מ-m, שווים לו וקטנים ממנו (מימין לשמאל). תיאור חלוקה ל-≤ ו-<: נאתחל את אינדקס i לתחילת המערך, ונריץ אינדקס j על המערך. אם נתקלנו באיבר קטן ממש מ-m, נחליפו עם האיבר במקום ה-i, ונעלה את i ב-1. • כעת בודקים את i: אם הוא נופל על m, אז ברור שהוא שווה ל-m. אחרת נבצע קריאה רקורסיבית על החלק המתאים (ימני או שמאלי – גדול או קטן מ-m).	1. חמישיות - Θ(n). 2. מציאת חציון מדויק לחמישיות - Θ(n). 3. קריאה רקורסיבית למציאת חציון מקורב – T(n/5). 4. Partition - Θ(n). 5. קריאה רקורסיבית על אחד מחלקי המערך – לכל היותר T(7n + 12 / 10) – מתבצע ע"י חישוב של כמה איברים (למשל) גדולים מ-m לכל הפחות =:x, ועל זה נבצע n-1-x – כדי לקבל חסם עליון – כמה איברים לכל **היותר** קטנים מ-m. זהה לכיוון השני.

מבני נתונים:

הנחה יסודית: כל מצביע שיש לנו הוא דו כיווני.

פעולות בזמן קבוע לתור עדיפויות, לכל המימושים: make-priority-queue(), minimum(), empty?(), size()

מימוש	Insert	Decrease-key	Extract-min
ערימות בינאריות (ערימה יחידה)	**תיאור:** • מקיים את תנאי הערימה: A[i] < min{A[2i], A[2i+1]} – כל בן גדול מאביו.		
	• מעדכנים את size באחד. • מכניסים את הערך החדש למקום האחרון בערימה (עם מפתח שידאג שיהיה במקום האחרון). • מבצעים עליו decrease-key עם המפתח המקורי שקיבלנו (שמבצע heapify-up לשימור תנאי הערימה). • **סיבוכיות**: O(logn) – גובה מקסימלי של הערימה.	• מבצעים heapify-up עד הגעה לשימור תנאי הערימה. • **סיבוכיות**: O(logn) – גובה מקסימלי של הערימה.	• שומרים את הערך בשורש להחזרה אח"כ, מעדכנים את size פחות אחד. • נשים את האיבר האחרון בערימה במקום השורש. • נעשה לו heapify-down עד שיתקיים תנאי הערימה. • **סיבוכיות:** O(logn) – גובה מקסימלי של הערימה.
ערימות בינומיות (קבוצת עצים בינומים)	**תיאור – עצים בינומים:** • לכל דרגה k יש עץ ייחודי לה, כך שמספר האיברים בכל שורה שווה ל-k מעל מספר השורה (מקדם בינומי). • עץ מדרגה k בעל בדיוק 2^k צמתים. • עומק עץ בינומי מדרגה k הוא בדיוק k. • מחיקת שורש של עץ בינומי מדרגה k יוצרת k עצים בינומים מדרגות 0 עד k-1.	**תיאור – ערימה בינומית:** • אוסף עצים בינומים מדרגות שונות, בצמתים מאוחסן מידע ומתקיים תנאי הערימה. • ייצוג בינארי לערימה – כל 1 מייצג שבערימה יש עץ בינומי מדרגה [מספר הספרה]. • לכל צומת מצביע לבנו השמאלי ולאחיו (מהצדדים). • שורשי כל העצים בערימה מקושרים. • כמובן שיש מצביע ל-min ומשתנה size.	
	• יוצרים עץ בינומי חדש מדרגה 0 מהמפתח החדש. • מבצעים SL כל עוד יש שני עצים בעלי אותה דרגה בערימה. אם שניים משורשרים, השורש עם המפתח הקטן יותר יהיה האב. נעדכן את min במידת הצורך. • **סיבוכיות**: O(logn) (תלוי בהצגה הבינארית של n).	• נבצע כמו בערימות בינאריות heapify-up בעץ הרלוונטי. • **סיבוכיות:** O(logn) גובה מקסימלי של עץ בינומי בערימה בת n איברים (נניח וזה עץ יחיד בערימה).	• בעת סילוק ה-min יתפרק העץ שלו ל-O(logn) עצים. • נבצע חיבור בינארי של מה שהיה (בלי העץ שפורק) עם רשימת העצים שקיבלנו, לקבלת התמונה החדשה. • נבצע SL. • **סיבוכיות:** O(logn) מאותם שיקולים.

מימוש	Insert	Decrease-key	Extract-min
ערימות פיבונאצ׳י (קבוצת עצי פיבונאצ׳י)	**תיאור – מקיימים את התנאים:** • שילוב שני עצים יתבצע רק אם הם בעלי אותה דרגה (אחד הופך לבן של השני). • כאשר מנתקים בן מאביו (והוא הופך לשורש) מסומן האב, ובניתוק בן נוסף גם האב מתנתק מאביו (יכול לגרור סדרת ניתוקים), הופך לשורש בערימה, וספירת ניתוקי הבנים מתאפסת. • מספר הקודקודים בעץ פיבונאצ׳י מדרגה k הוא לפחות Fk (מספר פיבונאצ׳י באינדקס k כאשר F0 = 1, F1 = 2). • כדאי לזכור על סדרת פיבונאצ׳י: Fk = Fk-2 + Fk-3 +…+F2 + 2×F1 + F0 ; חסם חשוב: Fk ≥ 2^(k/2). • עץ בעל n קודקודים יהיה מדרגה לכל היותר 2logn, כלומר O(logn).		
	• יצירת עץ חדש והכנסת המפתח אליו (אין SL) • **סיבוכיות:** O(1) • **סיבוכיות amortized:** O(1)	• אם מפר את כלל הערימה, ננתק את הצומת מאביו ונעבירו להיות שורש חדש בערימה (אחרת שנה את המפתח). • אם אביו לא איבד בן, נסמנו. אם איבד – ננתקו, ויתכן ויגרור ניתוקים נוספים. • **סיבוכיות:** O(logn) בגלל חסם עליון של ניתוקים נגררים - O של כמות הניתוקים הנדרשת. • **סיבוכיות amortized:** O(1)	• מוציאים את ה-min ואת כל בניו הופכים לשורשים בערימה. • מבצעים SL על **כל** הערימה כך שלא יהיה יותר מעץ אחד. • **סיבוכיות:** O של מספר השורשים טרם ביצוע ה-E, עלול להיות גם O(n). • **סיבוכיות amortized:** O(logn)
	ניתוח amortized – על מה כל אחד משלם:		
	• הכנסתו לערימה. • עלות חיבורו בעתיד ע״י SL של E. • **סה״כ:** O(1)	• ניתוקו והפיכתו לשורש בערימה. • ניתוק עתידי של אביו. • חיבור עתידי שלו ושל אביו בעת SL של E (אם הוא הבן הראשון שנותק ממנו, הבן השני לא צריך לשלם). • **סה״כ:** O(1)	• SL של לכל היותר 2logn העצים שנוצרו עכשיו. • SL של לכל היותר 2logn העצים שנוצרו ע״י E הקודם. • SL של עצים שנוצרו מ-insert: **לא משלם.** • SL של עצים שנוצרו מ-decrease-key: **לא משלם.** • **סה״כ:** O(logn)

<u>הערות לגבי ערימות:</u>

- לא ניתן לבצע find יעיל בערימות – רק ב-O(n).
- (לפיכך) בערימות תמיד נקבל מצביע, בעצים נקבל ערך.
- **Delete:** יתבצע ע״י decrease-key ל--∞ ואחריו Extract-min.

מימוש	Insert	Decrease-key	Extract-min
עץ אדום שחור (מימוש מילון ושימוש בפעולותיו)	• Insert רגיל של העץ. • **סיבוכיות:** O(logn)	• ביצוע delete של הערך. • הכנסת הערך מחדש לעץ עם המפתח הרצוי. • **סיבוכיות:** O(logn)	• Delete לאיבר המינימלי אליו תמיד יש מצביע, והזזת המצביע לעוקב שלו. • **סיבוכיות**: O(1)

<u>עצים בינאריים:</u>

- עץ בו כל קודקוד הוא מדרגה 0, 1 או 2.
- המפתח בכל קודקוד גדול מכל המפתחות בתת העץ השמאלי שלו, וקטן מכל המפתחות בתת העץ הימני שלו.
- עץ מאוזן הוא עץ שאורך כל מסלול בו הוא O(logn).

<u>עצים אדומים-שחורים:</u>

- לכל קודקוד יש **שני בנים** בדיוק.
- ערכים מוחזקים בצמתים, ובתחתית ישנם עלים שחורים ללא ערכים.
- האורך השחור זהה בין כל המסלולים מהשורש לעלים.
- אין שני אדומים משורשרים אחד אחרי השני.
- תכונות הנובעות מהנ"ל: עץ אדום שחור הינו **עץ מאוזן**, כלומר אורך כל המסלול הוא O(logn).

מימוש מילון באמצעות עץ אדום-שחור (תיקון צבעים עמוד הבא):

Insert	Delete	Find
• מבצעים חיפוש (בינארי) עד לעלה בו אמור להיכנס הערך החדש. • מכניסים את הערך בתוך קודקוד אדום חדש עם שני עלים. • מבצעים תיקון במידת הצורך (הפרת הכלל האדום). • **סיבוכיות:** O(logn) כתוצאה מחיפוש המיקום ותיקון צבעים לאורך המסלול לשורש במידת הצורך.	• אם לקודקוד אין בנים עם תוכן, נמחק אותו ואת עליו, ונשים במקום עלה (שחור). • אם יש לו בן אחד בלבד עם תוכן, נמחק אותו ואת בנו העלה ונשרשר ישירות את הסבא לבן עם התוכן. אם הופר הכלל האדום, נצבע את הבן בשחור (כיוון שאביו בטוח היה שחור). • אם יש לו שני בנים עם תוכן, נמצא את העוקב/הקודם שלו, נמחק אותו ואת תוכנו נשים במקום תוכן הקודקוד שרוצים למחוק. • **סיבוכיות:** O(logn) למשל למקרה השלישי ותיקון צבעים.	• חיפוש בינארי. • **סיבוכיות: O(logn)**

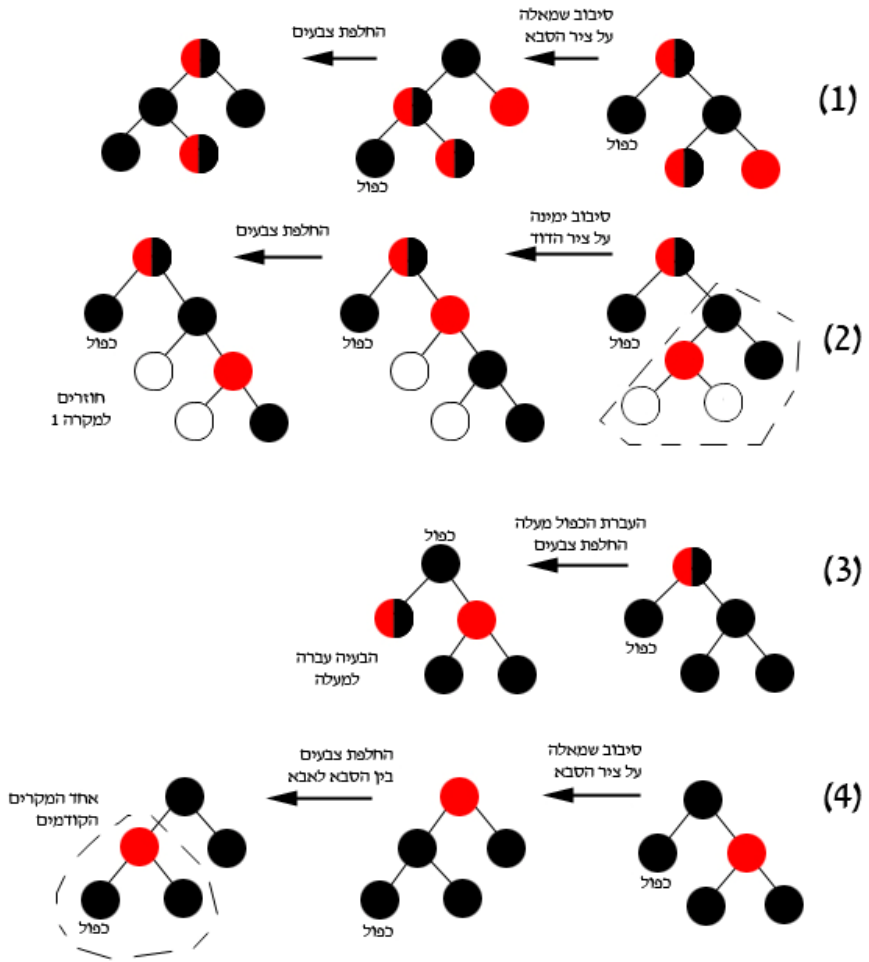
תיקון צבעים ל-insert במקרה של הפרת הכלל האדום – הבן שהוכנס ואביו אדומים:

הבן הוכנס ישירות תחת שורש העץ	נשנה את שורש העץ לשחור – לא פוגע בכלל השחור כי משנה את האורך השחור לכל המסלולים.
הדוד גם אדום (הסב כמובן שחור)	נחליף את הסבא לאדום ואת האבא והדוד לשחור. כך חלחלנו את האדום מעלה, ואם יש שם בעיה נמשיך לפי המקרה המתאים.
הדוד שחור, הבן בקו אחד עם האבא והסבא	נבצע סיבוב סביב הסבא, ונחליף צבעים בין הסבא לאבא (הסבא נהיה אדום, האבא נהיה שחור).
הדוד שחור, הבן לא בקו אחד עם האבא והסבא	נבצע סיבוב סביב האבא כדי להביא את הסבא-בן-אבא לקו אחד (בסדר זה) ומשם למקרה הקודם (בו האבא והבן החליפו תפקידים).

תיקון צבעים ל-delete במקרה של הפרת הכלל השחור – הבן שנמחק היה שחור, נסמן במקומו "שחור כפול":

- לשלושת המקרים הראשונים: שורש תת העץ שחור/אדום עם בן שחור כפול ובן שחור.

1. לבן השחור יש בן חיצוני אדום ובן פנימי שחור/אדום ← סיבוב שמאלה ואיזון צבעים.
2. לבן השחור יש בן חיצוני שחור ובן פנימי אדום ← סיבוב ימינה לתת-העץ ששורשו הבן השחור והחלפת צבעים לכדי הגעה למקרה 1.
3. לבן השחור שני בנים שחורים ← השחור הכפול מתחלף עם האדום/שחור, והבן השחור הופך אדום. הבעיה עברה למעלה.

- רק למקרה האחרון: שורש תת העץ שחור עם בן שחור כפול ובן אדום.

4. לבן האדום שני בנים שחורים (מן הסתם) ← סיבוב שמאלה והחלפת צבעים בין הסבא לאבא, וקיבלנו בתת העץ השמאלי את אחד המקרים הקודמים.