שיטות מיון במודל ההשוואות

שם	תאור האלגוריתם	סיבוכיות
Selection Sort מיון בחירה	בכל שלב מחפשים את האיבר המקסימלי במערך ממקום 0 עד מקום i ומחליפים בינו לבין האיבר במקום i-1.	Θ(n^2)
Insertion Sort מיון הכנסה	בכל פעם מחפשים את המיקום המתאים של האיבר שבמקום i-1 (האיבר האחרון בחלק הלא ממוין) ומעבירים אותו למיקומו המתאים בחלק הממוין (המיקום שמצאנו). **גרסאות:** ניתן לבצע את החיפוש של המיקום גם באמצעות חיפוש בינארי. שיפור נוסף הוא לשמור את האיבר בצד, לפנות את מקומו ולהעתיקו לשם במקום פעפוע.	Θ(n^2) עבור קלטים מסוימים, האלגוריתם יכול לרוץ ב-Θ(n). השיפור של החיפוש הבינארי מאפשר עבור קלטים נוספים לרוץ ב Θ(n*logn)
Merge Sort מיון מיזוג	מחלקים את המערך ל2 תתי מערכים באורך n/2. מיינים כל אחד מהם ע"י קריאה רקורסיבית לMerge Sort ואז ממזגים את 2 המערכים הממוינים למערך אחד.	Θ(nlogn)
Quick Sort מיון מהיר	מוצאים את החציון של המערך ומבצעים Partition פעמיים. ממשיכים במיון באמצעות קריאה רקורסיבית לQuick Sort על האיברים שגדולים וקטנים מהחציון. **גרסאות:** ניתן במקום למצוא את החציון לבחור מראש איבר במערך לפיו נחלק. כמו כן, ניתן במקום לבחור מראש, להגריל את האינדקס של הבחירה בכל פעם מחדש.	O(nlogn) במקרה הגרוע ביותר בגרסאות הנוספות, האלגוריתם ירוץ ב-O(n^2) אך במקרה הממוצע הריצה תיקח O(nlogn). חוסכים את הסרבול של מציאת החציון.

שיטות מיון מחוץ למודל ההשוואות

שם	הנחה	תאור האלגוריתם	סיבוכיות
Counting Sort מיון ספירה	תחום המספרים ידוע מראש	1. יוצרים מערך חדש C כגודל המספר המקסימלי למיון (c). 2. מאפסים את המערך שיצרנו. 3. עוברים על המערך A שברצוננו למיין וסופרים כמה פעמים מופיע כל מספר (ע"י קידום המונה בתא המתאים במערך החדש שיצרנו). 4. מבצעים במערך C שיצרנו חיבור של כל הסכומים שלפניו (לדוגמה: {1,4,4,6} -> {1,3,0,2}). 5. עוברים על מערך A, ומכנסים את האיבר במקום i למערך חדש B במקום שמצוין בתא המתאים במערך C (כלומר B[C[A[i]]] <- A[i]). כמו כן דואגים להחסיר 1 מהתא המתאים במערך C.	1. Θ(c) 2. Θ(c) 3. Θ(n) 4. Θ(c) 5. Θ(n) סה"כ Θ(c+n)
Radix Sort מיון בסיס	תחום הערכים בקלט ידוע. לכל המספרים במערך אותו מספר ספרות.	האלגוריתם מתחיל למיין את המספרים לפי ספרת האחדות שלהם ע"י שימוש ב Counting Sort, ממשיכים לספרת העשרות וכך הלאה.	O(dn) כאשר ממיינים n מספרים בני d ספרות
Bucket Sort מיון דלי	תחום הערכים בקלט ידוע מראש וגם פיזורם (כלומר ניתן לחלק את הקטע לדליים כך שכמות הערכים בכל דלי תהיה דומה)	מחלקים את התחום ל-c חלקים כך שבכל חלק (לאחר פעולת האלגוריתם) יהיה מספר דומה של מספרים. 1. מחלקים את המערך לדליים. בכל דלי תתקבל רשימה של מספרים בטווח של הדלי. 2. ממיינים כל רשימה בנפרד. 3. מאחדים את הרשימות לרשימה אחת ממוינת (פשוט משרשרים את הרשימות).	1. Θ(n) 2. Θ(n) 3. Θ(c) סה"כ O(n) נשים לב שמבקרה שn<c הסבוכיות היא O(c).

מיון ערימה

1. מכניסים את כל האיברים במערך לערימת מינימום.
2. מבצעים שליפה של המינימלי ומכניסים את האיבר שנשלף למערך.

סיבוכיות האלגוריתם: O(nlogn) (WC ובממוצע).

שלבים בחיפוש האינדקס ה-g (בזמן ליניארי)

1. מחלקים את המערך לחמישיות. בכל חמישייה מוצאים את החציון המדויק.
2. מוצאים את החציון של החציונים.
3. עושים Partition פעמיים עם החציון שמצאנו.
4. נסמן בi את האינדקס שלפניו כל האיברים קטנים מהחציון, וב-j את האינדקס שאחריו כל האיברים גדולים מהחציון שמצאנו. אם האינדקס שאנו מחפשים הוא בין i ל-j, מחזירים את החציון. אם האינדקס קטן מi, נריץ את האלגוריתם שוב רק על החלק של המערך בין 0 ל-i. אם האינדקס גדול מ-j, נריץ את האלגוריתם שוב על החלק השמאלי (מ-j עד סוף המערך ונחפש את האיבר הg-j).

מבני נתונים

מחסנית

מימוש 1 – באמצעות מערך עם הגבלת גודל ומשתנה עזר המציין את מספר האיברים במערך כעת.

שם הפעולה	תאור האלגוריתם	סבוכיות
makeStack()	יצירת מחסנית חדשה	Θ(1)
Empty?(s)	בודק האם מספר האיברים הוא 0	Θ(1)
Full?(s)	בודק האם מספר האיברים שווה לגודל המחסנית	Θ(1)
Size(s)	מחזיר את משתנה העזר	Θ(1)
top(s)	אם לא ריק, מחזיר את הערך בתא של משתנה העזר פחות 1	Θ(1)
pop(S)	אם לא ריק מחזיר את הערך בתא של משתנה העזר פחות אחד ומקטין את משתנה העזר ב1.	Θ(1)
Push(s,x)	אם לא מלא, מכניס את x למיקום של משתנה העזר במערך ומגדיל את משתנה העזר ב1.	Θ(1)

מימוש 2 – באמצעות רשימה מקושרת ומשתנה עזר המציין את מספר האיברים ברשימה.

שם הפעולה	תאור האלגוריתם	סבוכיות
makeStack()	יצירת מחסנית חדשה	Θ(1)
Empty?(s)	בודק האם מספר האיברים הוא 0	Θ(1)
Size(s)	מחזיר את משתנה העזר	Θ(1)
top(s)	אם לא ריק, מחזיר את הערך של ראש הרשימה.	Θ(1)
pop(S)	אם לא ריק מחזיר את הערך של ראש הרשימה. הופך את המצביע לראש הרשימה להצביע על האיבר הבא ומקטין את משתנה העזר ב-1.	Θ(1)
Push(s,x)	יוצרים איבר חדש שיצביע על ראש הרשימה. הופכים את המצביע לרשימה להצביע עליו ומגדילים את משתנה העזר ב1.	Θ(1)

מימוש 3 – באמצעות מערך ללא הגבלת גודל ומשתנה עזר המציין את מספר האיברים במערך כעת.

שם הפעולה	תאור האלגוריתם	סבוכיות
makeStack()	יצירת מחסנית חדשה	Θ(1)
Empty?(s)	בודק האם מספר האיברים הוא 0	Θ(1)
Size(s)	מחזיר את משתנה העזר	Θ(1)
top(s)	אם לא ריק, מחזיר את הערך בתא של משתנה העזר פחות 1	Θ(1)
pop(S)	אם לא ריק מחזיר את הערך בתא של משתנה העזר פחות אחד ומקטין את משתנה העזר ב1.	Θ(1)
Push(s,x)	אם המערך מלא, יוצר מערך הכפול בגודלו ומעתיק את האיברים למערך החדש. לבסוף מכניס את האיבר למיקום של משתנה העזר במערך ומגדיל את משתנה העזר ב-1.	W.C. Θ(n) Amortized Θ(1)

תור

מימוש 1

באמצעות רשימה מקושרת, משתנה עזר המציין את מספר האיברים ברשימה ומשתנים לקדמת התור ולזנב.

שם הפעולה	תאור האלגוריתם	סבוכיות
makeQueue()	יצרית תור חדש.	Θ(1)
Enqueue(Q,x)	יוצר איבר חדש, מכניס לתוכו את הערך x, מכוונים את מצביע הnext שלו להצביע על null. אם הרשימה לא ריקה, משנים את המצביע של הזנב ומצביע next של האיבר הזנב להצביע על האיבר החדש.	Θ(1)
Dequeue(Q)	אם לא ריק, מקטין את משנתה העזר ב1, הופכים את מצביע הראש להצביע על האיבר הבא ואם לאחר השינוי אין יותר איברים בתור. משנים את מצביע הזנב להצביע על null.	Θ(1)

מימוש 2

באמצעות מערך, משתנה עזר המציין את מספר האיברים ברשימה ומשתנים לקדמת התור ולזנב.

שם הפעולה	תאור האלגוריתם	סבוכיות
makeQueue()	יצרית תור חדש.	Θ(1)
Size()	מחזיר את משתנה העזר.	Θ(1)
Enqueue(Q,x)	אם המספר האיברים לא גדול או שווה לגודל המערך, שומרים את x במקום r במערך ומעדכנים את r להצביע על r+1 mod length(A) (כשA הוא המערך ו r הוא מצביע הזנב). אם המערך מלא מדפיסים שגיאה (או מכפילים אותו).	Θ(1)
Dequeue(Q)	אם לא ריק מחזירים את האיבר במקום f ומעדכנים את f להצביע על f+1 mod length(A).	Θ(1)

תור עדיפויות
באמצעות ערימה בינארית

שם הפעולה	תאור האלגוריתם	סבוכיות
makeQueue()	יצרית תור חדש.	Θ(1)
Minimum()	מחזיר את האיבר שבראש הערימה.	Θ(1)
Insert(Q,x)	מגדילים את משתנה העזר. מוסיפים את האיבר כעלה אחרון ועושים heapifyUp.	Θ(logn)
ExtractMin(Q)	מקטינים את משתנה העזר ב1. שומרים את ראש הערימה. מעבירים את העלה להיות ראש הערימה ועושים heapifyDoWn ומחזירים את האיבר ששמרנו.	Θ(logn)
DecreaseKey	מעדכנים את הערך של הצומת ועושים hepifyUp.	Θ(logn)
Merge	מיזוג של שתי ערימות והפעלת איחוד עצים.	O(n)
Delete	מחליפים עם העלה האחרון ועושים hepifyup/heapifydoWn בהתאם למפתח. (יש גם דרך נוספת).	O(logn)

גובה	דרגת השורש	מספר עלים	מספר צמתים
Logn	-	n/2	n

הערה: ראינו שניתן לממש תור עדיפויות באמצעות ערימה בינארית. ניתן לעשות זאת גם באמצעות מבנה מקושר כאשר בכל צומת יש 3 מצביעים: אב ו2 בנים. סבוכיות זמן הריצה של הפעולות במימוש זה הינה זהה לסבוכיות זמן ריצה בערימה בינארית.

באמצעות ערימה בינומית

שם הפעולה	SL	תאור האלגוריתם	סבוכיות
makeQueue()	לא	יצרית תור חדש.	Θ(1)
Minimum()	לא	מחזיר את האיבר שבראש הערימה.	Θ(1)
Insert(Q,x)	כן	יוצרים עץ חדש מדרגה 0 ומכניסים את הפתח לצומת. כל עוד יש שני עצים מדרגה k משלבים אותם לעץ חדש עם דרגה k+1 (השורש הוא בעל המפתח הנמוך).	Θ(logn) (תלות בהצגה הבינארית)
ExtractMin(Q)	כן	מסלקים את השורש עם המפתח המינימלי והעץ יתפרק לk עצים. מאחדים עצים מאותה הדרגה ומוצאים את הדרגה המינימלית ברשימת השורשים.	O(logn)
DecreaseKey	לא	מעדכנים את הערך של הצומת ועושים hepifyUp.	O(logn)
Merge	כן	מיזוג של שתי ערימות והפעלת איחוד עצים.	O(logn)

גובה	דרגת השורש	מספר עלים	מספר צמתים
לכל היותר k	k	-	בדיוק 2^k

תכונות ערימה בינומית:

1. בעץ בינומי מדרגה k יש בדיוק 2^k צמתים.
2. עומק של עץ בינומי מדרגה k (דרגה k = מספר הקודקודים מתחת לשורש) הוא k.
3. אם מוחקים את השורש של עץ בינומי מדרגה k הוא מתפרק לn עצים בינומים מדרגות 0,1,...n-1.
4. כאשר נתון מספר צמתים נתון, אנו יכולים להסתכל על הייצוג הבינארי שלו כדי לדעת את מבנה הערימה.
5. העץ עם הדרגה המקסימלית האפשרית בערימה בינומית הוא logn.

באמצעות ערימה בינומית עצלה (Lazy)

שם הפעולה	SL	תאור האלגוריתם	סבוכיות WC	סבוכיות Amortized
makeQueue()	לא	יצרית תור חדש.	Θ(1)	Θ(1)
Minimum()	לא	מחזיר את האיבר שבראש הערימה.	Θ(1)	Θ(1)
Insert(Q,x)	לא	יוצרים עץ חדש מדרגה 0 ומכניסים את הפתח לצומת.	Θ(1)	Θ(1)
ExtractMin(Q)	כן	מסלקים את השורש עם המפתח המינימלי והעץ יתפרק לk עצים. מאחדים עצים מאותה הדרגה ומוצאים את הדרגה המינימלית ברשימת השורשים.	O(n)	O(logn)
DecreaseKey	לא	מעדכנים את הערך של הצומת ועושים hepifyUp.	O(logn)	O(logn)
Merge	תלוי	מיזוג של שתי ערימות והפעלת איחוד עצים – תלוי בגירסה.	תלוי בגירסה	

גובה	דרגת השורש	מספר עלים	מספר צמתים
לכל היותר k	k	-	בדיוק 2^k

באמצעות ערימת פיבונאצ'י

שם הפעולה	SL	תאור האלגוריתם	סבוכיות WC	סבוכיות Amortized
makeQueue()	לא	יצרית תור חדש.	Θ(1)	Θ(1)
Minimum()	לא	מחזיר את האיבר שבראש הערימה.	Θ(1)	Θ(1)
Insert(Q,x)	לא	יוצרים עץ חדש מדרגה 0 ומכניסים את הפתח לצומת.	Θ(1)	Θ(1)
ExtractMin(Q)	כן	מסלקים את השורש עם המפתח המינימלי והעץ יתפרק לk עצים. מאחדים עצים מאותה הדרגה ומוצאים את הדרגה המינימלית ברשימת השורשים.	O(n)	O(logn)
DecreaseKey	לא	מקטינים את הערך של הצומת ואם יש צורך מנתקים אותו מהאבא (יכול לגרום שרשרת ניתוקים).	O(n)	Θ(1)
Merge	לא	מיזוג של שתי ערימות.	Θ(1)	Θ(1)
Delete	כן	הופכים את הערך של הצומת ל - (∞-) ומבצעים extractMin	O(n)	O(logn)

גובה	דרגת השורש	מספר עלים	מספר צמתים
לכל היותר n	k	-	לפחות Fk

תוכנות ערימת פיבונאצ'י:

1. בעץ פיבונאצ'י מדרגה k יש לכל הפחות F_k איברים.
2. אם בעץ השייך לערימת פיבונאצ'י יש n קודקודים אז דרגת השורש קטנה שווה ל-2logn.
3. הגדרת סדרת פיבונאצ'י במקרה זה מתחילה עם הערכים $F_0=1$ ו$F_1=2$ (כדי שתתקיים ההנחה שבעץ עם דרגה k יש לפחות F_k קודקודים.
4. גובה עץ פיבונאצ'י הוא לכל היותר logn.
5. לעץ פיבונאצ'י מדרגה k יש k בנים.

נוסחת איבר כללי בסדרת פיבונאצ'י:

$$F_n = \frac{1}{\sqrt{5}}\left(\phi_+^n - \phi_-^n\right)$$ כאשר $$\phi_\pm = \frac{1 \pm \sqrt{5}}{2}$$.

עץ חיפוש בינארי

עץ חיפוש בינארי הוא עץ בינארי שבו כל קודקוד הוא מדרגה 0,1,2 ואינו חייב להיות מאוזן. כמו כן, צומת V גדולה מכל צומת בתת העץ השמאלי וקטנה מכל צומת בתת העץ הימיני.

עץ חיפוש בינארי ייקרא מאוזן אם מובטח לנו שכל המסלולים בעץ עם n קודקודים הם באורך O(logn).

עצים אדומים שחורים

עצים אדומים שחורים הם עצי חיפוש בינאריים שבהם לכל קודקוד נוסף צבע ומקיים את התכונות הבאות:

1. לכל קודקוד שאינו עלה יש 2 בנים.
2. בעלים לא מחזיקים מפתחות וצבעם תמיד שחור.
3. בשאר הקודקודים מוחזקים מפתחות ומתקיים תנאי עצי חיפוש בינארי.
4. אין אבא ובן ששניהם אדומים.
5. "האורך השחור" הוא ערך משותף.

הערה: בעץ אדום שחור עם n קודקודים. אורך כל מסלול (יורד) הוא O(logn).

שם הפעולה	תאור האלגוריתם	סבוכיות WC
delete	מחיקת איבר מהעץ.	O(logn)
Insert(Q,x)	הוספת איבר לעץ. מוצאים את המקום המתאים במערך, הופכים את העלה לקודקוד פנימי ויוצרים לו שני עלים. את הקודקוד החדש נצבע באדום ואת העלים בשחור. במקרה של הפעלה נפעל ע"פ אחד מ4 המקרים.	O(logn)
Search	חיפוש איבר במילון לפי המפתח שלו. אם אין איבר כזה מוחזרת שגיאה או ערך null.	O(logn)
DecreaseKey	מבוצע ע"י delete ולאחריו insert.	O(logn)
extractMin	מבוצע ע"י delete לאיבר המינימלי והזזת המצביע לעוקב שלו.	O(logn)
Successor	מציאת העוקב: אם ל-x יש בן ימיני, תמצא את המינימום בתת העץ הימיני (פשוט רד שמאלה עד הסוף). אם ל-x אין בן ימיני, טפס מעלה כל עוד x הוא בן ימיני. ברגע שאתה בן שמאלי, האבא שלך הוא העוקב. אם הגעת לשורש, אתה המקסימלי.	O(logn)

גובה	דרגת השורש	מספר עלים	מספר צמתים
O(log n)	2	-	n

Union-Find

מימוש 1 - באמצעות רשימה מקושרת

שם הפעולה	תאור האלגוריתם	סבוכיות WC
Make-set	יצירת set חדש.	Θ(1)
Find	מציאת הנציג.	Θ(1)
Union	מחברים את זנב הרשימה הארוכה לראש הרשימה הקצרה ונזיז את מצביע הנציג של כל אברי הרשימה הקצרה לנציג של הרשימה הארוכה.	Θ של אורך הרשימה הקצרה

ניתוח amortized – נגדיר מדיניות חיוב כך שכל הפעולות יחויבו ב- Θ(1) וכל איבר יחויב בעלות הטיפול בו בעת Union. כל איבר יכול להיות מחויב לכל היותר logn פעמים ולכן הסיכום הכולל של l פעולות הינו O(l +nlogn).

מימוש 2 - באמצעות עצים הפוכים

בעצים הפוכים הבנים מצביעים על האבא. במימוש זה כל צומת יצביע על הנציג שלו. איחוד יכול לפעול ע"פ אחת מ-2 גישות: union-by-side או union-by-rank. שתי השיטות טובות באותה המידה (אנחנו התעמקנו בגישה השנייה). נשים לב שבעת ביצוע find במימוש זה, נבצע כיווץ מסלול של הצמתים שנעבור דרכם בfind.

סיבוכיות: במונחי WC הפעולות make-set וunion מבוצעים ב- Θ(1) וfind יבוצע ב- O(logn).
במונחי amortized רצף של l פעולות יתבצע בזמן $O(r*\log^{*}n)$.