מיונים:

	מקרה טוב	מקרה גרוע	מ. ממוצע	יציב	במקום	הנחה על קלט
מיון הכנסה Insertion-Sort	O(n)	O(n^2)	O(n^2)	כן	כן	אין
מיון בועות bubble-sort	O(n)	O(n^2)	O(n^2)	כן	כן	אין
מיון מיזוג merge-sort	O(nlgn)	O(nlgn)	O(nlgn)	כן	לא	אין
מיון ערמה HeapSort	O(n)	O(nlgn)	O(nlgn)	לא	לא	אין
מיון מהיר quick-sort	O(nlgn)	O(n^2)	O(nlgn)	לא	לא	אין
מיון מנייה Counting-Sort	O(k+n)	O(k+n)	O(k+n)	כן	לא	n מספרים שלמים בין 0 לk
מיון בסיס Radix-Sort	O(d*(n+k))	O(d*(n+k))	O(d*(n+k))	כן	לא	n מספרים שלמים עם d ספרות כל אחד, כאשר יש k ספרות אפשריות
מיון דלי Bucket-Sort	O(n) בתוחלת	O(n^2)	O(n) בתוחלת	כן	לא	n מספרים בין 0 ל-1 עם פיזור אחיד

ערמת מקסימום:

Max-Heapify(A, i)	לאחר קריאה זו, יש ערימה חוקית מתחת לאינדקס i	O(lgn)
Build-Max-Heap(A)	בניית ערמת מקסימום ממערך לא ממוין A	O(n)
Heap-Maximum(A)	החזרת האיבר המקסימלי בערמת מקסימום (ללא מחיקה)	O(1)
Heap-Extract-Max(A)	מחיקת האיבר המקסימלי מערימת מקסימום	O(lgn)
Heap-Increase-Key(A, i, key)	הגדלת הערך של key בו בערמה	O(lgn)
Max-Heap-Insert(A, key)	הכנסת ערך key לתוך הערמה	O(lgn)
Max-Heap-Delete(A, key) ! *** אין בספר צריך להוכיח *** ! **יש במדריך הלמידה בעמוד 79 מימוש אחר**	מחיקת ערך key מערימת מקסימום. - מוצאים את האיבר - מוחקים אותו - שמים במקומו את האיבר האחרון במערך - עושים עליו Max-Heapify - עושים Max-Heapify הפוך כלפי מעלה	O(lgn)
מבנה שימושי מחיקת חציון בO(logn)	בונים ממערך 2 ערימות: אחת מינימום ואחת מקסימום. בכל ערמה יש n/2 איברים. החציון יהיה תמיד השורש של ערמה אחת. דואגים לתחזק insert כדי שב-2 הערמות יהיו תמיד אותם איברים.	
תוספת שימושית לערמה:	שמירת הערך המקסימלי / מינימלי בכל תת ערימה.	

ערכי מיקום וחלוקה:

Select(A,p,r,i)	מציאת ערך מיקום ה-i במערך A, החל מ-p ועד r	O(n)
Partition(A, p, r)	שיטת החלוקה של A מp עד r. איבר הציר הוא האחרון.	O(n)
Randomize-Partition(A, p, r)	שיטת החלוקה כאשר איבר הציר הוא אקראי.	O(n)

<u>רמזים לשימוש במבני נתונים:</u>

- **Build בזמן לינארי** – כנראה ערמה (אלא אם Build ממערך ממויין – אז אפשר גם עץ)
- **מחיקת חציון** – ערמת מינימום ומקסימום מחוברות
- **משחק עם ערכי מיקום / חציון בזמן O(logn)** – עץ א"ש עם ערכי מיקום
- **אלגוריתמים שרצים בתוחלת** – טבלת גיבוב.
- **חיפוש בזמן O(logn)** – עץ א"ש. בטוח לא ערמה.
- **בקשות בזמן O(1)** – נצטרך לשמור בכל איבר במבנה. (נותן כיוון טוב)
- **סכומים / ממוצע וכו' של X איברים** – עץ א"ש עם הרחבה של סכום / ממוצע / חציון... בכל צומת.
- **איבר ותיק / חדש ביותר** – עץ א"ש עם רשימה מקושרת של היסטורית הכנסות או עת א"ש נוסף עם איברים בסדר ההכנסה וקישור בין העצים.

סכום סדרה חשבונית: $S = \frac{(a_1 + a_n) \cdot n}{2}$

סכום סדרה הנדסית: $S = \frac{a1 \cdot (q^n - 1)}{q - 1}$

סכום טור הנדסי מתכנס: $S = \frac{a_1}{1 - q}$

<u>מציאת חציון</u> –

n אי זוגי – החציון הוא: $i = \frac{(n+1)}{2}$.

n זוגי – החציון התחתון הוא: $i = \left\lfloor \frac{(n+1)}{2} \right\rfloor$

או פשוט $i = \left\lfloor \frac{(n+1)}{2} \right\rfloor$ לשני המקרים.

מבני נתונים:

מחסנית

Stack-Empty(S)	מחזיר אמת אם יש איברים במחסנית .אחרת מחזיר שקר	O(1)
Push(S, x)	מכניס למחסנית את x	O(1)
Pop(S)	מוציא מהמחסנית את האיבר האחרון שהוכנס (העליון)	O(1)
Top[S]	האינדקס של האיבר האחרון שנכנס למחסנות	O(1)

תור

Enqueue(Q, x)	הכנסת איבר x לסוף התור	O(1)
Dequeue(Q)	הוצאה והחזרת האיבר שבראש התור	O(1)

רשימה מקושרת

next[x]	מחזיר את האיבר הבא של x	O(1)
prev[x]	ברשימה דו כיוונית בלבד: מחזיר את האיבר הקודם של x	O(1)
head[L]	החזרת האיבר הראשון ברשימה	O(1)
Tail[L]	החזרת האיבר האחרון ברשימה – אם שומרים אותו	O(1)
key[x]	החזרת ערך איבר x	O(1)
List-Search(L, x)	חיפוש איבר עם מפתח x ברשימה	O(n)
List-Insert(L, x)	הכנסת איבר חדש x לראש הרשימה	O(1)
List-Delete(L, x)	מחיקת איבר x מהרשימה (x מצביע לאיבר שמוחקים)	O(n)

טבלת גיבוב

Hash-Insert(T, k)	הכנסת מפתח k לטבלה	O(1) בתוחלת
Hash-Search(T, k)	חיפוש איבר בעל מפתח k בטבלה	O(1) בתוחלת
Hash-Delete(T, k)	מחיקת איבר בעל מפתח k מהטבלה	O(1) בתוחלת

עץ חיפוש בינארי

Tree-Insert(T, z)	הכנסת צומת z לעץ	O(h)
Tree-Delete(T, z)	מחיקת צומת z מהעץ	O(h)
Tree-Search(T, z)	חיפוש צומת בעל המפתח z בעץ	O(h)
Tree-Minimum(T)	מציאת הצומת בעל הערך המינימלי	O(h)
Tree-Maximum(T)	מציאת הצומת בעל הערך המקסימלי	O(h)
Tree-Successor(x)	מציאת האיבר העוקב בסדר ממוין בעץ	O(h)
Tree-Predecessor(x)	מציאת האיבר הקודם בסדר ממוין בעץ	O(h)
בניית עץ ממערך לא ממוין	עושים n פעמים Tree-Insert	O(h*n)
בניית עץ ממערך ממוין	- מוצאים חציון O(1) - מכניסים את החציון לשורש O(1) - עושים באופן רקורסיבי ל2 חצאי המערך. (החלק השמאלי יהיה התת עץ השמאלי והימני התת עץ הימני.)	O(n)

עץ אדום-שחור

RB-Insert(T, z)	הכנסת צומת z לעץ	O(lgn)
RB-Delete(T, z)	מחיקת צומת z מהעץ	O(lgn)
Left-Rotate(T, x)	סיבוב שמאלה סביב צומת x בעץ T	O(1)
Right-Rotate(T, x)	סיבוב ימינה סביב צומת x בעץ T	O(1)
Tree-Search(T, z)	חיפוש צומת בעל המפתח z בעץ	O(lgn)

עץ א"ש עם ערכי מיקום

Size[x]	מחזיר כמה צמתים יש מתחת לצומת x	O(1)
OS-Select(x, i)	מחזיר מצביע לצומת בעל ערך המיקום ה-i בתת עץ המושרש ב-x	O(lgn)
OS-Rank(T, x)	מחזיר את ערך המיקום של x בתת עץ T	O(lgn)

<u>רעיונות להרחבות נוספות של עץ א"ש:</u>

- **שמירת סכום** כל האיברים של התת עץ המושרש בכל צומת.
- שילוב של **עץ א"ש עם רשימה דו כיוונית ממוינת** עם מצביעים מהצמתים בעץ לאיברי הרשימה. ככה מוצאים עוקב בO(1).
- **עץ הפרשים** - במקום לשמור ערכים מקוריים, שומרים את השורש כמו שהוא וכל הבנים הם

ההפרש בין השורש לעץ. זה עוזר לנו בתרגילים עם הפרשים

- **שמירת החציון** של התת עץ המושרש בכל צומת בעץ. טוב לשאלות על חציונים.
- **עץ שכיחויות** - שומרים בעץ לכל צומת את מספר הפעמים שהוא מופיע. ואז אם מכניסים לעץ משהו קיים אז במקום להכניס אפשר להגדיל את השדה של השכיחות ב-1.

משפט האב:

1. אם $f(n) = O(n^{\log_b a-\epsilon})$ קבוע כלשהו $\epsilon > 0$ כאשר
אז $T(n) = \Theta(n^{\log_b a})$
2. אם $f(n) = \Theta(n^{\log_b a})$
אז $T(n) = \Theta(n^{\log_b a}) \cdot \log n$

<u>מקרה 2 המורחב:</u>

אם $f(n) = \Theta(n^{\log_b a} \log^k n)$ כאשר $k \geq 0$
אז $T(n) = \Theta(n^{\log_b a} \log^{k+1} n)$

3. אם $f(n) = \Omega(n^{\log_b a+\epsilon})$ כאשר $\epsilon > 0$ קבוע כלשהו
וגם $a \cdot f^{n_b} \leq c \cdot f(n)$ עבור $c > 1$ והחל מ-n גדול מספיק
אזי $T(n) = \Theta(f(n))$

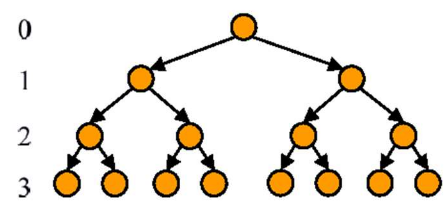
סריקות על עץ בינארי:

סריקה תחילית (Pre-Order): מתחילים בשורש , עוברים לשמאלי , ואז לימני.
סריקה תוכית (In-Order): הולכים לעלה הכי שמאלי , אחר כך עוברים לשורש ואז לימני. <u>(הסריקה הזאת מחזירה את העץ בסדר עולה ממוין)</u>
סריקה סופית (Post-Order) : עוברים לעלה הכי שמאלי , אחריו הולכים לימני ואז לשורש.

תכונות עצים בינריים שלמים

<u>בעץ בינרי שלם בעל n צמתים, L עלים, וגובה h:</u>

1. מספר הצמתים בעומק i: $n_i = 2^i$
2. מספר העלים: $L = n_h = 2^h$
3. מספר הצמתים: $n = \sum_{i=0}^{h} n_i = \sum_{i=0}^{h} 2^i = 2^{h+1} - 1$
4. הגובה: $h = \log_2(n+1) - 1$
5. מספר הצמתים הפנימיים: $n - L = 2^h - 1 = L - 1$



$$
\begin{aligned}
\log_a(bc) &= \log_a(b) + \log_a(c) \\
\log_a(b^c) &= c\log_a(b) \\
\log_a(1/b) &= -\log_a(b) \\
\log_a(1) &= 0 \\
\log_a(a) &= 1 \\
\log_a(a^r) &= r \\
\log_{1/a}(b) &= -\log_a(b) \\
\log_a(b)\log_b(c) &= \log_a(c) \\
\log_b(a) &= \frac{1}{\log_a(b)} \\
\log_{a^m}(a^n) &= \frac{n}{m}, \qquad m \neq 0
\end{aligned}
$$

טריקים שימושיים:

 # <u>הכנסה והוצאה של איברים והחזרת המינימום ב O(1).</u>
נשתמש בשתי מחסניות, אחת ראשית ואחת ששומרת מינימום. נכניס את האיבר הראשון לשניהם. כל איבר אחר שייכנס, ייכנס למחסנית הראשית.
אם הוא <u>**קטן**</u> מהאיברים שבראש המחסנית מינימום, נכניס אותו למחסנית המינימום גם כן, אחרת לא.
כאשר מוציאים איבר מהמחסנית הראשית, נבדוק אם הוא בראש מחסנית המינימום, אם כן נוציא אותו גם ממנה בנוסף

 # <u>הכנסה של מערך ממוין בגדול N לעץ חיפוש בינארי</u>
במקרה זה, עלות ההכנסה תהיה <u>**לינארית**</u> מאחר וזהו המקרה הטוב של הכנסה לעץ, מתבצעים מינימום צעדים מהשורש למקום בו צריך למקם את האיבר החדש.

 # <u>המספרים הנתונים המערך שלמים וחסומים</u>
במקרה זה ניתן להשתמש במיון מניה או מיון בסיס.
זמני ריצה:
מיון בסיס – O(d(n+k)) - כאשר D מייצג את מספר הספרות הנדרשות אם נמיין למשל לפי בסיס N כאשר הטווח הוא מ N עד 2^N אז D=3, תמיד לK ועוד 1
מיון מניה – זמן ריצה O(n+k) כאשר K גודל המספר המקסימלי בטווח

 # <u>החזרת המפתח ה-T השכיח ביותר במבנה עץ ערכי מיקום</u>
נמצא את הערך המקסימלי בעץ, נחסיר ממנו T , נחפש את תוצאת החיסור באמצעות השגרה OS-SELECT , נקבל צומת ואותה נחזיר (כמובן אם צריך לשמור מצביעים דו כיוונים בין מבני הנתונים).

 # <u>מציאת חציון ושימוש בו בערימות</u>
נתון מערך בגדול n נשתמש ב SELECT ע"מ למצוא את החציון.
לאחר שמצאנו נכניס את n/2 האיברים הגדולים מהחציון לערימת מינימום ואת החצי השני, כלומר את הקטנים מהחציון, לערימת מקסימום. נשמור מונה עבור כל ערמה המונה את מספר האיברים שנמצאים בה, זאת ע"מ שנדע אם צריך להעביר איברים בין הערמות כאשר מבצעים פעולות הכנסה ומחיקה (איזון בין ערימות).
אם נמחק את החציון (שנמצא בראש ערימת המקסימום) אם מספר האיברים הוא אי זוגי בערימת המקסימום לאחר המחיקה נצטרך להעביר איבר מערימת המינימום למקסימום, אחרת נעשה מחיקה רגילה בערימת המקס'.

 # <u>החזרת ערך המיקום ה7+ n/2 למשל של S</u>
פעולה שתמיד מופיעה בבניית מבנה נתונים S, ונדרשת להתבצע בסיבוכיות קבועה, לא תמיד יופיע 7 יכול להופיע כל מספר קבוע אחר.
לרוב יהיה לנו ערימת מינימום ומקסימום למציאת החציון כמו שהוצג לעיל, בערמת המינימום יש חצי מהאיברים בסה"כ, אבל חצי האיברים **הגדולים מהחציון**, ראש הערימה הוא הבא אחרי החציון ז"א ערך המיקום האן חלקי 2. לכן, ערך המיקום הנדרש n/2 +7 נמצא לכל היותר ברמה השביעית של הערמה, ז"א שיש מספר קבוע של איברים, 2 בחזקת 0 +...+2 בחזקת 7, שבניהם יימצא ערך המיקום הנדרש, ברמה נמוכה יותר הוא **לא** יכול להיות, זהו מספר קבוע של איברים, ולכן החזרת ערך המיקום המתבקש נעשית בסיבוכיות <u>**קבועה**</u>.