גידול של פונקציות

We say that **f(n)**has order of growth **Θ(g(n))** if there are constants $c_1 \neq 0$ and $c_2 \neq 0$ such that for all **n** > 0 $c_1 g(n) \le f(n) \le c_2 g(n)$
במקרה זה נאמר כי g(n) הוא חסם הדוק אסימפטוטית עבור f(n).
We say that **T(n) ∈ O(f(n))** if there is a constant c ≠ 0 such that for all **n** > 0 **T(n) ≤ c f(n)**
במקרה זה נאמר כי g(n) הוא חסם אסימפטוטי עליון עבור f(n).
We say that **T(n) ∈ Ω(f(n))** if there is a constant c ≠ 0 such that for all **n** > 0 **c f(n) ≤ T(n)**
במקרה זה נאמר כי g(n) הוא חסם אסימפטוטי תחתון עבור f(n).
הסימון (f(n)o משמש להגדרת חסם עליון שאינו הדוק אסימפטוטית.
הסימון (f(n)ω משמש להגדרת חסם תחתון שאינו הדוק אסימפטוטית.
שיטות לפתרון נוסחאות נסיגה: הצבה- ניחוש ובדיקה, ניתן גם עם קבוע, למצוא הדוק ביותר, איטרציה, עץ רקורסיה, שיטת אב.

מבני נתונים

טבלאות דינאמיות
מערך עם פעולות הכנסה ומחיקה. כאשר התמלא השטח המוקצה, מוקצה שטח חדש (בד"כ כפול בנפחו), מועתקים אליו הערכים הישנים ושאר התאים מאותחלים.
עבור n פעולות הכנסה, סדר הגודל הוא 3n ולכן עבור פעולה יחידה סדר הגודל הוא 3 (Amortized).

עצים - כללי
אורך מסלול- מספר הקשתות בין זוג צמתים.
גובה העץ- אורך המסלול הגדול ביותר בין שורש העץ לאחד העלים.
עומק של צומת- אורך המסלול ממנה לשורש העץ.
אם לא הוגדר אחרת, לכל צומת בעץ יש מצביע להורה שלו (Null בשורש).

סריקות עצים
Preorder: DLR- בקר בשורש, סרוק את תת-עץ שמאלי, סרוק תת-עץ ימני.
Inorder: LDR -סרוק תת-עץ שמאלי, בקר בשורש, סרוק תת-עץ הימני.
Postorder: LRD- סרוק תת-עץ שמאלי, סרוק תת-עץ ימני, בקר בשורש.

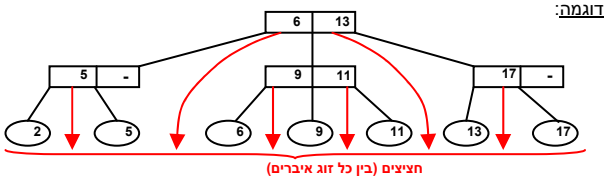
מימוש עצים
באמצעות מערך הורים- מעניקים לכל צומת מספר ומחזיקים שני מערכים. במערך הראשון ב A[i] נמצא המספר של ההורה של i ובמערך השני ב B[i] נמצא הערך של i (הנתונים). בעייתיות: לא ניתן לדעת את סדר הבנים (משמאל לימין או מימין לשמאל) ולכן יש להגדיר זאת מראש.
ע"י רשימות בנים- מחזיקים מערך עבור כל הצמתים בעץ כך שכל תא במערך מצביע לרשימה (רצוי בהקצאה דינאמית) של הבנים שלו.
ע"י מערך סמנים- לכל תא במערך יש ערך וכתובות של הבן הימני והשמאלי במערך (Null אם אין כאלו).
בהקצאה דינאמית- ידוע.

עץ חיפוש בינארי
לכל תת-עץ, כל הערכים בתת-העץ השמאלי קטנים מאלו של השורש וכל הערכים בתת העץ הימני גדולים ממנו.
מבנה זה רלוונטי רק כאשר יש יחס סדר בין האיברים.
סריקת Inorder של עץ כזה תציג רשימה ממויינת של האיברים בו.

```
TREE-MINIMUM (x)
1 while left[x] ≠ NIL
2 do x ← left[x]
3 return x

TREE-MAXIMUM(x)
1 while right[x] ≠ NIL
2 do x ← right[x]
3 return x

TREE-INSERT(T, z)
1 y ← NIL
2 x ← root[T]
3 while x ≠ NIL
4 do y ← x
5 if key[z] < key[x]
6 then x ← left[x]
7 else x ← right[x]
8 p[z] ← y
9 if y = NIL
10 then root[T] ← z
11 else if key[z] < key[y]
12 then left[y] ← z
13 else right[y] ← z
```

```
TREE-SUCCESSOR(x)
1 if right[x] ≠ NIL
2 then return TREE-MINIMUM (right[x])
3 y ← p[x]
4 while y ≠ NIL and x = right[y]
5 do x ← y
6 y ← p[y]
7 return y

TREE-DELETE(T, z)
1 if left[z] = NIL or right[z] = NIL
2 then y ← z
3 else y ← TREE-SUCCESSOR(z)
4 if left[y] ≠ NIL
5 then x ← left[y]
6 else x ← right[y]
7 if x ≠ NIL // ללא התניה בעץ אדום-שחור
8 then p[x] ← p[y]
9 if p[y] = NIL
10 then root[T] ← x
11 else if y = left[p[y]]
12 then left[p[y]] ← x
13 else right[p[y]] ← x
14 if y ≠ z
15 then key[z] ← key[y]
16 copy y's satellite data into z
17 return y
```

עצי 2-3
לכל צומת פנימי 2-3 ילדים, לכל העלים אותו עומק. האלמנטים הנשמרים בעלים מסודרים בסדר עולה משמאל לימין. בכל צומת פנימי שומרים את ערך המפתח הנמוך ביותר בתת-העץ האמצעי ואת הנמוך ביותר בתת-העץ הימני (אם קיים).
דוגמה:



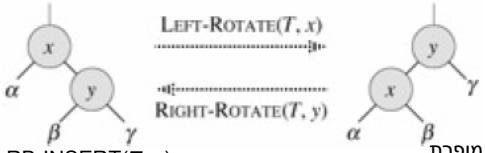
Insert
- v צומת חדש
- p צריך להיות אבא של v
- g אבא של p (סבא של v)
- הכנס v כבן נוסף של p
- אם כעת ל p שלושה ילדים - גמרת.
- אם ל p יש ארבעה ילדים:
  פצל p ל- p שמאל ו- p' מימין ושתול את p' כבן של g

Delete
v- צומת, p- אב, u- דוד.
- השמט v מ p
1) אם ל v שני אחים: השמט וסיים.
2) אם ל v אח בודד:
א) אם ל u שלושה בנים:
p יאמץ אחד מבני הדוד
ב) אם ל u שני בנים:
u יאמץ את האח של v ועכשיו p
ערירי ויש להשמיטו רקורסיבית

ייצוג קבוצות יעיל התומך בפעולות איחוד (זר) ו-Find ו-Insert.
האיברים ממוספרים 1, 2.. והקבוצות A, B …
1. נשמור מערך ובו בתא i נשמור את הקבוצה אליה I משתייך.
Find, Insert: O(1) Merge: O(n)
2. נשמור לכל קבוצה את גודלה ואת האיבר הראשון בה ולכל איבר נשמור את הקבוצה אליה הוא משתייך ואת האיבר הבא אחריו.
Merge: עוברים על איברי הקבוצה הקטנה ומשנים את הקבוצה שלהם לגדולה, את האיבר האחרון משרשרים לאיבר הראשון בקבוצה הגדולה ומשנים את מאפייני הקבוצה הגדולה בהתאם. מאחר וכל איבר שעובר לבעלים חדשים מגדיל לפחות פי 2 את הבעלים, כל איבר עובר לכל היותר lg(n) פעמים במהלך n פעולות Merge וכן הסיבוכיות לח פעולות כאלו היא O(n*lg(n)) .
3. נשמור עץ עבור כל קבוצה בו כל צומת מצביע לאביו (שהוא האיבר הבא אחריו) ובשורש העץ נמצא שם הקבוצה.
Merge: תולים שורש של אחת הקבוצות על השנייה. O(1).
Find: רצים כלפי מעלה. O(n).
אך אם נקפיד לתלות עץ קטן על עץ גדול נגדיל את עומק העץ ב-1 בכל תלייה ובכל תלייה מספר האיברים בקבוצה לפחות מוכפל. הסיבוכיות לח פעולות Find היא O(n*lg(n)).

עצים אדומים שחורים
עץ חיפוש בינארי "כמעט מאוזן" בו לכל צומת יש צבע שחור או אדום. ערכו של כל עלה הוא Null וצבעו שחור. לכל צומת אדום שני ילדים שחורים. כל מסלול מצומת מסויים לכל עלה הוא בעל אותו מספר של צמתים שחורים. bh(T) הוא הגובה השחור של עץ- מספר הצמתים השחורים מהשורש לעלה (לא כולל השורש).
משפט: גובהו של עץ אדום-שחור בעל n צמתים פנימיים הוא לכל היותר 2lg(n+1).

```
LEFT-ROTATE(T, x)
1 y ← right[x] □ Set y.
2 right[x]←left[y] //Turn y's left subtree into x's right subtree.
3 p[left[y]] ← x
4 p[y] ← p[x] // Link x's parent to y.
5 if p[x] = nil[T]
6 then root[T] ← y
7 else if x = left[p[x]]
8 then left[p[x]] ← y
9 else right[p[x]] ← y
10 left[y] ← x // Put x on y's left.
11 p[x] ← y
```



רוטציה: פעולת סיבוב השומרת על הסדר התוכי (Inorder) של העץ. כאשר מפעילים רוטציה שמאלית של צומת x, אנו מניחים שהבן הימני שלו y אינו Null. הרוטציה הופכת את y לשורש החדש של תת-העץ, את x לבן השמאלי של y ואת הבן השמאלי של y לבן הימני של x.
זמן ריצה: O(1).

הכנסה: O(lg(n)).
לאחר הכנסה רגילה של צומת לעץ התכונה היחידה שיכולה להיות מופרת היא שלצומת אדום יהיה בן אדום (זה שהכנסנו). במקרה זה מתקנים רקורסיבית כלפי מעלה לפי מקרים. מקרה 1 בו הדוד של הצומת שהכנסנו גם הוא אדום: צובעים את האב והדוד בשחור, את הסבא באדום וממשיכים. במקרה 2 בו הצומת שהכנסנו הוא בן ימני נבצע רוטציה שמאלית לאביו. במקרה 3 בו הצומת שהכנסנו הוא בן שמאלי של אביו נבצע רוטציה ימנית לסבו.
מחיקה: יש למחוק כמו בעח"ב. אם מחקנו צומת אדום- אין בעייה, אך אם מחקנו (בפועל) צומת שחור ייתכן ויש הפרה ואז נבצע תיקון.

```
RB-INSERT(T, z)
 Tree-Insert(T,z)
 Color[z] <- Red
1 while z =/ root and color[p[z]] = RED
2  do if p[z] = left[p[p[z]]]
3  then y ← right[p[p[z]]]
4   if color[y] = RED
5    then color[p[z]] ← BLACK // Case 1
6         color[y] ← BLACK // Case 1
7         color[p[p[z]]] ← RED // Case 1
8         z ← p[p[z]] // Case 1
9    else if z = right[p[z]]
10           then z ← p[z] // Case 2
11             LEFT-ROTATE(T, z) // C.2
12        color[p[z]] ← BLACK // Case 3
13        color[p[p[z]]] ← RED // Case 3
14        RIGHT-ROTATE(T, p[p[z]]) /C3
15  else (same as then clause
        with "right" and "left" exchanged)
16 color[root[T]] ← BLACK
```

Maintaining Dynamic Tree with Min Subject to Constraints
כאשר רוצים לבצע שאילתות לפי שני מפתחות (למשל ציון וגובה) נשמור שני עצי 2-3, בכל אחד מהם נשמור מפתחות שונים (באחד גבהים ובשני ציונים) ונשמור מצביעים בין הרשומות (בין הציון של תלמיד לגובה שלו למשל, נניח כי הציונים ייחודיים).כשנרצה לחפש למשל את הציון המינימלי של התלמידים החל מגובה מסויים נסרוק את תת-העץ של התלמידים מהגובה הרצוי (קל למצוא) ומציץ כלפי מעלה את הציון המינימלי שלהם. כל פעולה לוקחת O(lg(n)).

Splay Trees
עץ חיפוש בינארי בו פעולה בודדת לוקחת O(n) אבל m פעולות לוקחות O(mlg(n)) מבנה זה יעיל לשימוש כאשר הסיכויים לגישה חוזרת גבוהים.

ערימה מערך המייצג עץ בינארי "כמעט" שלם: העץ מלא לחלוטין פרט אולי לרמתו האחרונה, המלאה משמאל ועד לנקודה מסויימת. ידועים גודל המערך וגודל הערימה. Parent(i)=[i/2], Left(i)=2i, Right[i]=2i+1. הערך בכל צומת קטן או שווה מהערך של ההורה.

Heapify- פעולה המקבלת מערך וצומת i כך שהבנים של i הם ערימות חוקיות ויוצרת ערימה חוקית בצורה רקורסיבית כלפי מטה (Θ(lg(n)).

בניית ערימה- סורקים חצי המערך העליון ומבצעים Heapify. זמן הריצה הוא לכאורה (Θ(nlg(n)) אך ניתן למצוא חסם הדוק יותר והוא (Θ(n).

ExtractMax- מוחקים השורש, במקומו שמים את הקטן ביותר ומבצעים heapify.

שיטות מיון – מודל ההשוואות במודל ההשוואות מתארים את המיון כעץ החלטה לפי יחס הסדר בין האיברים. העלים בעץ הם הסדרים האפשריים (עבור n איברים יהיו !n סדרים- עלים).משפט: גובהו של עץ החלטה הממיין n איברים הוא (Ω(nlg(n)), כלומר לא ניתן למיין בזמן ריצה טוב יותר מ- (Θ(nlg(n, אלא עד כדי קבוע.

הוכחה: עץ בינארי מכיל לכל היותר 2^h עלים, מכאן 2^h=>!n ומכאן h >= lg(n!) ומאחר ו- lg(n!)>=n/2lg(n/2) נובע h >= nlg(n).

לפי מודל זה, החסם התחתון לעץ השוואות בעל n עלים הוא (Ω(lg(n)).

מיון הכנסה – ($\Theta(n^2)$)

בכל איטרציה i האיברים 1…i ממויינים ע"י החלפות.

מיון בחירה – ($\Theta(n^2)$)

בכל איטרציה i שמים במקום ה-i את האיבר הקטן מבין האיברים n…i.

מיון ערימה – Θ(nlg(n))

```
HEAPSORT(A)
1 BUILD-MAX-HEAP(A)
2 for i ← length[A] downto 2
3   do exchange A[1] ↔ A[i]
4   heap-size[A] ← heap-size[A]-1
5   MAX-HEAPIFY(A, 1)
```

מיון מיזוג – O(nlg(n))

מחלקים את המערך לשני חלקים שווים, ממיינים כל חלק רקורסיבית וממזגים.

מיון מהיר – WC – $O(n^2)$ Avg, O(nlg(n))

בוחרים Pivot ומחלקים את המערך לערכים גדולים ממנו וקטנים ממנו (Partition) וממשיכים רקורסיבית. Partition(i,j) לוקח (O(j-i+1 מאחר ומבקרים בכל איבר פעם אחת. המקרה הגרוע הוא זה בו מחלקים את המערך לחלקים בגודל 1 ו- (n-1):

$T(n)=T(n-1)+\Theta(n) \in \Theta(n^2)$, המקרה האידאלי הוא בו מחלקים את המערך תמיד לשני חלקים שווים: $T(n)=2T(n/2)+\Theta(n) \in \Theta(n\lg(n))$. כל Partition ביחס קבוע (למשל 1:9) ייתן זמן ריצה של Θ(nlg(n)), זהו המקרה הממוצע.

שיטות מיון נוספות

מיון מנייה (Count) – O(n+k)

אלגוריתם למיין n מספרים בתחום 1..k: למיין מערך A בגודל k נסרוק אותו ונשמור במערך C בגודל k את מספר המופעים של כל איבר בתחום. נסרוק את C ונצבור בכל תא את מספר האיברים הקטנים ממנו. נסרוק שוב את A ועבור כל איבר נכניס למערך B בגודל n את האיבר Am לפי המיקום עליו מורה C ונפחית 1 מ-C.

מיון בסיס (Radix) – O(d*n)

ממיין n מספרים בני d (קבוע) ספרות כל אחד. בכל שלב ממוינים המספרים בשיטת מיון "יציבה" (למשל Count Sort) לפי ספרות החל בLSD.

דוגמה למיין n איברים בתחום n^2-1: מספר הספרות של המספרים בתחום הוא לכל היותר $\log_n n^2$=2, נייצג כל מספר כזה במערך בגודל 2 ונבצע Radix עם שימוש ב CountSort. סה"כ: $O(n)=O(n+n) * O(\log_n n^2)$

O(n) – Bin Sort / Bucket Sort

מחלקים את הקלט במערך בן n האיברים ל-n קבוצות. נעזרים במערך B של n רשימות מקושרות וסורקים את A כדי להכניס כל איבר בו לרשימה המתאימה (הקבוצה) ב-B. לבסוף ממיינים את רשימות B בעזרת מיון הכנסה. טוב למחרוזות באורך קבוע או תחומים שידוע עליהם אינפורמציה, למשל: n^k…1.

Order Statistics

ככלל, נוכל למיין מערך ולמצוא את האיבר ה-i בו ב(O(nlg(n.

שיטה נוספת היא לחלק את המערך באמצעות Partition אקראי ולחפש את האיבר בחלק המתאים. בתוחלת O(n), WC, O(n2).

מציאת statistic ב- O(n) worst case

בחר אלמנט i מתוך n איברים: (select)
1) חלק האיברים ל n/5 קבוצות של 5 איברים
2) מצא את החציון של כ"א מהקבוצות (מיון פשוט)
3) השתמש רקורסיבית ב select למציאת חציון החציונים
4) בצע partition מסביב לחציון החציונים
יהי k מספר האיברים בחלק הנמוך
יהי n-k מספר האיברים בחלק הגבוה
5) השתמש ב select רקורסיבית למציאת i בחלק הנמוך או i-k בחלק הגבוה

טבלאות Hash

מבנה נתונים טוב לייצוג קבוצות, בממוצע זמן קבוע, תחליף לוקטור ביטים כאשר התחום גדול.

Hash פתוח

מחזיקים מערך בן B "סלים" (תאים) כך שכל תא מצביע לרשימה מקושרת ממויינת. הגישה לכל סל היא O(1) והחיפוש ברשימה הוא לינארי. פונקציית Hash h(x) היא פונקציה המתאימה לכל ערך x את הסל המתאים לו בין 0 לבין (B-1).

עבור N איברים ו-B סלים יהיו בממוצע N/B (מקדם העומס- α) איברים בכל סל ועבור פונקציה שמפזרת בצורה אחידה (ההסתברות של מפתח אקראי ליפול בכל תא היא 1/B) חיפוש (כושל או מצליח) אורך בממוצע Θ(1+α) (1 עבור חישוב הפונקציה ו-α עבור סריקת הרשימה). אם נבחר B ביחס ישר למספר האיברים: N = O(B) אזי α=1 ואז פעולת החיפוש לוקחת O(1).

דוגמאות לפונקציות: עבור 1<k>=0: h(k)=[kB], עבור k שלם: h(k)=k mod B (רצוי B ראשוני).

Hash סגור

מחזיקים מערך בן B "סלים" ללא רשימות מקושרות. אם הפונקציה גרמה להתנגשות מחשבים פונקציות נוספת $h_1(x)..h_n(x)$. למשל: $H_i(x)=(H(x)+i) \bmod B$

בחיפוש איבר נסרוק את התאים לפי $h_1(x), h_2(x)$… ועד שנגיע למקום ריק ואז נדע שהאיבר לא נמצא. (יש לשמור ערך עבור "ריק" ו- "deleted").

Hash אוניברסלי

אם רוצים למנוע ממשתמש "זדוני" לחבל ביעילות, נבחר בכל פעם ובאופן אקראי את הפונקציה שלנו מתוך קבוצה אוניברסלית- H- קבוצה המבטיחה לנו: H = { f | f: U -> {0,…, B-1} }

ולכל x,y∈U מתקיים |H|/B = | { h ∈ H : h(x) = h(y) } | כלומר אם פונקצייה נבחרת באופן אקראי Hm אוניברסלית אז הסיכוי להתנגשות של x ו-y הוא 1/B. קבוצה 2-אוניברסלית: לכל 2 מפתחות שונים x,y ופונקציה h הנבחרת רנדומלית מתוך H אז הסדרה <h(x),h(y)> היא כל אחת מm^2 הסדרות האפשריות בהסתברות שווה. כל קבוצה 2-אוניברסלית היא אוניברסלית. לא כל קבוצה אוניברסלית היא 2-אוניברסלית.

Expected Average Case	Expected Worst Case	Open Hash
O(1)	O(logn)	Search(Key)
O(1)	O(logn)	Insert(Key)
O(1)	O(logn)	Delete(Key)
O(n)	O(n)	Minimum()
O(n)	O(n)	Maximum()

משפט: אם h נבחרת מאוסף אוניברסלי של פונקציות hash ומשומשת להכניס N מפתחות לתוך טבלה בגודל B כך ש- N <= B, אזי תוחלת ההתנגשויות הקשורות במפתח x קטנה מ 1.

Perfect Hash: בניית הטבלה בתוחלת O(n), פעולת Find עולה WC O(1).

$1=O(n^{1/\log n}) = o(\log(\log n)) = o(\sqrt{\log n}) = o(\log n) = o(\log^2 n) = o(n^{1/3})$

$= o(n) = o(2^{\log n}) = o(n^2) = o(\log n)! = o(\log n)^{\log n} = o(n^{\log(\log n)}) =$

$o(3/2^n) = 0(2^n) = o(n*2^n) = o(n!)$

$\log(n!) = \Theta(n*\log n) \quad n\log n = o(n^{1.5}) \quad \log_b n \in \Theta(\lg n)$

$\sum_{i=1}^{\log n} 1/i = \Theta(\log(\log n)) \quad \sum_{i=0}^{\infty} x^i = 1/(1-x) \quad \sum_{i=1}^{n} \sqrt{i} = \Theta(n*\sqrt{n}) \quad \sum_{i=1}^{n} i^k = \Theta(n^{k+1})$

$\sum_{i=1}^{n} 1/i = \Theta(\log n) \quad \sum_{i=1}^{n} \log i = \Theta(n \log n)$

סדרי גודל:

$\log n$

$\log^2 n$

$[\log(\log n)]^{\log(\log n)}$

$2^{\log_3 n}$

$(\log n)^{\frac{\log n}{\log(\log n)}}, n$

$n\log n, \log(n!)$

$n^{\frac{2n+1}{n+1}}, n^2, \frac{n^4+1}{n^2+1}$

$2^{\log^2 n}$

$n^{2\log n}$

2^n

3^n

$n!$

$(n+1)!$

n^n

$T(n)=2T(\frac{n}{2})+\sqrt{n}=\theta(n)$

$T(n)=T(\frac{n}{2})+T(\frac{n}{7})+n=\theta(n)$

$T(n)=T(n-1)+\frac{1}{n}=\theta(\log n)$

$T(n)=T(\frac{n}{3}-4)+1=\theta(\log n)$

$T(n)=9T(n/3)+n\in\theta(n^2)$

$T(n)=T(2n/3)+1\in\theta(\lg n)$

$T(n)=3T(n/4)+n\lg n\in\theta(n\lg n)$

$T(n)=T(n/2)+c\in\theta(\lg n)$

$T(n)=T(n-1)+\lg n\in\theta(n\lg n)$

$T(n)=2T(n/2)+\theta(n^{0.6})\in\theta(n)$

$T(n)=T((n/3)-4)+1\in\theta(\lg n)$

$T(n)=2T(\frac{n}{2})+\theta(n)=\theta(n\log n)$

$T(n)=2T(\frac{n}{2})+\theta(1)=\theta(n)$

$T(n)=\sqrt{n}T(\sqrt{n})+n=\theta(n\lg(\lg n))$

$T(n)=2T(\lfloor\sqrt{n}\rfloor)+\lg(n)=\theta(\lg(n)(\lg\lg n))$

$T(n)=3T(\lfloor n/4\rfloor)+n\in\theta(n)$

$T(n)=2T(n/2)+n^2\in\theta(n^2)$

$T(n)=T(n/3)+T(2n/3)+n\in\theta(n\lg n)$

$T(n)=T((n/3)-4)+1\in\theta(n)$

$T(n)=T(n-1)+\theta(n)=\theta(n^2)$

$T(n)=T(3n/4)+T(n/4)+\theta(n)=\theta(n\lg n)$

$T(n)=T(9n/10)+T(n/10)+n\in\theta(n\lg n)$

$T(n)=T(n/5)+T((7n/10)+6)+\theta(n)\in\theta(n)$

$T(n)=T(n/2)+T(n/7)+6n\in\theta(n)$

$T(n)=T(2n/3)+T(n/3)+n\in\theta(n\lg n)$

סכום סדרה חשבונית- $n/2[2a_1+(n-1)d]$ **סכום סדרה הנדסית-** $\frac{a_1(q^n-1)}{q-1}$

$\log_x y = \log_z y/\log_z x$; $z^{\log_x y}=y^{\log_x z}$; $\log_x y=1/\log_y x$; $x^{\log_x y}=y$

$\log xy=\log x+\log y$; $\log x/y=\log x-\log y$; $\log 1=0$; $a^b=x \Leftrightarrow \log_a x=b$

Amortized = O(פעולות הגרועות ביותר) / n (n פעולות)

AVL – עץ בינארי בו ההפרש בין גובה כל תת-עץ ימני לשמאלי הוא לכל היותר 1.

גובה עץ של רקורסיה מהצורה $T(n)=cT(an/b)$ הוא $\Theta(\lg_{b/a} n)$

מבנה הנתונים	Insert	Del	Find	Last	Prev	Min	Max	Succ
רשימה ממומשת במערך	O(n) WC	O(n) WC	O(n) WC	O(1) WC	O(1) WC	O(n) WC	O(n) WC	O(n) WC
רשימה **מקושרת** ממומשת במערך	O(1) WC	O(1) WC נתון מצביע	O(n) WC	O(n) WC	O(n) WC	O(n) WC	O(n) WC	O(n) WC
רשימה **מקושרת** בהקצאה דינאמית	O(1) WC	O(1) WC נתון מצביע	O(n) WC	O(n) WC	O(n) WC	O(n) WC	O(n) WC	O(n) WC
רשימה דו-מקושרת (כאשר נתון רק מצביע לתחילת הרשימה)	O(1) WC	O(1) WC נתון מצביע	O(n) WC	O(n) WC	O(1) WC	O(n) WC	O(n) WC	O(n) WC
מחסנית בהקצאה דינאמית	O(1) WC	O(1) WC	O(n) WC	-	-	O(n) WC	O(n) WC	O(n) WC
תור בהקצאה דינאמית-מחזיקים מצביע להתחלה ולסוף	O(1) WC	O(n) WC	O(n) WC	-	-	O(n) WC	O(n) WC	O(n) WC
תור מעגלי ממומש במערך	O(1) WC	O(1) WC	O(n) WC	-	-	O(n) WC	O(n) WC	O(n) WC
עץ חיפוש בינארי	O(h) WC	O(h) WC	O(h) WC	-	-	O(h) WC	O(h) WC	O(h) WC
עץ 2-3 (B-Tree)	O(lg n) WC	O(lg n) WC	O(lgn) WC	O(lgn) WC	-	O(lgn) WC	O(lgn) WC	O(lgn) WC
עץ אדום-שחור	O(lg n) WC	O(lg n) WC	O(lgn) WC	O(lgn) WC	-	O(lgn) WC	O(lgn) WC	O(lgn) WC
עץ Splay	O(lg n) Amort	O(lg n) Amort	O(lgn) Amort	O(lgn) Amort	-	O(lgn) Amort	O(lgn) Amort	O(lgn) Amort

h הוא גובה העץ. כאשר הערך מאוזן: h=lg(n), הפעולות בעת"ב הן O(n) בWC אבל O(lg(n)) בממוצע.