```
i = n
while (i > 1)
    i = i - 1   || i = i - k
```

סיבוכיות זמן $\Theta(n)$ (מחיר פקודה)

```
i = n
while (i > 1)
    i = i / k
```

סיבוכיות זמן $\Theta(\log_k n)$

```
i = 1
while (i < n)
    i = i * 2
```

```
i = 1
while (i < n)
    i = i * k
```

סיבוכיות זמן $\Theta(\log_2 n)$

```
for (i = 1 to n)
    for (j = 1 to i)
        O(1)
```

סיבוכיות זמן $\Theta(n^2)$

```
for (i = 1 to n)
    for (j = 1 to n)
        O(1) סיבוכיות
```

```
i = n
while (i < n)
    i = i^k
```

```
i = 2
while (i < n)
    i = i * i
```

סיבוכיות זמן $\Theta(\log(\log(n)))$

$$\sqrt{\log n} < 1 < \log n < (\log n)^k$$

$$< (\log n)^{k+1} < n^\varepsilon < \sqrt{n} < n$$

$$< n \log n < n^k < n^{k+1} < 2^n$$

$$< n! < n^n$$

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

הוא $i^2 = i$ כך ש $\log_3$
בקלף שעליו כתוב n
נוסף בזמן $\log n$

$$c < \ldots < \log(\log(n)) < \log(n)$$

$$< n^k \quad (k \geq 1) < n < n\log(n) < n^2$$

$$< n^3 < \ldots < 2^n < 3^n < \ldots$$

$$< n! < n^n$$

רשימה מקושרת - linked list

* כל איבר ברשימה מכיל:
  * מידע (כלל שדות) * מצביע לאיבר הבא

פעולות בסיסיות:

- create list() - מאתחל רשימה ריקה.
- is empty(L) - מחזיר אמת אם הרשימה L ריקה.
- find(L, k) - מחזיר את האיבר שאפתח k ברשימה L.
- insert first(L, k) - מכניס איבר x בתחילת הרשימה L.
- insert(L, x, y) - מכניס איבר x אחרי האיבר y ברשימה L.
- delete first(L) - מוחק את האיבר שבתחילת הרשימה L.
- delete(L, x) - מוחק את האיבר x מהרשימה L
- delete after(L, y) - מוחק את האיבר שאחרי האיבר y ברשימה L.

תור Queue

* תור הוא מבנה נתונים כזה:

- מוגבל בצד אחד שנקרא head/front ובצד שני נקרא tail/rear שבו ניתן להכניס איברים חדשים לתור
- שמוציא אלמנט שנמצא הכי זמן בתור FIFO

פעולות על תור:

- יצירת תור makeEmpty()
- בדיקה אם התור ריק isEmpty()
- הכנסת איבר חדש לסוף התור enqueue()
- שליפת איבר הראשון מהתור value = dequeue()
- הצצה לאיבר הראשון בלי שליפה value = peek()

מבוא stack

* מבנה נתונים שבו ניתן להוסיף ולהוציא איבר אך ורק מקצה אחד של המבנה.

- מבנה נתונים שבו איבר שנכנס ראשון יוצא אחרון. FIFO או LIFO.

פעולות של מבנה:

- הכנסת איבר, נקרא לפעמים push(x)
- מציצה שבו מחזיר את האיבר שנמצא בראש המבנה top() -> value
- הוצאת איבר, נקרא גם מחזיר את הערך pop() -> value
- איפוס המבנה makeEmpty()
- מציצה שבו מחזיר האם המבנה ריק isEmpty()

שגיאת סוג ראשון: $\log(n)\Theta = 1 - (1+n)\log = \eta$

חישוב הגודל הקריטי של n בהינתן

n - הסתברות לפחות טעות אחת: $1 - 2^{-\frac{\text{מספר הבדיקות}}{1+\eta}}$ גודל הטבלה.

* הערה חשובה: לפי הנוסחה, ככל שגודל הטבלה

גדל

- לפי הניסוי, ערך η הוא בערך 2 בקירוב.
- לפי הסבר - לפי שהוא משמעות מספר הבדיקות.
- לעומת זאת - ערך ממוצע מספר הבדיקות בפועל
- יכול לגדול.
- לעומת לפי - ערך ממוצע מספר הבדיקות בהינתן שהמפתח
- אינו קיים (ערך ברירת המחדל - 1).
- ערך η - ערך ממוצע מספר הבדיקות בהינתן שהמפתח
- קיים - לעומת זאת לא מלא.
- לעומת -

* הערה: לפי הנוסחה, ככל שגודל הטבלה

מקסימה: $2^{h+1}-1$

מינימה: $N(h) = N(h-1) + N(h-2) + 1$

* עץ AVL עם n קודקודים בגובה h:

$$\log_{\phi}(n+2) - 2 \le h \le 1.44\log_2(n+2)$$

* עץ AVL הוא עץ חיפוש בינארי שמקיים את תכונת הגובה $\Theta(\log n)$ ולכן פעולת החיפוש לוקחת $O(\log n)$.

* מע"ד עץ בינארי מאפשר לרשימה מצביע ל-NIL.

סריקת inorder (תוכית) עץ חיפוש בינארי מדפיסה את המפתחות לפי סדר עולה.

* עץ חיפוש בינארי שמקיים שכל קודקוד פנימי שווה לקודקוד הבא אחריו בסריקת inorder.

- סריקה סופית (postorder) - שמאל, ימין, שורש.
- סריקה תכונית (preorder) - שורש, שמאל, ימין.
- סריקה רוחבית (inorder) - שמאל, שורש, ימין.

אלגוריתם שמוצא את הקודקוד הבא בסריקת inorder.

* סריקה עץ חיפוש בינארי (בינרי) עץ כל הקודקודים

* עץ TAV - כל שפה יוצר TAV בלי שגיאה מעגל יהיה 1-10/1.

* סעיף צמתון ו"בר הולכים ימינה וכל רק שמאלה ו"ו שמאלה וכל ימינה (מסדר מלמעלה ו"ו למטה).

* שגיאה ו"ו/ו-מפה ל"ו שגיאה בין עץ הפעולה. בדיקה עץ ומ'.

- שלו"ו כל, אפשר עץ ב TAV (נקרא גיא מפסל גיא)
  - RR - הבן משמאל אחרי הראשון
  - TR - בלבד משמאל הראשון
  - TT - הבן/משמאל אחרי הראשון
  - RT - בלבד משמאל הראשון

כאשר אנו שקולים עם הולכים אחרי הראשון שמצא ו/ו מלבד ו/ו הולכים ב בלבד את"ו הולכים שמאלית ואחרי"כ ל' הכוון, ו/ו עם TT/RR הבן משמאל אחרי הראשון, ו/ו עם TR/RT בלבד משמאל אחרי הראשון.

* בנ"ד עץ TAV מעגל אחד/ הפך מס

* אם מספרי ואי/אפ: $N(0) = 1$, $N(1) = 2$

$$N(n) = N(n-1) + N(n-2) + 1$$

* נראה באופן אחד ויכה עץ מס

ערימה - Heap

מספרו של האיבר בקודקוד i

בניו של האיבר באינדקס $2i$, $2i+1$

$A[i]$ = איבר

$A[2i]$ = בן שמאלי. (ואפשר קודקוד i כך $2/i$)

$A[2i+1]$ = בן ימני.

* הבדיל שמתקיים-הפוכה בצד אחד

* מוגדר מלמטה למעלה בשיא

* שורש הערימה $A[0]$

מעלה שאפשר לעלות בעץ הערימה:

* שלוקח ניתוח הערימה heapify (מתרץ $O(\log n)$)

* בונים ערימה מתוך Heap Building

הקוד

```
BuildHeap(A)
    heap-size ← A.length
    for i ← ⌊A.length/2⌋ downto 1
        heapify(A, i)
```

סופי מתן ריבוי (n)

* מתן ריבוי מיון הערימה ($n\log n$)

(נוסח הגרסה האחרונה).

אלו האופנים השונים שבהם אלגוריתם $n \log n$ עלול להתנהג
(נוסח אחר) כפי שהוא מופיע בהרצאה? (נתחיל מהקל
כלפי הקשה, ונסיים בשאלה שבה נדרש ניתוח מדויק)

$\Theta(n \log n) = \log(n!)$

ועל $\Omega(n \log n)$ כלשהו?

כלומר, נסיים עם שני כיוונים, שעל כל אחד מהם
ועל $n \log n$

תזכורת: נוסח הגרסה האחרונה שהופיעה
בכיתה עם שני כיוונים ועל $\log(n!)$
כ.ז. אלו אחת הפונקציות שבהן נתקלנו עד כה.
ובאלו אופנים אלגוריתם מיון יכול לנהוג אחרת
בהשוואה

* מיון ערימה $O(n \log n)$
* ערמת מינימום $[1]$
* ערמת מקסימום $(\log n)$
* בניית ערימה מתוך מערך $(n \log n)$
* כל פעולת ערימה $O(h)$, $(n \log n)$ Heapify Insert
* ניתן לחפש איבר בערמה (אבל לא יעיל)
* $n-1, \frac{n}{2}+1$: עלים/פנימיים

* בערמת מקסימום (כלל הערימה) כל צומת הוא גדול מצאצאיו (מימין ומשמאל) בניגוד לעץ חיפוש בינארי שבו כל איברי תת העץ השמאלי קטנים מהשורש ואיברי תת העץ הימני גדולים מהשורש. לכן בניגוד לשם זה לא מאפשר חיפוש יעיל של איבר בערמה, ויותר מזה, לא ניתן לדעת אם איבר נמצא בערמה או לא בלי לעבור על כל איברי הערמה (אלא אם כן מדובר בעלה).

כאשר גודל הקלט משתנה משמעותית - האלגוריתם לא יעיל.

n = כמות האיברים, x = גודל האיבר

למה כדי לייצג את כמות האיברים

כלומר, צריכים את האלגוריתם כאשר גודל האיברים

קבוע יהיו צריכים ביטים את האלגוריתם כפי.

כך, צריכים את האלגוריתם ב k ביטים גודל את גודל האיברים.

ומניחים את הסיבוכיות שתלוי בגודל(ביטים).

וגם נסמנים גודל כמה פעמים (k גודל מספר).

גודל האלגוריתם: נקרא מלא מלא (מורכב מגודל

האלגוריתם את מבוצע כמות אבל למה יבוא.

* אלגוריתם - ממן ריבוי $O(n^2+n)$

אלגוריתם פרופורציה:

האלגוריתם מבוצע כמות ובכל למה את יבוא:

* אלגוריתם מטריצה - ממן ריבוי $O(nlog(n))$ (מחלקה צורך n)

האלגוריתם מבוצע כמות ובמה את יבוא.

* אלגוריתם פרימה - ממן ריבוי $O(nlog(n))$

האלגוריתם את מבוצע כמות ובמה יבוא.

* אלגוריתם מיזוג - ממן ריבוי $O(n^2)$

אלגוריתם

* אין כמעט (explicit) נוסחה שמחשבת כמה ראשוניים יש.
  המחקר הזה לגבי מספרים קט'.
* אין ידוע האם אין שאינם לפחות סוף בין
  שני מספרים טבעיים n ו $(n+1)!$
* ישנה הוכחה לאי-קיום ראשוניים בטווח סגור
  שאורך n כלשהו $(n+1)!+2$ ועד $(n+1)!+n$
* נניח, כי n הוא הגדול ביותר, כך שכל p_i ראשוני
  $(n!)^2 < n! \cdot 2^{n!} \Leftarrow n! < 2^{n!} = f(2^n)$
  ישנו 2^n מספרים טבעיים עד $(n!)^2$ כך ש
  ישנה לפחות חזקה אחת של ראשוני עד n.
* אין סוף (כמו אין) סוגים שונים של אינסוף.
  שני אינסופים שונים נקראים שווי עוצמה
  אם יש ביניהם התאמה חד-חד ערכית ועל.
  למשל קטע $[0,1]$ ומספרים ממשיים חיוביים.
* אין בין $\mathbb{Z}$ - לעומת $\mathbb{R}$ (משיקולי אלכסון).
  אין בין $\mathbb{Q}$ לעומת $\mathbb{R}$ (משיקולי אלכסון).
* אין סוף - אין רציונלי לעומת $(n+1)!$
  אין סוף - אין אי-רציונלי לעומת $(n+1)!$
  n = סוף המספרים, x = מספר שלם (חיובי)
  p = סוף המספרים הטבעיים x. (כמו שהראינו קודם).
  כך, שבקטע $[x, x+n]$ אין אף מספר טבעי.
  $(n+1)!+x$ ... $(n+1)!+2$ $(n+1)!+1$
  אין אף מספר משני

אלגוריתם בניית ערימה מבנה נתונים מסוג מערך
שיטה: ג' לולאה לפי גודל ה"ב בסדר הפוך מסוג
על גובה $O(n)$
פי גלה, וכך סדה גודל / מילוי מערך ומערך.
מערכת שכלל את מלא מסוגים שונים גם
בניה ה"ב ממלך down - going
על גובה (מסוג) O (כיוון עץ)

לדוג'
ומה שלילי הוא לא מתלבש כך וזוכר
ומלבנית, תא בוחרים וזה שלא שם אין הוא
ת מבקרת fixdown, מגלים וזה ומה לא
שה"ב לו שלילי (סוגר ומסו לכן אלו)
- שלבים ב fixdown - מעבר שלוי ה"ב, מה
- עץ: מעלה עם
- כך וזוכר בעץ שלו, כלו
- $\lfloor \frac{n}{2} \rfloor$ ומסוגר ומלבנית,
* ה"ב מבנה
- ה"ב - O כלו

- פתיחה לתיבה מינימלי שווה ל-0

* הכנסת איבר - מוסיפים בסוף הערימה ומבצעים heapify

* הוצאת מקסימום - מוחקים את השורש ומעבירים אליו את האיבר האחרון בערימה ומבצעים heapify

סה"כ זמן ריצה $O(\log n)$

- בניית ערימה - מתחילים מהעלה הראשון שיש לו בן ומבצעים heapify עד השורש.

סה"כ זמן ריצה $O(n)$,

הוכחה בהרצאה.

- מיון ערימה - מוציאים את האיבר הכי קטן (השורש) ומחליפים אותו עם האיבר האחרון בערימה. מבצעים heapify על השורש - מעלים את האיבר הגדול ביותר לשורש.

* תור קדימויות/עדיפויות Priority Queue

הכנסת ומחיקת איבר מקסימלי.

סיכום מיון

	מיון	רעיון	סיבוכיות ממוצע	במקום Inplace	יציב stable
מיון השוואתי Comparison	Bubble sort מיון בועות	משווים זוגות סמוכים ומחליפים במידת הצורך	$O(n^2)$	✓	✓
	Insersion sort מיון הכנסה	כמו משחק שולי קלף MMI.	$O(n^2)$	✓	✓
	Selection sort מיון מחירה	מוצאים מינימום ומחליפים עם האיבר הראשון, ואז עם השני וכו'	$O(n^2)$	✓	✓
	Merge sort מיון מיזוג	מחלקים את המערך לשניים, ממיינים כל חלק ומאחדים בשלבים	$O(n\log n)$	לא במקום	✓
	Quick sort מיון מהיר	בוחרים ציר pivot ומחלקים סביבו	ממוצע $O(n\log n)$ גרוע $O(n^2)$	✓	✗
	Heap sort מיון ערימה	משתמשים heapify וכל הערימה	$O(n\log n)$	✓	✗
מיון לא השוואתי Non Comparison	Counting sort מיון מניה	מערך מונים באורך $[1,k]$ לכל ערך בטווח המספרים	$\Theta(n+k)$	$\Theta(n+k)$ לא במקום	✓
	Radix sort מיון ספרות	ממיינים לפי הספרות מהנמוכה לגבוהה: משתמשים ב p פעמים ב $O(n+b)$ כאשר b בסיס	$\Theta(d(n+b))$	$\Theta(n+b)$ לא במקום	✓
	Bucket sort מיון דליים	מחלקים את המספרים לדליים שווים $[0,1)$	ממוצע $O(n)$ גרוע $O(n^2)$	$\Theta(n)$ לא במקום	✓

* אילו על ידי - אור אי.אי.
* אילו על ידי מחלה - עלייה באינזין
* אחד - אינזין אור אי.אי = (מסוים) O
* בבעל - הלוואת מקבל בעיה = $O(n^2)$
אילו - סגר, זקף:
* אי.אי ילדי ממוצע מיליונרים $1+\frac{2}{n}$ ל- n.
* אילו מאבד כלפי עובדה (ין) סוכן ראשי של)
* אינזין (בק. אילן) לוקח סא זקף אם (ין)O
- מחלה ריבה משפחתו:

מסקנה: $1-2^{-(1+n)}$

אינדוקציה: $(1+(2-n))N + (1-n)N = (n)N$

* של 7AV אם בעיה כלפי בראשי n:

$$\log\Theta = 1-(1-n)\log = n$$

* נשוב ראשי של ב"א, אם:

כאשר מליונרים $n = 1-2^{-(1+n)}$

* אם שבעים כלפי מילי, אם בראשי n:

* $n\Theta = n^{\log 2}$

* $(n)\log\Theta = n\log n$

$$\text{סכום סדרה גאומטרית} = (\text{איבר ראשון} + \text{איבר אחרון}) * \frac{\text{מס' האיברים}}{2}$$

* סכום של סדרה גאומטרית:

סכום

* חיפוש לא יעיל $O(n)$
* הוספה לא יעילה $O(n)$
* הוספה יעילה $O(\log n)$ אך מחיקה $O(n \log n) = O(\log n!)$
* מחיקה יעילה $O(\log n)$ אך מחיקה לא יעילה $O(n)$
* ניתן לממש בעזרת מערך, קל יותר לממש מאשר רשימה מקושרת
* ממשק של מחסנית רשימה מקושרת $O(n)$

ממשק של ערימה:

Heapify-I , Insert , Delete = $O(\log n)$

Build-Heap = $O(n)$, Maximum-Heap = $O(1)$

* תור - אם נממש בעזרת מערך יעיל.
* חיפוש בתור, JAVA-קל יותר במימוש $O(\log n)$
הכנסה / מחיקה / איברים מקומיים / חיפוש וכו'.
Insert , Delete = $O(1)$, Search = $O(n)$
* רשימה מקושרת - הכנסה ומחיקה $O(1)$
IsEmpty , Enqueue , Dequeue כל $O(1)$
* תור - כל הפעולות לוקחות $O(1)$ (מבנה FIFO)
Push , Pop , IsEmpty כל $O(1)$
* מחסנית - כל הפעולות לוקחות $O(1)$ (מבנה LIFO)

סיכום: