הוכן ע"י אמיר רובינשטיין

מבני נתונים ומבוא לאלגוריתמים

נושא 5

תורי קדימויות ומיון ערימה
Priority queues and heap-sort

בתוכנית

פרק 6 בספר הלימוד

- נכיר את מבנה הנתונים ערימה (heap) ונכיר שני שימושים חשובים שלו:
  - תור קדימויות (priority queue)
  - האלגוריתם מיון-ערימה (Heap-Sort)

תור קדימויות (priority queue)

תור קדימויות הוא ADT המוגדר ע"י הפעולות הבאות:

- $\mathrm{Create}(PQ, L)$ – יצירת תור קדימויות מתוך רשימה נתונה L של איברים
- $\mathrm{Insert}(PQ, x)$ – הכנסת האיבר אליו מצביע x
- $\mathrm{Max}(PQ)$ – החזרת האיבר בעל המפתח המקסימלי
- $\mathrm{Del\text{-}Max}(PQ)$ – מחיקת האיבר בעל המפתח המקסימלי
- $\mathrm{Change\text{-}Priority}(PQ, x, k)$ – שינוי המפתח של האיבר אליו מצביע x ל- k

זוהי הרחבה של תור, אלא שכאן הוצאה היא לפי הקדימות, ולא לפי הוותק.

למפתחות של האיברים ניתנת בד"כ משמעות של קדימות או עדיפות.

לתורי קדימויות שימושים רבים ומגוונים, למשל:

- סימולציה של אירועים בציר הזמן (המפתח: זמן האירוע)
- תזמון משימות (ביצוע עבודות על מעבד, גישה לרשת וכו')

אילו מבני נתונים מממשים תור קדימויות ביעילות?

תור קדימויות (priority queue)

- Create(PQ, L) – יצירת תור קדימויות מתוך רשימה נתונה L של איברים
- Insert(PQ, x) – הכנסת האיבר אליו מצביע x
- Max(PQ) – החזרת האיבר בעל המפתח המקסימלי
- Del-Max(PQ) – מחיקת האיבר בעל המפתח המקסימלי
- Change-Priority(PQ, x, k) – שינוי המפתח של האיבר אליו מצביע x ל- k

מימוש תור קדימויות במערך ובמערך ממוין

- m הוא גודל הרשימה ההתחלתית L
- n הוא מספר האיברים ברגע נתון

סיבוכיות זמן	מערך	מערך ממוין
Create	$\Theta(m)$	$\Theta(m\log m)$**
Insert	$\Theta(1)$	$\Theta(n)$
Max	$\Theta(n)$ *	$\Theta(1)$
Del-Max	$\Theta(n)$	$\Theta(1)$
Change-Priority	$\Theta(1)$	$\Theta(n)$

* אפשר לשפר ל- $\Theta(1)$, על חשבון Change-Priority

** אם לא ידוע דבר על איברי הקלט, לא ניתן למיין בפחות מזמן זה (יוכח בהמשך הקורס)

שאלה (מופיעה בשאלות החזרה):

מה הסיבוכיות במימוש כרשימה מקושרת, ורשימה מקושרת ממוינת?

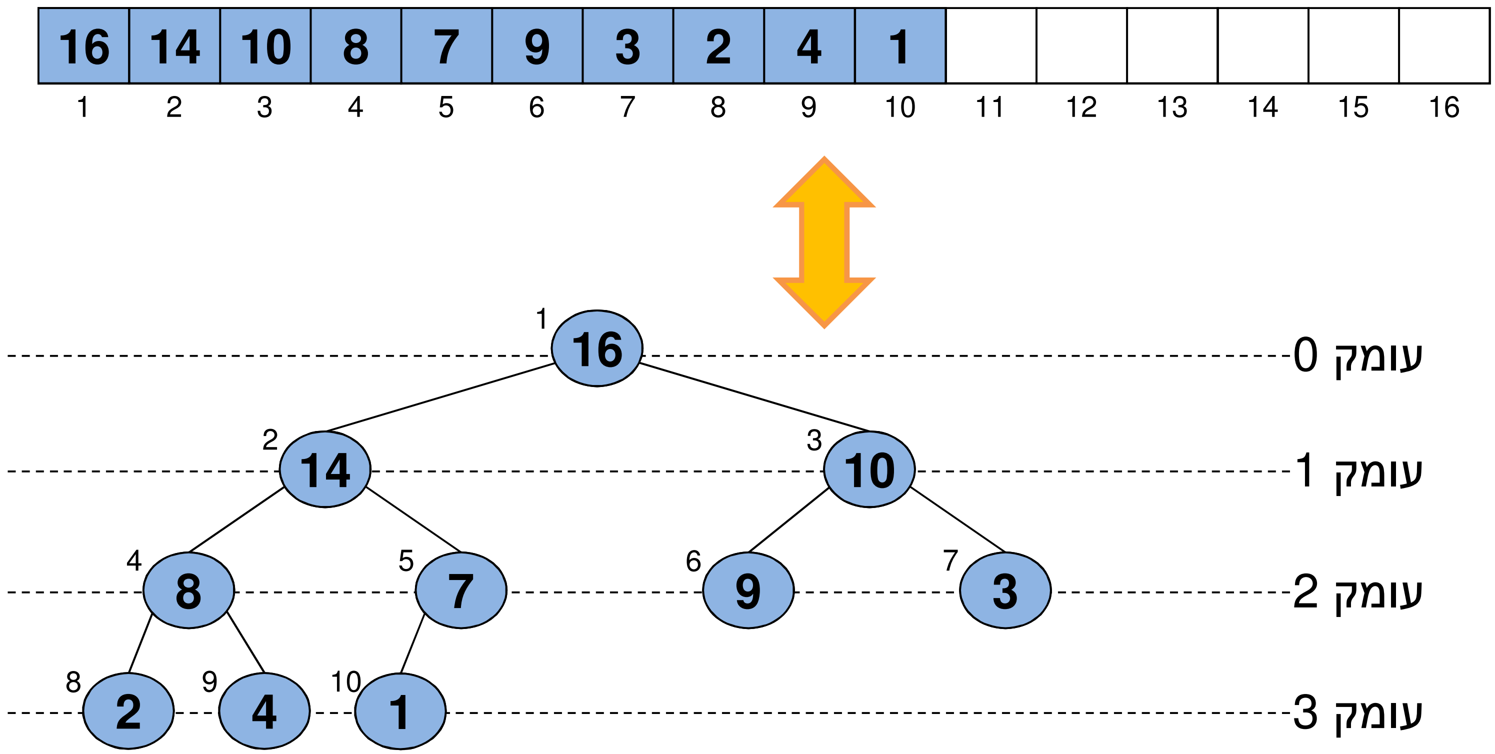
תור קדימויות (priority queue)

נראה כעת מבנה נתונים יעיל עבור תור קדימויות, שנקרא **ערימה**

היתרון של מימוש בערימה: אף פעולה, למעט Create כמובן, לא תדרוש זמן ליניארי !

ערימה (heap)

מבנה הנתונים **ערימה** הוא למעשה **מערך**, שמייצג עץ בינארי כמעט שלם.



בנוסף למערך אנו מחזיקים משתנה בשם *heapSize* ששומר את מספר האיברים בערימה.

- בדוגמה: $heapSize = 10$.

בהינתן איבר באינדקס i אפשר לחשב את האינדקסים של בניו ושל אביו:

Parent(i)
1. return $\lfloor i/2 \rfloor$

Left(i)
1. return $2i$

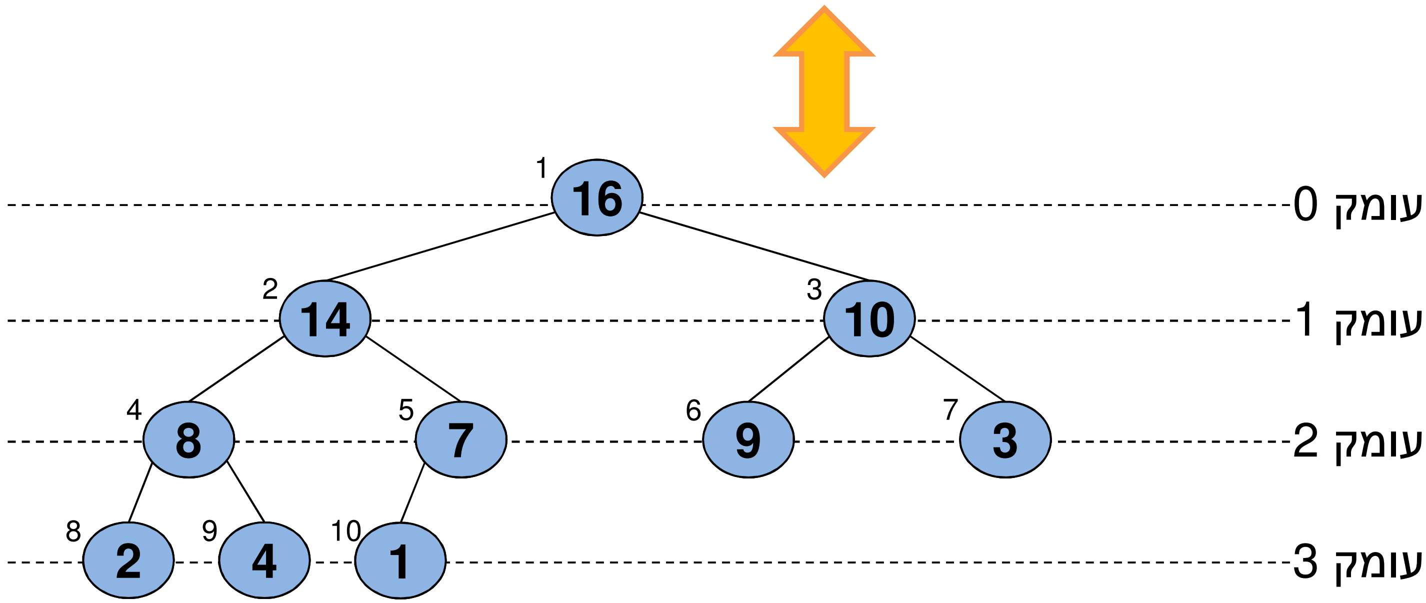
Right(i)
1. return $2i+1$

ערימת מקסימום (max-heap)

<u>בערימת מקסימום</u> מתקיים:

אם x בן של y אז $\text{key}[x] \le \text{key}[y]$

16	14	10	8	7	9	3	2	4	1						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16



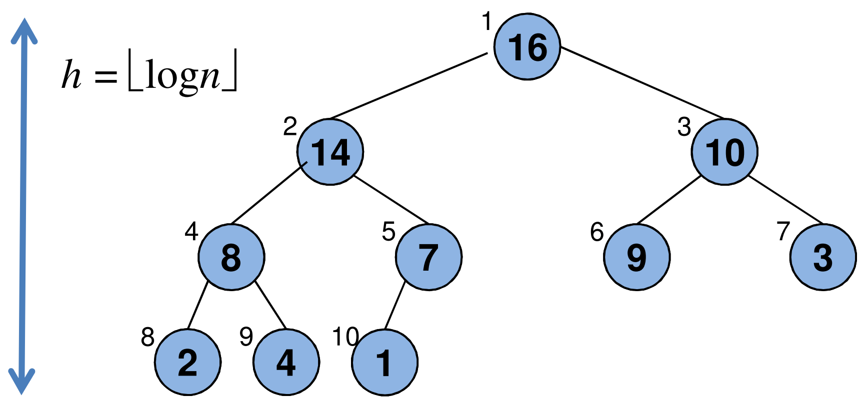
<u>בערימת מינימום</u> מתקיים: אם x בן של y אז $\text{key}[x] \ge \text{key}[y]$

מעתה, אם לא צוין אחרת, נדבר על ערימות מקסימום.

תכונות של ערימות

- מהו מספר האיברים בערמה שגובהה h ?

$$2^h \le n \le 2^{h+1}-1$$



- מהו הגובה של ערמה בת n איברים ?

מצד אחד: $h \le \log n$

ומצד שני: $h \ge \log(n+1)\text{-}1 > \log n\ \text{-}1$

ולכן $h = \lfloor \log n \rfloor = \Theta(\log n)$

- באילו אינדקסים נמצאים העלים, וכמה עלים יש? כמה צמתים פנימיים יש?

עלים נמצאים באינדקסים $\lfloor n/2 + 1 \rfloor$ עד n.

יש $\lceil n/2 \rceil$ עלים, ו- $\lfloor n/2 \rfloor$ צמתים פנימיים

<u>תרגיל</u>: הוכיחו באינדוקציה על n

- צומת באינדקס i הוא עלה אם"ם $2i > n$.

פעולות תיקון על ערימות

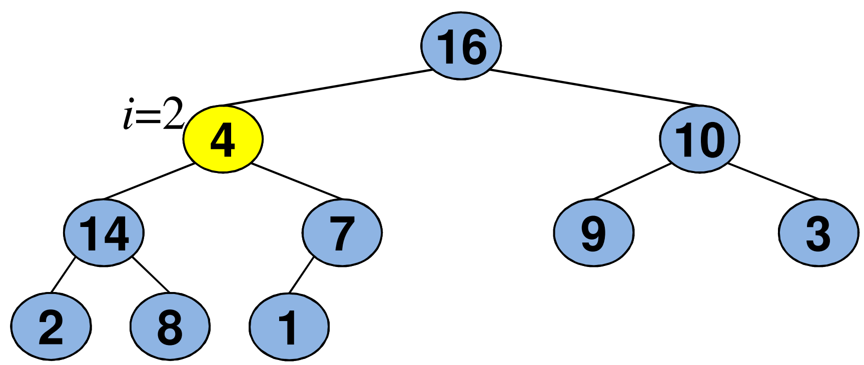
נכיר כעת 2 פעולות בסיסיות על ערימות.

מטרתן: תיקון ערימה שיש בה הפרה כלשהי של תכונת הערימה.

ישמשו אותנו בהמשך בפעולות האחרות.

<u>הפעולה Heapify-Down:</u>

- מקבלת מערך A ואינדקס i.
- מניחה שתתי-העצים הימני והשמאלי של i הם ערימות תקינות.
- ייתכן ש- $A[i]$ "עבריין": קטן (מלפחות אחד) מבניו.
- מחליקה את "העבריין" כלפי מטה, ע"י החלפתו עם בנו המקסימלי שוב ושוב

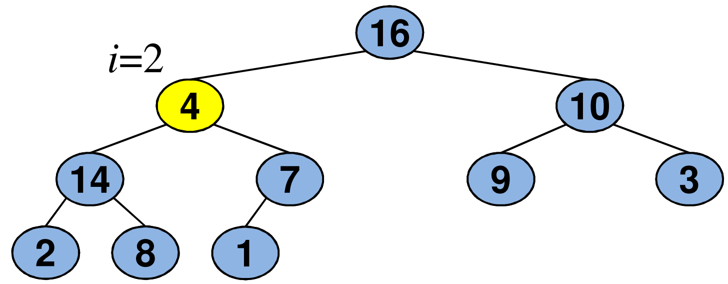


Heapify-Down - פעולות תיקון על ערימות

<u>example</u>: Heapify-Down(A,2)

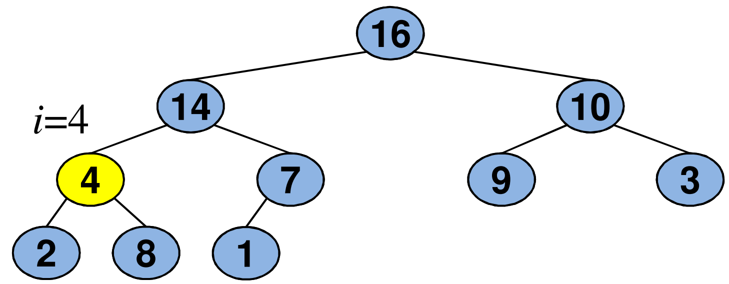
A

16	4	10	14	7	9	3	2	8	1						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16



A

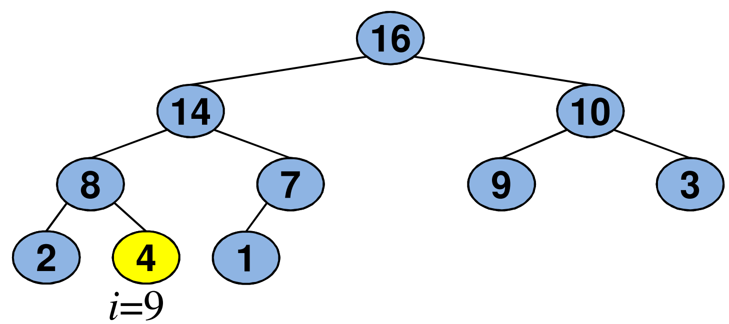
16	14	10	4	7	9	3	2	8	1						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16



A

16	14	10	8	7	9	3	2	4	1						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

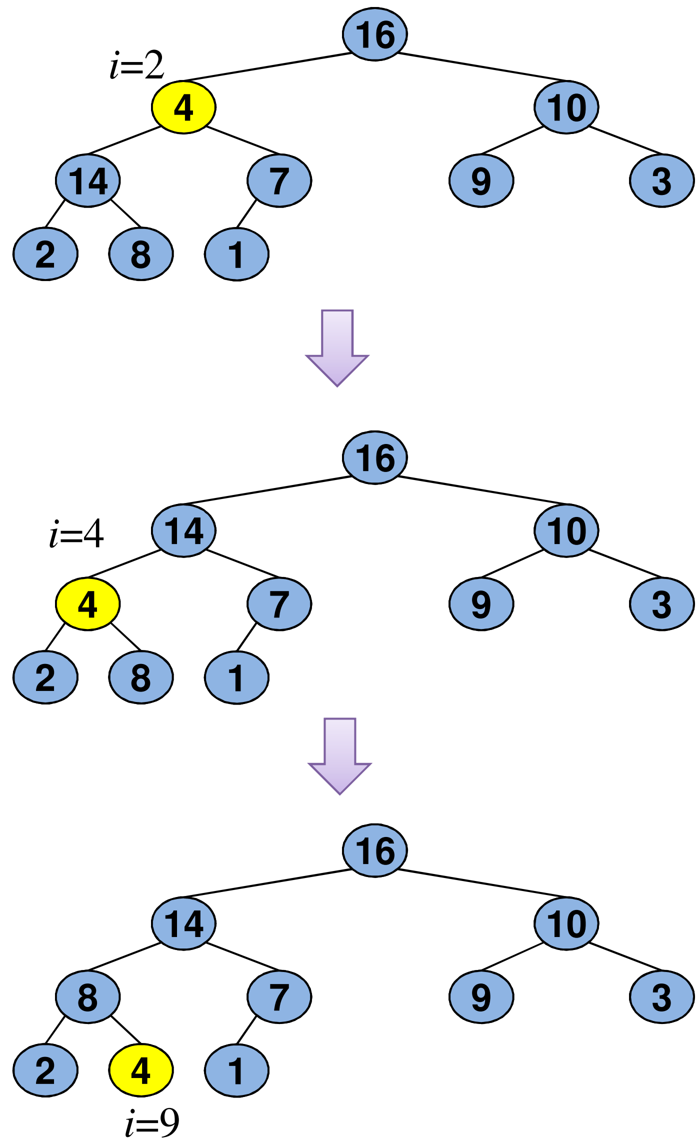
ערימה תקינה



Heapify-Down - פעולות תיקון על ערימות

Heapify-Down(A, i)

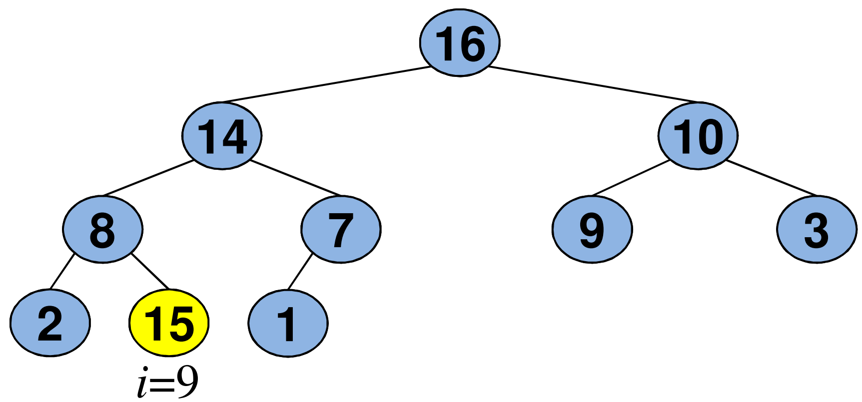
1. $l \leftarrow$ Left(i), $r \leftarrow$ Right(i), $n \leftarrow heapSize[A]$
2. $largest \leftarrow i$
3. **if** $l \leq n$ **and** $A[l] > A[i]$ **then** $largest \leftarrow l$
4. **if** $r \leq n$ **and** $A[r] > A[largest]$ **then** $largest \leftarrow r$
5. **if** $largest \neq i$
6. exchange $A[i] \leftrightarrow A[largest]$
7. Heapify-Down(A, $largest$)



פעולות תיקון על ערימות - Heapify-Up

<u>הפעולה Heapify-Up:</u>

- מקבלת מערך A ואינדקס i.
- מניחה ש- A ערימה תקינה, למעט אולי $A[i]$ ה"עבריין" שגדול מאביו
- מחליקה את "העבריין" כלפי מעלה, ע"י החלפתו עם אביו שוב ושוב



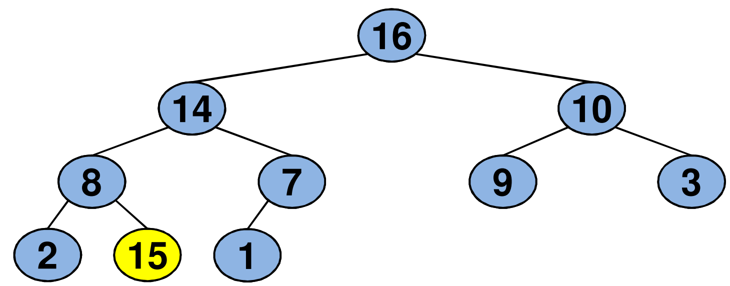
<u>תרגיל</u>: ממשו את Heapify-Up ב- pseudo-code.

Heapify-Up - פעולות תיקון על ערימות

example: Heapify-Up(A,9)

A

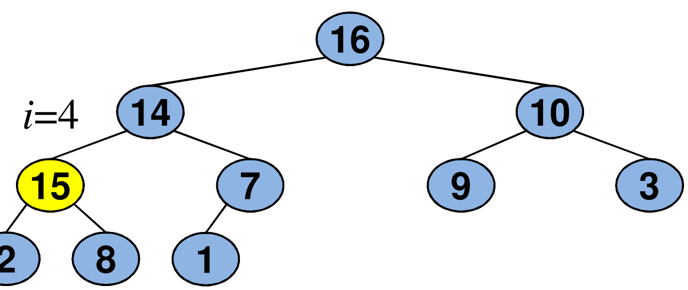
16	14	10	8	7	9	3	2	15	1						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16



i=9

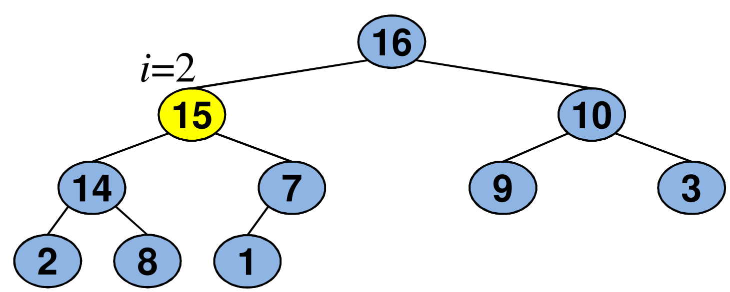
A

16	14	10	15	7	9	3	2	8	1						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16



A

16	15	10	14	7	9	3	2	8	1						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16



ערימה תקינה

פעולות תיקון על ערימות - ניתוח סיבוכיות

<u>סיבוכיות זמן</u>

במקרה הגרוע מבחינת כמות ההחלפות:

- Heapify-Down(A,i) רצה בזמן $\Theta(h(i))$ כאשר $h(i)$ הוא <u>גובה</u> הצומת i

- Heapify-Up(A,i) רצה בזמן $\Theta(d(i))$ כאשר $d(i)$ הוא <u>עומק</u> הצומת i

במקרה הגרוע מבחינת i:

אם מפעילים Heapify-Down על השורש, או Heapify-Up על עלה,

זמן הריצה הוא $\Theta(\log n)$.

<u>סיבוכיות זיכרון נוסף</u>

Heapify-Down דורשת $\Theta(h(i))$ זיכרון נוסף לצורך הרקורסיה.

אבל: אפשר לממש גרסה איטרטיבית (ללא רקורסיה), ואז $\Theta(1)$.

פעולות על ערימות

עד כאן Heapify-Down ו- Heapify-Up.

כעת נפנה למימוש פעולות תור הקדימויות, וניעזר בשתי פעולות התיקון הנ"ל.

- Create(PQ, L) – יצירת תור קדימויות מתוך רשימה נתונה L של איברים
- Insert(PQ, x) – הכנסת האיבר אליו מצביע x
- Max(PQ) – החזרת האיבר בעל המפתח המקסימלי
- Del-Max(PQ) – מחיקת האיבר בעל המפתח המקסימלי
- Change-Priority(PQ, x, k) – שינוי המפתח של האיבר אליו מצביע x ל- k

פעולות על ערימות - בנייה

בניית ערימה מרשימת איברים נתונה

קלט: מערך A כלשהו עם n איברים.

פלט: אותם איברים מסודרים ב- A כערימה חוקית.

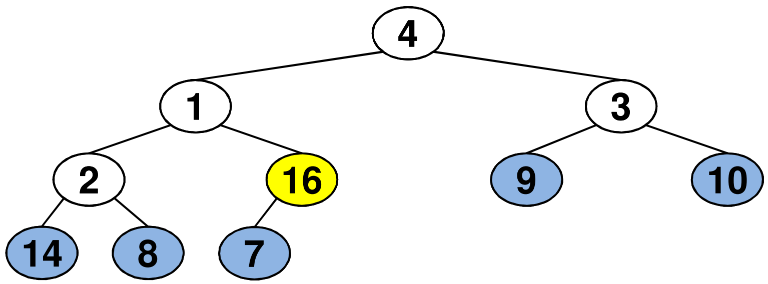
האלגוריתם: עוברים על כל הצמתים הפנימיים bottom-up, ומפעילים עליהם Heapify-Down.

Build-Heap(A, n)

1. $heapSize[A] \leftarrow n$
2. **for** $i \leftarrow \lfloor heapSize[A]/2 \rfloor$ **downto** 1
3. Heapify-Down(A, i)

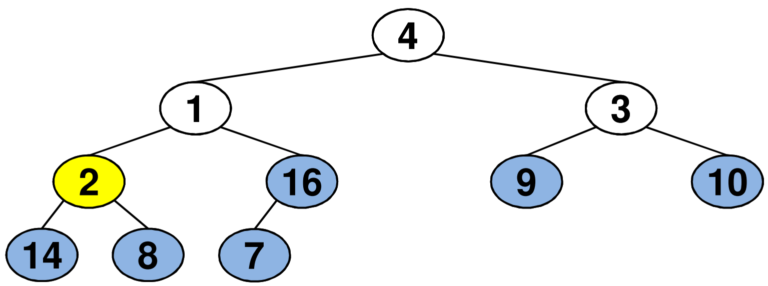
פעולות על ערימות - בנייה

בניית ערימה

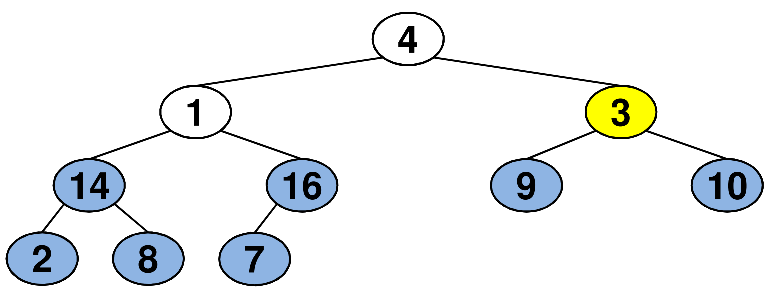


4	1	3	2	16	9	10	14	8	7

מערך הקלט



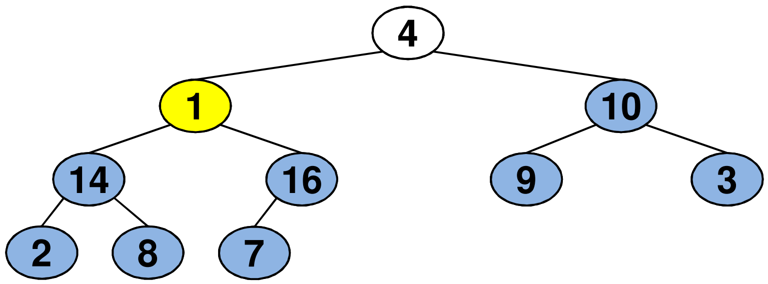
4	1	3	2	16	9	10	14	8	7



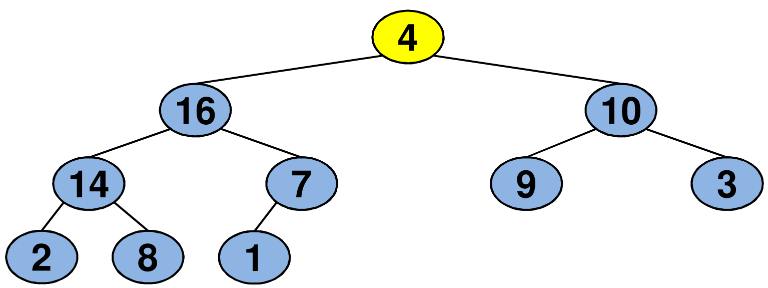
4	1	3	14	16	9	10	2	8	7

פעולות על ערימות - בנייה

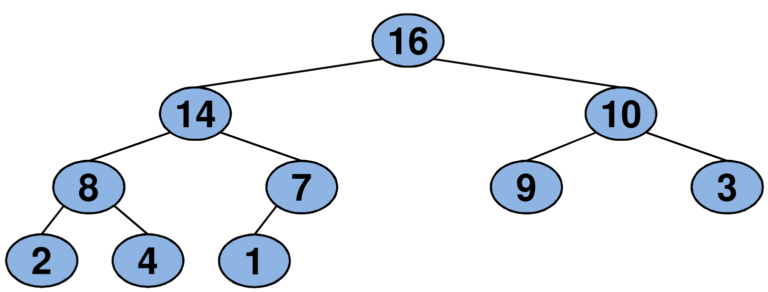
בניית ערימה



4	1	10	14	16	9	3	2	8	7



4	16	10	14	7	9	3	2	8	1



16	14	10	8	7	9	3	2	4	1

ערימה תקינה

פעולות על ערימות - בנייה

Build-Heap(A, n)
1. $heapSize[A] \leftarrow n$
2. **for** $i \leftarrow \lfloor heapSize[A]/2 \rfloor$ **downto** 1
3. Heapify-Down(A, i)

- מדוע עוברים רק על הצמתים הפנימיים?
- מדוע bottom-up?

ניתוח זמנים – בניית ערימה

ניתוח גס

$\lfloor n/2 \rfloor$ קריאות ל- Heapify-Down כל אחת ב- $O(\log n)$. סה"כ $O(n\log n)$.

ניתוח עדין יותר

נוכיח כעת שזמן הריצה הוא $\Theta(n)$.

אינטואיציה: יש הרבה קריאות "זולות" ומעט קריאות "יקרות".

נספור כמה החלפות (exchange) מתבצעות (לזמן ריצה כולל יש לכפול בקבוע).

- יש לכל היותר $n/2$ עלים, $n/4$ אבות של עלים, $n/8$ סבים של עלים וכו' (השמטנו ערכי תקרה)
- לצומת בגובה y מתבצעות לכל היותר y החלפות.
- לכן מספר ההחלפות הכולל הוא לכל היותר:

$$\frac{n}{4}\cdot 1+\frac{n}{8}\cdot 2+\ldots+\frac{n}{2^{i+1}}\cdot i+\ldots+1\cdot\lfloor \log n \rfloor$$

$$\begin{aligned}
&= (\frac{n}{4}+\frac{n}{8}+\frac{n}{16}+\ldots+1) && \leq && n/2\\
& && && +\\
&\quad +(\frac{n}{8}+\frac{n}{16}+\ldots+1) && \leq && n/4 && \leq\; n\\
& && && +\\
&\quad\quad +(\frac{n}{16}+\ldots+1) && && \vdots\\
&\quad\quad\quad \ldots && && +\\
&\quad\quad\quad +1 && \leq && 1
\end{aligned}$$

$$\boxed{\sum_{i=1}^{\lfloor \log n \rfloor}\frac{i}{2^{i+1}}\leq 1}$$

<u>פעולות על ערימות – מחיקת מקסימום</u>

<u>מחיקת מקסימום</u>

הרעיון:

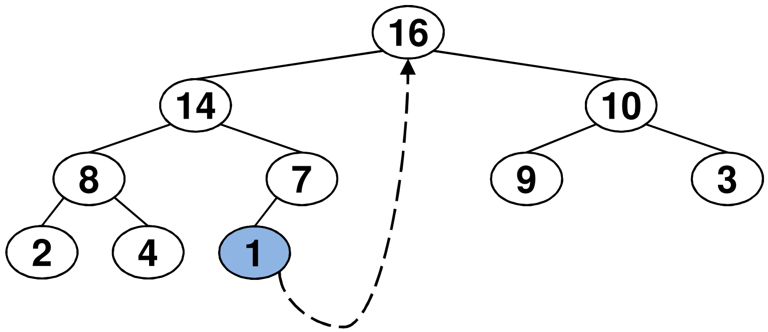
- נעביר לשורש את העלה האחרון
- נקטין את *heapSize* ב- 1
- נקרא ל- Heapify-Down על השורש כדי לתקן הפרה אפשרית.

```
Heap-Extract-Max(A)
if  heapSize[A] < 1
        error "heap underflow"
max ← A[1]
A[1] ← A[heapSize[A]]
heapSize[A] ← heapSize[A]  –1
Heapify-Down(A, 1)
return max
```

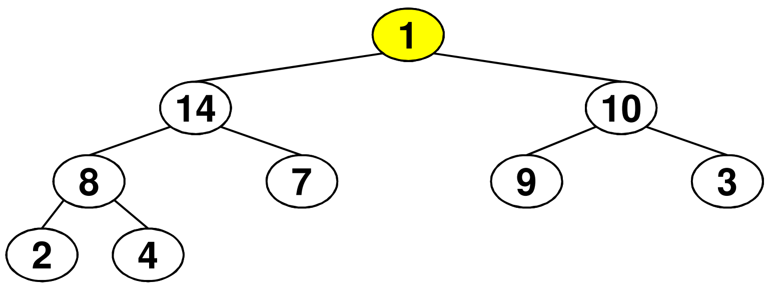
סיבוכיות המקרה הגרוע: $\Theta(\log n)$

פעולות על ערימות – מחיקת מקסימום

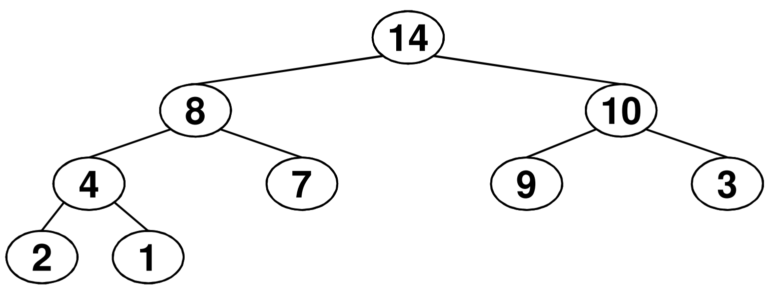
מחיקת מקסימום



16	14	10	8	7	9	3	2	4	1



1	14	10	8	7	9	3	2	4	



14	8	10	4	7	9	3	2	1	

פעולות על ערימות – הכנסה

הכנסה

הרעיון:

- נוסיף את האיבר החדש כעלה הבא
- נגדיל את *heapSize* ב- 1
- נקרא ל- Heapify-Up על העלה החדש

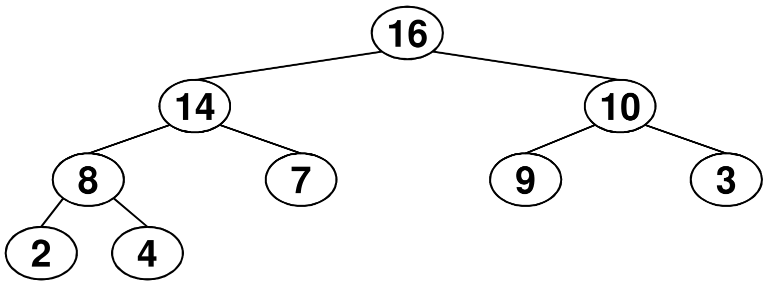
```
Heap-Insert(A, key)
1.   heapSize[A] ← heapSize[A] + 1
2.   A[heapSize[A]] ← key
3.   Heapify-Up(A, heapSize[A] )
```

סיבוכיות המקרה הגרוע: $\Theta(\log n)$

הערה: אם נתון מצביע לאיבר x אותו רוצים להכניס (שיכול להיות בעל שדות רבים),
אז נכניס לערימה את המצביע באופן דומה, כאשר את התיקון נבצע לפי המפתח של x.

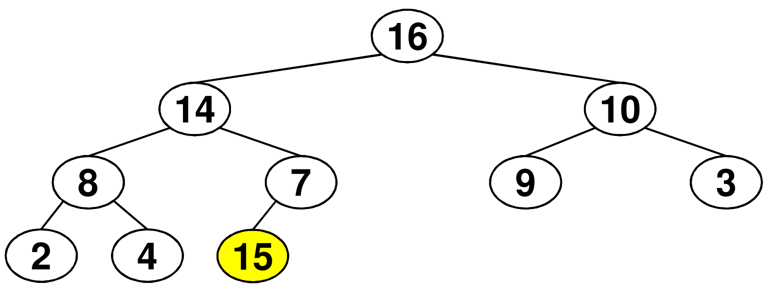
פעולות על ערימות – הכנסה

<u>הכנסה</u>

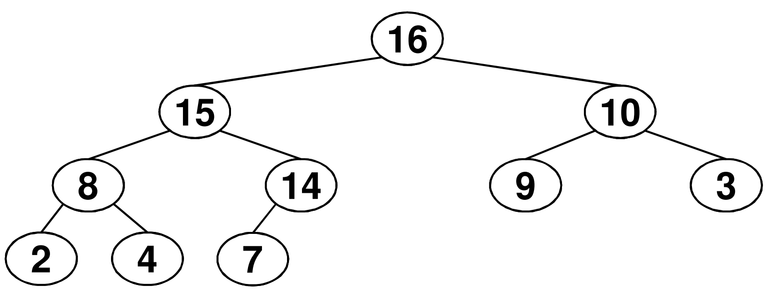


15

16	14	10	8	7	9	3	2	4	



16	14	10	8	7	9	3	2	4	15



16	15	10	8	14	9	3	2	4	7

פעולות על ערימות – הכנסה

בעיה 6-1 מספר הלימוד

בהינתן מערך A בגודל n, ניתן לבנות ממנו ערימה בדרך שונה מזו שראינו, ע"י הכנסת האיברים לערימה בזה אחר זה:

Build-Heap'(A, n)
1. $heapSize[A] \leftarrow 1$
2. **for** $i \leftarrow 2$ **to** n
3. Heap-Insert(A, $A[i]$)

א. הדגימו בניית ערימה בדרך זו על המערך

1	**2**	**3**

והשוו לערימה המתקבלת מ- Build-Heap.

ב. האם שיטה זו יעילה מבחינת זמן ריצה?

פעולות על ערימות – שינוי מפתח

שינוי מפתח

הרעיון:

- אחרי שינוי המפתח, יכולה להיווצר הפרה כלפי מעלה או כלפי מטה.
- לכן נתקן כלפי מעלה או כלפי מטה.

Heap-Change-Key(A, i, key)
1. $A[i] \leftarrow key$
2. Heapify-Down(A, i)
3. Heapify-Up(A, i)

סיבוכיות המקרה הגרוע: $\Theta(\log n)$

- האם ייתכן שאחרי שינוי המפתח ישנה הפרה גם מעלה וגם מטה?

מיון ערימה

מיון ערימה (heap-sort)

שימוש נוסף של ערימות הוא האלגוריתם מיון-ערימה

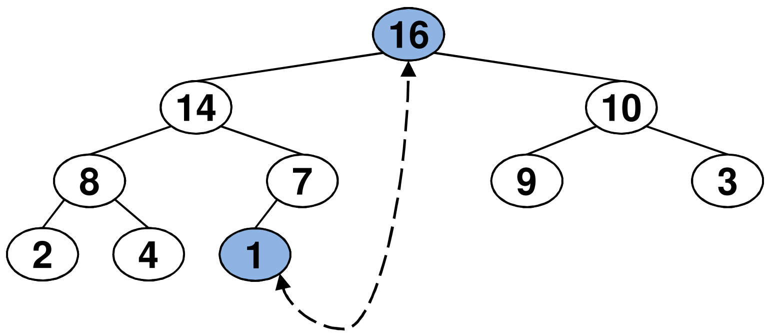
```
Heap-Sort(A, n)
    Build-Heap(A)
    for i ← n downto 2
        exchange A[1] ↔ A[i]
        heapSize[A] ← heapSize[A] – 1
        Heapify-Down(A, 1)
```

שלב בניית הערימה

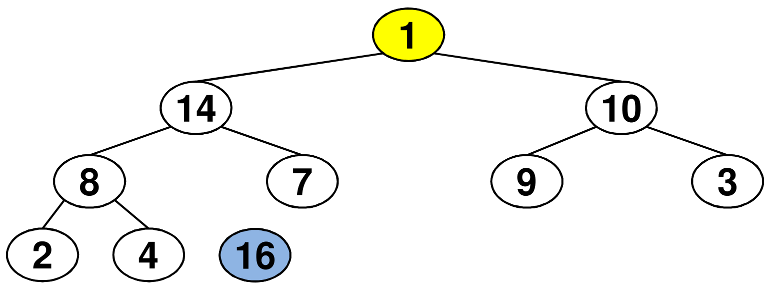
שלב סידור האיברים

מיון ערימה (heap-sort)

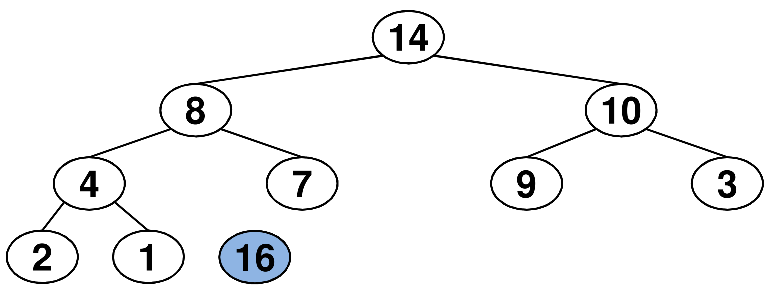
הדגמת שלב סידור האיברים (הערימה כבר נבנתה):



16	14	10	8	7	9	3	2	4	1

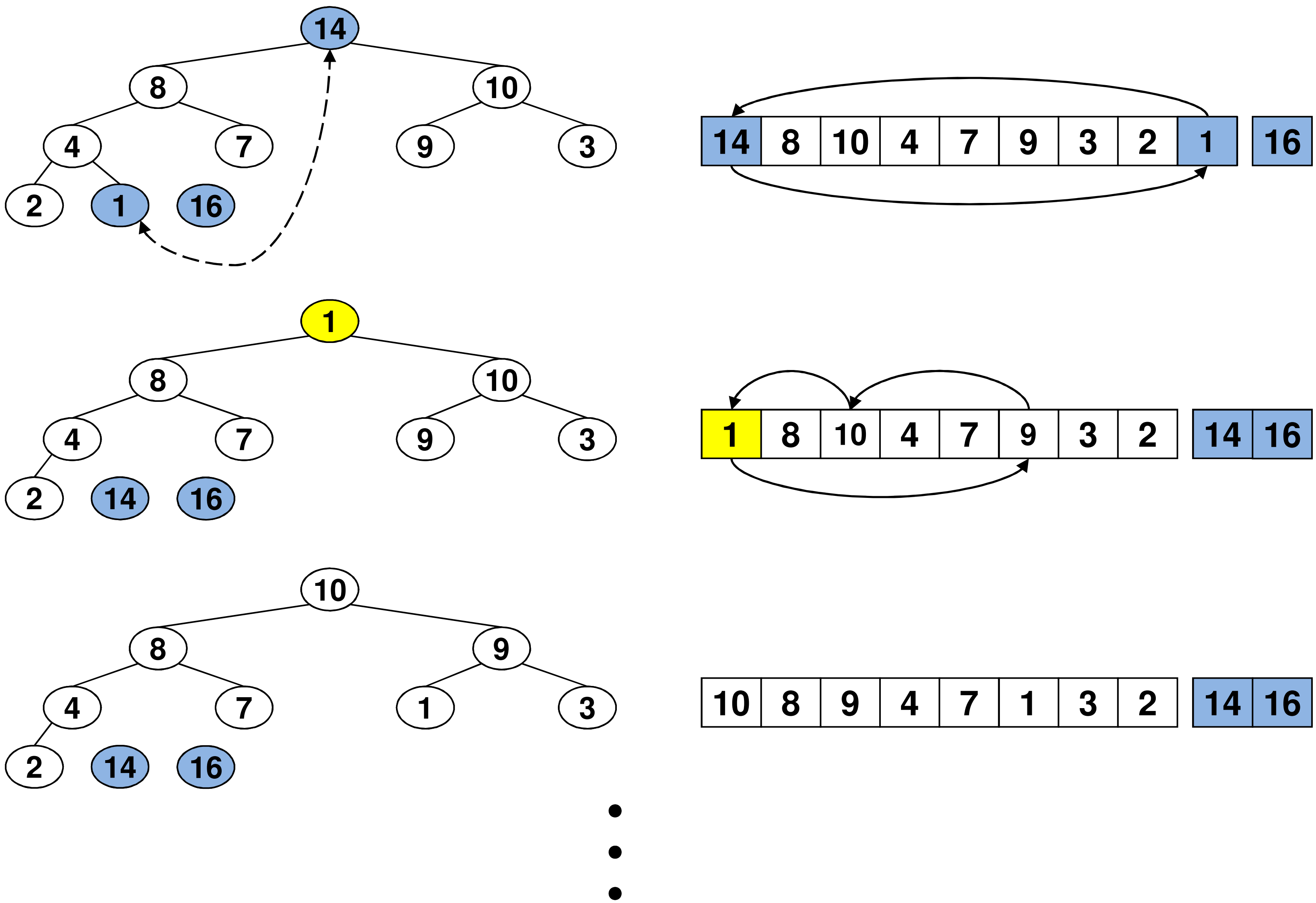


1	14	10	8	7	9	3	2	4		16



14	8	10	4	7	9	3	2	1		16

מיון ערימה (heap-sort)



מיון ערימה (heap-sort)

Heap-Sort(A, n)
1. Build-Heap(A)
2. **for** $i \leftarrow n$ **downto** 2
3. exchange $A[1] \leftrightarrow A[i]$
4. $heapSize[A] \leftarrow heapSize[A] - 1$
5. Heapify-Down(A, 1)

$\Theta(n)$

$\Theta(\log k) \quad 1 \le k \le n\text{-}1$

סיבוכיות המקרה הגרוע:

$$\Theta(n)+\sum_{k=1}^{n-1}\Theta(\log k)=\Theta(n)+\Theta(\log n!)=\Theta(n)+\Theta(n\log n)=\Theta(n\log n)$$

<u>תכונות מיון ערימה:</u>

- ממיין מערך בגודל n בזמן $\Theta(n\log n)$ במקרה הגרוע.
- ממיין במקום (in-place), ודורש $\Theta(1)$ זיכרון נוסף (מעבר למערך הקלט) (בתנאי ש- Heapify-Down ממומשת ללא רקורסיה)
- האם מיון יציב?

שאלות חזרה

1. ניתן לממש תור קדימויות ברשימה מקושרת, וברשימה מקושרת ממוינת. מהי סיבוכיות כל אחת מהפעולות של תור קדימויות בכל אחד מהמימושים האלו?

2. האם המערך הבא הוא ערימת מקסימום? $<23,17,14,6,13,10,1,5,7,12>$

3. האם מערך ממוין הוא ערימת מינימום?

4. בערמת מקסימום, האיבר המקסימלי נמצא באינדקס 1, והאיבר השני הכי גדול נמצא באינדקס 2 או 3. נתונה ערמת מקסימום בגודל 15, עם איברים שונים זה מזה.

   א. באילו אינדקסים יכול להימצא האיבר ה- 3 הכי גדול?

   ב. באילו אינדקסים יכול להימצא האיבר ה- 4 הכי גדול?

5. באילו אינדקסים יכול להימצא המינימום בערימת מקסימום בגודל n?
   הניחו כי כל האיברים שונים זה מזה.

6. מהו זמן הריצה של חיפוש איבר נתון בערימה?

7. הראו באמצעות דוגמה שמיון ערימה איננו מיון יציב.

תשובות לשאלות חזרה

1. ניתן לממש את הפעולות בסיבוכיות הבאה (ודאו שאתם מבינים כיצד):

	רשימה מקושרת	רשימה מקושרת ממוינת
Create	$\Theta(n)$	$\Theta(n\log n)$
Insert	$\Theta(1)$	$\Theta(n)$
Max	$\Theta(n)$	$\Theta(1)$
Del-Max	$\Theta(n)$	$\Theta(1)$
Change-priority	$\Theta(1)$	$\Theta(n)$

2. לא. המפתח 6 מפר את תכונת הערימה כלפי בנו הימני.

3. כן. כל צומת מקיים את תכונת ערימת המינימום.

4. א. באינדקסים 2 עד 7 (עומק 1 או 2)
ב. באינדקסים 2 עד 15 (עומק 1 עד 3)

5. המינימום חייב להיות עלה, כלומר באינדקסים $\lfloor n/2 + 1 \rfloor$ עד n.

6. חיפוש בערימה דורש במקרה הגרוע זמן ליניארי. ניתן להוכיח זאת למשל בעזרת התשובה לשאלה הקודמת: אם מחפשים את המינימום, יש לעבור על $\Theta(n)$ צמתים.

7. למשל מיון ערימה על $<1, 1'>$ לא שומר על הסדר היחסי בין 1 ל- 1'.

תרגילים

תרגילים נוספים

1. מה זמן הריצה של Build-Heap על מערך הממוין בסדר הפוך?

2. ממשו את הפעולה $\mathrm{Del}(A, i)$ המוחקת מערמת מקסימום A בגודל n את הצומת באינדקס i, בסיבוכיות זמן $\Theta(\log n)$.

3. תארו מבנה נתונים המאפשר ביצוע הפעולות הבאות בסיבוכיות הנדרשת:
   - Init(S) – אתחול המבנה, בהינתן סדרת איברים S באורך m, בזמן $O(m)$.
   - Insert(x) – הוספת x למבנה, בזמן $O(\log n)$ (n הוא מספר האיברים הנוכחי).
   - Find-min – החזרת המינימום, בזמן $O(1)$.
   - Find-max – החזרת המקסימום, בזמן $O(1)$.
   - Del-min – הוצאת המינימום מהמבנה, בזמן $O(\log n)$.
   - Del-max – הוצאת המקסימום מהמבנה, בזמן $O(\log n)$.

   הערה : מס' האיברים המקסימלי במבנה נתונים הוא m.

4. במכללה מסוימת מלמדים n מרצים. נתוני המרצים (שם, כתובת, שכר) שמורים בקובץ. אנו מעוניינים להדפיס את שמותיהם של m המרצים שמרוויחים את השכר הגבוה ביותר.

   תנו פתרון יעיל תחת ההגבלות הבאות:
   - סיבוכיות הזיכרון הנוסף המותרת: $O(m)$. נתון $m=o(n)$.
   - מותר לעבור על נתוני הקובץ פעם אחת בלבד.

פתרון 1

בכל אחת מ- $\lfloor n/2 \rfloor$ האיטרציות, השגרה Heapify-Down תסתיים בקריאה הראשונה.

זמן הריצה הוא עדיין $\Theta(n)$, אבל הקבוע החבוי בזמן הריצה הוא קטן יותר.

פתרון 2

נעביר את העלה האחרון לאינדקס i, ונתקן.

Del(A, i)

1. $n \leftarrow heapSize[A]$
2. $A[i] \leftarrow A[n]$
3. $heapSize[A] \leftarrow heapSize[A] - 1$
4. Heapify-Down(A, i)
5. Heapify-Up(A, i)

נשים לב שתיתכן הפרה כלפי מעלה או מטה.

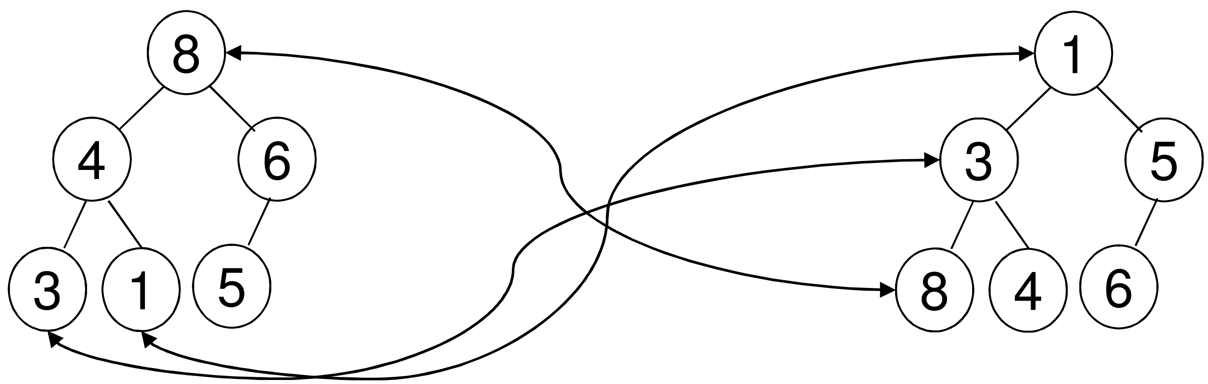
האם אתם יכולים לתת דוגמה לכל מקרה?

<u>פתרון 3</u>

נשמור את האיברים בשתי ערמות בו זמנית: ערימת מינימום וערימת מקסימום.

לכל איבר נשמור גם את האינדקס של ה"תאום" שלו בערמה השנייה.

ערימת מקסימום ערימת מינימום



	1	2	3	4	5	6	7	8
element	8	4	6	3	1	5		
index of “twin”	4	5	6	2	1	3		

1	2	3	4	5	6	7	8
1	3	5	8	4	6		
5	4	6	1	2	3		

<u>בשקף הבא - מימוש הפעולות</u>

Init(S): נעתיק את איברי S לשני מערכים נפרדים. בשלב זה "תאומים" נמצאים באותו אינדקס, לכן לכל איבר נאתחל את אינדקס ה"תאום" שלו כאינדקס שלו עצמו.

נריץ Build-heap פעם לערמת מינימום ופעם לערמת מקסימום. כל תזוזה של איבר תגרור עדכון של האינדקס ששמור אצל אחיו "התאום" (תוספת של קבוע לסיבוכיות).

זמן: $2\cdot\Theta(m)$ עבור ההעתקה ועוד $2\cdot\Theta(m)$ עבור בניית הערמות. סה"כ $\Theta(m)$.

Find-max / Find-min: החזרת שורש הערמה המתאימה. זמן: $\Theta(1)$.

Insert(x): נכניס את x לכל אחת משתי הערמות. שוב, בכל תזוזה של איבר בערמה אחת, נעדכן את האינדקס ששמור אצל אחיו "התאום" בערמה השנייה בהתאם.

זמן: כל הכנסה $\Theta(\log n)$, סה"כ $\Theta(\log n)$.

Del-min: נמחק את שורש ערימת המינימום ובעזרת האינדקסים נגיע ל"תאום" ונמחק גם אותו מערימת המקסימום. מחיקת שורש תתבצע כרגיל, ואילו מחיקת ה"תאום" תתבצע כפי שראינו בתרגיל קודם. מימוש Del-max סימטרי ל- Del-min.

זמן: כל מחיקה $\Theta(\log n)$, סה"כ $\Theta(\log n)$.

<u>הערה</u>: הצומת התאום של שורש ערימה אחת הוא עלה בערימה השנייה.

<u>פתרון 4</u>

נשתמש בערמת מינימום בגודל m.

1. נעתיק את m האיברים הראשונים בקובץ למערך, ונאתחל ממנו ערמת מינימום, כאשר המפתח הוא השכר, והשם והכתובת הם נתונים נלווים.
2. כעת נעבור על יתר המרצים בקובץ החל ב- $m+1$, ולכל מרצה x :

   אם שכרו של x גבוה מזה שמופיע בשורש הערמה, אז נבצע:

   2.1. Heap-Extract-Min

   2.2. Heap-Insert(x)
3. לבסוף נדפיס את איברי הערמה.

<u>סיבוכיות</u>

זיכרון נוסף: אנו משתמשים בערימה שגודלה m, כלומר $\Theta(m)$.

זמן: - שלבים 1 ו- 3 רצים ב- $\Theta(m)$.

- בשלב 2 מבצעים $n\text{-}m$ פעולות, שכל אחת מורכבת במקרה הגרוע משתי פעולות ב- $\Theta(\log m)$.

סה"כ: $\Theta(m) + \Theta((n\text{-}m)\log m) = \Theta(n\log m)$.