

# חידה לחימום



בסל מקש יש 100 כדורי צמר.

שני שחקנים משחקים בתורות: כל שחקן, בתורו, צריך להוציא כמות כלשהי של כדורי צמר מהסל –

לפחות כדור אחד, אך לא יותר ממחצית מכמות כדורי הצמר שבסל.

מי שלא יכול לעשות מהלך (מתי זה יקרה?) – מפסיד במשחק.

פתחו אסטרטגיה מנצחת עבור אחד השחקנים במשחק.

# מבני נתונים ויעילות אלגוריתמים

(4.2.2015)

## 1. הגדרת הערימה

בשיעור קודם ראינו מבני נתונים שונים למימוש טיפוס הנתונים המופשט (טנ"מ) הקרוי 'תור קדמויות'. אחד ממבני הנתונים היעילים ביותר למימוש טנ"מ זה הוא מבנה נתונים הנקרא **ערימה בינארית** (Binary Heap), או בקיצור – **ערימה** (Heap).

ערימה בינארית היא עץ בינארי כמעט שלם, הממומש באמצעות מערך, והמתאפיין בתכונה נוספת הנקראת **תכונת הערימה** (Heap Property).

כזכור, עץ בינארי כמעט שלם הוא עץ שמלא לחלוטין בכל רמותיו, פרט אולי לאחרונה, המלאה משמאל ועד לנקודה מסוימת. המערך  $A$ , שבו נאחסן את איברי העץ הבינארי הכמעט שלם, יתאפיין בשתי התכונות הבאות:  $A.length$  – מספר התאים במערך,  $A.heap-size$  – מספר האיברים בערימה המאוחסנת ב- $A$ . במילים אחרות, אף כי  $A[0..A.length-1]$  זהו המערך כולו, איבר שנמצא אחרי  $A[A.heap-size-1]$  אינו שייך לערימה.

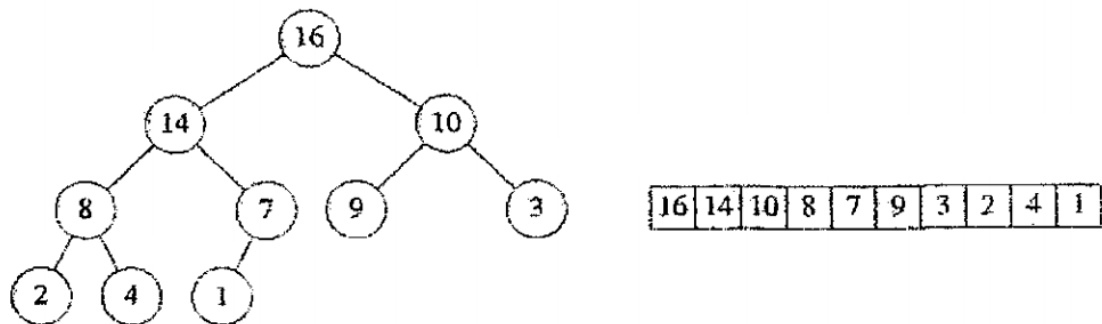
שורשו של העץ הוא  $A[0]$ , ובהינתן האינדקס  $i$  של צומת, אפשר לחשב בקלות את האינדקסים של אביו, של בנו השמאלי, ושל בנו הימני:

```
LEFT(i)  
  return 2i+1
```

```
RIGHT(i)  
  return 2i+2
```

```
PARENT(i)  
  return  $\lfloor (i-1)/2 \rfloor$ 
```

כעת, כדי שעץ בינארי כמעט שלם המיוצג במערך ייחשב כערימה, עליו לקיים את תכונת הערימה: לכל צומת  $i$ , פרט לשורש, מתקיים:  $A[PARENT(i)] \geq A[i]$ . כלומר, ערכו של כל צומת הוא לכל היותר ערכו של ההורה שלו. לפיכך, הערך הגדול ביותר בערימה מאוחסן בשורש, והערך שצומת מכיל הוא גדול או שווה לכל ערכי צאצאיו.



### שאלה

האם המערך  $A = (23,7,10,3,6,4)$  מייצג ערימה? אם כן – שרטטו את ייצוג הערימה כעץ בינארי כמעט שלם.

מושג הערימה, כפי שהגדרנו אותו לעיל, נקרא **ערימת מקסימום** (Maximum Heap). קיים סוג נוסף של ערימה, הנקרא **ערימת מינימום** (Minimum Heap), שזוהה בכל לערימת מקסימום, פרט לכך שתכונת הערימה נראית במקרה הזה כך:

$$A[\text{PARENT}(i)] \leq A[i]$$

כלומר, ערכו של כל צומת הוא לכל הפחות ערכו של ההורה שלו. לפיכך, הערך הקטן ביותר בערימת מינימום מאוחסן בשורש, והערך שצומת מכיל הוא קטן או שווה מכל ערכי צאצאיו.

### שאלה

- א. הראו שהמערך  $A = (11,10,7,9,5,6,4,8,2,3,1)$  מייצג ערימת מקסימום, ושרטטו את ייצוג הערימה כעץ בינארי כמעט שלם.
- ב. הראו כיצד ניתן לסדר מחדש את האיברים במערך  $A$ , כך שהוא ייצג ערימת מינימום.

### שאלה

- א. האם מערך הממוין בסדר יורד בהכרח מייצג ערימת מקסימום?
- ב. האם מערך הממוין בסדר עולה בהכרח מייצג ערימת מינימום?

בערימה בינארית מתקיימות שלוש הטענות הבאות:

**טענה 1:** אם  $A$  הוא מערך המייצג ערימה בינארית, ואם נסמן ב- $n$  את  $A$ .heap-size, אז כל האיברים שבמערך שנמצאים בין האינדקס  $\lfloor n/2 \rfloor$  לבין האינדקס  $n-1$ , הם עלים.

**טענה 2:** גובהה של ערימה בינארית בעלת  $n$  צמתים הוא  $\lfloor \log_2 n \rfloor$ .

**טענה 3:** בערימה בינארית בעלת  $n$  צמתים, מספר הצמתים שגובהם  $h$  הוא לכל היותר  $\lfloor \frac{n}{2^{h+1}} \rfloor$ .



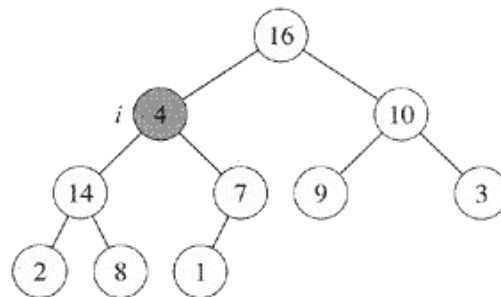
## 2. שמירה על תכונת הערימה

נכיר אלגוריתם חשוב לטיפול בערימות מקסימום בשם MAX-HEAPIFY. האלגוריתם מקבל כקלט מערך A ואינדקס i למערך. ההנחה על הקלט היא, שכאשר מתבצעת קריאה לאלגוריתם MAX-HEAPIFY, אז העצים הבינאריים המושרשים ב-LEFT(i) וב-RIGHT(i) הם ערימות מקסימום חוקיות, אולם ייתכן ש-A[i] קטן מבניו, ובכך מפר את תכונת הערימה.

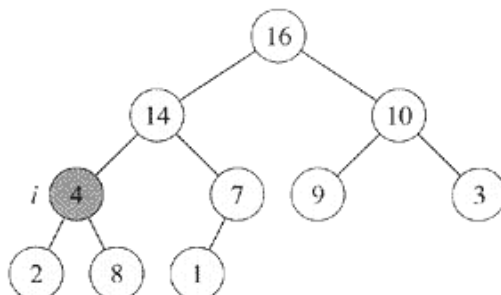
תפקידה של MAX-HEAPIFY הוא "להחליק" את A[i] במורד הערימה, עד שהתת-עץ המושרש באינדקס i יהפוך לערימת מקסימום חוקית.

```
MAX-HEAPIFY(A, i)
1   L ← LEFT(i)
2   R ← RIGHT(i)
3   if L ≤ A.heap-size and A[L] > A[i]
4     then largest ← L
5     else largest ← i
6   if R ≤ A.heap-size and A[R] > A[largest]
7     then largest ← R
8   if largest ≠ i
9     then SWAP(A[i], A[largest])
10  MAX-HEAPIFY(A, largest)
```

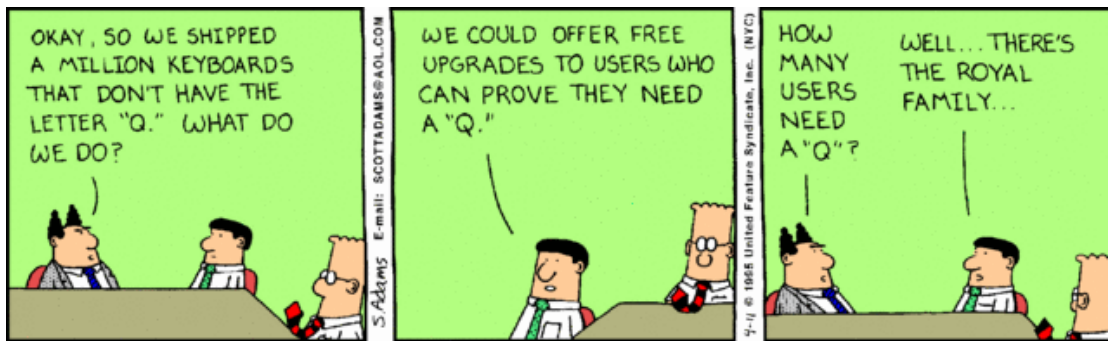
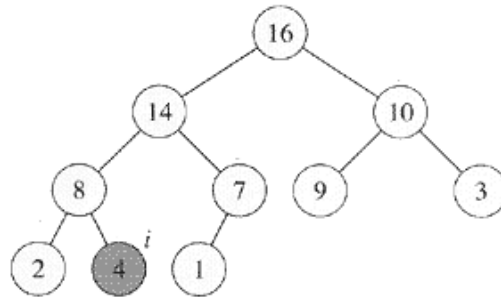
נדגים את פעולת האלגוריתם MAX-HEAPIFY על מערך A המקיים  $A.heap-size = 10$ . אפשר לראות כי האיבר A[1] במערך הנתון מפר את תכונת הערימה, שכן הוא אינו גדול משני בניו:



החלפת A[1] ו-A[3] תוביל לכך שצומת 1 אכן יקיים את תכונת הערימה, אולם צומת 3 אינו מקיים אותה, ולכן - נבצע קריאה רקורסיבית MAX-HEAPIFY(A, 3).



החלפת  $A[3]$ -ו- $A[8]$  זה בזה מסדרת את צומת 3, והקריאה הרקורסיבית  $\text{MAX-HEAPIFY}(A, 8)$  אינה גורמת לשינוי נוסף.



מהו זמן הריצה של האלגוריתם  $\text{MAX-HEAPIFY}$  על תת-עץ בגודל  $n$ ? שורות 1-9 אורכות זמן קבוע, וגודלו של העץ, עליו מפעילים את הזימון הרקורסיבי בשורה 10, הוא לכל היותר  $2n/3$  (זה קורה כאשר בדיוק מחצית מהשורה התחתונה בעץ היא מלאה, ומפעילים את הזימון הרקורסיבי על תת-העץ השמאלי).

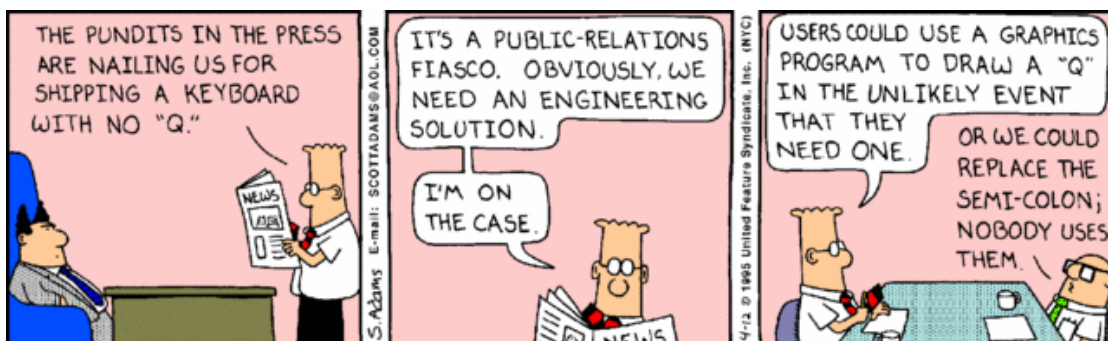
לכן, נוסחת הנסיגה המבטאת את זמן הריצה של האלגוריתם  $\text{MAX-HEAPIFY}$  היא:

$$T(n) = T(2n/3) + \Theta(1)$$

לפי משפט האב, נקבל שהפתרון הוא  $T(n) = \Theta(\log n)$ , כלומר – לוגריתמי במספר הצמתים בעץ (או: לינארי כגובהו של העץ  $\Theta(h)$ ).

### שאלה

תארו בעזרת שרטוט את פעולת האלגוריתם  $\text{MAX-HEAPIFY}(A, 1)$  כאשר מפעילים אותו על המערך  $A = (15, 4, 12, 9, 8, 10, 11, 5, 2, 7, 8, 8)$ .



### 3. בניית ערימה

ניתן להשתמש באלגוריתם MAX-HEAPIFY כדי ליצור ערימת מקסימום ממערך  $A[0..n-1]$ . מכיוון שהאיברים שנמצאים בין האינדקס  $\lfloor n/2 \rfloor$  (אמצע המערך) לבין האינדקס  $n-1$  (סוף המערך) הם כולם עלים של העץ, הרי שבתחילת התהליך כל אחד מהם הוא ערימת מקסימום בת איבר אחד. האלגוריתם BUILD-MAX-HEAP עובר על שאר הצמתים בעץ הבינארי, ומפעיל את MAX-HEAPIFY על כל אחד מהם. סדר המעבר על הצמתים, מהצמתים בעלי אינדקס גבוה לצמתים בעלי אינדקס נמוך (ובמערך זה בא לידי ביטוי במעבר מימין לשמאל) מבטיח שהתת-עצים המושרשים בבניו של צומת  $i$  הם ערימות מקסימום חוקיות, לפני ש-MAX-HEAPIFY מופעל על צומת זה.

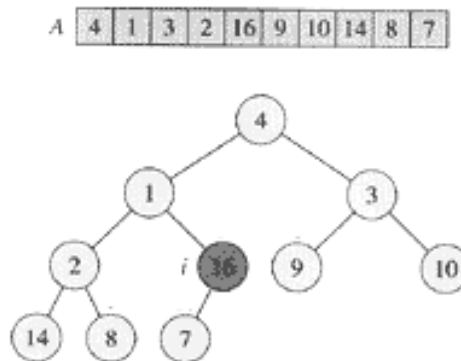
BUILD-MAX-HEAP(A)

```

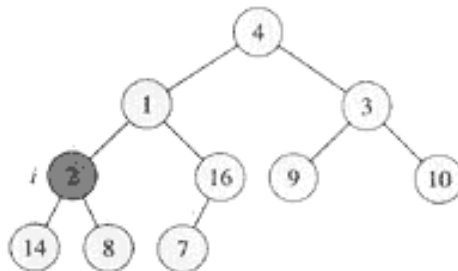
1   n ← A.heap-size ← A.length
2   for i ←  $\lfloor n/2 \rfloor - 1$  downto 0
3     do MAX-HEAPIFY(A, i)

```

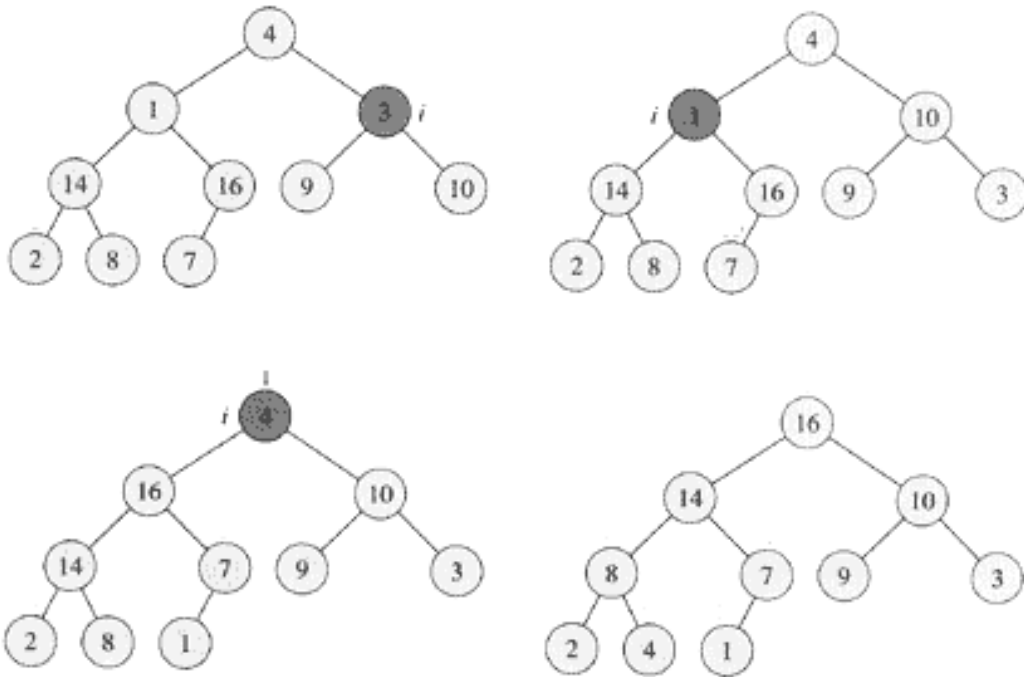
נדגים את פעולת האלגוריתם, על המערך הבא (בן 10 איברים) המייצג עץ בינארי כמעט שלם. ניתן לראות שהאינדקס  $i$  מקבל את הערך ההתחלתי 4 (באיטרציה הראשונה של הלולאה):



כך נראה העץ לאחר קריאה אחת ל-MAX-HEAPIFY. האינדקס  $i$  יקבל כעת את הערך 3:



כך יראה העץ באיטרציות הבאות של הלולאה. נשים לב שבכל פעם שמזמנים את MAX-HEAPIFY על צומת מסוים, אז שני תתי-העצים של הצומת הם ערימות מקסימום חוקיות:



מהי סיבוכיות זמן הריצה של האלגוריתם BUILD-MAX-HEAP?

ניתן למצוא חסם עליון אסימפטוטי, באופן הבא: כל קריאה ל-MAX-HEAPIFY רצה בזמן לוגריתמי ויש  $n/2$  קריאות כאלה, לכן – החסם העליון הוא  $O(n \log n)$ . חסם זה, הגם שהוא נכון, איננו חסם הדוק.

לצורך מציאת החסם ההדוק, יהיה עלינו להיעזר, בנוסף לטענות שראינו לעיל, גם בנוסחה הבאה:

$$\sum_{k=1}^{\infty} k \cdot x^k = \frac{x}{(1-x)^2} \text{ אז } |x| < 1, \text{ אם מתקיים}$$

כעת, נוכל לקבל חסם הדוק יותר, אם נשים לב כי זמן הריצה של MAX-HEAPIFY על צומת משתנה עם גובה הצומת בעץ, וכי רוב הצמתים אינם גבוהים. נסתמך על שתי הטענות הבאות: גובהה של ערימה בת  $n$  איברים הוא  $\lfloor \log_2 n \rfloor$ , ומספר הצמתים בכל גובה  $h$  בערימה כזו הוא לכל היותר

$$\left\lceil \frac{n}{2^{h+1}} \right\rceil$$

הזמן הדרוש להרצת MAX-HEAPIFY על צומת שגובהו  $h$  הוא  $\Theta(h)$ , ולכן מתקיים גם  $O(h)$ .

ניתן לבטא את זמן הריצה הכולל של האלגוריתם BUILD-MAX-HEAP באופן הבא :

יש לכל היותר  $\left\lceil \frac{n}{2^{1+1}} \right\rceil$  צמתים בגובה 1, ועבור כל אחד מהם מבצעים  $O(1)$  פעולות.

יש לכל היותר  $\left\lceil \frac{n}{2^{2+1}} \right\rceil$  צמתים בגובה 2, ועבור כל אחד מהם מבצעים  $O(2)$  פעולות.

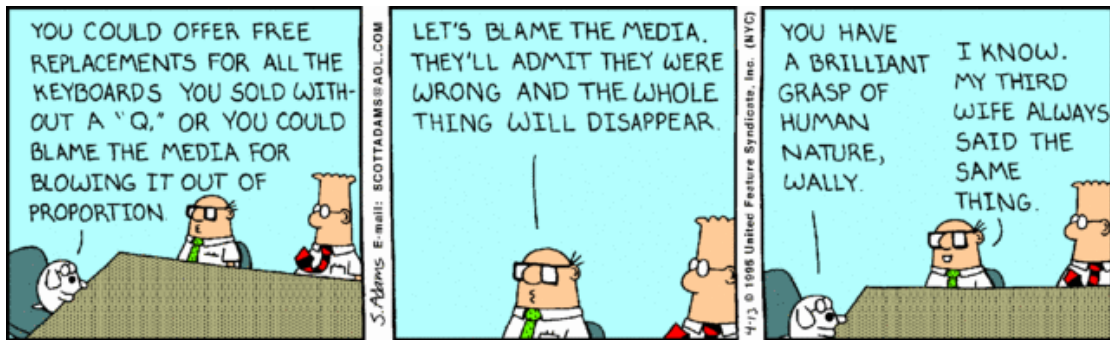
יש לכל היותר  $\left\lceil \frac{n}{2^{3+1}} \right\rceil$  צמתים בגובה 3, ועבור כל אחד מהם מבצעים  $O(3)$  פעולות.

...

יש לכל היותר  $\left\lceil \frac{n}{2^{\lfloor \log_2 n \rfloor + 1}} \right\rceil$  צמתים בגובה  $\lfloor \log_2 n \rfloor$ , ועבור כל אחד מבצעים  $O(\log n)$  פעולות.

נסכם הכול יחד, ונקבל :

$$\sum_{h=1}^{\lfloor \log n \rfloor} \left\lceil \frac{n}{2^{h+1}} \right\rceil \cdot O(h)$$



נכניס את הסכום לתוך הסימון האסימפטוטי  $O$ , ונקבל :

$$O\left(\sum_{h=1}^{\lfloor \log n \rfloor} \left(\frac{n}{2^{h+1}} \cdot h\right)\right)$$

ניתן להשמיט את ערך התקרה מבלי לפגום בנכונות הטענה :

$$O\left(\sum_{h=1}^{\lfloor \log n \rfloor} \left(\frac{n}{2^{h+1}} \cdot h\right)\right)$$

כפל זו פעולה המקיימת את חוק החילוף :

$$O\left(\sum_{h=1}^{\lfloor \log n \rfloor} \left(\frac{h}{2^{h+1}} \cdot n\right)\right)$$

ניתן להוציא את  $n$  אל מחוץ לסכום :

$$O\left(n \cdot \sum_{h=1}^{\lfloor \log n \rfloor} \frac{h}{2^{h+1}}\right)$$



כדי למצוא חסם עליון לסכום שבסוגריים, ניעזר בטענה 4 :

$$\sum_{k=1}^{\infty} k \cdot x^k = \frac{x}{(1-x)^2}$$

נציב  $x = 0.5$ , ונקבל :

$$\sum_{k=1}^{\infty} k \cdot 0.5^k = \frac{0.5}{(1-0.5)^2}$$

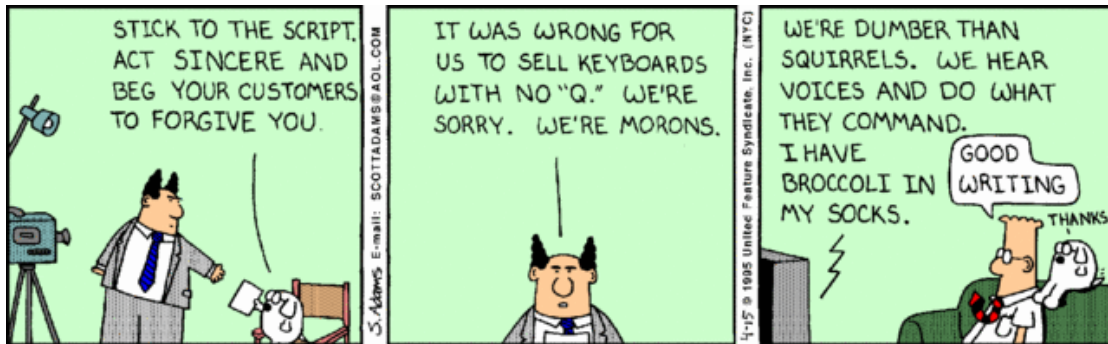
$$\sum_{k=1}^{\infty} \frac{k}{2^k} = \frac{0.5}{(-0.5)^2} = \frac{0.5}{0.25} = 2$$

ואת השיוויון  $\sum_{k=1}^{\infty} \frac{k}{2^k} = 2$  אפשר לחלק ב-2, ואז נקבל :  $\sum_{k=1}^{\infty} \frac{k}{2^{k+1}} = 1$

כעת :

$$O(n \cdot \sum_{h=1}^{\lfloor \log n \rfloor} \frac{h}{2^{h+1}}) \leq O(n \cdot \sum_{h=1}^{\infty} \frac{h}{2^{h+1}}) = O(n \cdot 1) = O(n)$$

בסך הכל, הראינו שזמן הריצה של האלגוריתם BUILD-MAX-HEAP חסום מלמעלה על-ידי ביטוי לינארי. מאידך, ברור שהוא חסום גם מלמטה על-ידי ביטוי לינארי (כי עוברים בלולאה על  $n/2$  איברים במעריך). מכך שמתקיים גם  $O(n)$  וגם  $\Omega(n)$ , נסיק שמתקיים  $\Theta(n)$ .



#### 4. מיון-ערימה

ערימה היא מבנה נתונים שניתן להשתמש בו כדי לממש אלגוריתם מיון יעיל, הנקרא 'מיון ערימה' (Heap Sort). בהינתן מערך, נפעיל את BUILD-MAX-HEAP כדי להפוך אותו לערימת מקסימום.

כעת, האיבר הגדול ביותר במערך מאוחסן ב- $A[0]$ . ניתן להציב אותו במקומו הסופי הנכון, על-ידי החלפתו עם  $A[n-1]$ .

נשים לב שאם "מסלקים" עכשיו את הצומת  $n$  מן הערימה, על-ידי הקטנת  $A.heap\text{-}size$  ב-1, אפשר להפוך בקלות את  $A[0..n-2]$  לערימה חוקית: בניו של השורש נותרו ערימות חוקיות, אולם ייתכן שהשורש החדש מפר את תכונת הערימה. כל מה שדרוש כדי להחזיר את תכונת הערימה, זו

קריאה ל-  $\text{MAX-HEAPIFY}(A, 0)$ . עם סיום הקריאה,  $A[0..n-2]$  יכול ערימת מקסימום חוקית המורכבת מ- $n-1$  איברים. על ערימה זו נוכל לחזור על התהליך שוב ושוב, ובכל פעם הערימה תקטן באיבר אחד.

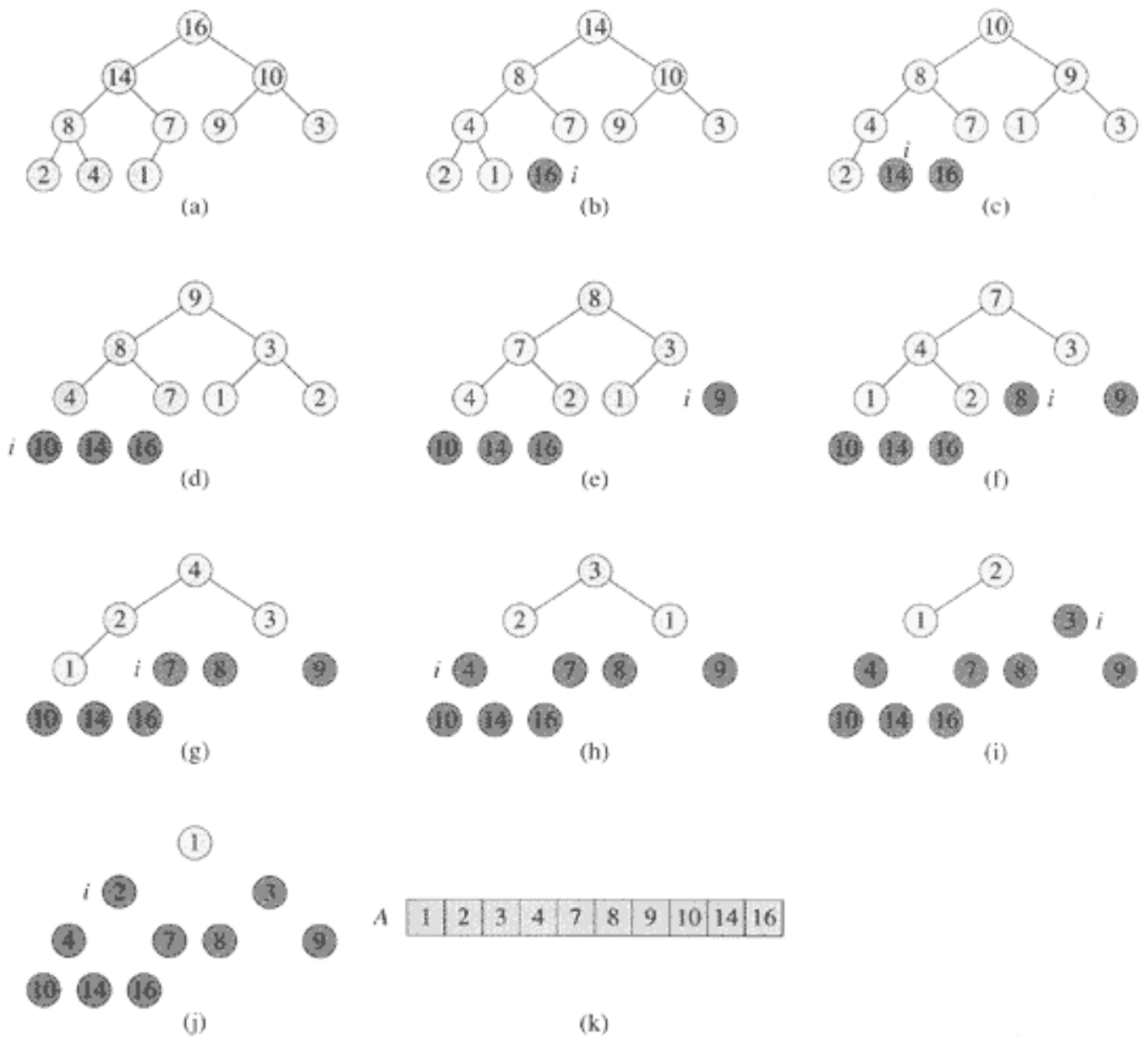
HEAP-SORT(A)

```

1  BUILD-MAX-HEAP(A)
2  for i ← A.length-1 downto 1
3      do SWAP(A[0],A[i])
4      A.heap-size ← A.heap-size - 1
5      MAX-HEAPIFY(A, 0)

```

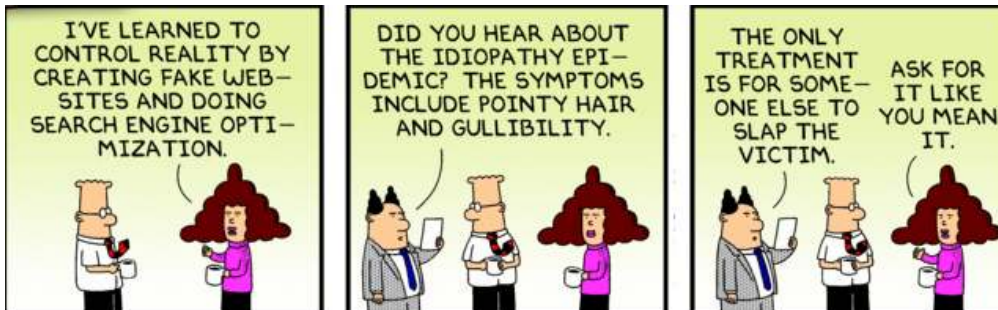
נדגים את פעולת האלגוריתם למיון-ערימה:



מהי סיבוכיות זמן הריצה של האלגוריתם? צעד מספר 1 אורך זמן לינארי, וכל אחת מ-1-n הקריאות ל-MAX-HEAPIFY מתבצעת בזמן לוגריתמי לגודל הערימה, אבל על ערימה שגובהה הולך וקטן. סך כל הפעולות שמתבצעות על-ידי זימונים אלו הוא :

$$\begin{aligned} & \log(n-1) + \log(n-2) + \log(n-3) + \dots + \log(2) + \log(1) = \\ & = \log((n-1) \cdot (n-2) \cdot \dots \cdot 2 \cdot 1) = \\ & = \log((n-1)!) = \\ & = \log\left(\frac{n!}{n}\right) = \\ & = \log(n!) - \log n = \\ & = \Theta(n \log n) - \log n = \\ & \Theta(n \log n) \end{aligned}$$

לכן, בסך-הכול, הסיבוכיות של מיון-ערימה היא  $\Theta(n \log n)$ .



## 5. תור קדימויות

מבנה הנתונים 'ערימה בינארית' מהווה דרך טובה מאוד לממש את טיפוס הנתונים המופשט 'תור קדימויות'. לשם כך, עלינו לספק מימוש לשלוש פעולות הממשק הבאות: הוספת איבר חדש, אחזור ערכו של האיבר המקסימלי, והוצאת האיבר המקסימלי.

אחזור ערכו של האיבר המקסימלי ימומש בצורה טריוויאלית, ובזמן קבוע  $\Theta(1)$ , באופן הבא :

```
MAXIMUM(A)
```

```
    return A[0]
```

את הפעולה של הוצאת האיבר המקסימלי נממש כך :

```
HEAP-EXTRACT-MAX(A)
```

```
1   max ← A[0]
2   A[0] ← A[A.heap-size-1]
3   A.heap-size ← A.heap-size - 1
4   MAX-HEAPIFY(A, 0)
5   return max
```

ברור שסיבוכיות זמן הריצה של הפעולה HEAP-EXTRACT-MAX היא  $\Theta(\log n)$ , שכן לבד מהזימון של MAX-HEAPIFY, המתבצע בזמן לוגריתמי, כל יתר הפעולות מתבצעות בזמן קבוע.

### שאלה

הדגימו את פעולת האלגוריתם HEAP-EXTRACT-MAX על המערך  $A = (10, 7, 5, 1, 6, 4)$ .

את הפעולה של הוספת איבר חדש ל- $A$ , בעל עדיפות  $key$ , נממש באמצעות הוספת עלה חדש לעץ. לאחר מכן, נסרוק מסלול מהעלה הזה לעבר השורש, כדי למצוא מקום מתאים לאיבר החדש:

```
HEAP-MAX-INSERT(A, key)
1   A.heap-size  $\leftarrow$  A.heap-size + 1
2   i  $\leftarrow$  A.heap-size-1
3   while i > 0 and A[PARENT(i)] < key
4       do A[i]  $\leftarrow$  A[PARENT(i)]
5         i  $\leftarrow$  PARENT(i)
6   A[i]  $\leftarrow$  key
```

זמן ריצת האלגוריתם על ערימה בת  $n$  איברים הוא  $\Theta(\log n)$ , שכן אורכו של המסלול מן העלה החדש ועד לשורש הוא  $\Theta(\log n)$ .

### שאלה

הדגימו את פעולת האלגוריתם HEAP-MAX-INSERT על המערך  $A = (11, 8, 9, 6, 7, 3, 5, 2, 4, 1)$ .