מבנה נתונים

עפיף חלומה

14 במאי 2009

תוכן עניינים

פרק 1

הרצאה מס.1

אתר: www.cs.huji.ac.il/~dast

פרק 2

הרצאה מס.2

סיקום של הרצאה קודמת: o, Θ, Ω, O

1. קבועים בפנים לא משנים
2. $f(n) = \overset{O}{\underset{\Theta}{\Omega}}(g(n)+h(n)) = \overset{O}{\underset{\Theta}{\Omega}}(g(n))$ כלומר $h(n)$ זניח יחסית ל $g(n)$ אז אפשר להשמיט את $h(n)$
3. $f(n) = n^d, g(n) = c^n$ אזי $g(n)$ מנצח את $f(n)$. $f(n) = O(g(n))$
4. $f(n) = \log^c(n), g(n) = n^d$ [1] אזי $g(n)$ מנצח את $f(n)$.

2.1 מיון בועות

לחןעיפטם 1.2 Bubble Sort

```
Bubble Sort A[1...n] {
if n>1 {
bubble(A[1...n])
bubblesort(A[1,...,n-1])
}
}
Bubble(A[1...n]) {
for j=1 to n-1 {
if A(j)>A(j+1) then swap
}
}
```

סיבוכיות של $Bubble$ היא .

$$\begin{aligned} T(n) &= T(n-1) + \Theta(n) \\ T(1) &= \Theta(1) \end{aligned}$$

[1] כאשר אומרים log מתקוונים ל$\log_2$ שהוא שווה ל$\log_{10}$ עד כדי קבוע

נרצה לפתור את הבעיה הזו ולנצל את זה לראות כל מיני שיטות לפתור בעיות ריקורסיה.

2.1.1 השיטה האיטרטיבית

זה פשוט "לא יודעים אז בואו ננסה"

$$\begin{aligned} T(n) &\leq T(n-1) + c \cdot n \\ &\leq T(n-2) + c(n-1) + cn \\ &\leq T(n-3) + c(n-2) + c(n-1) + cn \\ &\leq T(1) + 2c + 3c + \cdots + c(n-1) + cn \\ &\leq T(1) + \sum_{i=2}^{n} c \cdot i \\ &= T(1) + c\left(\frac{n(n-1)}{2} - 1\right) \\ &= O\left(n^2\right) \end{aligned}$$

מאותו שיקול

$$\begin{aligned} T(n) &\geq T(n-1) + c'n \\ &\vdots \\ &\geq \Theta\left(n^2\right) \end{aligned}$$

מסיקים כי $T(n) = \Omega\left(n^2\right)$
לכן $T(n)$ הוא גם $O\left(n^2\right)$ וגם $\Omega\left(n^2\right)$ ולכן $T(n) = \Theta\left(n^2\right)$

2.2 שיטת הניחוש וההוכחה באינדוקציה

ננסה להוכיח באינדוקציה $T(n) \leq cn^2$

$$\begin{aligned} T(n) &= T(n-1) + \Theta(n), \Theta(n) \leq cn \\ T(1) &= \Theta(1) \leq c \end{aligned}$$

עבור $n = 1$: $T(1) \leq c$ בסדר.
נניח עבור n ש $T(n) \leq cn^2$
נוכיח עבור $n+1$ כי $T(n+1) = T(n) + c(n+1)$.

$$\begin{aligned} T(n+1) &= T(n) + c(n+1) \\ &\overset{\text{מהנחת האינדוקציה}}{\leq} cn^2 + c(n+1) \\ &= c\left(n^2 + n + 1\right) \\ &\leq c(n+1)^2 \end{aligned}$$

זה מראה כי $T(n) = O\left(n^2\right)$.

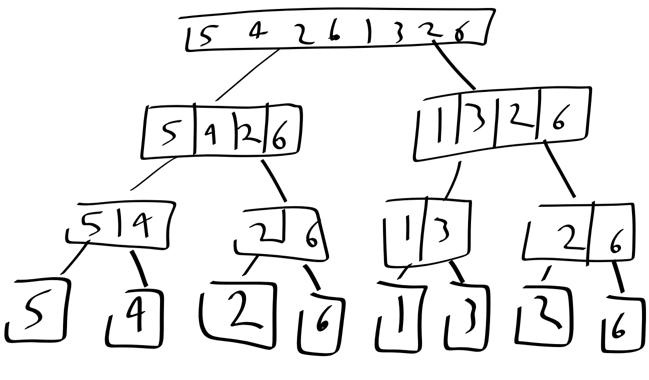
באותה צירה אפשר להוכיח הצד השני של האינדוקציה

2.3 מיון מיזוג - Merge Sort

2.2 לחןעיפטם Merge Sort

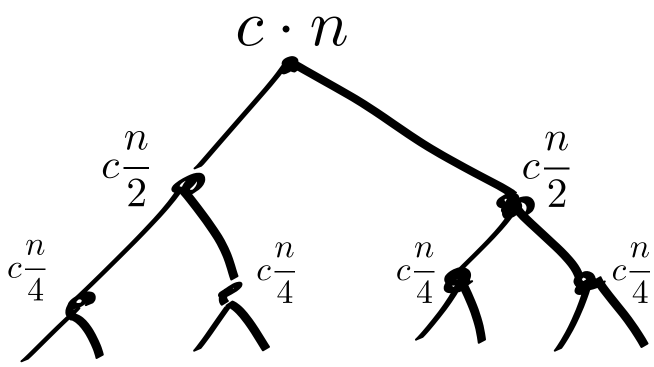
```
Merge Sort(A[1...n]){
if n>1 {
mergesort(A[1...n/2])
mergesort(A[n/2 + 1...n])
merge(A[1,...n/2],A[n/2 + 1...n],A[1...n])
}
}
Merge(A,B,C) {
l=1,i=1,j=1
do while i<n/2 or j<n/2 {
c[l]=min(A[i],B[i])
if min was A(i) increment i, else incrememnt j
increment l
}
```

עץ הרקורסיה:



איור 2.1: עץ הריקורסיה של מיון מיזוג

נבנה עכשיו את עץ הריקורסיה ונכתוב כמה פעולות קוראות בכל קודקוד.

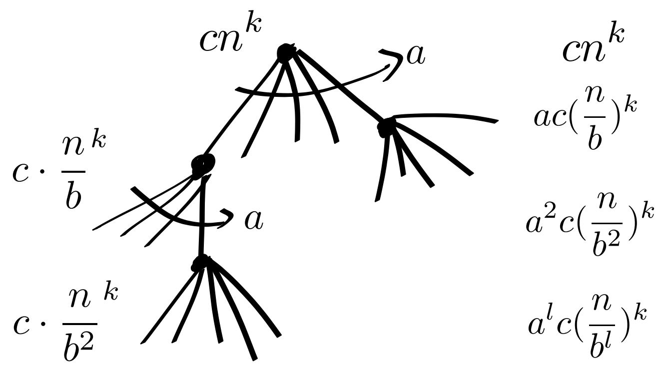


איור 2.2: משקל כל קודקוד במיזוג מיון

ניתן לראות כי בכל שורה בעץ יש cn פעולות. אזי האלגורתם הזה עולה $cn \cdot (\text{גובה העץ}) = cn \cdot \log_2(n)$

2.4 סיבוכיות של אלגורתם אלפרדו משהוא

$$T(n) = aT(n/b) + \Theta\left(n^k\right)$$



איור 2.3: עץ קריאה של אלגוריתם אלפרדו

$m = \log_b n$ אזי מספר השורות הוא $m+1$

העבודה בשורה העליונה היא $\Theta\left(cn^k\right)$

העבודה השורה התחתונה היא $\Theta\left(a^{\log_b n}\right)$

אזי

$$
\begin{aligned}
T(n) &\leq cn^k \sum_{l=0}^{m} \left(\underbrace{\frac{a}{b^k}}_{k} \right)^l \\
&\leq cn^k \left(\frac{q^{m+1}-1}{q-1} \right) = cn^k \left(\frac{1-q^{m+1}}{1-q} \right) = cn^k \left(\frac{1-q^{m+1}}{1-q} \right) \\
&\underbrace{\leq}_{q<1} cn^k \frac{1}{q-1} = O\left(n^k\right) \\
T &= O\left(n^k\right)
\end{aligned}
$$

מה אם $q = 1$ כמו ב merge sort?

$$
\begin{aligned}
T(n) &\leq cn^k (m+1) \\
&= cn^k \left(\log_b (n+1)\right) \\
&= \Theta\left(n^k \log n\right)
\end{aligned}
$$

מה אם $q > 1$?

$$
\begin{aligned}
T(n) &\leq cn^k \frac{q^{m+1}-1}{q-1} \\
&= \Theta\left(n^k q^m\right) \\
&= \Theta\left(n^k \left(\frac{a}{b^k}\right)^m\right) \\
&= \Theta\left(n^k \frac{a^m}{a^{mk}}\right) \\
&\underbrace{=}_{b^m=n} \Theta\left(a^m\right) \\
&= \Theta\left(a^{\log_b n}\right) \\
&\underbrace{=}_{\text{תרגיל}} \Theta\left(n^{\log_b a}\right)
\end{aligned}
$$

פרק 3

תרגול מס.2

מעריכים אלגוריתם לפי זמן ריצה וכמה מקום הוא לוקח בזכרון, מתארים את זה דרך פונקציות $\mathcal{O},\theta,\Omega$

3.1 שיטות לפתור ריקורסיה

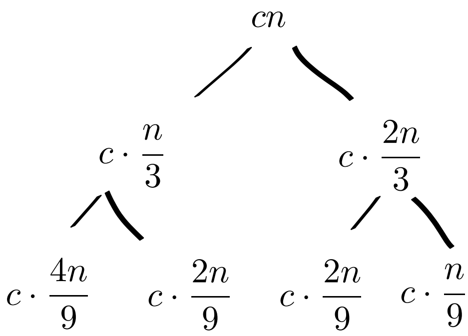
1. שיטה איטרטיבית
2. לנחש ולהוכיח באינדוקציה
3. עץ ריקורסיה
4. משפט המאסתר - Master Theorem
5. לחסום מלמעלה ומלמטה על ידי פומקציה אחרת

3.1.1 שיטת עץ הריקורסיה

משתמשים בזה לנחש את הפתרון
אחרי שמנחשים צריך להוכיח
למשל, תמצא חסם עליון על $T\left(1\right)=6,T\left(2\right)=7,T\left(3\right)=10$

$$\begin{aligned} T\left(1\right) &= 6,T\left(2\right)=7,T\left(3\right)=10 \\ T\left(n\right) &= T\left(\frac{n}{3}\right)+T\left(\frac{2n}{3}\right)+\Theta\left(n\right) \end{aligned}$$

את $\Theta\left(n\right)$ נכתוב בצורת cn



איור 3.1: דוגמה לעץ ריצה

אפשר לראות כי בכל שורה יש cn פעולות. אבל זה לא בדיוק נכון כי הענף הימני ארוך יותר מהענף השמאלי.

הגובה המקסימאלי של עץ זה הוא $h = \log_{3/2}(n)$

אזי נרצה להוכיח כי $cn \log_{3/2} n = \mathcal{O}(n \log n)$

פרק 4

הרצאה מס.3

4.1 אלגוריתם Quicksort

Quicksort	לחןעיפטם 1.4

```
Quicksort[Left, Right]
–if(right>left)
—-m←partition[A, Left, Right]
—-QuckisortA [Left, m − 1]
—-QuicksortA [m + 1, Right]
Partition A [Left, Right]
–If Left < Right
—-L = Left, R = Right, pivot = A (Right)
—-for j = Left to Right
——if A (j) < pivot
———B (L) = A (j) , L + +
——else
———B (R) = A (j) , R − −
—-B (L) = A [Right]
—-A [Left, Right] = B [Left, Right]
—-return L
```

יש גרסה יותר יעילה לפונקציה $Partition$:

Partition In Place	לחןעיפטם 2.4

```
Partition In Place A [Left, Right]
–pivot = A [Right]
–scan A from Left to find A(j) > pivot
–scan B from Right to find A(j) < pivot
–Swap
–continue until pointers cross
–swap pivot with that location(where pointers cross)
–return pointer
```

אז רוצים לנתח את $Quicksort$:

$$T\left(n\right)=T\left(m-1\right)+T\left(n-m\right)+\Theta\left(n\right)$$

אבל את כל זה תלוי ב m שאי אפשר לבטא אותו באופן חד חד ערכי. אז נסתכל על כמה מקרי קיצון:

1. אם $m=n-1$ תמיד מקבלים $T\left(n\right)=T\left(n-2\right)+T\left(1\right)+\theta\left(n\right)$ ועומק הריקורסיה הוא n אזי $T\left(n\right)=\Theta\left(n^{2}\right)$
2. אם $m=\frac{n}{2}$ תמיד מקבלים $T\left(n\right)=T\left(\frac{n}{2}-1\right)+T\left(\frac{n}{2}\right)+\Theta\left(n\right)$ אזי $T\left(n\right)=\Theta\left(n\log n\right)$ כמו Mergesort
3. נניח $m=\frac{9}{10}m$ אזי מקבלים $T\left(n\right)=T\left(\frac{9}{10}n\right)+T\left(\frac{n}{10}\right)+\Theta\left(n\right)$ אזי העומק המקסימאלי הוא $\log_{10/9}n=\frac{\log_{2}n}{\log_{2}\frac{10}{9}}$ אבל הרבה ענפים נפסקים באמצע אזי מקבלים כי זה $\mathcal{O}\left(n\log n\right)$[1]

טענה: לכל קלט $T\left(n\right)=\mathcal{O}\left(n^{2}\right)$ וכן $T\left(n\right)=\Omega\left(n\log n\right)$
הקלט האכי גרוע הוא מערך ממיון וסיבוכיות של Worst Case של Quicksort הוא $\mathcal{O}\left(n^{2}\right)$

4.1.1 סיבוכיות ממוצעת

$$\left\langle T\left(n\right)\right\rangle =\sum_{\delta}\frac{\left(T\left(n\right)\ for\,input\,\delta\right)}{n!}$$

δ הוא מיון מסויים של המספרים $\left\{ k_{1},\ldots,k_{n}\right\}$
מקבלים כי $\left\langle T\left(n\right)\right\rangle =\Theta\left(n\log n\right)$ אבל את זה לא נוכיח
דרך אחרת לחשב התוכלת היא להגריל את m שזה די נכון לעשות ואז נחשב את התוחלת $E\left(T\left(n\right)\right)$

$$E\left(T\left(n\right)\right)=\sum_{m=1}^{n}\frac{1}{n}\left(E\Big(T\left(m-1\right)\Big)+E\Big(T\left(n-m\right)\Big)\right)+\Theta\left(n\right)$$

[1] שים לב, זה $\mathcal{O}$ ולא Θ

פרק 5

הרצאה מס.4

התוכלת של Randomized Quicksort היא

$$\begin{aligned} E\left(T\left(n\right)\right) &= \sum_{m=1}^{n}\frac{1}{n}\left(E\Big(T\left(m-1\right)\Big)+E\Big(T\left(n-m\right)\Big)\right)+\Theta\left(n\right) \\ &= \sum_{m=0}^{n-1}\frac{1}{n}\cdot 2\cdot E\left(T\left(m\right)\right)+\Theta\left(n\right) \end{aligned}$$

בקורס זה תמיד $T\left(1\right)=\times\left(1\right)$ אזי

$$\sum_{m=0}^{n-1}\frac{2}{n}E\left(T\left(m\right)\right)+c_2 n \qquad \leq \qquad E\left(T\left(n\right)\right)\leq\sum_{m=0}^{n-1}\frac{2}{n}E\left(T\left(m\right)\right)+c_1 n$$

$$c_2\leq \quad E\left(T\left(1\right)\right) \quad \leq c_1$$

אז נגדיר

$$\begin{aligned} U_C\left(n\right) &= \sum_{m=0}^{n-1}\frac{2}{n}U_c\left(m\right)+cn \\ U_c\left(1\right) &= U_c\left(0\right)=c \end{aligned}$$

טענה:

$$U_{c_2}\leq E\left(T\left(n\right)\right)\leq U_{c_1}\left(n\right)$$

ניתן להוכיח באינדוקציה(ההוכחה בתרגיל)

$$=$$

$$=$$

$$-\begin{cases} nU_c(n) &= 2\sum_{m=0}^{n-1} U_c(m) + cn^2 \\ (n+1)\,U_c(n+1) &= 2\sum_{m=0}^{n} U_c(m) + c(n+1)^2 \end{cases}$$

$$\begin{aligned} (n+1)\,U_c(n+1) - nU_c(n) &= 2U_c(n) + c(n+1)^2 - cn^2 \\ (n+1)\,U_c(n+1) &= (n+2)\,U_c(n) + 2cn + c \\ U_c(n+1) &= \frac{n+2}{n+1}U_c(n) + \frac{2cn+c}{n+1} \end{aligned}$$

טענה קלה:

$$c \le \frac{2cn+c}{n+1} \le 2c$$

אזי

$$U_c(n+1) \le \frac{n+2}{n+1}U_c(n) + 2c$$

נפתור את זה בשיטת האיטרציה:

$$\begin{aligned} U_c(n+1) &\le \frac{n+2}{n+1}\left(\frac{n+1}{n}u_c(n-1) + 2c\right) + 2c \\ &= \frac{n+2}{n+1}\cdot\frac{n+1}{n}U_c(n-1) + \frac{n+2}{n+1}\cdot 2c + \frac{n+2}{n+2}\cdot 2c \\ &\le \frac{n+2}{n+1}\cdot\frac{n+1}{n}\left(\frac{n}{n-1}U_c(n-2) + 2c\right) + \frac{n+2}{n+1} + \frac{n+2}{n+2}2c \\ &\le \frac{n+2}{n+2}\cdot 2c + \frac{n+2}{n+1}\cdot 2c + \frac{n+2}{n} + 2c + \frac{n+2}{n+1}\cdot\frac{n+1}{n}\left(\frac{n}{n-1}U_c(n-2) + 2c\right) \end{aligned}$$

רואים כי כל כופלי $U_C(\dots)$ מצתמצמים ונשאר המונה הראשון והמכנה האחרון
אזי

$$U_c(n+1) \quad \le \quad (n+2)\,2c\sum_{j=1}^{n+2}\frac{1}{j} + (n+2)\cdot 2c$$

פרק 6

הרצאה מס.?

השיעורים הקודמים היו סתם הכנה למתמטיקה. היום נכנס למבנה הנתונים.
סוג נתונים אבסטרקטי[1] הוא אוסף נתונים + פעולות. דוגמעות

- מחסנית: push, pop LIFO
- תור: enqueue, dequeue FIFO

6.1 תור קדימיות[2]

קיימים שתי סוגים של תור זה:

- Min Priority Queue
- Max Priority Queue

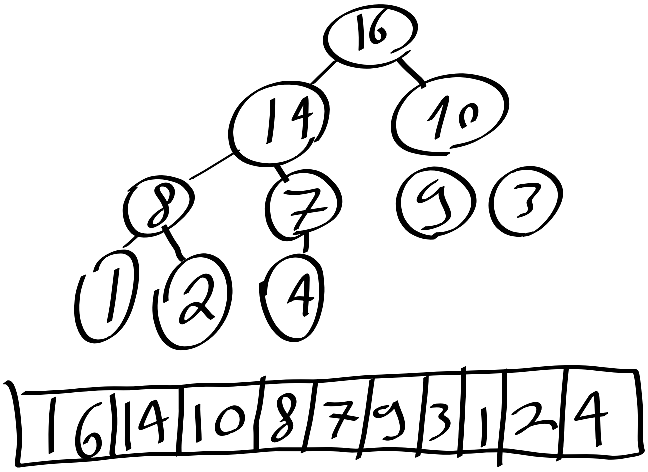
פעולות:

- $max\left(S\right)$ מחזירה רשומה עם ערך קדימות הכי גבוה.
- $extract-max\left(S\right)$ אותו דבר אבל עם הוצאת הרשומה מהתור.
- $insert\left(S,x\right)$ מכניסה לתור רשומה עם קדימות x
- $increase\_key\left(S,i,key\right)$ מעלה את קדימות של האיבר הi בתור לkey

6.1.1 מימוש בעזרת ערימה

ערימה זו עץ קמעט שלם שמקיים תכונת הערימה כלומר: $\forall i:A\left[parent\left(i\right)\right]\geq A\left(i\right)$

[1] Abstract Data Type
[2] Priority Queue



איור 6.1: עץ קמעט שלם עם הצגתו במערך

במערך של העץ מתקיים:

$$
\begin{aligned}
left\left(i\right) &= 2i \\
right\left(i\right) &= 2i+4 \\
parent\left(i\right) &= \left\lfloor \frac{i}{2} \right\rfloor
\end{aligned}
$$

רוצים לממש את הפעולות של ה ADT של תור הקדימות כך שמבנה הערימה ישמר.

אז נממש את הפעולות שהגדרנו:

אלגוריתם 1.6 פעולת $extract\_max\left(S\right)$ בערימת קדימות

```
max(S){
return A(1);
}
extract-max(S){
max=S(1)
swap(min (S) , S (1))
heapsize–;
max_heapify(A,1);
}
min(S) {
return S [heapsize (S)]
}
max_heapify(A,i) {
l=left(i);
r=right(i)
if(l ≤ heapsize [A]) then {
if(r > heapsize [A]) then r=reapsize(A)
if(A (l) > A (i)) largest=l, else largest=i
if(A (r) > A (largest)) largest=r
if(largest ≠ i) then swap (A [largest] , A [i]), max_heapify(A,largest)
}
}
```

הסיבוכיות של maxheapify היא $\mathcal{O}\left(D\right)=\mathcal{O}\left(\log n\right)$

אלגוריתם 2.6 פעולת $increase\_key\left(A,i,key\right)$ בערימת קדימות

```
increase_key(A, i, key) {
if(key < A [i]) error "new key smaller than old"
A(i)=key
while i>1 and A[parent (i)] < A [i] {
swap(A [i] , A [parent (i)])
i=parent(i)
}
}
```

סיבוכיות של פעולה זו היא גם $\mathcal{O}\left(\log n\right)$

אלגוריתם 3.6 פעולת insert

```
heapsize(a)++
A(heapsize (A)) = −∞
increase_key[A, heapsize (A) , key]
```

6.1.2 איך מייצרים ערימה?

יש כל מיני פתרונות שעובדות ב $\mathcal{O}\left(n\log n\right)$ אבל יש גם דרך לעשות את זה ב$\mathcal{O}\left(n\right)$:

אלגוריתם 4.6 בנית ערימה

```
build_heap(A) {
heapsize[A]=length[A]
for(i=n to 1) max_heapify(A,i)
}
```

זה נראה כמו $\mathcal{O}(n\log n)$ אבל באמת זה יוצא $\mathcal{O}(n)$
האלגוריתם הזה עובד כי:

1. כל שורש בקודקוד בערימת מקסימום זה שורש של ערימת מקסימום.
2. כל קודקוד בעץ כמעט שלם הוא שורש של עץ קמעט שלם
3. אם מפעילים max_heapify על קודקוד i ששני בניו שורשי ערימות מקסימום אזי i יהיה שורש של ערימת מקסימום

פרק 7

הרצאה מס.?

ערמה זה עץ כמעט שלם
כל קודקוד גדול אושווה לבניו.
נרצה לכתוב את האלגוריתם לבנות ערמה חדשה ממערך

אלגוריתם 1.7 בנית ערימה BuildHeap(A)

```
buildHeap(A) {
n=length(A)
for i=n to 1 do{
max_heapify(A, i)
}
}
```

נרצה עכשיו לראות למה זה עובד ומה הזמן ריצה של זה.[1]
הדרך להוכיח את זה הוא דרך אינדוקציה.
אינטואיציה: האלגוריתם כל פעם מתקן הערימה הקטנה שיש לו ומתחיל מהסוף עד להתחלה. אז כל פעם שאנחנו מפעילים max_heapify יש לנו כבר שתי תתי ערימות תקינות שמחוברים לקודקוד שאנחנו מפעילים אותה עליו)

הוכחה פורמלית: נשתמש במשהוא שנקרא Loop Invariant שיש לו שלוש תכונות:

1. קודקוד בעץ כמעט שלם הוא שורש של עץ כמעט שלם.
2. קודקוד בערימת מקסימום הוא שורש של ערימת מקסימום.
3. אם מריצים max_heapify על קודקוד שבניו הם ערימת מקסימום אז מתקבלת ערימת מקסימום.

בשלו הi הקודקודים $n, n-1 \ldots n-i+1$ הם שורשים של ערימת מקסימום

הוכחה:

בסיס אינדוקציה: עבור בסיס $k = 1$ זה טריוויאלי כי כל עלה הוא כבר ערימת מקסימום.

הנחת האינדוקציה: $n, n-1 \ldots n-k+1$ הם שורשים לערימת מקסימום.

[1] זאת הבעיה הראשונה שאנחנו נוכיח כי היא עובדת.

היכחה עבור $k+1$: לפי הנחת האנדוקציה $right\left(n-k\right), left\left(n-k\right)$(כי האינ-קסים האילו יותר גדולים מ$n-k$) הם ערימות מקסימום. לכן מותר לבצע max_heapify. לכן אחרי הפעלת max_heapify על $n-k$ הוא נהפך שורש של ערימת מקסימום. ולפי ההנחה כל תת-עץ הוא גם ערימת מקסימום. זה מה שרוצים להוכיח.

7.1 זמן ריצה של Max_Heapify

יודעים כבר כי max_heapify הוא רץ בסיבוכיות זמן $\mathcal{O}\left(\log n\right)$. אבל זה זמן ריצה מקסימאלי($\mathcal{O}\left(\log n\right)$ ולא $\Theta\left(\log n\right)$) אז נרשום ביטוי יותר מדויק: מסמנים עימק העץ ב D אז ברמה הראשונה(השורש) יש D פעולות. ברמה השניה יש לכל קודקוד $D-1$ פעולות אזי $\left(D-1\right)2^{1}$. הביטוי הכללי עבור כל רמה d הוא $\left(D-d\right)2^{d}=h2^{D-h}$. אזי

$$\sum_{n=1}^{D}h\cdot2^{D-h}=2^{D}\sum_{h=1}^{d}h2^{-h}=2^{D}\sum_{h=1}^{d}h\left(\frac{1}{2}\right)^{-h}$$

יודעים כי אם $x\in\left(0,1\right)$ אזי $\lim_{h\rightarrow\infty}\sqrt{x}^{h}=0$ אז קיים k' כך שבו מתקיים $\sqrt{x}^{k'}=1$

$$\begin{aligned}
\sum_{k=1}^{\infty}kx^{k} & = \sum_{k=1}^{k'}kx^{k}+\sum_{k=k'+1}^{\infty}kx^{k}\\
 & = C+\sum_{k=k'+1}^{\infty}\underbrace{k\sqrt{x}^{k}}_{<1}\sqrt{x}^{k}\\
 & \leq C+\sum_{k=k'+1}^{\infty}1\cdot\sqrt{x}^{k}\\
 & \leq C+\frac{1}{1-\sqrt{x}}+c
\end{aligned}$$

שהביטוי הזה תלוי רק ב x שזה במקרה שלנו $\left(\frac{1}{2}\right)$

7.2 עץ חיפוש בינארי

עץ בינארי בו עבור כל קודקוד x מתקיים

- עבור y בתת עץ שמאלי של $key\left(y\right)\leq key\left(x\right)$
- עבור y בתת עץ ימיני של $key\left(y\right)>key\left(x\right)$

רוצים שעץ הזה יתמוך בפעולות הבאות:

- Insert(T,x)
- Remove(T,x)
- Search(T,x)
- min(T)

- max(T,x)
- Successor(T,x)
- Pre....(T,x)

אז הקל ביותר זה Max,Min ומחזירים האיבר הימיני ביותר או השמאלי ביותר.

לחןעיפטם 2.7 Search(node,value)

```
search(n,x) {
if(key(n)==x || n==null) return n
if(key(n)<x) return search(right(n),x);
else return search(left(n),x);
}
```

למצא successor:

- אם ל x יש בן ימני y אזי min(T,y) עוקב של x
- אין בן ימיני אזי נעלה בעץ עד שנמצא עץ שx הוא בן ימיני שלו.
- אם x הוא הmaximum