מבני נתונים ומבוא לאלגוריתמים
20407

ממן 12

מוגש ע"י אורנית כהן גינדי

(המטלה מוקלדת בPPTX אוכל לשלוח קובץ
מהסוג הזה במייל אם תרצי)

ממן 12 שאלה 1

א. נתון המערך A= [45, 33, 55, 15, 28, 66, 64, 18, 14 , 49] ונבצע עליו BUILD-MAX-HEAP n=10 וזה גם יהיה גודל הערימה heap-size = 10, ערך i התחלתי הוא i = n/2 = 5 כדי לדלג על העלים שממילא מקיימים ערימת מינימום:

נבצע MIN-HEAPIFY(A,5) . 28 הוא אב של 49, ומקיים ערימת מינימום לכן לא ישתנה כלום בשלב זה:

i=5 →

1	2	3	4	5	6	7	8	9	10
45	33	55	15	28	66	64	18	14	49

כל אלו עלים

MIN-HEAPIFY(A,4) ימצא ש 15 מפר את ערימת המינימום וכי בנו 14 קטן יותר והם יתחלפו

i=4 →

1	2	3	4	5	6	7	8	9	10
45	33	55	**14**	28	66	64	18	**15**	49

MIN-HEAPIFY(A, 3) ימצא ש 55 מקיים ערימת מינימום ולכן לא ישתנה כלום בשלב זה:

i=3 →

1	2	3	4	5	6	7	8	9	10
45	33	55	14	28	66	64	18	15	49

MIN-HEAPIFY(A, 2) ימצא ש 33 מפר את ערימת המינימום ולכן 33 יתחלף עם בנו 14, אבל גם במקומו החדש הוא עדיין מפר את ערימת המינימום ויתחלף עם בנו החדש 15 :

i=2 →

1	2	3	4	5	6	7	8	9	10
45	**14**	55	**15**	28	66	64	18	**33**	49

MIN-HEAPIFY(A, 1) ימצא ש 45 מפר את ערימת המינימום ולכן יתחלף עם בנו 14, אבל גם במקומו החדש הוא עדיין מפר את ערימת המינימום ויתחלף עם 15 אך גם במקום זה לא מצא את מנוחתו וימשיך ויתחלף עם 18 שקטן ממנו:

i=1 →

1	2	3	4	5	6	7	8	9	10
14	**15**	55	**18**	28	66	64	**45**	33	49

i=0 **done!**

ממן 12 שאלה 1 המשך

ב. נתונה ערימת מינימום A בגודל n

אלגוריתם למציאת איבר מקסימלי בערימת מינימום :

```
FIND-MAX(A)
if n  < = 2
then return A[n]
start ← ceiling (A.size /2)
max ← A[ start ]
for  i ← start  to  n  do
    if A[ i ] > max
    then max  ← A[ i ]
return max
```

האיבר הגדול ביותר בערימת מינימום חייב להיות איבר ללא בנים (אחרת אינו האיבר הגדול ביותר, מהגדרת ערימת מינימום) , מכאן שהוא עלה של הערימה ויש לחפשו בין n/2 (ערך עליון) איברים האחרונים של הערימה.

את הדרישה שמספר השוואות קטן מ n-1 , מקיים התנאי שבראש האלגוריתם, אשר בודק אם גודל הערימה הוא עד 2 איברים: כאשר יש איבר אחד אז הוא המינימום והמקסימום של הערימה, וכאשר יש 2 איברים אז האיבר האחרון הוא המקסימלי.

זמן הריצה O(n)

ג. נתונה ערימת מינימום A בגודל n

אלגוריתם למציאת איבר השביעי בגודלו בערימת מינימום:

```
FIND-SEVENTH(A)
max ← 127
if n < 127  then max ← n
for i ← 1 to n
    do  tmpArray[ i ] ← A[ i ]
Insertion-Sort(tmpArray)
return tmpArray[ 7 ]
```

האיבר השביעי בגודלו יכול להיות לכל היותר ברמה ה7 של ערימת המינימום, כמות האיברים המקסימלית עד הרמה השביעית שהיא רמה מספר 6 (כאשר הרמה הראשונה נספרת כרמה 0) היא $2^7 - 1 = 127$

האלגוריתם אוסף את כל האיברים בתחום הזה למערך בגודל קבוע O(1) כך שבטוח שהאיבר המבוקש ביניהם ומכיון שהמספר 127 הוא קבוע, כל מיון של המערך יהיה בסיבוכיות של O(1) ומציאת השביעי בגודלו תהיה גם היא בO(1)

ד. אלגוריתם למציאת איבר השביעי בגודלו מבין אברי המסלול הימני בערימת מינימום A:

```
FIND-SEVENTH-IN-RIGHT(A)
return A[ 127 ]
```

נתון שגובה הערימה גדול מ7, כך שיש לפחות 7 רמות שהן מלאות לחלוטין (מתחייב מהגדרת ערימה שלא יתכנו חוסרים בצמתים מלבד ברמה התחתונה)

אם נסתכל על מסלול כלשהו בערימת מינימום, שוב מהגדרת הערימה, מתחייב שהערך בכל צומת יהיה גבוה מצומת שמעליו במסלול, מכאן שהאיבר השביעי בגודלו במסלול הימני הוא פשוט האיבר הימני ביותר ברמה השביעית אשר נספרת מהשורש. מכיוון שהשורש נחשב רמה מספר 0 אז הרמה השביעית היא רמה מספר 6.

כמות האיברים המקסימלית עד הרמה השביעית היא $2^7 - 1 = 127$, וזה יהיה האינדקס של האיבר השביעי בגודלו במסלול הימני בייצוג של הערימה כמערך.

הסיבוכיות: O(1)

ממן 12 שאלה 2

א. האלגוריתם מתבסס על הענין שהבנים של הצומת Z שומרים על תכונת הערימה גם אחרי שמתווסף קבוע C>0 אליהם ולצאצאיהם.

ההוספה של C יכולה לגרום להפרה של חוקיות הערימה אצל צומת Z ואז יש צורך בביצוע פעולת התיקון HEAPIFY על Z (שבגובה k).

פעולה זו אמנם תתקן את Z אבל יכול להיות שכתוצאה מכך אבא של Z יפר את חוקיות הערימה וכן הלאה עד לשורש העץ.

האלגוריתם רץ בלולאה מצומת Z (גובה k) כלפי מעלה עד לשורש העץ, כך שמספר האיטרציות הוא lgn –k

ובכל איטרציה מתבצעת פעולת HEAPIFY שעולה lgn

ובסך הכל זמן הריצה הוא O(lgn * (lgn –k))

```
CORRECT-H-1 (H, Z)
i ← lgZ
 //commen:  take Z as index in the heap array
while i  >  1/2 do
   if left(H ,i) > H[ i] or right(H, i) > H[i]
   then   heapify (H , i)
          i ← parent (H ,i)
   else return
```

ב. האלגוריתם עובר רקורסיבית על כל הצמתים שמתחת לשורש Z, ומפעפע אותם מעלה (BUBBLE UP שגרת עזר שמחליפה ערך של צומת עם ערך של צומת אבא במידה והוא גדול יותר מהאבא וכן הלאה כלפי מעלה עד למציאת מקומו , כמו שמתואר באלגוריתם הכנסת איבר חדש לערימה) .

כמות הצמתים מתחת לZ היא 2^(k+1) -1 וזה מספר האיטרציות של הלולאה.

פעולת הפעפוע נעשית מקסימום lgn פעמים כך שסה"כ זמן הריצה הוא O(2^k * lgn) (כמובן לאחר הזנחת קבועים)

```
CORRECT-H-2 (H, node)
 //call with node =Z
if node exists
then BUBBLE-UP( node)
        CORRECT-H-2 (H, left(node))
        CORRECT-H-2 (H, right(node))
else return
```

ג.

איזו שגרה עדיפה	**2**. O(2^k * lgn)	**1**. O(lgn * (lgn –k))	
2 עדיפה כיוון ש O(lgn) < O(lg^2n)	O(2^C * lgn) = O(C'*lgn) = **O(lgn)**	O(lgn * (lgn –C)) = **O(lg^2n)**	K = C
שתי השגרות באותה עדיפות	O(2^(lglgn) *lgn) = **O(lg^2n)**	O(lgn * (lgn – lg lgn))= O(lg^2n – lgn * lglgn) = **O(lg^2n)**	k= lg lgn
1 עדיפה כיוון ש O(lg^2n) < O(n^2 * lgn)	O($2^{lgn / 2}$ * lgn) =O($2^{lg(n^{\wedge}2)}$ *lgn)= **O(n^2 * lgn)**	O(lgn * (lgn – lgn/2))= O(lgn * lgn/2) = **O(lg^2n)**	k = lgn / 2
1 עדיפה כיוון ש O(lgn) < O(nlgn)	O($2^{(lgn-C)}$ * lgn) = O((2^{lgn} /2^C)*lgn) = O(C'2^{lgn} * lgn)= **O(nlgn)**	O(lgn * C) = **O(lgn)**	lgn –k = C

ממן 12 שאלה 3

נתון מערך $A[1..n]$ המקיים את התנאי הבא: אם $A[i] > A[j]$, $1 \le i < j \le n$, אזי $.\ j = i+1$

<u>Insertion-Sort (A)</u>

אלגוריתם זה רץ עבור $A[i]$ בכל המערך משמאל ל- i , $A[1...i-1]$. אם $A[i-1] > A[i]$ אז כל מה יש לאלגוריתם לבצע הוא להפוך ביניהם ולהמשיך ל$A[i+1]$ וזה המקרה הגרוע. (**בניגוד** למערך שאינו בהכרח המערך המוגדר להלן, בו במקרה הגרוע היה צורך להזיז i-1 איברים כדי למצוא את מקומו הנכון של $A[i]$)

ולכן במקרה הגרוע של אלגוריתם הכנסה עבור מערך מהסוג המתואר להלן, זמן הריצה יהיה $O(n)$

<u>Merge-Sort(A)</u>

אם נסתכל על האלגוריתם המוצג בספר (עמ 26-27) נשים לב שהאלגוריתם לא מתעניין בערכי המערך עד שהוא מפורק לגורמיו בעומק הרקורסיה. רק שם בעומק הרקורסיה מתבצעת פעולת הMERGE. מכיוון שכך זמן הריצהשל מיון זה אינו תלוי במצב מיון המערך. ולכן ישאר $O(n\lg n)$

<u>Quick-Sort(A)</u>

במקרה הגרוע של מיון מהיר, כאשר הוא ממוין לגמרי בסדר עולה או בסדר יורד ואז הפיבוט תמיד קיצוני ופעולת המיון בכל קריאה נעשית על קלט שקטן באחד , זמן הריצה הוא כידוע
$T(n) = T(n-1) +n = O(n^2)$

במערך המתואר לעיל, יכולים להיות היפוכים רק בין שני איברים סמוכים. כך שגם אם יש מקסימום היפוכים המערך קרוב מאוד למצב של מערך ממוין.

נסתכל על הקצה הימני של המערך: אם אין היפוך, הפיבוט יהיה קיצוני כמו במערך ממוין ואם יש היפוך אז הפיבוט הוא לכל היותר צעד אחד שמאלה. במקרה **הטוב** יהיו מקסימום היפוכים והקלט יקטן ב2 בכל קריאה רקורסיבית. זמן הריצה בדומה למקרה הגרוע של מיון מהיר, יהיה $T(n-2) + n = O(n * n/2) = O(n^2)$

ממן 12 שאלה 4

א. פעולת חיפוש איבר מינימלי מצריכה מעבר על כל קבוצת האיברים.
במקרה הנ"ל נחפש איבר מינימלי בתוך מחצית מערך, אשר תחום במרכזו (בין הרבע הראשון לרבע האחרון). לא מובטח בשום מצב שהחציון של המערך הכללי יימצא בתחום הזה כאיבר מינימלי, ובסיכויים שווים אנו עלולים למצוא כפיבוט איבר קיצוני. זמן ריצת חיפוש מינימום הוא O(n)
פעולת סידור האיברים סביב הPIVOT גם היא O(n)
מכאן ששגרת החלוקה (partition) תשאר בסיבוכיות O(n)
וזמן הריצה של מיון מהיר נשאר O(n^2) במקרה הגרוע וO(nlgn) במקרה הטוב
מה שכן משתנה הם המקרה הטוב והמקרה הגרוע. כעת מיון מהיר כאשר המערך ממוין (עולה או יורד) עשוי להיות בעל זמן ריצה פחות גרוע מכשהיה כאשר נבחר פיבוט הממוקם בקצה.

ב. חציון של המקטע המדובר, הנו איבר שמחצית מהאיברים **במקטע** גדולים ממנו ומחצית קטנים ממנו.
אבל האיברים שבשני רבעי המערך **שבקצוות** , **מחוץ לתחום המקטע**, לא מובטח שהם גדולים או קטנים ממשהו ולכן במקרי הקצה תהיה:

1. חלוקה מאוזנת. החציון שנבחר , הוא **החציון** של הקלט כולו – מצב הידוע כאידיאלי
2. חלוקה מאוד לא מאוזנת ז"א כל האיברים הללו בצד אחד (**כולם** גדולים או **כולם** קטנים מהאיבר הנבחר) כך שהציר שבחרנו יחלק את הקלט הכולל ל- ¼ ו- ¾.

מציאת החציון לוקחת O(n) ושגרת ה Partition בסך הכל לוקחת O(n)
נסתכל על מצב 2, בו החלוקה לא מאוזנת ונפתח את נוסחת זמן הריצה על המצב הגרוע בו החלוקה היא שוב ושוב ¼ ו- ¾:

$$T(n) = T(n/4) + T(3n/4) + O(n)$$

אם בכל שלב חותכים אחוז מסוים מהקלט אז עומק עץ הרקורסיה יהיה logn על בסיס כלשהו.
נניח שO(nlgn) חסם עליון לפתרון נוסחת הנסיגה :

$T(n) = O(n\lg n)$

נראה ש: $T(n) \le d*n\lg n$ (קבוע $d > 0$)

$T(n/4) \le d(n/4* \lg (n/4))$

$T(3n/4) \le d(3n/4 *\lg (3n/4))$

$T(n) \le d(n/4* \lg (n/4)) + d(3n/4* \lg (3n/4)) + cn$ (קבוע $c>0$)

$= dn/4* \lg n – dn/4* \lg 4 + 3dn/4 *\lg 3 + 3dn/4* \lg n – 3dn/4 *\lg 4 + cn$

$= \lg n (dn/4 + 3dn/4) + n (-d/4 *\lg 4 + 3d/4 *\lg 3 - 3d/4* \lg 4 + c)$

$= n\lg n* d + n (-d*\lg 4 + 3d/4 *\lg 3 + c)$

$\le \underline{d*n\lg n}$ מש"ל

עבור d ו- c שמקיימים:

$-d\lg 4 +3d/4\lg 3 + c \le 0 \Rightarrow -d(\lg 4 – ¾ \lg 3) \le - c \Rightarrow \boxed{d \le c/(\lg 4 – ¾ \lg 3)}$

ממן 12 שאלה 5

נתון מערך לא ממוין ורוצים למצוא קבוצת ערכי מיקום בטווח נתון ולהחזירה כאשר היא ממוינת.

הרעיון של האלגוריתם הוא מציאת ערכי המיקום הקיצוניים של הקבוצה המבוקשת, מיון הקבוצה והחזרת הקבוצה הממוינת.

ערך המיקום התחתון הוא החציון (שאינו מבוקש אבל כן קבוצת הערכים שמעליו).

בהפעלת סלקט , מוחזר מערך אשר ערכיו מסודרים כך שכל הערכים הגדולים מהחציון, נמצאים מעליו.

בשלב השני, הפעלת סלקט על הקצה השני המבוקש: n/2 + n/lgn . שוב מוחזר מערך אשר ערכיו מסודרים כך שכל האיברים הקטנים מהערך המבוקש נמצאים מתחתיו.

בשלב זה כל הערכים המבוקשים נמצאים בטווח אינדקסים ידוע אך יש למיינם, ואת הפעולה הזו מבצע מיון מהיר.

סיבוכיות של כל פעולת סלקט היא O(n) וסבוכיות של מיון מהיר O(nlgn) אבל הקלט שמקבל המיון המהיר אינו n אלא חלק קטן ממנו והראיתי באמצעות הצבה (במלבן הגדול) שבסופו של דבר הסיבוכיות של המיון המהיר היא O(n) וזו גם הסיבוכיות הכללית של האלגוריתם.

(נניח שהחלוקות של n הן שלימות מכיון שזהו פסיאודו-קוד, כמובן בכתיבת קוד אמיתי יש להתיחס לחלוקות הלא שלימות)

```
FIND-VALUES(A)
n ← A.length
select(A,1,n, n/2)
select(A, n/2+1, n, n/lgn)
QUICK-SORT(A[ n/2+1 … n/2+n/lgn])
return A[n/2 +1 … n/2 + n/lgn]
```

O(n)

O(n)

O(NlgN), N=n/lgn → n/lgn * lg (n/lgn) →
(n/lgn) * (lgn – lglgn) → nlgn/lgn – nlglgn/lgn
→ n – nlglgn/lgn = O(n)