מבני נתונים
תרגיל 2 - פתרונות

מיון מהיר

1. הריצו את השיטה partition על המערך הבא. הראו את שלבי הריצה השונים.

$$6, 10, 20, 4, 2, 15, 5, 99, 12, 1$$

אחרי שלב זה המשיכו והריצו את מיון מהיר על המערך. תארו את כל שלבי הרקורסיה, אך עתה אין צורך להיכנס לתיאור הריצה של partition.

פתרון: בתאור מסומנים p ו־q בכל שלב אחרי החלפה ראשונה:

$$6, 1^p, 20, 4, 2, 15, 5, 99, 12, 10^q$$

אחרי שנייה:

$$6, 1, 5^p, 4, 2, 15, 20^q, 99, 12, 10$$

ואז q זז:

$$6, 1, 5^p, 4, 2^q, 15, 20, 99, 12, 10$$

ואז p:

$$6, 1, 5, 4, 2^{q,p}, 15, 20, 99, 12, 10$$

לבסוף מתבצעת ההחלפה עם ה pivot:

$$2, 1, 5, 4, \mathbf{6}, 15, 20, 99, 12, 10$$

ומוחזר המיקום של 6.

עתה ניכנס לתיאור הריצה של מיון מהיר:

(א) הוא רץ על צד שמאל: $2, 1, 5, 4$. partition יתן: $1, \mathbf{2}, 5, 4$ ואז
   i. צד ימין יחזור מיד כי גודל הקטע הוא 1.

ii. צד שמאל יריץ partition ונקבל: $4, \mathbf{5}$. ואז שתי הרקורסיות יחזרו מיד.

(ב) עכשיו על צד ימין: $15, 20, 99, 12, 10$. partition יתן: $12, 10, \mathbf{15}, 99, 20$. ואז הרקורסיות:

i. צד שמאל, partition יתן $10, \mathbf{12}$ ושתי הרקורסיות שלו יחזרו מייד.

ii. בצד ימין, partition יתן $20, \mathbf{99}$ ושתי הרקורסיות של יחזרו גם מייד.

סך הכל נקבל את המערך ממוין:

$$1, \mathbf{2}^2, 4, \mathbf{5}^3, \mathbf{6}^1, 10, \mathbf{12}^3, \mathbf{15}^2, 20, \mathbf{99}^3$$

כאשר סימנתי את כל מי שהיה pivot ומעליו את עומק הרקורסיה בה הוא תיפקד ככזה.

2. נבחן את הקוד של הפונקציה partition:

```
1  int partition(int a[], int l, int r)
2  {
3    int pivot = a[l];
4    int p = l;
5    int q = r;
6
7    while(p < q)
8    {
9      while(a[q] > pivot && p < q) q--;
10     while(a[p] <= pivot && p < q) p++;
11
12     swap(a, p, q);
13   }
14   swap(a, l, p);
15   return p;
16 }
```

בכל סעיף מוצע שינוי לקוד של הפונקציה. בדקו לכל שינוי האם הוא משפיע על נכונות הפונקציה. אם הפונקציה נשארת נכונה נסו להסביר מדוע, ואם לא תנו דוגמאת הרצה קטנה בה הפונקציה החדשה אינה נכונה.

(א) הורדת התנאי $p < q$ בשורה 9.

פתרון: הפונקציה תמשיך לעבוד נכון. נשים לב שתמיד בזמן התנועה של q, p נמצא על מישהו שהוא קטן שווה מהפיבוט - בהתחלה זה כך כי הוא על הפיבוט עצמו, ואחרי זה, תמיד ה swap יגרום לזה שוב.

לכן, אף פעם q לא יכול לעבור אותו ולכן אין צורך בתנאי.

(ב) הורדת התנאי $p < q$ בשורה 10.

פתרון: פה זה לא יעבוד, וזה למשל בדוגמא בה כל המספרים במערך קטנים מהפיבוט.

(ג) שינוי שורה 4:

```
int p = l+1
```

פתרון: זה נראה הגיוני, הרי אין טעם לבדוק את הפיבוט עצמו. אבל הבעיה יכולה לקרות בחילוף האחרון של שורה 14. זה יקרה אם כל המספרים במערך גדולים מהפיבוט. אז נצא מהלולאה כש $p = q = l + 1$ ואז החילוף האחרון יהיה לא טוב.

(ד) הוספת שתי שורות אחרי שורה 12:

```
p++
q--
```

פתרון: זה נשמע רעיון טוב כי ברור שאחרי החילוף אפשר כבר לזוז קדימה בשני המונים שלנו. נוצרת רק בעיה במקרי קצה. למשל במערך הבא:

$$20, 30, 10$$

30 ו־10 מתחלפים ואז p נהיה 2, והחילוף האחרון יכניס את הפיבוט במקום הלא נכון.

אפשר לפתור את זה ואולי אפילו זה שווה מבחינת זמן ריצה.

ערמות

1. נתון המערך הבא:

$$10, 5, 20, 4, 3, 7, 19, 2, 1, 40, 30, 14, 12$$

(א) הריצו את buildHeap על המערך.

(ב) הוציאו את שורש הערמה ותקנו אותה - קחו את העלה האחרון ושימו אותו בשורש ואז תקנו.

(ג) חזרו על הסעיף האחרון שוב פעם.

פתרון:

(א) לאחר buildHeap של השכבה התחתונה $(4, 3, 7, 19)$:

$$10, 5, 20, 4, \mathbf{40}, \mathbf{14}, 19, 2, 1, \mathbf{3}, 30, \mathbf{7}, 12$$

לאחר שכבה שנייה $(5, 20)$:

$$10, \mathbf{40}, 20, 4, \mathbf{30}, 14, 19, 2, 1, 3, \mathbf{5}, 7, 12$$

ואז השורש:

$$\mathbf{40}, \mathbf{30}, 20, 4, \mathbf{10}, 14, 19, 2, 1, 3, 5, 7, 12$$

(ב) קודם נשים את העלה בשורש:

$$\mathbf{12}, 30, 20, 4, 10, 14, 19, 2, 1, 3, 5, 7$$

ואז נפעפע מטה:

$$\mathbf{30}, \mathbf{12}, 20, 4, 10, 14, 19, 2, 1, 3, 5, 7$$

(ג) קודם נשים את העלה בשורש:

$$\mathbf{7}, 12, 20, 4, 10, 14, 19, 2, 1, 3, 5$$

ואז נפעפע מטה:

$$\mathbf{20}, 12, \mathbf{19}, 4, 10, 14, \mathbf{7}, 2, 1, 3, 5$$

2. (א) תארו מערך בן 7 תאים עליו buildHeap תרוץ את הזמן המקסימלי האפשרי.

פתרון: המערך יהיה:

$$1, 2, 3, 4, 5, 6, 7$$

כל אבר חדש שאנו מגיעים אליו קטן מכל קודמיו ולכן יאלץ לפעפע כל הדרך למטה. כך: 3 יוחלף עם 7, 2 יוחלף עם 5, ואז 1 יוחלף עם 7 ואז עם 6

(ב) הכלילו את הדוגמא שמצאתם בסעיף הקודם למערך בן n תאים, ותנו מקרה בו זמן הריצה של buildHeap הוא מקסימלי.

פתרון: אם ניקח מערך שממוין מהקטן לגדול, אז בכל שלב אנו נתקלים במספר שקטן מכל אלה מתחתיו ולכן יאלץ לפעפע כל הדרך למטה. זמן הריצה לכן יהיה מקסימלי.

(ג) תנו דוגמא למערך בן n תאים, עליו תרוץ buildHeap את הזמן המינימלי.

פתרון: אם ניקח מערך שהוא כבר ממוין בסדר יורד, אז בכל שלב, ל heapify אין עבודה לעשות, ולכן הריצה תסתיים בסריקת המערך בלבד, ללא החלפות.

3. כתבו גרסא חדשה ל buildHeap העובדת הפוך, היא מתחילה מלמעלה ויורדת כלפי מטה. כלומר היא מתחילה מערמה ריקה וכל פעם מוסיפה עוד אבר, שמה אותו בתור העלה האחרון, ואז מפעפעת אותו למעלה. נתחו את זמן הריצה של השיטה שכתבתם - התמקדו במקרה שהעץ הוא בינארי מלא. שימו לב בעיקר לתרומה של השכבה התחתונה ביותר בעץ.

פתרון: נשתמש בשיטה שראינו בכיתה:

```
void addElement(int[] a, int n, int new)
{
  int p = n+1;
  while(p != 1 && a[p/2] < new) {
    a[p] = a[p/2];
    p = p/2;
  }
  a[p] = new;
}
void buildHeap2(int[] a)
{
  for (int n = 1; n < a.length; n++)
    addElement(a, n, a[n+1]);
}
```

זכרו שהמערכים שלנו הולכים מאינדקס 1 ועד אינדקס a.length, כולל האחרון.

מבחינת זמן הריצה, הראנו כבר שכל קריאה ל addElement לוקחת $O(\log n)$, ולכן סך הכל זמן הריצה הוא $O(n \log n)$.

הפעם לא ניתן לשפר את חישוב זמן הריצה, וזה כי אם נסתכל על השכבה האחרונה, יש שם חצי מהקודקודים וכל אחד יכול בהחלט לפעפע כל הדרך למעלה - חישבו על מערך ממוין בסדר עולה. לכן רק שכבה זאת תתרום $\Omega(\frac{1}{2}n \log n) = \Omega(n \log n)$ ומכאן שזמן הריצה הוא באמת $\Theta(n \log n)$.

4. תכננו ערמה בה לכל קודקוד יש שלושה ילדים. תארו היכן נמצאים ילדיו של כל קודקוד, תארו ערמה לדוגמא עם 15 קודקודים, וכיתבו את השיטה heapify. מה יהיה עומק העץ במקרה של ערמה עם n קודקודים?

פתרון: ערמה לדוגמא:

$$90, 10, 40, 20, 7, 4, 8, 30, 20, 6, 1, 10, 13, 6, 2$$

ילדיו של קודקוד יהיו במקומות $3i-1, 3i, 3i+1$

```
void heapify(int[] a, int i, int n)
{
  while(true) {
    int max = i; // max is actually an index.
    if (3*i - 1 <= n && a[3*i - 1] > a[max])
      max = 3*i - 1;
    if (3*i <= n && a[3*i] > a[max])
      max = 3*i;
    if (3*i + 1 <= n && a[3*i + 1) > a[max])
      max = 3*i + 1;
    if (max == i)
      return
    swap(a, i, max);
    i = max;
  }
}
```

עומק העץ יהיה $\log_3(n)$.

5. נתון מערך לא ממוין בעל n מספרים. עליכם למצוא את k המספרים הקטנים ביותר במערך. אלגוריתם פשוט יעבור k פעמים על המערך וימצא כל פעם מינימום וימחוק אותו. אלגוריתם זה יקח $O(n \cdot k)$ זמן.

אלגוריתם אחר הוא למיין את המערך ואז לקחת את k המספרים התחתונים. אלגוריתם זה יקח $O(n \log n)$ זמן.

השתמשו בערמה ומצאו אלגוריתם יותר יעיל משניהם, הרץ בזמן $O(n + k \log n)$.

פתרון:

נהפוך את כל המערך שלנו לערמה - זה לוקח $O(n)$ זמן. אבל הערמה שנבנה היא הפוכה למה שראינו בשיעור, כלומר המינימום הוא בשורש. נוציא את השורש ונכתוב בצד - ואז נתקן את הערמה כרגיל, ניקח את העלה האחרון, נשים בשורש ונריץ Heapify. זה יקח $O(\log(n))$. נמשיך ככה k פעמים ונקבל את k האברים הכי קטנים.

```
buildHeap(a, n);
for(i=0; i<k; i++) {
  print a[1];
  a[1] = a[n];
  n--;
  heapify(a, 1, n);
}
```

זמן הריצה הוא $O(n + k\log(n))$.