<u>עבודה 5 – מבני נתונים</u>

<u>מגישים: ניצן גואטה ואחיעד שייקר</u>

1. נתאר כיצד ניתן לשנות את אלגוריתם QuickSort כך שירוץ בזמן O(nlogn) במקרה הגרוע ביותר.
האלגוריתם בו נשתמש יעבוד לפי השלבים הבאים:

א. נשתמש במיון מהיר כמעט רגיל, כלומר, השינוי הוא שבמקום לקחת פיבוט רנדומלי או מאחד הקצוות, נפעיל את האלגוריתם למציאת חציון בזמן דטרמיניסטי (O(n)). נבחר בו להיות הפיבוט ונמקם אותו במיקומו (במרכז המערך).

ב. שאר האלגוריתם המקורי של מיון מהיר יישאר זהה למקור.

נחזור על אלגוריתם זה עד אשר המערך יהיה ממוין, כך שזמן ריצתנו הוא O(nlogn) וזהו שיפור לאלגוריתם מיון מהיר שבזמן הגרוע ביותר הוא $O(n^2)$.

2. אנחנו יודעים שחיפוש בינארי במערך ממוין הוא O(logn). אנו מקבלים כקלט איבר מסוים ומערך ממוין "חלקית", כלומר במערך יש 2 חלקים ממוינים. כלומר יש איבר במערך שלא מקיים A[i]<A[i+1]. אנו נרצה למצוא את i כדי לדעת את הטווחים של כל אחד מ-2 החלקים הממוינים במערך שלנו, וכך נדע היכן לחפש את האיבר שקיבלנו כקלט (בעזרת חיפוש בינארי).

FindKeyIndex(A[], key)

```
Flag<-false
l<-0 , h<-A.length-1 , mid<-⌊A[i] < A[i + 1] (h+l)/2⌋
while(NOT flag) //when break out – h is the index of min value
if(l+1 = h)
        flag<-true
else  if(A[l]<A[mid]  AND  A[mid+1]<A[h])
        flag<-true
else
        if(A[mid+1]<A[h])
            h<-mid
            mid<-⌊(h+l)/2⌋
         else
             l<-mid+1
             mid<-⌊(h+l)/2⌋

if(h=A.length-1 AND l=0) //sorted array
    return binarySearch(A, 0/*low*/, A.length-1/*high*/,key)
else if (key≥A[h] AND key≤A[A.length-1])
    return binarySearch(A, h/*low*/, A.length-1/*high*/,key)
else if (key≥ A[0] AND key≤A[l])
    return binarySearch(A, 0 /*low*/, l /*high*/,key)
return -1
```

ניתוח זמן ריצה:

חיפוש בינארי במערך ממוין O(logn).

<u>חיפוש האינדקס שמפר את התנאי</u> A[i]<A[i+1] O(logn), נסביר:

נוסחת הנסיגה המתאימה <u>למקרה זה</u> $T(n) = T(\frac{n}{2}) + 6$, בכל פעם אנו מפצלים את המערך לשני חצאים וממשיכים בחיפוש על האינדקס שמפר את התנאי:

$A[i] < A[i+1]$ רק באחד מבין שני החצאים (בדומה לחיפוש בינארי).

בנוסף, מכיוון ש6 הפעולות (מציאת טווח שבו האיבר באינדקס המינימלי גדול מהאיבר באינדקס המקסימלי של הטווח) הן סדר גודל של O(1) ניתן לרשום $T(n) = T(\frac{n}{2}) + 1$.

אנו מכירים נוסחת נסיגה זו מהכיתה וזמן ריצתה הוא O(logn).

בסה״כ זמן ריצה: O(logn)+O(logn)=2O(logn)=O(logn).

3. א.

2	3	4	5
8	9	12	∞
14	∞	∞	∞
16	∞	∞	∞

ב. נניח כי Y[1,1]=∞. נראה כי Y טבלת יאנג ריקה בגודל m*n.

מתכונות טבלת יאנג, האיברים בכל שורה ממויינים בסדר עולה משמאל לימין והאיברים בכל עמודה ממויינים בסדר עולה מלמעלה למטה. אנו יודעים שאיבר לא קיים מוגדר להיות ∞,

בנוסף, כל איבר הקיים בטבלה יהיה מספר קטן מ ∞. מכיוון ש Y[1,1]=∞ והוא האיבר הראשון בשורה הראשונה, כל האיברים הבאים אחריו יהיו גם כן ∞. לכן בשורה הראשונה ישנם n איברים שלא קיימים

נתבונן על העמודות 1 עד n, האיבר העליון בכל עמודה הוא ∞, לכן, כל האיברים שמתחתיו צריכים להיות גדולים או שווים לו, כלומר כל האיברים מתחתיו יהיו ∞.

לכן, בכל עמודה ישנם m איברים שלא קיימים.

אם נסתכל על המטריצה, ישנן n עמודות כשבכל עמודה ישנם m איברים שלא קיימים כלומר – n*m איברים שלא קיימים (∞) בסה״כ – ולכן הטבלה ריקה.

נניח כי Y[m,n]< ∞ ונראה כי Y מלאה.

Y[m,n]< ∞ , כלומר Y[m,n] מכיל מספר כלשהו. מתכונות טבלת יאנג אנו יודעים שבשורה הm, Y[m,n] הוא האיבר האחרון בשורה– ולכן גדול או שווה לכל האיברים הקודמים לו באותה שורה, בפרט כל האיברים הקודמים לו יהיו מספרים קטנים מאינסוף גם כן (וקטנים ממנו בהכרח). כלומר שורה m מכילה n איברים.

כעת נתבונן בעמודה ה-n, Y[m,n] מכיל מספר כלשהו שהוא האיבר התחתון בעמודה – ולכן גדול מכל האיברים הקודמים לו באותה העמודה, בפרט כל האיברים שמעליו יהיו מספרים קטנים ממנו ומאינסוף גם כן. לכן עמודה n מכילה m איברים.

בנוסף נשים לב שבכל עמודה בין העמודה הראשונה לעמודה ה-n-1, האיבר בשורה אחרונה (m) – כלומר האיבר התחתון ביותר בכל עמודה יהיה מספר קטן מאינסוף לכן, כל העמודות האלו בהכרח יכילו מספרים קטנים מהמספר התחתון ביותר. לכן, בכל עמודה בין 1 ל n-1 יהיו m מספרים.

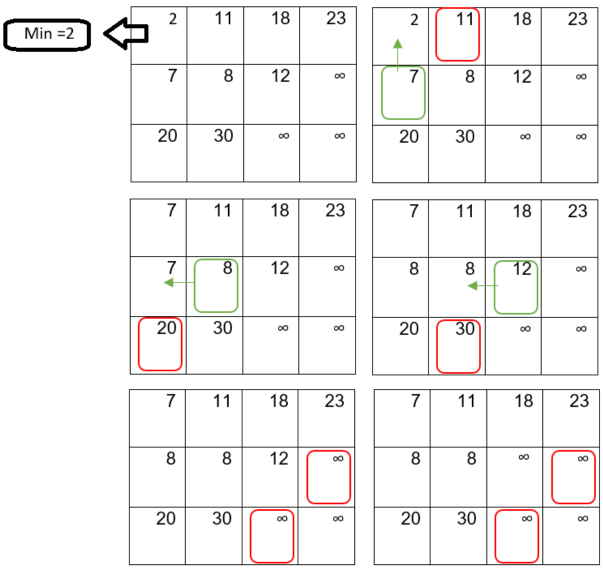
כלומר, ישנן n עמודות המכילות m איברים כנול אחת – m*n איברים בסה״כ, כנדרש.

ג. תחילה נשים לב שמהגדרת טבלת יאנג האיבר המינימלי בהכרח ימצא באינדקס (1,1) ואותו נרצה להחזיר.

נתאר אלגוריתם ל extract-min:

1. נגדיר משתנה חדש min ונשמור בו את Y[1,1].
2. כעת, בצורה איטרטיבית, בכל שלב נבדוק מיהו האיבר המינימלי מבין האיברים הסמוכים לy[i,j] : Y[i+1,j] או Y[i,j+1]. נעדכן את הערך במקום ה-Y[i,j] להיות האיבר המינימלי מבין השניים שלעיל (או אחד מהם אם שניהם שווים), ונעבור למיקום האיבר המינימלי (i,j=i+1,j או i,j=I,j+1). ונמשיך לאיטרציה הבאה.
3. נעצור כאשר נגיע לאחד מהמצבים הבאים :
   - Y[i+1,j] וגם Y[I,j+1] הם ∞.
   - I=m וגם j=n (כלומר אין שכנים).
   - Y[i+1,n]= ∞ או Y[m,j+1] = ∞.

כשנגיע לתנאי עצירה, נצא מהלולאה נשנה את Y[I,j] להיות ∞, ונחזיר את ans.



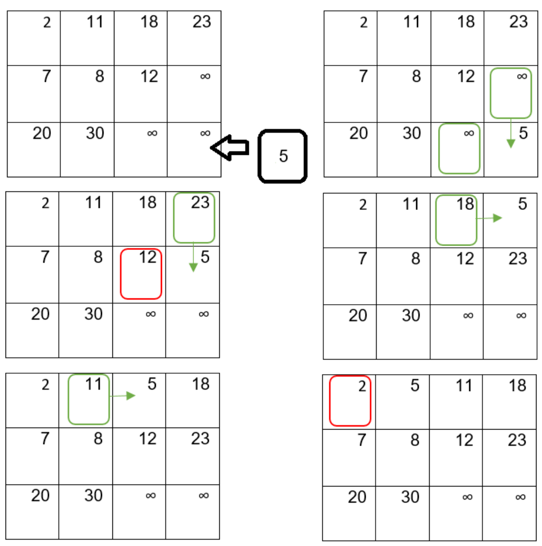
ניתוח זמן ריצה:

בכל איטרציה אנו מתקדמים לעמודה הימנית או לשורה התחתונה. במקרה הגרוע ביותר (טבלת יאנג מלאה) נגיע ל-$y[n,m]$, (כלומר בכל איטרציה בה התקדמנו ימינה בטבלה, באיטרציה הבאה ירדנו למטה בטבלה (או ההפך)) לכן החלפנו איברים והתקדמנו לכל היותר n צעדים ימינה ו-m צעדים למטה בטבלה, כלומר בסה״כ $O(n+m)$.

ד. תחילה נשים לב שמהגדרת טבלת יאנג יש מקום פנוי בטבלה רק אם $Y[m,n]=\infty$.
האלגוריתם יעבוד לפי אותו רעיון כמו בסעיף ג׳.
תיאור האלגוריתם:
נגדיר $i=m,\ j=n$ נבדוק האם $Y[m,n]=\infty$
אם כן, בצורה איטרטיבית, בכל שלב נבדוק האם האיבר אותו נרצה להכניס קטן מ $Y[i-1,j]$ וגם $Y[i,j-1]$. בנוסף, נבדוק מיהו האיבר המקסימלי מבין האיברים הסמוכים ל-$y[i,j]$: $Y[i-1,j]$ או $Y[i,j-1]$. נעדכן את הערך במקום ה-$Y[i,j]$ להיות האיבר המקסימלי מבין השניים שלעיל (או אחד מהם אם שניהם שווים), ונעבור למיקום האיבר המקסימלי ($i,j=l,j-1$ או $i.j=i-1,j$). ונמשיך לאיטרציה הבאה.
נעצור כאשר נגיע לאחד מהמצבים הבאים:

- $Y[i-1,j]$ וגם $Y[l,j-1]$ קטנים מהערך אותו נרצה להכניס.
- $i=1$ וגם $j=1$ (כלומר אין שכנים).
- $Y[i-1,1]$ או $Y[1,j-1]$ קטנים מהערך אותו נרצה להכניס.

כשנגיע לתנאי עצירה, נכניס את האיבר בתא ה$Y[i,j]$ ונסיים.



ניתוח זמן ריצה:

בדומה לסעיף הקודם, במקרה הגרוע ביותר לy[1,1], כלומר עשינו n צעדים שמאלה וm צעדים למעלה, כלומר O(m+n).

ה. נראה כיצד ניתן להשתמש בטבלת יאנג בגודל n על n בלבד על מנת למיין n^2 מספרים בזמן $O(n^3)$. תחילה, נכניס ע"י insert את כל אחד מ- n^2 האיברים לטבלה. נשים לב שמתכונות טבלת יאנג סדר הכנסת האיברים לא משנה וזמן ריצת ההכנסה הוא O(n+n)=O(n). נשתמש בextract-min כדי להוציא בכל פעם את האיבר המינימלי מהטבלה ונכניס (נדרוס) למקום הבא במערך (בסדר כרונולוגי – נתחיל מתא 0).

בסוף התהליך יהיה לנו מערך בגודל n^2 ממוין (מכיוון שבכל פעם הוצאנו את האיבר המינימלי).

זמן ריצת extract-min הוא O(n+n)=O(n) כלומר בסה"כ n^2 *O(n) איברים ולכן $O(n^3)$.

ו. נתאר אלגוריתם שבודק אם מספר נתון קיים בטבלת יאנג נתונה בגודל m על n בזמן O(m+n).

תחילה, נבדוק האם key (הקלט) קטן מy[1,1] – אם כן, המספר אינו קיים בטבלה.

כעת בדומה לסעיפים קודמים נתבונן בכל פעם ב Y[i+1,j] ו- Y[I,j+1] ונבדוק אם:

Key>(Y[i+1,j] וגם- Y[I,j+1]) כלומר המפתח גדול מ-2 הצמודים לy[I,j] (שגדולים ממנו) – אז נבדוק מי הוא המקסימלי מבין Y[i+1,j] ו- Y[I,j+1] ונתקדם בכיוון האיבר המקסימלי.

אחרת, אם key < Y[I,j+1] וגם key> Y[i+1,j] נעבור לתא Y[I,j+1]. (המקרה השני סימטרי).

אחרת, אם key < (Y[i+1,j] וגם- Y[i,j+1]), כלומר המפתח קטן מ-2 הצמודים לY[i,j] – אז המפתח לא נמצא.

במידה ויש רק צמוד יחיד אז נבדוק אם קטן מצמוד זה.

זמן ריצה: במקרה הגרוע ביותר, נתקדם עד Y[m,n] , כלומר נתקדם לאורך m שורות ולאורך n עמודות, לכן O(m+n).

4. א. נראה שעבור $x_1 + x_2 + \cdots + x_n$ ומשקלים $w_i = \frac{1}{n}$ לכל $i \in \{1,2,\ldots,n\}$ ה-median שווה ל-weighted median.

תחילה נשים לב שהאיבר האמצעי הוא $x_{\lceil\frac{n}{2}\rceil}$, כעת נתבונן על חלוקת האיברים ל-2 קבוצות כך ש $x_{\lceil\frac{n}{2}\rceil}$ הוא הmedian : מימין לmedian (לא כולל) יהיו לכל היותר $\frac{n}{2}$ איברים. משמאל ל median יהיו לכל היותר $\left\lfloor\frac{n-1}{2}\right\rfloor$ – נסביר – האיבר ה$\left\lceil\frac{n}{2}\right\rceil$ הוא ה-median ולא נכלל בשום קבוצה.

כעת נחשב את ה weighted median : יש לנו n איברים כך שכל המשקלים זהים לכן סדר הכנסתם לא משנה. נכניס אותם לפי הסדר, x1,...,xn. במקרה זה גם נקבע את xk להיות ה-weighted median (בדיוק לפי אותו סדר חלוקה) ונראה שאכן מתקיימים התנאים :

$$\sum_{xi<xk} wi = \sum_{xi<xk} \frac{1}{n} = \frac{n-1}{2} = \frac{1}{2} - \frac{1}{2n} < \frac{1}{2}$$

$$\sum_{xi>xk} wi = \sum_{xi>xk} \frac{1}{n} = \frac{n}{2} * \frac{1}{2} \leq \frac{1}{2}$$

התנאי האחרון יהיה שווה ל$\frac{1}{2}$ רק כאשר n זוגי.

ב. נראה אלגוריתם המחשב את ה weighted median עבור n איברים בזמן O(nlogn).
הפעולות שנעשה הן :

1) נמיין את האיברים לפי משקלם (למשל ע"י מיון מיזוג). O(nlogn)
2) נסכום את המשקלים של האיברים עד אשר נמצא כי הסכום גדול מחצי ונעצור, בשיטה זו נמצא את הweighted median . (O(n)).

האלגוריתם יפעל כך :

1) נגדיר משתנים i=1 (אינדקס האיברים), sum=0 (סכום האיברים).
2) כל עוד $\text{sum} + w_i < \frac{1}{2}$:

   $\text{sum} = \text{sum} + w_i$ (הסכום גודל ב-w_i),

   i=i+1 (האינדקס גדל ב-1).
3) נחזיר את w_i .

אלגוריתם זה ייעצר כאשר נגיע ל weighted median, כלומר כאשר משמאלו קיימים איברים שסכום משקליהם קטן מחצי.

זמן הריצה של האלגוריתם מורכב מזמן ריצת המיון + זמן שלקח עד למציאת ה-weighted median. ידוע כי ניתן למיין את המערך ע״י מיון שזמן ריצתו הוא O(nlogn), כמו כן, זמן הריצה עד למציאת weighted-median הוא O(n) , לכן זמן הריצה הכולל הוא O(nlogn).

ג. נראה אלגוריתם המחשב את ה-weighted median בזמן θ(n), ע״י שימוש באלגוריתם למציאת הmedian הרץ בזמן לינארי.

האלגוריתם יקבל כקלט מערך (נקרא לו Arr) שמכיל את הmedian, ומשתנה (מספר), (נקרא לו totalWeight) שהוא סך המשקל איברי של הקלט ההתחלתי, שהוא קטן מכל איברי Arr.

שלבי האלגוריתם יהיו כך :

1) נאתחל משתנים :

n – אורך המערך,

m- האיבר ה-median של Arr,

G – מערך שמקיים {G ={ Arr[i]≥m. מאותחל להיות מערך ריק.

H - מערך שמקיים {H={ Arr[i]<m. מאותחל להיות מערך ריק.

weightG – סכום המשקלים של G.

2) אם n=1, נחזיר את המערך [Arr[1 מכיוון שהוא היחידי במערך ולכן מקיים את תכונות weighted median.

3) נעבור באיטרציה מ1=i עד i=n,

אם Arr[i]<m, נבצע $weightG = weightG+w_i$. (נגדיל את הערך של סכום משקלי G) ונוסיף את [Arr[i למערך G.

אחרת, נוסיף את [Arr[i למערך H.

4) אם $totalWeight + weightG > \frac{1}{2}$, נקרא לפונקציה שוב עם מערך G ו-totalWeights. // האיבר weighted median נמצא בקבוצה G.

5) אחרת, נקרא לפונקציה שוב עם מערך H ו- weightG.

הקריאה ההתחלתית לפונקציה תתבצע עם הערכים : מערך (נקרא לו Arr) שמכיל את הmedian, ובנוסף עם קלט totalWeights=0.

האלגוריתם יסתיים תמיד מכיוון שגודל A נעשה קטן יותר בכל קריאה רקורסיבית. כשהאלגוריתם נגמר הפלט לעולם יהיה weighted median מכיוון ש weighted median תמיד יימצא ב-Arr, לכן כאשר רק איבר יחיד יישאר הוא יהיה ה-weighted median.

זמן ריצה של האלגוריתם הוא $O(n)$.

חישוב median ופיצול מערך Arr ל2 מערכים (G,H) לוקח $\theta(n)$. בכל קריאה רקורסיבית גודל המערך קטן מ-n ל- $\left\lceil \frac{n}{2} \right\rceil$, לכן

$T(n) = T(\frac{n}{2}) + \theta(n) = \theta(n)$.

5. א. A-B-D-E-C-F-H-G-I

ב. A-B-F-G-C-I-H-D-E

6. א. נתאר אלגוריתם שבודק האם הקשת e נמצאת על כל המסלולים הקצרים ביותר בין s ל-t ב-G.

תחילה נריץ BFS על הגרף. נשמור את t.d (המרחק הקצר ביותר בין s ל-t) במשתנה זמני. נסיר את e מהגרף G ונריץ שוב BFS. כעת נתבונן שוב ב-t.d, ונבדוק: אם $temp \geq t.d$ נחזיר false – כלומר היה קיים מסלול באותו הגודל ש-e לא עברה דרכו, ולכן בהסרת e , t.d לא השתנה.

אחרת, $t.d > temp$, אז נחזיר true (כל המסלולים הקצרים ביותר עברו דרך e).

זמן ריצה: $2\Theta(V + E)+O(1) = \Theta(V + E)$, הרצנו פעמיים BFS והגדרנו משתנה temp עליו עשינו 2 בדיקות לכל היותר.

ב. נתאר אלגוריתם הבודק האם הקשת e נמצאת על מסלול קצר ביותר כלשהו בין s ל-t ב-G.

תחילה נריץ BFS על הגרף מקודקוד s. נשמור במשתנה זמני (נקרא לו oldDistT) את t.d, ונשמור גם במשתנה זמני נוסף(נקרא לו oldDistU1) את u1.d. כעת נריץ שוב BFS, הפעם מהקודקוד u2. נשמור במשתנה זמני נוסף(נקרא לו newDistT) את t.d. כעת נבדוק האם newDistT+oldDistU1+1=oldDistT, נסביר, מכיוון ש"התעלמנו" מהצלע e בהרצה השנייה של BFS נוסיף +1 לחישוב. אם המשוואה נכונה אז בהכרח יש מסלול קצר ביותר שעובר דרך e (בגודל oldDistT) ולכן נחזיר true, אחרת נחזיר false.

זמן ריצה: $2\Theta(V + E)+O(1) = \Theta(V + E)$, הרצנו פעמיים BFS והגדרנו משתנים: oldDistT, newDistT,oldDistU1 עליהם ביצענו בדיקה.