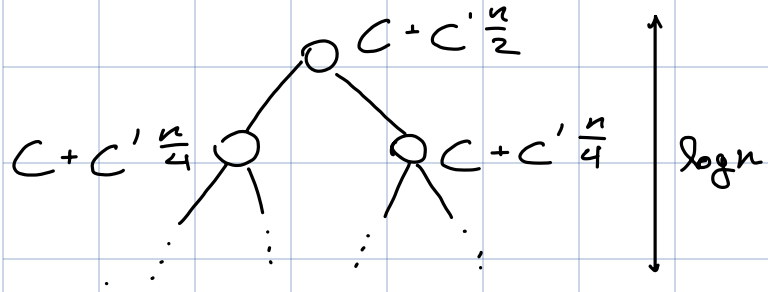
1) א. עבור מערך של אחדות, נתאר איך partition עובד:

המצביעים תמיד יעברו בכל איבר שיגיעו אליו (כי לכל $1 \le i \le n$, $A[i] = pivot$)

ולכן יתבצעו $\frac{n}{2}$ החלפות במערך. $A[i] \leftrightarrow A[j]$. q שיתקבל בסוף קריאות יהיה

בדיוק אמצע תת המערך שלו עשינו את הפרוצדורה, לכן, בכל רמה יתבצעו

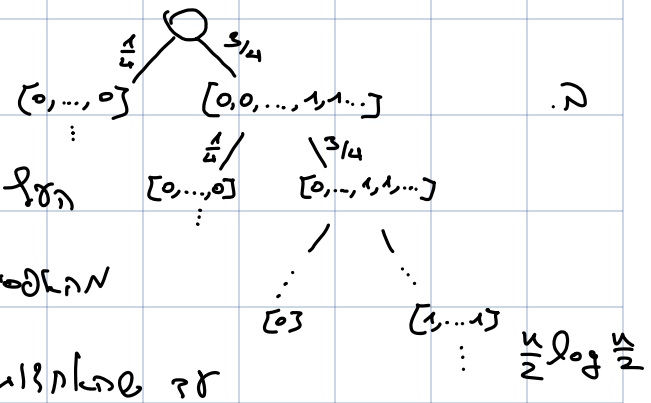
$c + c'\frac{n}{2}$ פעולות ויהיו $\log n$ רמות.



לכן, $g(n) = \log n\left(c + c'\frac{n}{2}\right)$

$\Rightarrow \Theta(n \log n)$

ב. לה העלה של כמה עבור המערך הנתון בכל קריאות,



העלה יתפצל ל- $\frac{1}{2}$ מהאפסים שבכל קריאה ולמערך של $\frac{1}{2}$

מהאפסים ועוד $\frac{n}{2}$ אחדות. לה יקרה $\log(\frac{n}{2})$ פעמים

עד שהאחדות יתפצלו מהאפסים לגמרי.

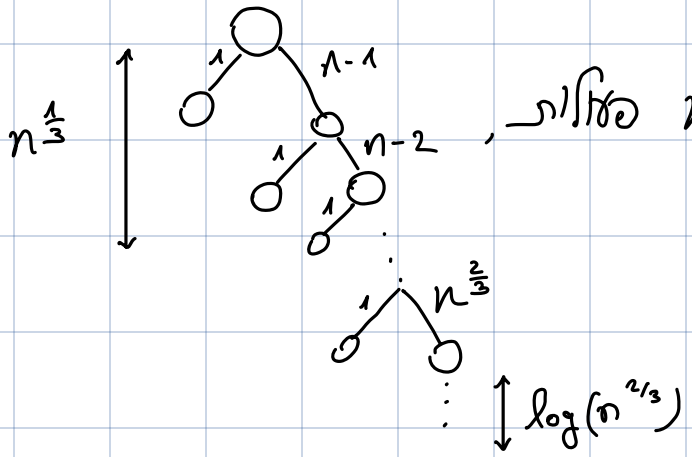
בכל קריאה של Partition עד הפיצול יהיה סדר גודל של n פעולות.

לכן, עד שהאחדות יתפצלו מהאפסים, נקבל זמן ריצה של $n \cdot \log(\frac{n}{2})$.

הרעש שבאחדות יתפצלו, אלחנו במקרה של מיון אותן, והראינו שצת סדר

גודל של מעלה פעולות. בסה"כ זמן הריצה $\Theta(n \log n)$.

ג. המערך ממוין, ולכן partition תפצל את הרשימה ל- תתי מערך באורך 1 ובאורך $n-1$ בכל פריצה. כה יקרה $n^{\frac{1}{3}}$ פעמים, עד שנגיע למערך באורך $n^{2/3}$ של הסריקה שמפיצה בתוך ה- $n^{\frac{1}{3}}$, שעורו זמן הריצה יהיה כפי שניתנו בסעיף ב, $n^{2/3}\cdot\log(n^{2/3})$.



בכל שלב בעלי יהיה סדר גודל של n פעולות, ולכן $g(n) = c'\cdot n\cdot n^{1/3} + c\cdot n^{2/3}\cdot\log(n^{2/3})$

$\Rightarrow \Theta(g(n)) = \Theta(n^{4/3})$

2) k. לא יתכן. קיים חסם תחתון על מיון מבוסס השוואות שהוא $n\log n$.

מכיוון שקיימים יותר מספרים שערכם לא ידוע, חייב לבצע השוואות כחלק מהאלגוריתם שימיין אותם. לכן, נהיה חייבים להשתמש באלגוריתם מבוסס השוואות ⇐ קיים חסם $n\log n$.

ב. לא יתכן. כדי להפוך מערך למערך פרבולי, נצטרך למיין כל אחד מחצאי המערך. במיון מבוסס השוואות, למיון חצי מערך יעלה לפחות $\frac{n}{2}\log\frac{n}{2}$ ולכן נקבל שהזמן הכולל יהיה $n\log(\frac{n}{2}) > O(n)$.

ג) יתכן. ראינו ש- $O(n^{1.01}) << O(n\log n)$. אם נמיין כל אחד מחצאי המערך באחד מאלגוריתמי המיון שראינו (לדוגמה מיזג), זה יעלה $n\log n$ שקטן משמעותית מ- $O(n^{1.01})$.

3) ג.

```
return max(A,P,q)
return MedianRecursive(A,P,q,K)
return MedianRecursive(A,q+1,r,K-left_size)
```

ב. ניקח את המקרה הגרוע ביותר, מערך ממוין הפוך: $\boxed{n \mid n-1 \mid \cdots \mid 2 \mid 1}$.

עבור קלט כזה, נקבל ש- partition יחלק תמיד רשימה פחות, n פעמים.

למן הכינה של partition עבור קלט באורך S הוא $\Theta(S)$.

$$\Rightarrow T_{\substack{worst\\case}} = \sum_{s=1}^{n} s = \frac{n\cdot(n+1)}{2} \Rightarrow T_{\substack{worst\\case}} = \Theta(n^2)$$

ג. נשתמש במשפט המאסטר: $T(n) = 1\cdot T(\frac{2}{3}n) + \overbrace{n}^{f(n)=n}$

$\Leftarrow$ $a=1$, $b=\frac{3}{2}$ $\Leftarrow$ $P = \log_{\frac{3}{2}} 1 = 0$.

אנחנו במקרה 3 של משפט המאסטר: $f(n) = n = \Omega(n^{P+\varepsilon}) = \Omega(n^{\varepsilon}) = \Omega(n)$, $\varepsilon = 1$

בנוסף, $\frac{2}{3}n = f(\frac{2}{3}n) \le c f(n) = cn$, $c = \frac{3}{4}$

לכן, $T(n) = \Theta(n)$.

3. נתאר את האלגוריתם: נבצע partition שנקבע עבורו שה- pivot=0 כך שכל המספרים הגדולים יהיו בצד שמאל, והיתר בצד ימין. ניקח את תת המערך של המספרים החיוביים (באורך $\frac{2}{3}n$) ונבצע עליו fourth_Partition. ניקח את הרבע השמאלי לאחר ההלוקה, נריץ עליו את הפרוצדורה max, ולזה יהיה החסיון שלנו.

fourth_Partition זו אותה פרוצדורה כמו third_Partition רק שהיא מחלקת את המערך לתת מערך באורך $\frac{n}{4}$ ואחד באורך $\frac{3n}{4}$. כך שכל האיברים בתת המערך באורך $\frac{n}{4}$, קטנים מהמספרים בתת המערך באורך $\frac{3n}{4}$.

<u>זמן ריצה</u>: שלושה חלקים לאלגוריתם:

1. ה- partition הראשון, רץ בזמן $\Theta(n)$.
2. ה- fourth_Partition, רץ בזמן $\Theta(n)$.
3. ה- max, רץ בזמן $\Theta(n)$.

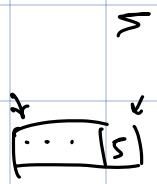
בסה"כ, נקבל שהאלגוריתם רץ ב- $\Theta(n)$.

4) בשורה ה-8 נכתוב $A[j-i+1] \leftarrow i$.

נסביר: עד שורה 6 כה אותו אלגוריתם שראינו בכיתה.

כאשר $C[i] > 0$, נכניס למערך A במקום ה- $j-i+1$ (בכניסה הראשונה $j-i+1=1$) את הספרה ה-i. i לא משתנה, j גדל ב-1 ולכן בפעם הבאה נכניס ספרה במקום הבא במערך. כה קורה עד ש- $C[i]=0$ ואז i ו-j שניהם יגדלו ב-1 ואת הספרה הבאה נכניס במקום המתאים וכך הלאה.

```
EMPTY(Q):
    if Q.num_of_elements = 0
        return True
    return False
```




```
ENQUEUE(Q,X):
    if Q.num_of_elements = MAXSIZE
        return overflow
    Q.rear = (Q.rear + 1) mod n
    Q.Elements[Q.rear] = X
```

```
DEQUEUE(Q):
    if (!EMPTY(Q)):
        Q_temp = Q.head
        Q.head = (Q.head + 1) mod n
        return (Q.elements[Q.temp])
    else
        return underflow
```

```
FRONT(Q):
    if (!EMPTY(Q))
        return Q.elements[Q.head]
    else
        return underflow
```