שא' 1

א. האלג' יתחיל ב אתחול את העזר, ההתחלתי p ולאחר מכן ירוץ בלולאה איטרטיבית על איברי המערך ויבדוק אם יש איברים ב A במקום האינדקס הגדולים ממנו ואז יחליף את תוכן האינדקס ב i הנוכחי, כך עד שימצא את האיבר הכי גדול.

- Iterative-Max(A,p,r)

```
index ← p
for i ← p+1 to r do
    if A[index] < A[i] then
        index ← i
return(index)
```

ב.

מס' פעולות	
	MAXB(A, p, r)
1	if p=r
	then return A[p]
T(n-1)	else temp ← MAXB(A, p, r-1)
1	if temp ≥ A[r]
1 (פעולה או השני)	then return temp
	else return A[r]

הערה
פה נתייחס לn
גודל המערך בקלט
כפי שהתייחסנו
בפתרון השאלות בכיתה.
(אפשר להתייחס לזה
כ: $n=r-p+1$)
ובמצב $T(n-1)=T(r-p+1-1)$

כאשר יש איבר אחד נקבל את תנאי העצירה ותתבצע פעולה אחת $\leftarrow T(1)=1$

ולכן נוסחאת הנסיגה:

$$T_1(n) = \begin{cases} T(n-1)+3 \\ T(1)=1 \end{cases}$$

ג'

הערה:
ע"פ ההנחה ניתן
להתייחס לn כאורך המערך בין r לp
לכן ← $\frac{r-p+1}{2} = \frac{n}{2}$

	מס' פעולות
Rec_Max(A,p,r)	
if p=r Then	1
return p	(אני נכליל את זה בחלק הif)
else	
one ← Rec_Max(A, p, $\lfloor\frac{p+r}{2}\rfloor$)	$T(n/2)$
two ← Rec_Max(A, $\lfloor\frac{p+r}{2}\rfloor$+1, r)	$T(n/2)$
if A[one]>A[two] Then	1
return one	1 איזה שהוא
else return two	

<u>הסבר:</u> האלגוריתם הרקורסיבי, כאשר יקבל גבולות שווים, תנאי עצירה
יחזיר את האינדקס היחיד, אחרת יחלק את המערך
לשני חלקים, וישלח בחזרה לעצמו, כך עד שיגיע לתנאי
העצירה. - משם בצורה "זה מעלה" בחזרה כך שכל
פעם תהיה לנו השוואה בין one לtwo
ונחזיר את הגדול מביניהם.

3. נוסחאות נסיגה:

$$T(n) = \begin{cases} T(n) = 2T(n/2)+3 \\ T(1)=1 \end{cases}$$

2. א.

מס' פעולות	
	`Rec_Num(A,r,i,num)`
1	`if A[r] = num then`
	`    i += 1`
1	`if r = 1 then`
	`    return i`
	`else`
T(n-1)	`    Rec_Num(A,r-1,i,num)`

<u>הסבר:</u> נקרא לאלגוריתם בראשונה כאשר r יגביל את אורך המערך ו i מאותחל לאפס. ⇐ Rec_num(A,n,0,num) האלגוריתם יבדוק כל פעם אם האיבר הנוכחי שווה ל num ואם כן יגדיל אותו; תנאי העצירה הוא כאשר אנו נמצאים באיבר האחרון ואז יוחזר i: כל עוד אנחנו לא באיבר האחרון האלגוריתם יפעל בצורה רקורסיבית כאשר הוא מוריד אחד מאורך המערך – r המתפקד גם כמונה.

ב. $$T(n) = \begin{cases} T(n) = T(n-1) + 2 \\ T(1) = 1 \end{cases}$$

ג. אין שום הבדל בין המקרה הטוב למקרה הרע, בכל מקרה האלגוריתם יעבור על כל איברי המערך ויבדוק "אחד אחד" אותם עד שיגיע לתנאי העצירה.

.ג

```
Rec-num_k(A,r,i,num,k)
 if A[r] = num then
    i+=1
    if i=k then
       return true
 if r=1 then
    return false
 else
    Rec_num_k(A,r-1,i,num,k)
```

הסבר: האלגוריתם זהה לקודם בסעיף א, חוץ מהבדיקה שביצעתי לאחר הגדלת i, שבודקת האם הגענו לk ואם כן נחזיר שtrue.
לאחר מכן שיניתי - שאם הגענו לסוף המערך ⇐ $r=1$ כלומר $i<k$, לא נכנסנו לתנאי ולכן נחזיר שfalse.
(כמובן i מתחיל באפס)

ה. האלגוריתם הזה יעצור ברגע שנמצא מספיק מספרים שווים לnum ולכן המקרה הטוב שלו הוא כאשר $k=1$ והמספר הראשון במערך שווה לnum, האלגוריתם ירוץ פעם אחת בלבד (אפשר גם ש$n=k$ ואז גם האלגוריתם ירוץ פעם אחת)
המקרה הגרוע הוא כאשר אין מספיק מספרים שווים לnum, כלומר $i<k$ ולכן הוא ירוץ על כל המערך (מקרה זה גם יקרה שהאיבר האחרון מקיים את $i=k$)

ש' 3

א. נציב במשוואה את הנתונים

ונפתור את המשוואה:

$$100n^2 < 2^n \quad / lg$$

$$\Downarrow$$

$$lg100n^2 < n$$

$$\Downarrow$$

$$2(lg100 + lg n) < n$$

$$2(\lceil 6.6 \rceil + lg n) < n$$

נתייחס לכך כי אנחנו במספרים שלמים

$$\downarrow$$

$$14 + 2lg n \le n$$

אנו רואים כי כאשר $n \ge 14$ האלגוריתם שזמן ריצתו $100n^2$ רץ יותר מהר מהאלגוריתם שזמן ריצתו 2^n.

ב. נציב במשוואה:

לפי הנתונים, פקודה מתבצעת ב $10^{-9} = \frac{1}{10^9}$ שניות כלומר המחשב יבצע בשנייה 10^9 הוראות $\Rightarrow \frac{10^9}{10^9} = 1$, נציב בנוסחה ונבדוק מתי $n!$ יבצע יותר מ 10^9 פעולות בשנייה

$$\frac{n!}{10^9} < 1$$

$$n! < 10^9$$

$$n < 13$$

ג. כמו בתרגיל הקודם

$$\frac{\lg n}{10^9} < 1 \quad / \cdot 10^9$$

$$\Downarrow$$

$$\lg n < 10^9$$

$$\Downarrow$$

כאשר מתקיים $n < 2^{10^9}$

האלגוריתם ירוץ בפחות משנייה.

ד. נדרוש ונשווה לפי הנתון בתרגיל:

$$\lg n < n^{0.1}$$

נרצה לבדוק מתי בעצם $\lg n$ יהיה גדל מ $n^{0.1}$

$$\Downarrow$$

לפי חוקי לוגים:

$$n < 2^{n^{0.1}}$$

$$\Downarrow$$

כאשר $1 < n$, כלומר האלגוריתם חייב להתבצע לפחות על איבר אחד $\Leftarrow$ $1 < n^{0.1}$ חייב להתקיים ולכן $2 < 2^{n^{0.1}}$ חייב להתקיים גם כן.

לסיכום, כאשר $2 < n$ זמן הריצה של האלגוריתם $n^{0.1}$ יהיה מהיר יותר מ זמן הריצה של האלגוריתם $\lg n$

שא' 4

א. הטענה בשא' נכונה, נסביר את הפתרון:

① נמיין את המערך בעזרת מיון-מיזוג שזמן הריצה במקרה הגרוע הוא $O(n \log n)$ (לפי עמ' 13 בספר)

② לאחר מכן נעבור בסריקה סדרתית על המערך ונמנה כל איבר שחוזר על עצמו - מכיוון שהמערך ממוין כבר כל האיברים השווים יהיו אחד אחרי השני, לכן כל פעם שנגיע לאיבר חדש - נאפס מונה ונספור בנוסף את מספר הפעמים השווים הסמוכים, אם הגענו לאיבר שונה, נשמור בצד את המונה המקסימלי ואת האיבר, ונמשיך כך כשכל פעם נבדוק האם המונה של האיברים הנוכחיים גדול מהמקסימלי שמצאנו, לבסוף נדפיס את המונה המקסימלי ואת האיבר.

```
זמן          Find_Common(A,n)
T(n log n)   MERGE-SORT(A,1,n)
1            J←0, Max←0, Evar←0
T(n-1)       for i←1 to n-1 do
                 if A[i] = A[i+1] then
                     J←J+1
                     if J > max then
                         max←J
                         Evar←A[i]
                 else
                     J←0
1            return (Evar, Max+1)
```

הערה חשובה

בתרגיל אני מניח כי אם ישנם איברים המופיעים במערך בשכיחות שווה, הוא יחזיר את הראשון שהוא נתקל בו מבינהם. או לחלופין שבמערך יש רק איבר אחד שהוא השכיח ביותר.

זמן ריצה

מיון

מכיוון שזמן הריצה של המיון במקרה הגרוע הוא $O(n\log n)$ והוא הדומיננטי לעומת הלולאה הסדרתית שרצה בזמן $O(n-1)$ סה"כ זמן הריצה הוא $O(n\log n)$

ב. מבין כל אלגוריתמי המיון שלמדנו עד אתק הכי מתאים יהיה מיון הכנסה מכיוון שיהיו כמה זוגות שאיבריהם סמוכים אשר "הפוכים" בסדר ושאר המערך יהיה ממוין.
במיון הכנסה הוא עובר על כל איבר נוכחי ומשווה לאיברים משמאלו, מכיוון שלפי ההוראות הנתונה, רק "אי סדרים" במערך יהיו בזוגות סמוכים, כל פעם שירת ה"דחיפה" (ההכנסה) תרוץ פעם אחת - תציב את האיבר הנוכחי מקום אחד שמאלה, לפני האיבר שהתהפך איתו (לפי הנתון), לכן גם עבור הזוגות הללו, כל האיברים יוזזו מקום אחד ימינה

הערה

ישנו מקרה שבו ישנם הרבה זוגות הפוכים במערך וזמן הריצה של המיון יגיע למקרה הגרוע - $O(n^2)$, ולכן במקרה זה דווקא יתאים מיון מיזוג שלו הזמן $O(n\log n)$

שא' 5

א.

```
IsA&B(A,B,n)
for i←1 to n do                T(n)
  for j←1 to n do              T(n)
    if A[i] = B[j] then
      return TRUE
return FALSE
```

<u>הסבר:</u> האלגוריתם יעבור בלולאה מקוננת על כל איבר ב A ישווה ויעבור על כל איברי B, ברגע שנתקל במספר שמופיע ב B ושווה למספר הנוכחי ב A, יחזיר TRUE.

<u>ניתוח זמן:</u> בעזרת האלגוריתם, עבור את הלולאה הראשונה n פעמים, ועבור כל איטרציה של הלולאה הראשונה (מונה i) יבצע לולאה נוספת על כל מערך B גם היא n פעמים,

<u>סה"כ:</u> $\underbrace{(1..n)}_{\text{מעבר על A}}\cdot\underbrace{n}_{\text{מעבר על B}} = n\cdot n = n^2 = O(n^2)$

ב.

זמן

```
T(nlogn)   IsA&B2(A,B,n)
           merge-sort(B,1,n)
T(n)       for i←1 to n do
T(logn)        if True=Binary_search(B,1,n,A[i]) then
                   return true
           return false
```

<u>הסבר:</u> האלגוריתם מתחיל בביצוע מיון-מיזוג על מערך B, לאחר מכן יעבור בלולאה למערך איברי A ועבור כל איבר יבצע חיפוש בינארי ב-B, ואם החיפוש הצליח ונמצא איבר A[i] בתוך B נחזיר true, אחרת נחזיר false.

<u>ניתוח זמן:</u> מיון מיזוג על מערך B יקח לפי הספר $O(n\log n)$ (עמ' 31)

כפי שלמדנו בכיתה, זמן הריצה של חיפוש בינארי הוא $O(\log n)$.

מכיון שבלולאה עבור כל איבר של A נבצע חיפוש בינארי ב B, נבצע n פעמים חיפוש בינארי:

$$\underbrace{(1..n)}_{\text{מעבר על A}} \cdot \underbrace{O(\log n)}_{\text{זמן חיפוש בינארי אחד}} = O(n\cdot\log n) + \underbrace{O(n\log n)}_{\text{זמן של מיון מיזוג}}$$

לכן סה"כ זמן ריצת האלגוריתם מכיון ששני הזמנים מסדר גודל זהה: $O(n\log n)$.