שאלה 1

א. 1) $n(\log_2 n)^k = O(n(\log_{10} n)^k$

$n(\log_2 n)^k < c(n(\log_{10} 2)^k(\log_2 n)^k$

$1 < c(\log_{10} 2)^k$

לכן הפסוק הוא **אמת**.

2) $2^{(\log_2 n)} = O(2^{(\log_{10} n)}$

$2^{(\log_2 n)} < c*2^{((\log_{10} 2)*(\log_2 n))}$

$\log_2 n > (\log_{10} 2)*(\log_2 n)$

לכן הפסוק הוא **שקר**.

3) $(\sqrt{2})^{(n*\log n)} = O(n^{(\sqrt{(n/2)})}$

$2^{((n/2)*\log n)} < c*2^{(\sqrt{(n/2)}*\log n))}$

$(n/2) > \sqrt{(n/2)}$

לכן הפסוק הוא **שקר**

4) – נוסחאת המקור -

$((\log n)/2)*((\log n+1)/2^n) < c*1$

$(\log n)^2 + \log n < 2^{(n+1)}*c$

הפסוק הוא **אמת** בגלל שברור שהצד האקספוננציאלי תמיד ייגדל מהר יותר מהצד הלוגריתמי.

ב. 1) על כל איבר ברשימה הפונקציה מוסיפה לה כמות איברים לפי ערך האינדקס של האיבר. $\sum_{i=0 \to (n-1)} i = (n^2 - n)/2$, זאת סדרה חשבונית. לכן התשובה היא $O(n^2)$.

2) מספר האיטרציות בפונקציה כאורך הרשימה במקורית, ובכל איטרציה כמות האיברים ברשימה גדלה פי 2 לכן: $\sum_{i=0 \to n}(2^i)*n = n*(2^n - 1)$ סכום סדרה הנדסית, לכן התשובה היא $O(n*(2^n))$.

3) הפונקציה פועלת באותו אופן בדיוק כמו בסעיף 2, רק כתובה באופן קצת שונה. לכן גם כאן התשובה תהייה $O(n*(2^n))$.

.

4) מספר האיטרציות כאורך הרשימה, ובכל איטרציה נוספים לרשימה 10,000 איברים ללא תלות בשום דבר, כלומר מספר קבוע.

לכן מספר הפעולות תלוי לינארי בn, והתשובה תהייה $O(n)$.

ג. לפי מודל הזיכרון של פייטון, הפונקציה פועלת במקום, כלומר – הלולאה עוברת על כל איבר ברשימה, ומיד מוסיפה לה איבר נוסף ('a') עבורו, ורק אז ממשיכה לאיטרציה הבאה. לכן, גם עבור ה'a' הנוסף יוכנס איבר 'a' נוסף לרשימה, ולעולם לא נצא מהלולאה, או שנלחץ ctrl+c.

שאלה 2

ב)

```
>>> import math
>>> f1 = lambda x: x**2 - math.cos(x)
>>> find_root(f1,-10,0)
-0.823974609375
```

ג) התוכנית לא עוצרת כי בשלב מסוים, ממוצע ערכי הפונקציה בקצוות שווה לערך הפונקציה בקצוות עצמם בקירוב, וכך אנחנו נכנסים ללולאה אינסופית, כיוון שלמעשה הקצוות לא משתנים יותר (ובדיוק על זה מתבסס האלגוריתם). זה קורא בגלל הערך של אפסילון.
בנקודה x = 12345678901 הפונקציה לא עובדת באופן מדויק מספיק (אפסילון לא מספיק קטן) כדי לאפשר לערך הקצוות להשתנות ביחס לממוצע שלהם.

שאלה 3

```
def multi_merge_v1(lst_of_lsts):
    all = [e for lst in lst_of_lsts for e in lst]  #O(m*n)
    merged = [ ]
    while all != [ ]:
        minimum = min(all)          #∑(i = 0 -> m*n)m*n – i = O(((m^2)*(n^2))/2)
        merged  += [minimum]                #O(1)
        all.remove(minimum)                 #O(1)
    return merged
```

א) כאשר n מייצג את האורך המקסימלי של כל רשימה, ו-m מייצג את מספר הרשימות ברשימת העל.
לכן כפי שניתן לראות בניתוח הסיבוכיות של הפונקציה, יתקבל שהיא O((n^2)*(m^2)).

```
def multi_merge_v3(lst_of_lsts):
    m = len(lst_of_lsts)
    merged = [ ]

    for lst in lst_of_lsts:   # runs m itterations
        merged = merge(merged, lst)   #adds n items to list each iteration at worst case

    return merged
```

ד) כפי שניתן לראות מניתוח הפונקציה, הסיבוכיות תהייה O(m^2*n).

שאלה 4

א. n = 10,000 כי אם המחשב שלה רץ פי 2 יותר מהר, כלומר 2*log100, שזה שווה ל log(100^2).
ב. n = 200, הסיבוכיות היא לינארית ולכן ברור שאם הוא רץ פי 2 יותר מהר אז גם גודל הקלט שהיא יכולה לרוץ עליו יגדל פי 2.
ג. n = √20,000 = 141.42 , זה יהיה גודל הקלט המקסימלי שהיא תוכל לרוץ עליו כאשר המחשב רץ פי שתיים יותר מהר.
ד. n = 101, כיוון ש 2*2^100 = 2^101.

שאלה 5

ב. דרשתם שהמספרים יהיו שלמים, כי אחרת היה ניתן לשטות באלגוריתם ולגרום לו לפספס מקומות שהיו עונים על התנאי, למשל ע"י הכנסת הרשימה: [4, 3 , 2.5, 2 , 1]. כאשר האלגוריתם היה מגיע ל2.5, הוא היה מחפש למטה, כיוון שלפי החוקיות גם כל שאר המספרים מימין אמורים להיות גדולים מהאינדקסים שלהם. אלא שזה לא נכון למקרה הזה.

ג. דרשתם שהמספרים יהיו שונים מאותה סיבה ממקודם – זה מאפשר "לרמות" את האלגוריתם. כמו למשל ע"י הרשימה הזאת: [4,4,4,4,4]. הנימוק לא שונה מהסעיף הקודם.

שאלה 6

כדי שסיבוכיות האלגוריתם מסעיף א' תהייה עדיפה על quicksort צריך להתקיים:
$O(n+k) < O(n\log n)$, כלומר: $n + k < n\log n$, ומכאן: $k < n(\log n - 1)$.