

מבוא לתכנות מתמטי

חישוב סיבוכיות

תרגיל

נניח ונתונים שני אלגוריתמים הפותרים את אותה הבעיה.
אלגוריתם ראשון רץ במשך 0.32 דקות.
אלגוריתם שני רץ במשך 0.4 דקות.

באיזה אלגוריתם כדאי להשתמש? מי האלגוריתם היעיל ביותר?

חישוב יעילות

נרצה להעריך את האלגוריתמים ללא תלות במחשב עליהם הם רצים.

שאלה:

כמה זמן (בערך) יקח למחשב לחשב

$$X=y+z$$

?

חישוב יעילות

נרצה להעריך את האלגוריתמים ללא תלות במחשב עליהם הם רצים.

שאלה:

כמה זמן (בערך) יקח למחשב לחשב

$$X=y+z$$

?

קיים קבוע k שחוסם את זמן הריצה.

חישוב יעילות

שאלה:

מה הוא זמן ריצה של הקוד הבא?

```
for i=1:n  
    x=y+z;  
end
```

חישוב יעילות

שאלה:

מה הוא זמן ריצה של הקוד הבא?

```
for i=1:n  
    x=y+z;  
end
```

אנו "נכנסים" ללולאה n פעמים ובכל פעם למקסימום k יחידות זמן ולכן סה"כ
 $f(n) = k \times n$

במילים אחרות החסם העליון לזמן הריצה של קטעה קוד זה הוא בסדר גודל
(order) של n .

ובאופן כללי יותר :

כאשר יש לנו קטע קוד שזמן ריצתו חסום ע"י $k \times t(n)$ נאמר שזמן הריצה הוא $O(t(n))$. כלומר בסדר גודל של $t(n)$.

באופן פורמלי :

יהי $t(n)$ זמן ריצת קטע קוד מסוים כאשר n הוא גודל הקלט .

$$t(n) = O(\underline{f(n)}) \iff \exists N, c \mid \forall n > N, t(n) \leq c \times f(n)$$

תרגיל

• מה הסיבוכיות זמן של הקוד הבא?

```
for i=1:n
  for j=1:n
    x=y+z;
  end
end
```

תרגיל

• מה הסיבוכיות זמן של הקוד הבא?

```
for i=1:n
  for j=1:n
    x=y+z;
  end
end
```

נסתכל פנימה וכלפי חוץ : הלולאה הפנימית חסומה ע"י $k \times n$ (לפי דוגמא קודמת) ולולאה החיצונית "נכנסים" n פעמים. לכן סה"כ נקבל חסם של $(k \times n) \times n$ יחידות זמן או $O(n^2)$.

באופן כללי יותר :

$$O(t(n)) \times O(g(n)) \leq (k \times t(n)) \times (c \times g(n)) = k \times c \times t(n) \times g(n) =$$

$$O(t(n) \times g(n))$$

תרגיל

טענה: עבור k קבוע מתני

הוכחה: נמצא קבועים c ,

דוגמאות

$$f(n) = 1044 = O(1)$$

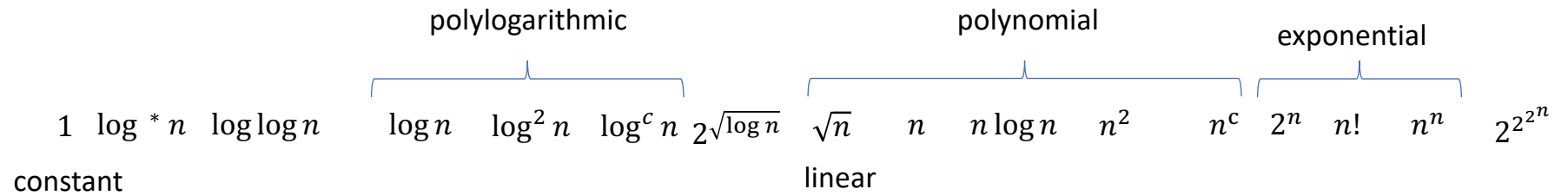
קבוע

$$f(n) = 5 + n = O(n)$$

לינארי

$$f(n) = n + n^2 = O(n)$$

היררכיה של פונקציות נפוצות



מה היעילות של הקוד הבא?

```
for i=1:n
  for j=1:n
    x=y+z;
  end
  x=1+x;
  for k=1:n
    x=y+x;
  end
end
```

מה היעילות של הקוד הבא?

```
for i=1:n
  for j=1:n
    x=y+z;
  end
  x=1+x;
  for k=1:n
    x=y+x;
  end
end
```

שוב נתחיל מבפנים כלפי חוץ :

בתוך הלולאה החיצונית הסיבוכיות היא :

$$O(n) + k + O(n) = O(n)$$

כלומר בכל איטרציה של הלולאה החיצונית אנו מבצעים חישוב בסדר גודל של n ולכן בסה"כ :

$$O(n) \times O(n) = O(n^2)$$

תרגיל

```
function flag = search_item (arr,k)
    flag = 0;
    for i=1:length(arr)
        if arr(i)== k
            flag = 1;
            break
        end
    end
end
```

חשבו את הסיבוכיות של הקוד.

תרגיל

```
function flag = search_item (arr,k)
    flag = 0;
    for i=1:length(arr)
        if arr(i)== k
            flag = 1;
            break
        end
    end
end
end
```

חשבו את הסיבוכיות של הקוד.

תמיד נסתכל על המקרה הכי גרוע

$O(n)$

תרגיל

```
algorithm merge(A, B) is
  inputs A, B : list
  returns list

  C := new empty list
  while A is not empty and B is not empty do
    if head(A) ≤ head(B) then
      append head(A) to C
      drop the head of A
    else
      append head(B) to C
      drop the head of B

  // By now, either A or B is empty. It remains to empty the other input list.
  while A is not empty do
    append head(A) to C
    drop the head of A
  while B is not empty do
    append head(B) to C
    drop the head of B

  return C
```

ויקיפדיה

תרגיל

```
algorithm merge(A, B) is
  inputs A, B : list
  returns list

  C := new empty list
  while A is not empty and B is not empty do
    if head(A) ≤ head(B) then
      append head(A) to C
      drop the head of A
    else
      append head(B) to C
      drop the head of B

  // By now, either A or B is empty. It remains to empty the other input list.
  while A is not empty do
    append head(A) to C
    drop the head of A
  while B is not empty do
    append head(B) to C
    drop the head of B

  return C
```

סיבוכיות זמן : החלק הראשון של הפונקציה (עד הלולאה הראשונה) לוקח זמן קבוע ולכן לא מעניין אותנו. נשאל כמה פעמים נכנסים ללולאה הראשונה במקרה הגרוע ביותר ? התשובה היא $m + n - 1$ פעמים ובכל פעם מבצעים מספר קבוע של פעולות חסומות. לכן סיבוכיות חלק זה היא $O(m+n)$. סיבוכיות שתי הלולאות האחרונות הינה או $O(m)$ או $O(n)$ ובכל מקרה מוכלת ב $O(m+n)$ ולכן סה"כ סיבוכיות זמן $O(m+n)$.

תרגיל

נתון בניין בעל 1024 קומות ונתון אוסף של כדורי זכוכית שבירים וזהים.
יכול להיות שקיימת קומה בבניין שעד אליה ניתן לזרוק את הכדורים מבלי
שיתנפצו ומעבר לקומה הזאת יתנפצו אם נשליך אותם למטה.

כיצד נוכל למצוא את הקומה הזאת, אם נתון לנו כדור אחד?

תרגיל

נתון בניין בעל 1024 קומות ונתון אוסף של כדורי זכוכית שבירים וזהים.
יכול להיות שקיימת קומה בבניין שעד אליה ניתן לזרוק את הכדורים מבלי
שיתנפצו ומעבר לקומה הזאת יתנפצו אם נשליך אותם למטה.

כיצד נוכל למצוא את הקומה הזאת, אם נתון לנו כדור אחד?
פתרון: נתחיל מקומה ראשונה, ונזרוק מכל קומה עד שישבר.
מקסימום זריקות: $O(n)$

תרגיל

נתון בניין בעל 1024 קומות ונתון אוסף של כדורי זכוכית שבירים וזהים.
יכול להיות שקיימת קומה בבניין שעד אליה ניתן לזרוק את הכדורים מבלי
שיתנפצו ומעבר לקומה הזאת יתנפצו אם נשליך אותם למטה.

כיצד נוכל למצוא את הקומה הזאת, אם נתונים לנו כמה כדורים שנרצה?

תרגיל

נתון בניין בעל 1024 קומות ונתון אוסף של כדורי זכוכית שבירים וזהים. יכול להיות שקיימת קומה בבניין שעד אליה ניתן לזרוק את הכדורים מבלי שיתנפצו ומעבר לקומה הזאת יתנפצו אם נשליך אותם למטה.

כיצד נוכל למצוא את הקומה הזאת, אם נתונים לנו כמה כדורים שנרצה?
פתרון: לזרוק מחצי גובה, אם נשבר לזרוק מחצי הגובה של הבנין התחתון.
אם לא, נזרוק מחצי גובה של הבניין העליון.
נמשיך ככה עד שנמצא את הקומה.

$O(\log n)$

תרגיל

נתון בניין בעל 1024 קומות ונתון אוסף של כדורי זכוכית שבירים וזהים. יכול להיות שקיימת קומה בבניין שעד אליה ניתן לזרוק את הכדורים מבלי שיתנפצו ומעבר לקומה הזאת יתנפצו אם נשליך אותם למטה.

כיצד נוכל למצוא את הקומה הזאת, אם נתונים לנו כמה כדורים שנרצה?
פתרון: לזרוק מחצי גובה, אם נשבר לזרוק מחצי הגובה של הבנין התחתון.
אם לא, נזרוק מחצי גובה של הבניין העליון.
נמשיך ככה עד שנמצא את הקומה.

$O(\log n)$

PSEUDOCODE OF COUNTING SORT

```
1 CountingSort(A)
2   //A[]-- Initial Array to Sort
3   //Complexity: O(k)
4   for i = 0 to k do
5     c[i] = 0
6
7   //Storing Count of each element
8   //Complexity: O(n)
9   for j = 0 to n do
10    c[A[j]] = c[A[j]] + 1
11
12   // Change C[i] such that it contains actual
13   //position of these elements in output array
14   ///Complexity: O(k)
15   for i = 1 to k do
16     c[i] = c[i] + c[i-1]
17
18   //Build Output array from C[i]
19   //Complexity: O(n)
20   for j = n-1 downto 0 do
21     B[ c[A[j]]-1 ] = A[j]
22     c[A[j]] = c[A[j]] - 1
23 end func
```
