CS-234218 מבני נתונים

נאדר בשותי

חדר 737

שעות קבלה: ג 10:30- 11:30

/http://webcourse.cs.technion.ac.il/234218/

ספר הלימוד העיקרי (קיים גם תרגום):

.Cormen, Leiserson, Rivest, Introduction to Algorithms

©Geiger&Itai,2001

מטרות הקורס

1. הֶכֵּרות עם מבני נתונים ומימושיהם היעילים
2. פיתוח כלים לניתוח יעילות
3. בחירת מבני נתונים לפתרון בעיות

דוגמאות למבני נתונים: מחסנית, תור, מילון, תור עדיפויות, טבלת ערבול...

דוגמאות לשימושים: מיון, מימוש שָׂפות תכנות, ארגון קבצים, מְנוֹעֵי חיפש ועוד, ועוד

©Geiger&Itai,2001

תוכנית הקורס

1. מבני נתונים בסיסיים וסימונים אסימפטוטיים
2. מערכים ורשימות מקושרות
3. עצים ועצי חיפוש
4. עצי AVL
5. עצי 2-3
6. רשימות דילוגים
7. טבלאות ערבול
8. אחזקת קבוצות זרות
9. מיון
10. טיפול במחרוזות
11. גרפים
12. איסוף אשפה

©cs,Technion

הגדרות בסיסיות

טיפוס נתונים אבסטרקטי (Abstract Data Type = ADT): הוא אוסף של פעולות וכללים על קבוצת נתונים.

מימוש של מבנה נתונים הוא אוסף פרוצדורות, אחת לכל פעולה, הממשות את הפעולות של מבנה הנתונים.

דוגמא: מהי מחסנית?

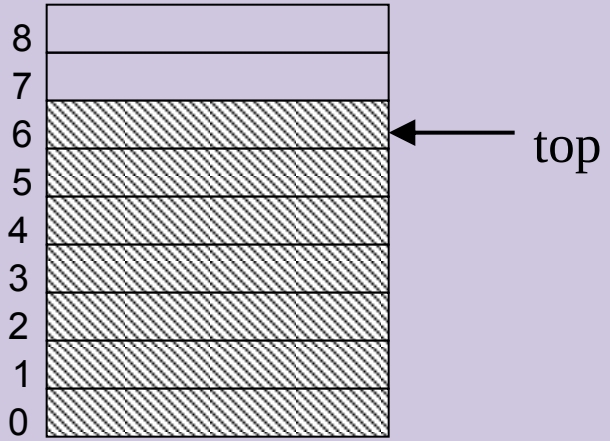
האם היא מערך עם מציין ל- top ?

אתחול: top = -1

הוספת איבר: top = top + 1; A[top] = x;

ראש המחסנית: if top >-1 returnA[top]

הוצאת איבר: top = top - 1



לא ! זהו מימוש של מחסנית באמצעות מערך ומצביע.

מחסנית כטיפוס נתונים אבסטרקטי

Last In -- First Out : LIFO אחרון נכנס – ראשון יוצא

מחסנית (Stack) מוגדרת ע"י הפעולות הבאות:

create(S) – מחזיר מחסנית S ריקה חדשה.

push(S,x) – מכניס איבר בעל ערך x לראש המחסנית S.

top(S) – מחזיר את האיבר שבראש המחסנית S (המחסנית אינה משתנה).

pop(S) – מוציא את האיבר שבראש המחסנית S.

is-empty(S) – מחזיר true אם המחסנית S ריקה ו-false אחרת.

פונקציה: **is-empty:{Stacks}** $\longrightarrow$ {true,false}

©cs,Technion

מחסנית כטיפוס נתונים אבסטרקטי (המשך)

פעולות המחסנית מקיימות את הכללים הבאים:

1. אפשר לבצע pop, top רק על מחסנית לא ריקה.
2. מיד לאחר create(S), is-empty(S) מחזיר ערך true.
3. לאחר ביצוע push, ואחריו pop, המחסנית לא משתנה.

כלל 2: **is-empty(create(S)) = true**

כלל 3: **pop(push(S,x)) = S**

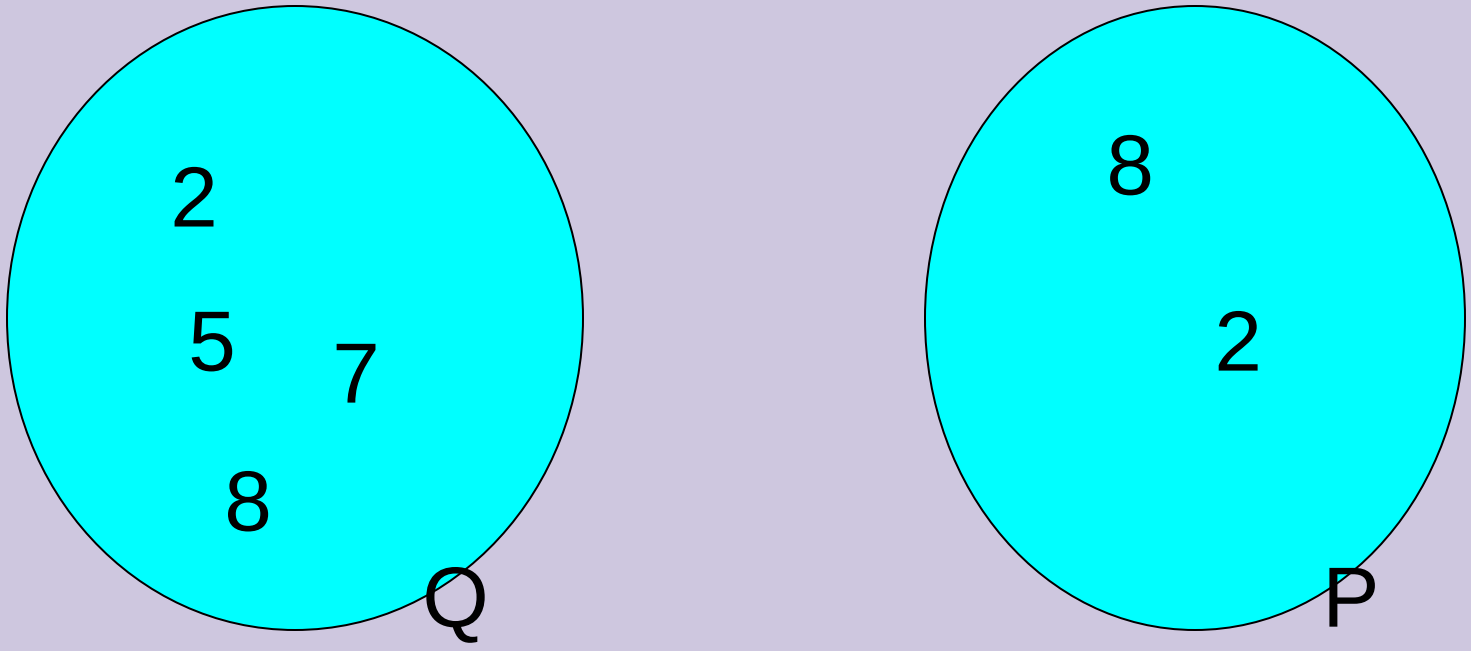
יש טענות הנובעות רק מהכללים. **לדוגמא:**

```
create(S);
push(S,17);
pop(S);
print is_empty(S);
```

מה יודפס - הוכח?

```
is_empty(pop(push(create(S))))=true;
```

כל מימוש חייב לאפשר את הפעולות ולקיים את הכללים.



Create(Q) Push(Q,2) Push(Q,5) Push(Q,7)

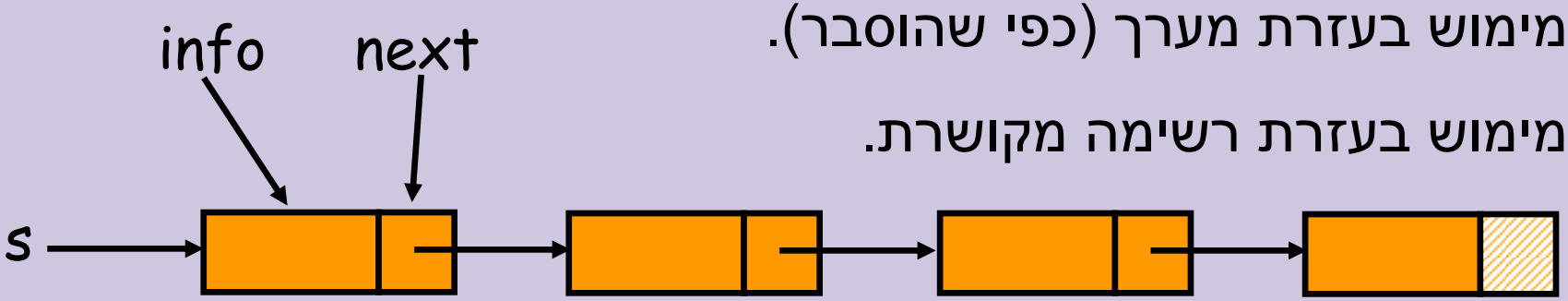
Pop(Q) Create(P) Push(Q,8) Is-Empty(P)

Push(P,8) Is-Empty(Q) Push(P,2) Pop(Q)

©cs,Technion

מימושי מחסנית

1. מימוש בעזרת מערך (כפי שהוסבר).
2. מימוש בעזרת רשימה מקושרת.



הגדרת צומת:

```
#define NULL 0
typedef struct node {
        DATA_TYPE info;
        struct node *next;
}  NODE;
typedef   NODE   *STACK;
```

יצירת מחסנית ריקה:

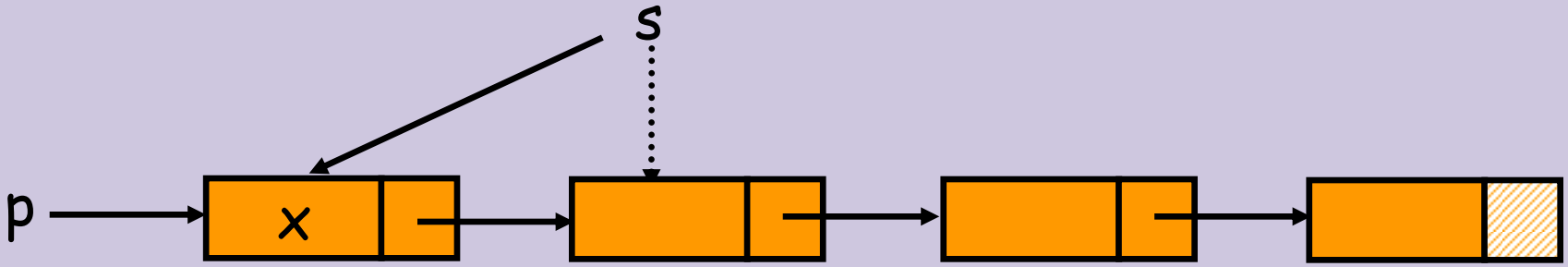
```
void create (STACK  *s) {
        (*s) = NULL ;
}
```

©cs,Technion

הכנסת איבר

פעולת push(s,x):

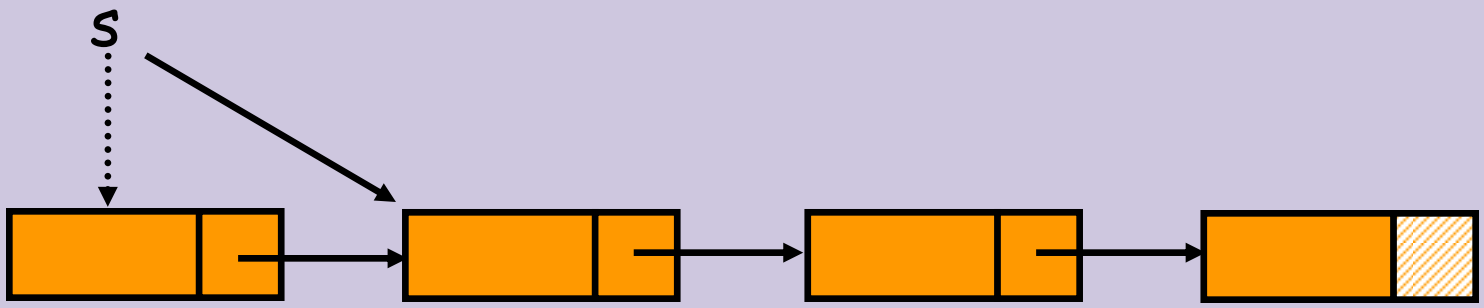
```
void  push ( STACK  *s,  DATA_TYPE x){
        NODE *P;
    P = malloc (sizeof (NODE));
    P -> info = x;
    P -> next = (*s) ;
    (*s) = p ;
}
```



אחזור והוצאת איבר

פעולת pop(s):

```
void  pop ( STACK  *s){
        STACK  t ;
        t = (*s) → next;
        free (*s) ;
        (*s) = t ;
}
```



פעולת top(s):

```
DATA_TYPE  top ( STACK  *s){
return (*s) → info ;
}
```

©cs,Technion

תור כמבנה נתונים

ראשון נכנס – ראשון יוצא First In -- First Out : FIFO

תור (Queue) מוגדר ע"י הפעולות הבאות:

create(Q) – מחזיר תור ריק.

head(Q)– מחזיר את ערך האיבר שבראש התור Q (התור אינו משתנה).

enqueue(Q,x) – מכניס איבר עם ערך x לסוף התור Q.

dequeue(Q) – מוציא את האיבר שבראש התור Q.

is-empty(Q) – מחזיר true אם התור Q ריק ו-false אחרת.

©cs,Technion

מימוש של תור בעזרת מערך

head(Q):	Q[f];
enqueue(Q,x):	Q[r] = x;
	r = (r+1) %n;
dequeue(Q)	f = (f+1) %n;
is_empty(Q):	f = = r;
create(Q):	f = r = 0;

מערך Q בן n איברים עם שני מציינים

r = מציין את מקום האיבר שאחרי סוף התור (rear)

f = מציין את מקום האיבר שבראש התור (front)

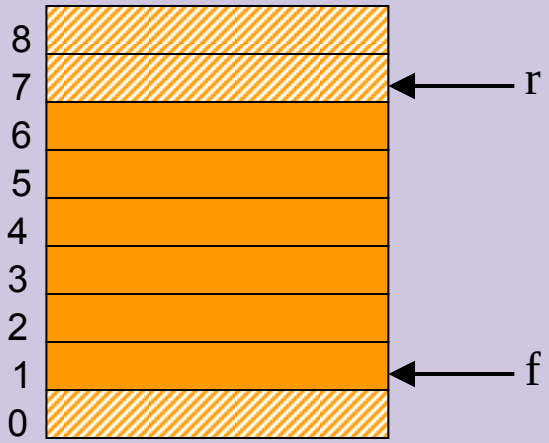
נקודות חשובות:

הפעולות האריתמטיות נעשות mod n (% n).

התור מלא אם f = = (r +1)% n .

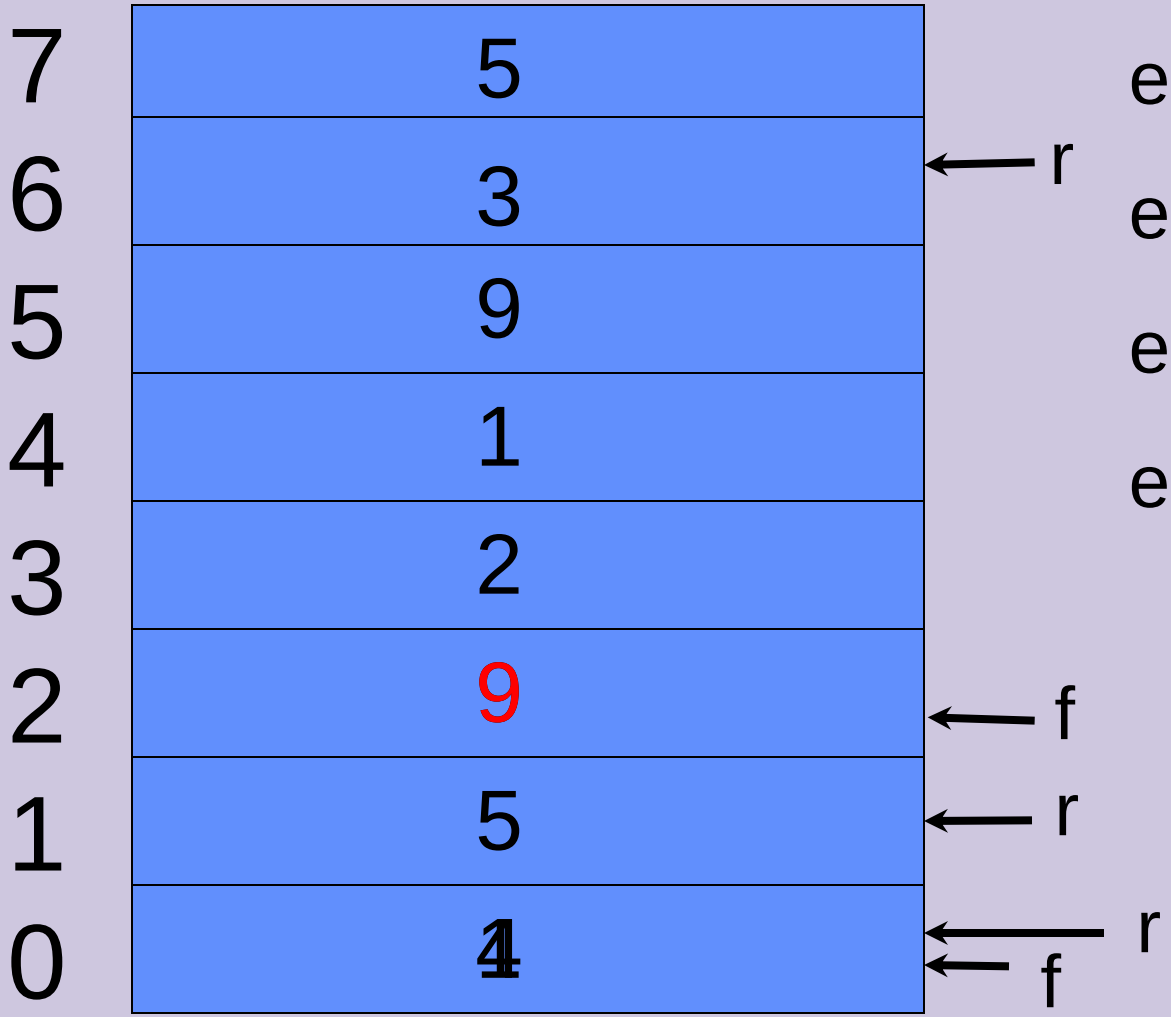
יש לבדוק "לא מלא" לפני enqueue

ו"לא ריק" לפני dequeue.



n=9

Creat(Q)



enqueue(Q,4)

enqueue(Q,5)

enqueue(Q,9)

enqueue(Q,2)

enqueue(Q,1)

enqueue(Q,9)

dequeue(Q)

dequeue(Q)

head(Q)?

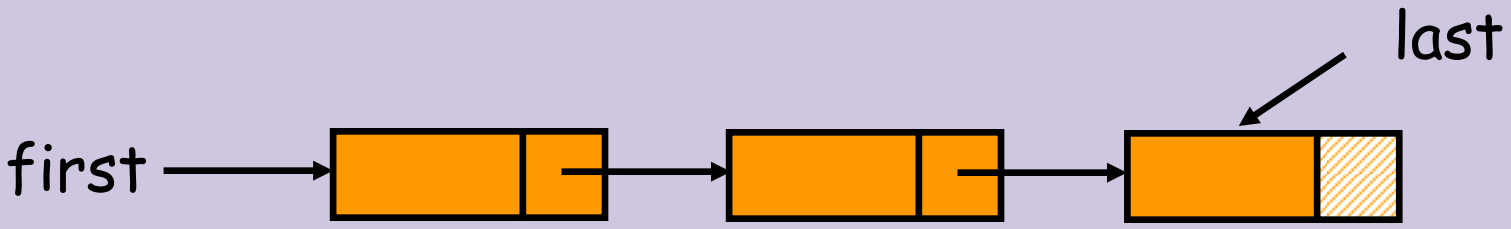
enqueue(Q,3)

enqueue(Q,5)

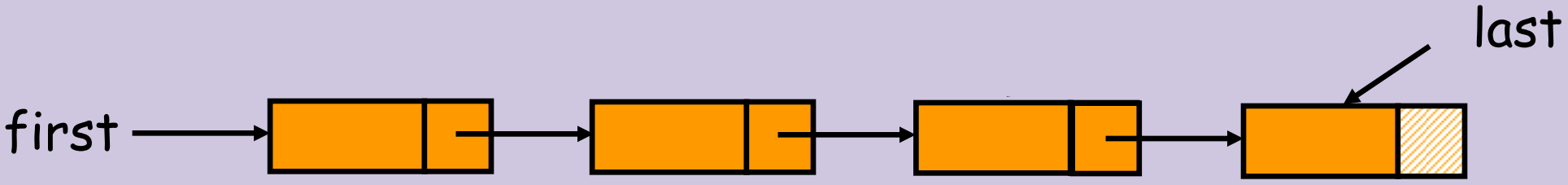
enqueue(Q,1)

©cs,Technion

מימוש תור ע"י רשימה מקושרת



הכנסה בסוף התור (last) :



הוצאה מראש התור (first).

שימו לב שלא ניתן להוציא איבר באמצעות המצביע last.

©cs,Technion

הערות לגבי ADT ומימושים

Abstract Data Type =ADT

המשתמש ב-ADT:

מכיר את הפעולות והשפעתן על הנתונים.

אינו נדרש להכיר את המימוש.

איכות המימוש מאופינת ע"י:

- **יעילות:**
  - זמן – מספר צעדי החישוב הנדרשים, לכל פעולה.
  - מקום – כמות הזיכרון הנדרשת.
- **פשטות התוכנית:** (מאפשר מימוש נכון ואחזקה יעילה).

©cs,Technion

זמן ריצה של אלגוריתם

זמן ריצה של אלגוריתם *A* עבור קלט x יסומן ב- $T_A(x)$.

זמן הריצה נמדד ע"י מספר פקודות מכונה שהאלגוריתם מבצע על קלט נתון.

מדד זה מתעלם מהבדלי המהירות בין הפקודות. (למשל, חבור לעומת כפל).

הגודל של קלט x יסומן ב- $|x|$.

לדוגמא: בתוכנית הַמְסַכֶּמֶת איברי מערך x, גודל הקלט הוא מספר איברי המערך.

זמן הריצה הגרוע ביותר (worst case) של אלגוריתם *A*

עבור קלט שגודלו n מוגדר ע"י

$$T_A(n) = \max_{|x|=n} T_A(x)$$

דוגמא 1:

נתון מערך a[]

A:
```
sum = 0
for (i = 0; i < n; i++)
sum = sum + a[i];
```

זמן הריצה של אלגוריתם זה עבור כל קלט n הוא $T_A(n) = c_1 \cdot n + c_2$

כאשר c_1, c_2 הם קבועים התלויים במימוש הפקודות בשפת מכונה.

©cs,Technion

זמן ריצה של אלגוריתם

דוגמא 2:

```
sum = 0
for (i = 0; i < n; i++)
        if sum < a[i+1]
                sum = sum + a[i];
        else
                terminate;
```

מהו זמן הריצה של האלגוריתם?

זמן הריצה **הגרוע ביותר** של אלגוריתם זה מקיים: $T_A(n) = c_1 \cdot n + c_2$

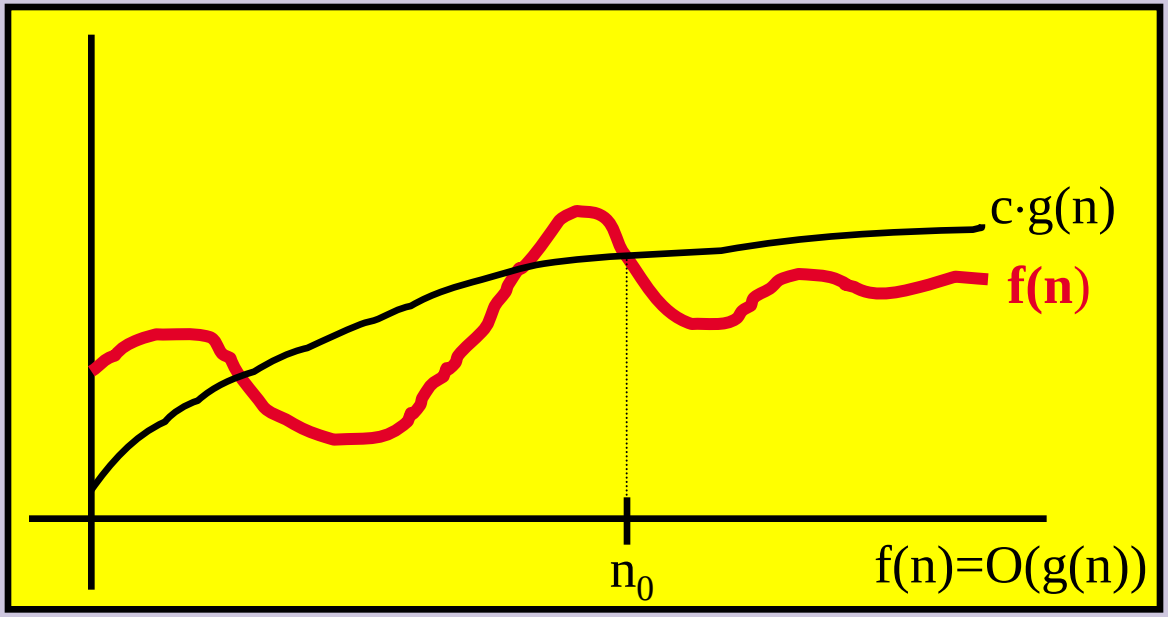
©cs,Technion

סיבוכיות והסימון O

הגדרה: יהיו $f(n), g(n)$ פונקציות חיוביות. נאמר שהפונקציה $f(n)$ נמצאת בקבוצת הפונקציות $O(g(n))$ אם קיימים קבועים c, n_0 כך שלכל $n \geq n_0$ מתקיים :

$$f(n) \leq c \cdot g(n)$$

כמו כן נאמר ש- $g(n)$ מהווה חסם עליון אסימפטוטי לפונקציה $f(n)$ ונסמן זאת ע"י $f(n) = O(g(n))$ במקום הסימון הרגיל $f(n) \in O(g(n))$.



נשתמש בסימון זה כאשר הפונקציה $f(n)$ היא פונקציה שקשה לתאר במדויק, למשל זמן הריצה של אלגוריתם, בעוד $g(n)$ פשוטה יותר לתיאור.

דוגמאות פולינומיאליות

טענה: עבור k קבוע מתקיים $f(n) = a_0 + a_1 n + a_2 n^2 + \ldots + a_k n^k = O(n^k)$

הוכחה: נמצא קבועים c, n_0 כך כך שלכל $n_0 \le n$ יתקיים $f(n) \le c \cdot n^k$.

$$f(n) \le |a_0| + |a_1| n + |a_2| n^2 + \ldots + |a_k| n^k$$

$$\le |a_0| n^k + |a_1| n^k + |a_2| n^k + \ldots + |a_k| n^k$$

$$= (|a_0| + |a_1| + |a_2| + \ldots + |a_k|) n^k = c n^k$$

דוגמאות: קבוע $f(n) = 10006 = O(1)$

ליניארי $f(n) = 4n + 27 = O(n)$

ריבועי $f(n) = n \log_2 n + 3n^2 = O(n^2)$

הדוגמא האחרונה נובעת מאי השוויון $\log_2 n < n$.

©cs,Technion

דוגמאות נוספות

לוגריתמי

$f(n) = \log_a n = O(\log_b n) = \boxed{O(\log n)}$

מכיוון שמתקיים $\log_a n = \log_a b \cdot \log_b n$.

לכל ε חיובי (שברים ושלמים) מתקיים: $f(n) = \log_a n = O(n^{\varepsilon})$

בין ליניארי וריבועי

$f(n) = 8 + 15\,n + 9\,n \log_2 n = O(n \log n)$

כיוון שמתקיים $f(n) < 32\,n \log_2 n$.

אקספוננציאלי

$f(n) = n^k + a^n = O(a^n) \qquad (a > 1)$

$f(n) = a^n = O(b^n) \qquad (b \geq a)$

©cs,Technion

דוגמאות

האם מתקיים ?

$f(n) = 3^n = O(2^n)$

$f(n) = 2^n = O(n^k)$ (k קבוע)

$f(n) = \log n = O(\log \log n)$

התשובה שלילית בשלושת המקרים. הוכיחו!

©cs,Technion

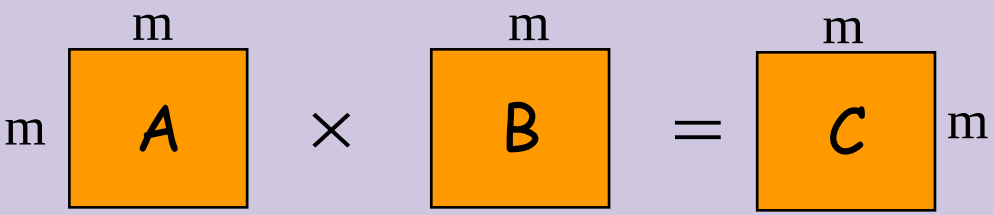
סיבוכיות זמן

דוגמא ראשונה: סכום איברי מערך. ראינו שמתקיים $T(n) < c\cdot n + 1$

ולפיכך $T(n) = O(n)$.

דוגמא שניה: כפל מטריצות ריבועיות בגודל $m\cdot m$.

גודל הקלט $n = 2m^2$.



מחשבים m^2 איברים במטריצת התוצאה C. כאשר כל איבר מחושב ע"י:

$$C[i,j] = \sum_{k=1}^{m} A[i,k]\cdot B[k,j]$$

סיבוכיות הזמן כפונקציה של m היא $O(m^3)$.

סיבוכיות הזמן כפונקציה של גודל הקלט n היא $O(n^{3/2})$ כיוון שמתקיים:

$$T(n) = O(m^3) = O\left(\frac{n^{3/2}}{2^{3/2}}\right) = O(n^{3/2})$$

$$\boxed{m = \sqrt{n/2}}$$

קבוע

סיבוכיות זמן (המשך)

דוגמא שלישית: חיפוש בינרי של x במערך ממוין בן n איברים.

בכל צעד משווים את x לאיבר האמצעי במערך העַכְשָׁוִי.

- אם האיבר האמצעי שווה ל- x החיפוש נגמר.
- אם האיבר האמצעי גדול מ- x ממשיכים עם חלק המערך המכיל את המספרים הקטנים.
- אם האיבר האמצעי קטן מ- x ממשיכים עם חלק המערך המכיל את המספרים הגדולים.

נסמן ב-T את סיבוכיות הזמן. מתקיימת משוואת הנסיגה הבאה:

$$T(1) = c_1 \qquad T(n) = c + T(\lceil n/2 \rceil)$$

בהנחה ש- n חזקה של 2 נקבל:

$$T(n) = c + c + T(n/4) = c \overbrace{+ c + \ldots +}^{i} c + T(n/2^i)$$
$$= c + c + \ldots + c + T(n/2^{\log_2 n}) \quad = c\log_2 n + T(1) = O(\log n)$$

הטענה נכונה גם אם n לא חזקה של 2 . פרטים על שיטות פתרון למשוואות נסיגה ניתן למצוא בספר הלימוד (פרק 4): Cormen, Leiserson, Rivest, Introduction to Algorithms.

סיבוכיות זמן (המשך)

דוגמא רביעית:

```
S = 0;
for ( i = 1;  i < n;  i ++)
        for ( j = 0;  j< n;  j + = i )
                S++ ;
```

אנליזה גסה: שתי לולאות מקוננות. $T(n) \le n\,(n-1)$.

אנליזה עדינה:

כאשר $i = 1$,	מתבצעות n פעולות.
$i = 2$,	$\lceil n/2 \rceil$
$i = 3$,	$\lceil n/3 \rceil$
•••	
$i = n$,	$n/n = 1$

סה"כ $$T(n) \le n + \frac{n}{2} + \frac{n}{3} + \ldots + \frac{n}{n} = n\left(1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n}\right) = nH_n$$

H_n נקרא המספר ההרמוני ה-n

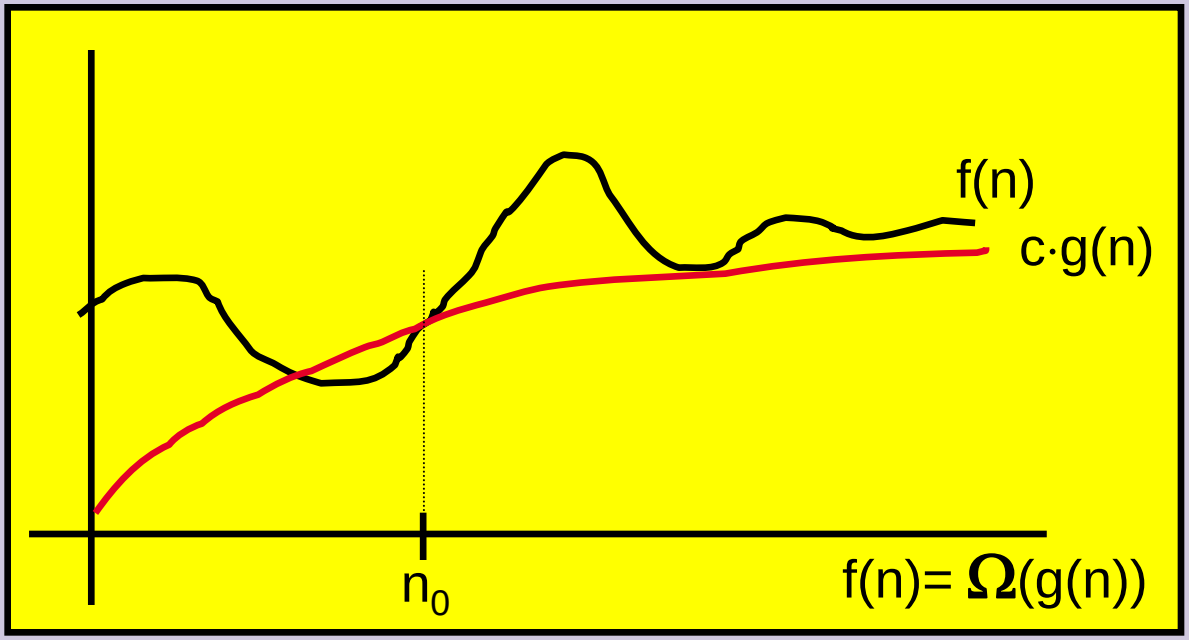
כיוון שמתקיים: $$\ln(n+1) = \int_1^{n+1} \frac{dx}{x} \le 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n} \le 1 + \int_1^{n} \frac{dx}{x} = 1 + \ln(n)$$

הערה: לשם דיוק היה עלינו לכפול את כל המשוואות בקבוע ולפיכך: $T(n) = O(n \cdot \log n)$

חסם תחתון אסימפטוטי

הגדרה: יהיו $f(n), g(n)$ פונקציות חיוביות. נאמר שהפונקציה $f(n)$ נמצאת בקבוצת הפונקציות $\Omega(g(n))$ (אומגה) אם קיימים קבועים c, n_0 כך שלכל $n \geq n_0$ מתקיים :

$$f(n) \geq c \cdot g(n)$$



דוגמאות פולינומיאליות

טענה: אם $a_k > 0$ עבור k קבוע מתקיים

$$f(n) = a_0 + a_1 n + a_2 n^2 + \ldots + a_k n^k = \Omega(n^k)$$

הוכחה: נמצא קבועים c, n_0 כך כך שלכל $n \geq n_0$ יתקיים $c \cdot n^k \leq f(n)$.

$$f(n) \geq -|a_0| - |a_1| n - |a_2| n^2 - \ldots - |a_{k-1}| n^{k-1} + a_k n^k$$

$$\geq -|a_0| n^{k-1} - |a_1| n^{k-1} - |a_2| n^{k-1} - \ldots - |a_{k-1}| n^{k-1} + a_k n^k$$

$$= (-(|a_0| + |a_1| + |a_2| + \ldots + |a_{k-1}|) + a_k n) n^{k-1}$$

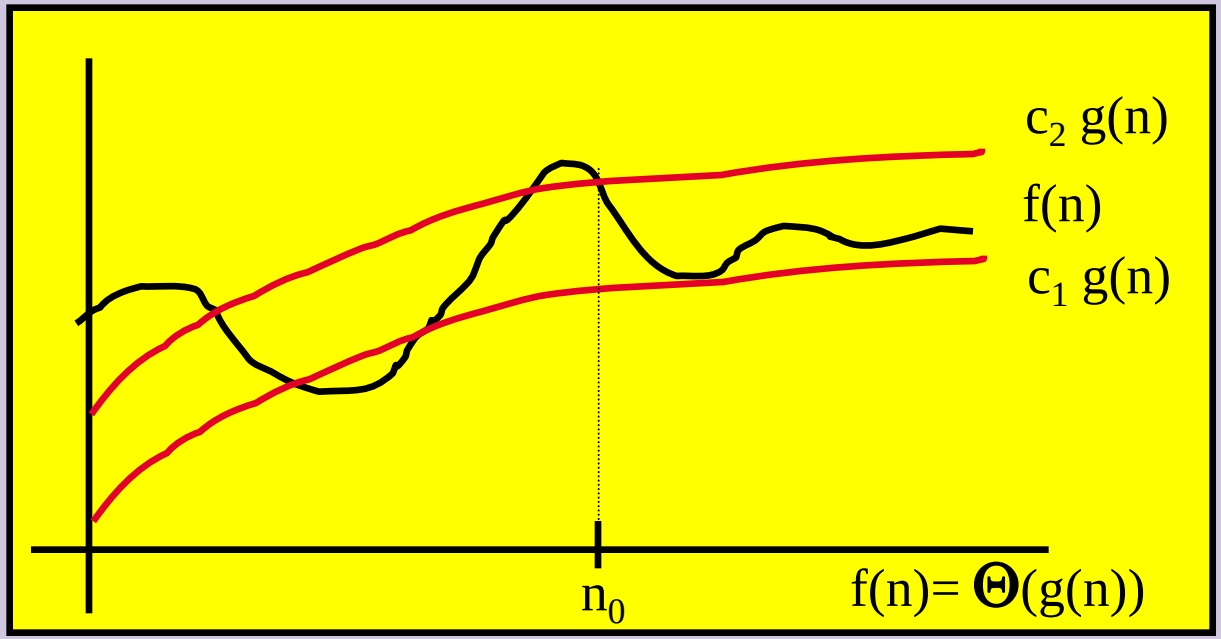
עבור $n \geq \boxed{n_0 = \frac{2(|a_0| + |a_1| + |a_2| + \ldots + |a_{k-1}|)}{a_k}}$

$$\geq \frac{a_k n}{2} n^{k-1} = c n^k$$

$$\boxed{c = \frac{a_k}{2}}$$

חסם הדוק אסימפטוטי

הגדרה: יהיו $f(n), g(n)$ פונקציות חיוביות. נאמר שמתקיים $f(n)=\Theta(g(n))$ (תטה) אם $f(n)=O(g(n))$ וגם $f(n)=\Omega(g(n))$.



הגדרה (אקויולנטית): נאמר שמתקיים $f(n)=\Theta(g(n))$ אם קיימים קבועים n_0,c_1,c_2 כך שלכל $n \geq n_0$ מתקיים $c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$.

דוגמא

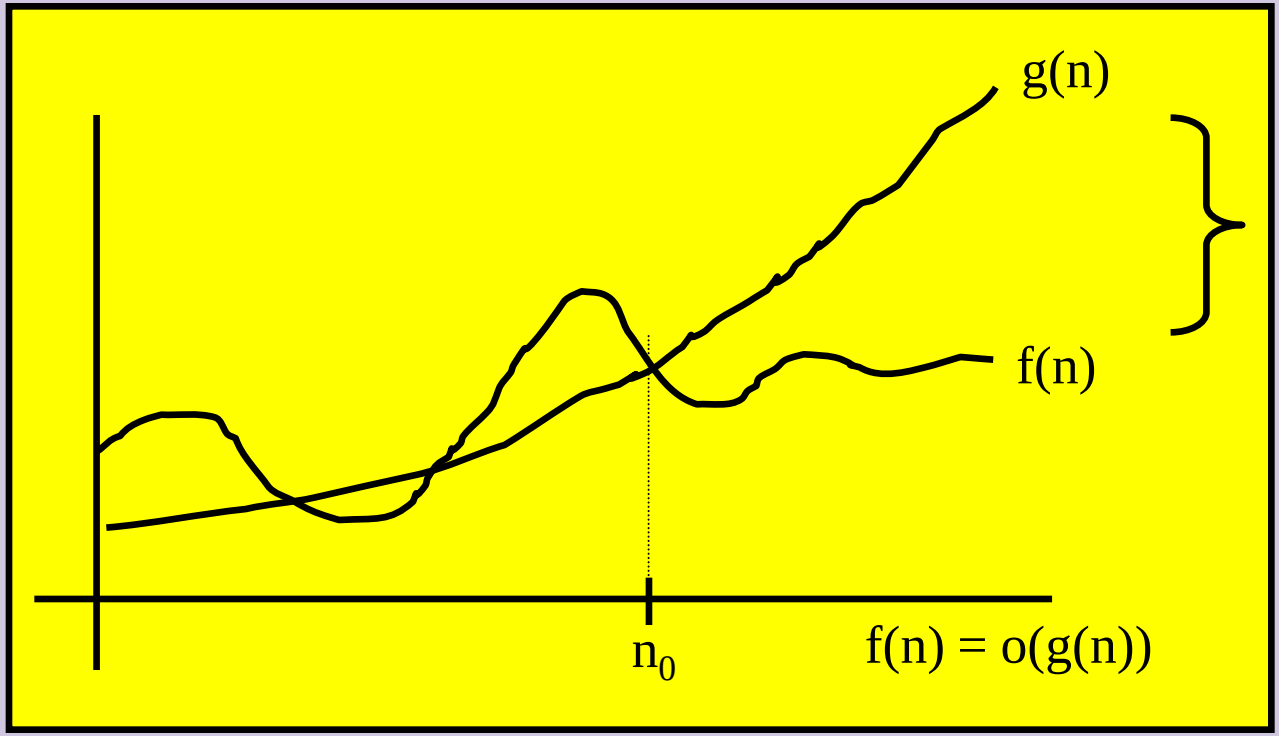
$$f(n) = a_0 + a_1 n + a_2 n^2 + \ldots + a_k n^k = \Theta(n^k)$$

©cs,Technion

הסימון o קטן

הגדרה: יהיו $f(n), g(n)$ פונקציות חיוביות. נאמר שהפונקציה $f(n)$ נמצאת בקבוצת הפונקציות $o(g(n))$ אם לכל קבוע c קיים קבוע n_0 כך שלכל $n \geq n_0$ מתקיים :

$$f(n) \leq c \cdot g(n)$$



$f(n)$ זניחה אסימפטוטית יחסית לפונקציה $g(n)$.

הגדרה (אקויולנטית): נאמר שמתקיים $f(n)=o(g(n))$ אם $\lim_{n\to\infty} f(n)/g(n) = 0$

דוגמאות: $n-100 \neq o(n)$, $\log n = o(n)$

©cs,Technion

מגבלות הסימון האסימפטוטי

נוח להשתמש בסימונים אסימפטוטיים מפני

1. הסימון מתעלם מקבועים
2. מאפשר ניתוח זמנים פשוט יותר.

אנו נשתמש בסימונים אלה לאורך הקורס.

אבל לסימון יש מגבלות מסוכנות...

ברור שנעדיף תוכנית הרצה בזמן $T(n) = n^2$

על פני תוכנית הרצה בזמן קבוע של $T(n) = 10^{80}$

כיון שבתוכניות ממשיות אנו משתמשים בגודל קלט n סופי הקטן באופן משמעותי מ- 10^{40}.

מסקנה: צריך לוודא שהקבועים n_0, c המתחבאים בהגדרות אסימפטוטיות O, Θ, Ω אמנם "סבירים".

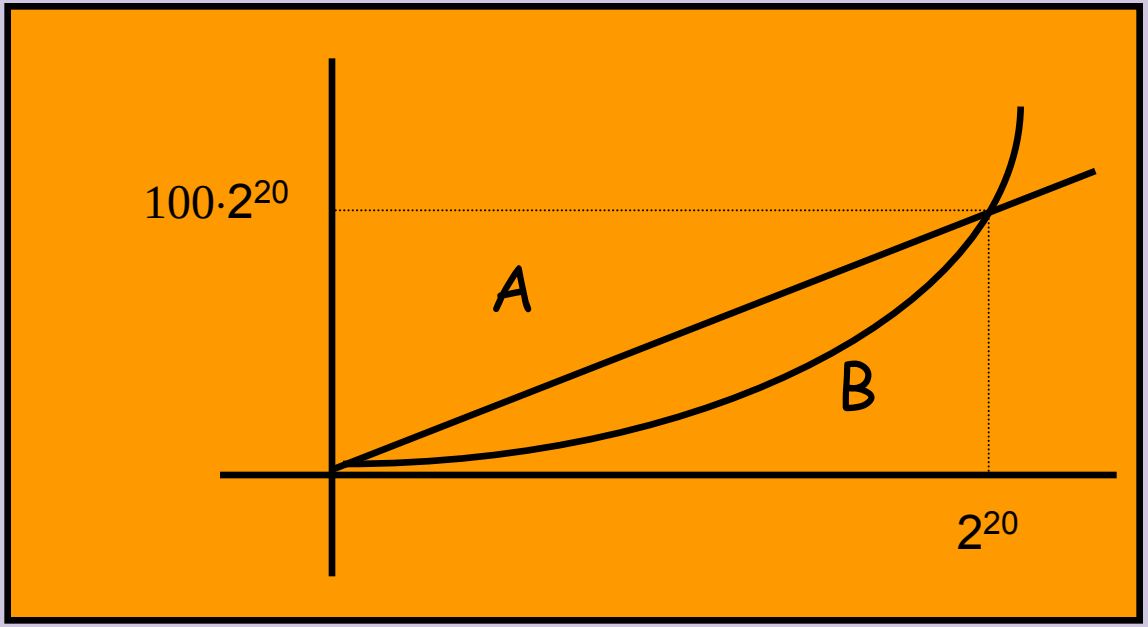
דוגמא.

נניח שאלגוריתם A רץ בזמן $T_A(n)= 100\ n$

ואלגוריתם B רץ בזמן $T_B(n)= 5n\log_2 n$.

בניתוח אסימפטוטי עדיף אלגוריתם A כיון שהאלגוריתם ליניארי,

אבל עבור קלטים המקיימים $n < 2^{20}$ עדיף אלגוריתם B.



©cs,Technion

שאלה מבוחן 1993

נתונה הפונקציה הבאה ב-C - נתחו את הסיבוכיות שלה.

```
Void func (int n)
{
        float x, delta;
        int i;
        for (i = 1 ; i <= n ; i++) {
                delta = 1.0 / i;
                x = i;
                while ( x > 0 ){
                        x = x - delta;
                }
        }
}
main()
{
        int n;
        scanf("%d",&n);
        func(n)
}
```

$$i = 1,2,3,\cdots,n$$

$$delta = 1/i$$

$$x = i, i-\frac{1}{i}, i-\frac{2}{i}, i-\frac{3}{i}, \cdots, 0$$

$$i^2+1$$

$$\sum_{i=1}^{n} i^2+1$$

$$\sum_{i=1}^{n} i^2+1 \le n\cdot n^2+n = n^3+n$$

$$\sum_{i=1}^{n} i^2+1 \ge \sum_{i=n/2}^{n} i^2 \ge \frac{n}{2}\cdot\left(\frac{n}{2}\right)^2 = \frac{n^3}{8}$$

$$\sum_{i=1}^{n} i^2+1 = \theta(n^3)$$