זמן ריצה ונוסחאות נסיגה

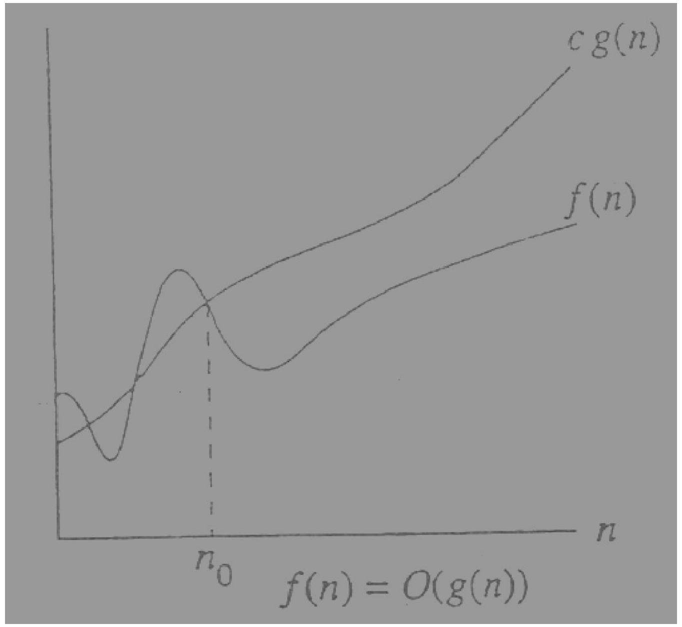
זמן ריצה של פונקציה

- סיבוכיות של אלגוריתם: נתון אלגוריתם A לפתרון הבעיה P. הסיבוכיות של A תוגדר כמספר הצעדים המתבצעים על ידי A עבור המקרה הגרוע ביותר של P.
- מתארים זמן ריצה של אלגוריתם כפונקציה של גודל הקלט
- גודל הקלט: מספר הפריטים בקלט

זמן ריצה של פונקציה

- המקרה הגרוע ביותר: זמן הריצה הארוך ביותר על איזשהו קלט n
- המקרה הממוצע: זמן הריצה הצפוי בממוצע על הקלט
- המקרה הגרוע ביותר מהווה חסם עליון על זמן הריצה עבור כל קלט אפשרי (מסומן ב O)

זמן ריצה של פונקציה
סימונים אסימפטוטיים



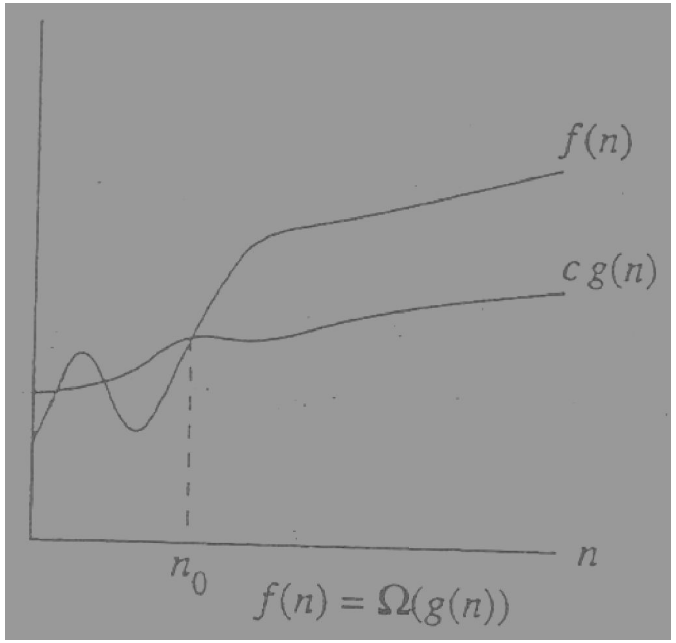
- הסימון O: חסם אסימפטוטי עליון

$O(g(n)) = \{ f(n) :$ קיימים קבועים חיוביים c ו־n_0
כך ש־ $0 \le f(n) \le cg(n)$ לכל $n \ge n_0$.$\}$

- הסימון O מהווה חסם עליון, כלומר f(n) במקרה הגרוע ביותר, לא תעבור את g(n)

זמן ריצה של פונקציה

סימונים אסימפטוטיים



- הסימון Ω: חסם אסימפטוטי תחתון

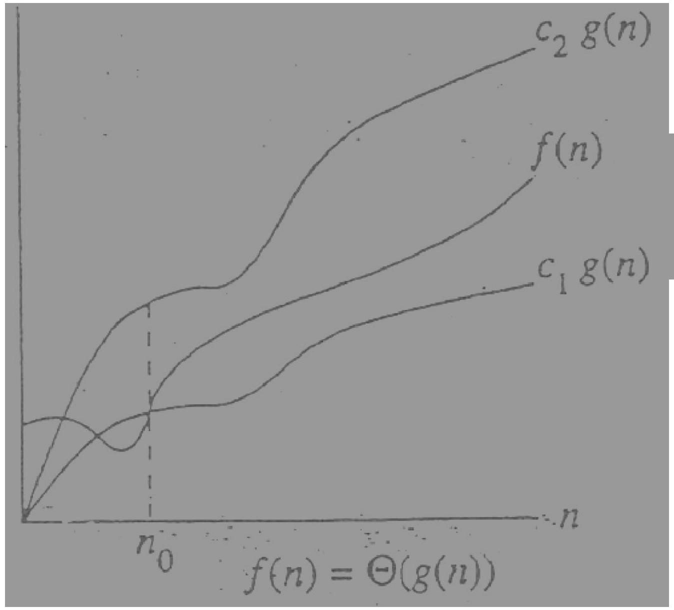
$\Omega(g(n)) = \{ f(n) :$ קיימים קבועים חיוביים c ו-n_0 כך ש- $0 \le cg(n) \le f(n)$ לכל $n \ge n_0$.$\}$

- הסימון Ω מהווה חסם תחתון, כלומר, f(n) לעולם לא תרוץ בזמן טוב יותר מאשר g(n)

- למשל, ישנו חסם תחתון של $\Omega(n \times \log_2 n)$ עבור כל מיון נתונים בגודל n (כאשר אין מידע נוסף על הקלט)

זמן ריצה של פונקציה

סימונים אסימפטוטיים



- הסימון Θ: חסם אסימפטוטי הדוק

$\Theta(g(n)) = \{ f(n) :$ קיימים קבועים חיוביים c_1, c_2 ו־n_0 כך ש־ $0 \le c_1 g(n) \le f(n) \le c_2 g(n)$ לכל $n \ge n_0$. $\}$

- כלומר, $f(n) = \Theta(g(n))$ אם ורק אם $f(n) = O(g(n))$ וגם $f(n) = \Omega(g(n))$

זמן ריצה של פונקציה

$$O(1)=O(10^9)=O(c) < O(\lg n)=O(\ln n)=O(\log_a n) < O(\sqrt{n}) < O(n)$$

סדר גודל קבוע   סדר גודל לוגריתמי   סדר גודל פולנומיאלי

$$O(n)=O(1000 * n)<O(n \lg n)<O(n^2)<O(n^3)$$

סדר גודל פולנומיאלי

$$< O(2^n)<O(3^n)<O(n^{\lg n})<O(n!)<O(n^n)$$

סדר גודל מעריכי

זמן ריצה של פונקציה

סדרי-גודל										גודל הקלט
n^n	n!	2^n	n^3	n^2	$\lg n$ n	$\sqrt{n}$	$\lg n$	2n	n	
1	1	2	1	1	0	1	0	2	1	1
4	2	4	8	4	2	1.41	1	4	2	2
256	24	16	64	16	8	2	2	8	4	4
16777216	40320	256	512	64	24	2.83	3	16	8	8
1.84E+19	2.09E+13	65536	4096	256	64	4	4	32	16	16
1.46E+48	2.63E+35	4.29E+9	32768	1024	160	5.66	5	64	32	32
לא סביר	לא סביר	לא סביר	262144	4096	384	8	6	128	64	64
לא סביר	לא סביר	לא סביר	2097152	16384	896	11.31	7	256	128	128

זמן ריצה

קבעו האם $f(n) = O(g(n))$, או $f(n) = \Omega(g(n))$, או $f(n) = \Theta(g(n))$. הוכיחו את תשובתכם.

- $g(n) = 5n\log_3 n^5$ $\qquad$ $f(n) = n\log_2 n + 5n$

זמן ריצה

1
2+1
3+2+1
4+3+2+1
n+(n-1) + ... + 1

- מהו זמן הריצה של האלגוריתמים הבאים:

.1

```
for i←1 to n
    for j←1 to i
        for k ← j to i
            print(i+j+k)
```

לולאה שלישית	i	
1	1	כל שורה הסכום שלה: $\sum_{k=1}^{i} k = (1+i) * \frac{i}{2} = \frac{i^2+i}{2}$
2+1	2	
3+2+1	3	
4+3+2+1	4	
n+(n-1) + ... + 1	n	

סכימת כל השורות:

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

מאחר ומדברים על סדרי גודל, ניתן להשמיט מהמשוואה (מהסכום) ערכים פחות משמעותיים

$$\sum_{i=1}^{n} i^2 = 1 + 4 + 9 + 16 + \cdots + n^2$$

הפונקציה שלנו: $f(n) = 1 + 4 + 9 + 16 + \cdots + n^2$

יש בסכום n אברים.

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

$$1 + 4 + 9 + 16 + \cdots + n^2 < n^2 + n^2 + n^2 + \cdots + n^2 = n * n^2 = c * n^3$$

C=1, n>1

זמן ריצה

- מהו זמן הריצה של האלגוריתמים הבאים:

.2

```
for i=1 to n
{
        for j = 1 to i
        {
                for k = 1 to j
                        print k
                k = 2
                while k < i
                        k = k^2
}       }
```

$$2^{2^x} = i$$

$$\log_2(2^{2^x}) = \log_2 i$$

$$2^x * \log_2(2) = \log_2 i$$

$$2^x = \log_2 i$$

$$\log_2 2^x = \log_2 \log_2 i$$

$$x = \log_2 \log_2 i$$

זמן בסך הכל:

$$\sum_{i=1}^{n} \log_2 \log_2 i$$

$$n^3 + \sum_{i=1}^{n} \log_2 \log_2 i$$

$$\log_2 \log_2 1 + \log_2 \log_2 2 + \log_2 \log_2 3 + \cdots + \log_2 \log_2 n$$
$$\leq \log_2 \log_2 n + \log_2 \log_2 n + \log_2 \log_2 n + \cdots + \log_2 \log_2 n$$
$$= n * \log_2 \log_2 n < n^2 < n^3$$

$$O(n^3)$$

זמן ריצה

- מהי הסיבוכיות של האלגוריתם הבא:

```
for i=1 to n
{
      for j = 1 to i
      {
            k = n
            while k > 2
                  k = ⌊k^(1/3)⌋
      }
}
```

$$n^{\frac{1}{3^x}} = 2$$

$$\log_2 n^{\frac{1}{3^x}} = \log_2 2$$

$$\frac{1}{3^x}\log_2 n = 1$$

$$\log_2 n = 3^x$$

$$\log_3 \log_2 n = \log_3 3^x$$

$$x = \log_3 \log_2 n$$

סדרה חשבונית:

$$\sum_{j=1}^{n} j = (1+n) * \frac{n}{2} = O(n^2)$$

מאחר והלולאה הפנימית ביותר היא עצמאית ולא תלויה בלולאות
החיצוניות, יש לבצע הכפלה בין שתי המשוואות.

בסך הכל:

$$O(n^2 * \log_3 \log_2 n)$$

זמן ריצה

- מהי הסיבוכיות של האלגוריתם הבא:

```
for(i = 2; i <= n; i++)
{
    x = 2;
    while(x < i)
    {
        x = x*2;
    }
    while(x > 2)
    {
        x = sqrt(x);
    }
}
```

$$2^p = i$$
$$p = \log_2 i$$

רק לפעם אחת כתלות ב i.

בסך הכל כולל הלולאה הראשונה:

$$\sum_{i=2}^{n} \log_2 i = \log 2 + \log 3 + \cdots + \log n = \log n!$$

$$\log n! < \log n^n = n * \log n$$

$$i^{\frac{1}{2^k}} = 2$$
$$\log_2 i^{\frac{1}{2^k}} = \log_2 2$$
$$\frac{1}{2^k}\log_2 i = 1$$
$$\log_2 i = 2^k$$
$$\log_2 \log_2 i = \log_2 2^k$$
$$k = \log_2 \log_2 i$$
$$\sum_{i=2}^{n} \log_2 \log_2 i$$

בסך הכל שתי הלולאות הפנימיות:

$$log\, n! + \sum_{i=2}^{n} \log_2 \log_2 i$$

מורידים את הפחות משמעותי ומקבלים:

$$\boldsymbol{\Theta}(\log n!)$$

לפעמים כותבים

$$O(n \log n)$$

נתון קטע הקוד הזה:

```
x=n ; y=0 ;
while(x>1)
{
    x=x-n/10 ;
    for(i=1;i<=x;i++)y++;
}
```

כמו־כן נתון כי כל המשתנים בקטע זה הם מטיפוס שלם.

מהי סיבוכיות זמן הריצה של קטע קוד זה כפונקציה של n ?

1. $O(n)$
2. $O(n^2)$
3. $O(n \log n)$
4. $O(1)$

זמן ריצה

ו. להלן הפונקציות האלה:

$$f(n) = (\log n)^n, \quad g(n) = 2^n, \quad h(n) = \Omega(\sqrt{n})$$

h(n) – כל הפונקציות שחסומות מלמטה על ידי שורש של n

בחר את ההיגד הנכון מבין ההיגדים שלהלן:

1. $h(n) = O(f(n))$
2. $g(n) = \Omega(f(n))$
3. $g(n) \cdot g(n) = O(f(n))$
4. $f(n) \cdot g(n) = \Theta(f(n))$

זמן ריצה של פונקציה

היחס $O(\ )$ מקיים את התכונות הבאות:

1. רפלקסיביות. לכל פונקציה $f: \mathrm{N} \rightarrow \mathrm{R}^+$ מתקיים $f(n)=O(f(n))$.
2. טרנזטיביות. אם $f(n)=O(g(n))$ ו-$g(n)=O(h(n))$ אז $f(n)=O(h(n))$.
3. אם $f(n)=O(g(n))$ אז לכל $c>0$ $f(n)=O(c*g(n))$.
4. אם $f_1(n)=O(g_1(n))$ ו-$f_2(n)=O(g_2(n))$ אז $(f_1+f_2)(n)=O(\max\{g_1(n),g_2(n)\})$.
5. אם $f_1(n)=O(g_1(n))$ ו- $f_2(n)=O(g_2(n))$ אז $f_1(n)*f_2(n)=O((g_1(n)*g_2(n))$.

חיפוש בינארי

בהינתן מערך ממויין, עלינו לקבל מספר X
ולבדוק האם הוא נמצא במערך
אם **כן** – להחזיר את המיקום (אינדקס) שלו במערך
אחרת – להחזיר -1

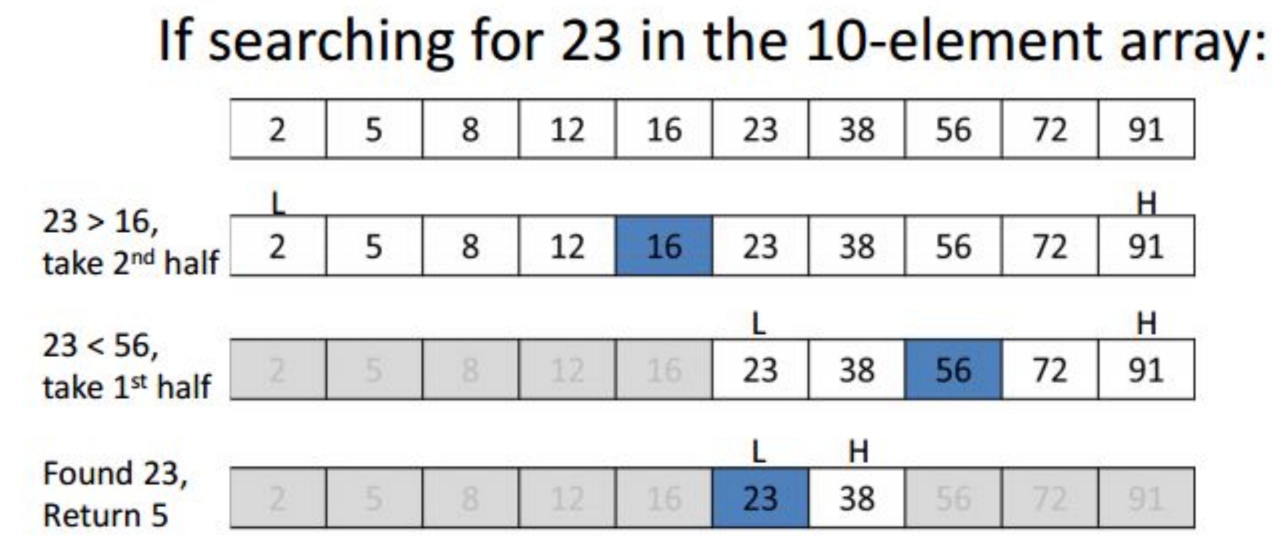
```c
int binarySearch(int arr[], int l, int r, int x)
{
  while (l <= r)
  {
    int m = l + (r-l)/2;

    // Check if x is present at mid
    if (arr[m] == x)
        return m;

    // If x greater, ignore left half
    if (arr[m] < x)
        l = m + 1;

    // If x is smaller, ignore right half
    else
         r = m - 1;
  }

  // if we reach here, then element was not present
  return -1;
}
```



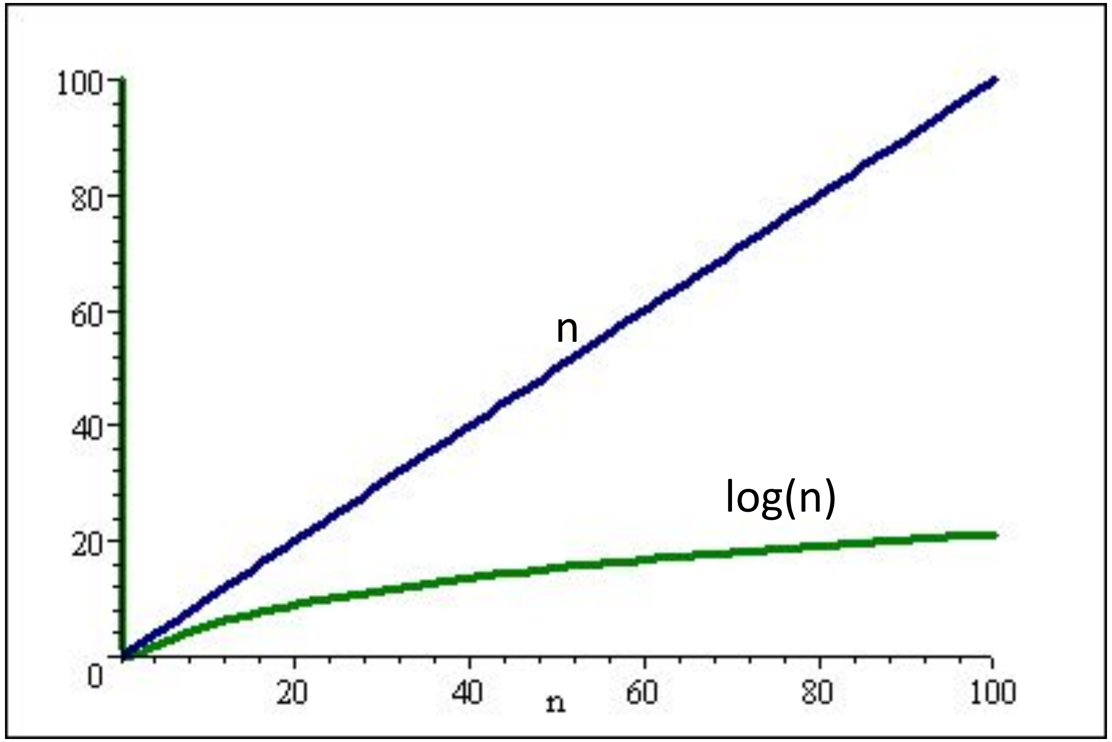
חיפוש בינארי

- זמן ריצה: בידינו מערך בגודל n איברים. בכל איטרציה של הלולאה, אנחנו מצמצמים את החיפוש למערך בגודל $\frac{n}{2}$ איברים
- לאחר איטרציה נוספת המערך יהיה בגודל $\frac{n}{4}$
- לאחר x איטרציות המערך יהיה בגודל $\frac{n}{2^x}$
- במקרה הגרוע, אנחנו נצמצם את המערך עד הסוף, כלומר $\frac{n}{2^x} = 1$

$$x = \log n$$

- ניתוח זמן הריצה – $O(\log n)$

חיפוש בינארי



מיזוג

- מיזוג מערכים ממויינים: נתונים שני מערכים ממוינים, עלינו למזג אותם למערך ממוין אחד
- הרעיון – להשוות תמיד בין שני איברים במערכים ואת הקטן יותר ניקח למערך הממויין

A	6	13	18	21

B	4	8	9	20

C	4	6	8	9	13	18	20	21

מיזוג

```
void mergeArrays(int arr1[], int arr2[], int n1,int n2, int arr3[])
{
   int i = 0, j = 0, k = 0;
   while (i<n1 && j <n2)
   {
        if (arr1[i] < arr2[j])
            arr3[k++] = arr1[i++];
        else
            arr3[k++] = arr2[j++];
   }

    while (i < n1)
        arr3[k++] = arr1[i++];

    while (j < n2)
        arr3[k++] = arr2[j++];
{
```

A	6	13	18	21

B	4	8	9	20

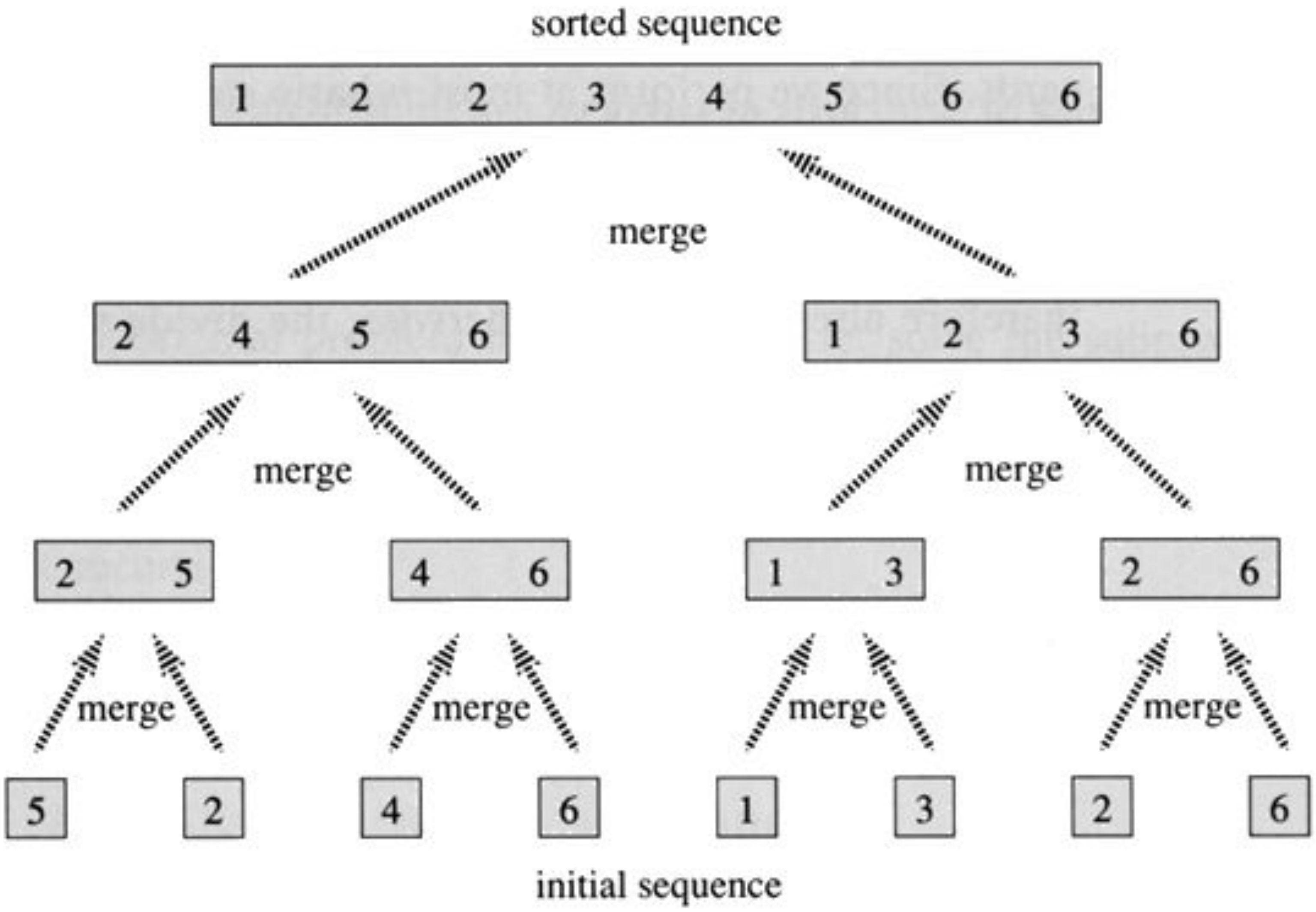
C	4	6	8	9	13	18	20	21

מיון מיזוג

- **המטרה: לקבל מערך לא ממוין ולמיין אותו בסדר עולה**

- מיון מיזוג: לפרק את המערך למערכים קטנים בגודל 1
- למזג כל זוג מערכים קטנים למערכים בגודל 2
- למזג כל זוג מערכים בגודל 2 למערכים בגודל 4
- ...
- להגיע למערך ממויין

מיון מיזוג



sorted sequence
1 2 2 3 4 5 6 6
merge
2 4 5 6
1 2 3 6
merge
merge
2 5
4 6
1 3
2 6
merge
merge
merge
merge
5
2
4
6
1
3
2
6
initial sequence

מיון מיזוג

```c
void merge(int a[],int i1,int j1,int i2,int j2)
{
    int temp[50];    //array used for merging
    int i,j,k;
    i=i1;    //beginning of the first list
    j=i2;    //beginning of the second list
    k=0;

    while(i<=j1 && j<=j2)    //while elements in both lists
    {
        if(a[i]<a[j])
            temp[k++]=a[i++];
        else
            temp[k++]=a[j++];
    }

    while(i<=j1)    //copy remaining elements of the first list
        temp[k++]=a[i++];

    while(j<=j2)    //copy remaining elements of the second list
        temp[k++]=a[j++];

    //Transfer elements from temp[] back to a[]
    for(i=i1,j=0;i<=j2;i++,j++)
        a[i]=temp[j];
}
```

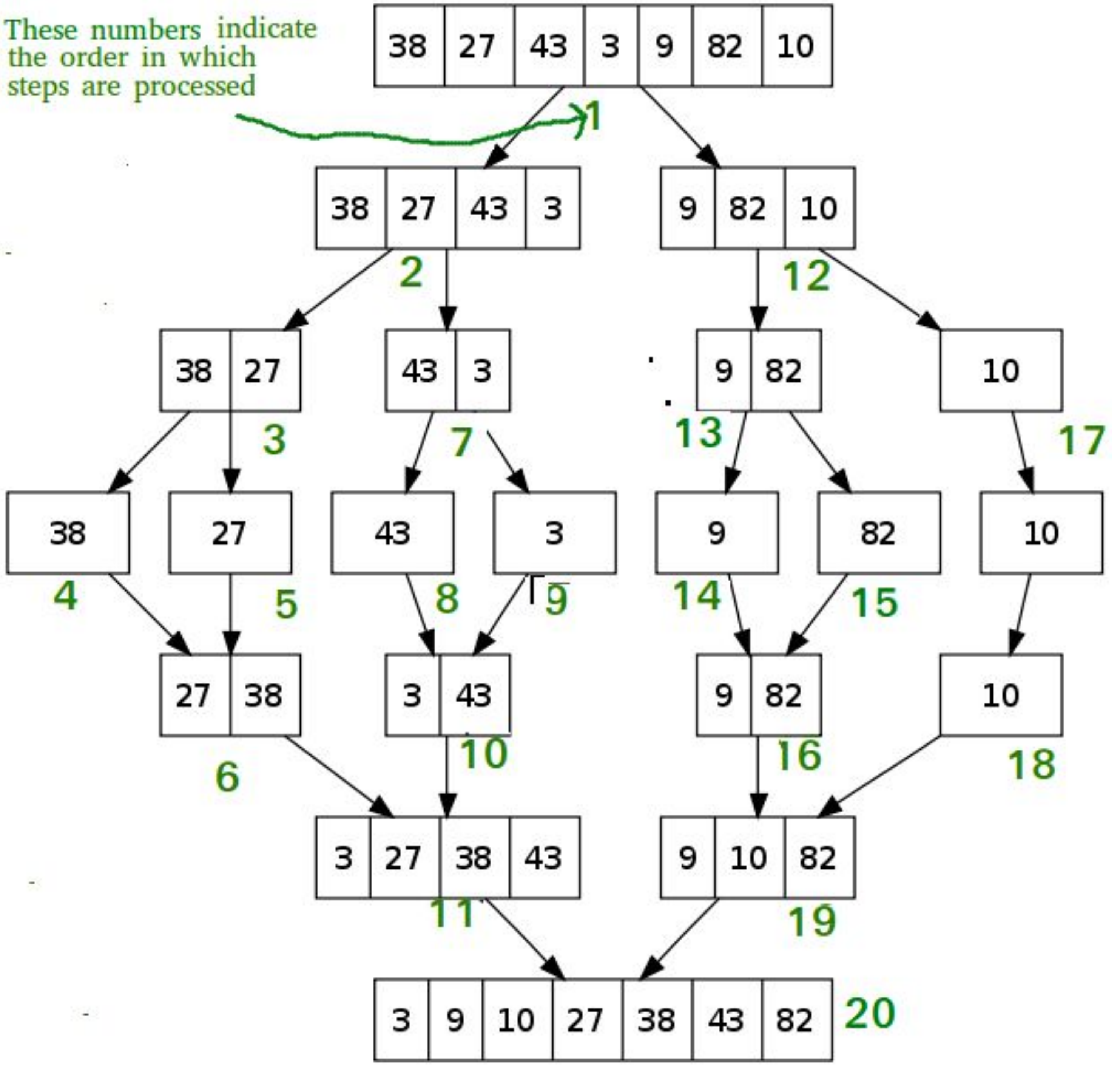
מיון מיזוג

```
void mergesort(int a[],int i,int j)
{
    int mid;

    if(i<j)
    {
        mid=(i+j)/2;
        mergesort(a,i,mid);          //left recursion
        mergesort(a,mid+1,j);     //right recursion
        merge(a,i,mid,mid+1,j);     //merging of two sorted sub-arrays
    }
}
```

```
void mergesort(int a[],int i,int j)
{
    int mid;

    if(i<j)
    {
        mid=(i+j)/2;
        mergesort(a,i,mid);
        mergesort(a,mid+1,j);    //
        merge(a,i,mid,mid+1,j);
    }
}
```



נוסחאות נסיגה

- רקורסיה – הפרד ומשול:
  1. חלק את הבעייה ל k תת בעיות זהות בגודל קטן יותר
  2. באופן רקורסיבי, פתור את k הבעיות הקטנות יותר
  3. חבר את k הפתרונות כדי לקבל את הפתרון של הבעייה הגדולה

- דוגמה: מיון מיזוג
  1. חלק את המערך לשני גדלים שווים
  2. באופן רקורסיבי, פתור את שני המערכים הקטנים באותה שיטה
  3. מזג את שני שני המערכים תוך שמירה על סדר המיון

נוסחאות נסיגה

- רקורסיה – הפרד ומשול:
  1. חלק את הבעייה ל k תת בעיות זהות בגודל קטן יותר
  2. באופן רקורסיבי, פתור את k הבעיות הקטנות יותר
  3. חבר את k הפתרונות כדי לקבל את הפתרון של הבעייה הגדולה
- נסמן ב T(n) את זמן הריצה על קלט בגודל n, ולכן
-

$$T(1) = c$$

$$T(n) = \sum_{i=1}^{k} T\left(\frac{n}{b_i}\right) + d \times f(n)$$

זמן ביצוע כל אחת מתת הבעיות הקטנות

עלות חיבור תת הבעיות הקטנות

נוסחאות נסיגה

- מיון מיזוג:
  1. חלק את המערך לשני גדלים שווים
  2. באופן רקורסיבי, פתור את שני המערכים הקטנים באותה שיטה
  3. מזג את שני שני המערכים תוך שמירה על סדר המיון

-

$$T(1) = 1$$

$$T(n) = \sum_{i=1}^{2} T\left(\frac{n}{2}\right) + n = 2T\left(\frac{n}{2}\right) + n$$

זמן ביצוע כל אחת מתת הבעיות הקטנות

עלות חיבור תת הבעיות הקטנות

נוסחאות נסיגה

- ישנן 4 שיטות עיקריות לפתירת נוסחאות רקורסיביות:
  - שיטת הניחוש ואינדוקציה
  - שיטת האיטרציה
  - משפט האב
  - החלפת משתנים

שיטת הניחוש ואינדוקציה

- מנחשים את צורת הפתרון, ומשתמשים באינדוקציה כדי למצוא את הקבועים, ולהוכיח את נכונות הפתרון
- דוגמא: נוכיח חסם עליון עבור הנוסחה:

$$T(n) = 2T(\lfloor n/2 \rfloor) + n$$

ניחוש: הפתרון הוא: $T(n) = O(n \lg n)$.

שיטת הניחוש ואינדוקציה

כאשר $T(1)=1$ אי אפשר לבחור c מספיק

גדול כך שיתקיים: $T(1) < c \cdot n \lg n$

שהרי: $1 = T(1) > c \cdot 1 \lg 1 = 0$

נתחיל מ- $n_0=2$.

מנוסחת הנסיגה: $T(2)=4 \;,\; T(3)=5$

נבחר: $c \geq 2$,

ונקבל:

$$T(2) \leq c \cdot 2 \cdot \lg 2$$

$$T(3) \leq c \cdot 3 \cdot \lg 3$$

מש"ל.

(הוכחנו: $T(n) = O(n \lg n)$

שיטת הניחוש ואינדוקציה

שלב ב. הוכחת האינדוקציה:

אם נכון עבור n אז נכון גם עבור 2n

מתקיים: $T(n) \le c_1 * n * \log n$

צריך להוכיח: $T(2n) \le c_2 * 2n * \log(2n)$

$$T(2n) = 2T(n) + 2n \le 2 * c_1 * n * \log(n) + 2n$$

צריך להוכיח:

$$2 * c_1 * n * \log(n) + 2n \le c_2 * 2n * \log(2n)$$

$$c_1 * n * \log(n) + n \le c_2 * n * \log(2n) \quad /n$$

$$c_1 * \log(n) + 1 \le c_2 * \log(2n) = c_2 \log 2 + c_2 \log n$$

$$c_1 * \log(n) + 1 \le c_2 + c_2 \log n$$

אם נבחר

C2>C1 אז מתקיים

וגם

C2>1

שיטת הניחוש ואינדוקציה

$$T(n) = 6T\left(\frac{n}{6}\right) + \frac{n}{\log_2 n}$$

ניחוש: הפתרון הוא: $T(n) = 6T\left(\frac{n}{6}\right) + \frac{n}{\log_2 n} = O(n \times \log n)$

עבור n=6 מתקיים:

$$T(6) = 6T\left(\frac{6}{6}\right) + \frac{6}{\log_2 6} = 6T(1) + 2.32 = 8.32 \le c * 6\log 6 \ \rightarrow 8.32 \le 15.5c$$

עבור c=1.

נניח שמתקיים עבור $\frac{n}{6}$ כלומר: $T\left(\frac{n}{6}\right) \le c \times \frac{n}{6} \times \log_2 \frac{n}{6}$ מטעמי נוחות, כשנכתוב log נתכוון לבסיס 2

לכן:

$$\mathrm{T(n)} = 6\mathrm{T}\left(\frac{\mathrm{n}}{6}\right) + \frac{\mathrm{n}}{\log_2 \mathrm{n}} \le 6\mathrm{c} \times \frac{\mathrm{n}}{6} \times \log_2 \frac{\mathrm{n}}{6} + \frac{\mathrm{n}}{\log_2 \mathrm{n}}$$

$$6c \times \frac{n}{6} \times \log_2 \frac{n}{6} + \frac{n}{\log_2 n} = c \times n(\log_2 n - \log_2 6) + \frac{n}{\log_2 n} \le c \times n(\log_2 n - \log_2 6) + n$$

$$= c \times n \log_2 n - c \times n \log_2 6 + n = c \times n \log_2 n + n\underbrace{(1 - c \times \log_2 6)}_{<0} \le$$

$$\le c \times n \log_2 n = O(n \log_2 n)$$

שיטת האיטרציה

- מפתחים את הנוסחה עד שמגיעים לסכום של איברים התלויים רק ב n ובתנאי ההתחלה.
- לבסוף חוסמים את הפתרון באמצעות סכומים
- דוגמא: $T(n) = 3T\left(\frac{n}{4}\right) + n$

$$* T\left(\frac{n}{4}\right) = 3T\left(\frac{\frac{n}{4}}{4}\right) + \frac{n}{4} = 3T\left(\frac{n}{16}\right) + \frac{n}{4}$$

$$** T\left(\frac{n}{16}\right) = 3T\left(\frac{\frac{n}{16}}{4}\right) + \frac{n}{16} = 3T\left(\frac{n}{4^3}\right) + \frac{n}{4^2}$$

$$T(n) = 3T\left(\frac{n}{4}\right) + n = 3\left[3T\left(\frac{n}{16}\right) + \frac{n}{4}\right] + n = 9T\left(\frac{n}{16}\right) + \frac{3n}{4} + n = 9\left[3T\left(\frac{n}{4^3}\right) + \frac{n}{4^2}\right] + \frac{3n}{4} + n =$$

$$3^3T\left(\frac{n}{4^3}\right) + \frac{3^2n}{4^2} + \frac{3n}{4} + n = 3^4T\left(\frac{n}{4^4}\right) + \frac{3^3n}{4^3} + \frac{3^2n}{4^2} + \frac{3n}{4} + n = \cdots =$$

- באופן כללי יותר:

$$n + \frac{3}{4}n + \frac{3^2}{4^2}n + \frac{3^3}{4^3}n + \frac{3^4}{4^4}n + \cdots + \frac{3^x}{4^x}n = n\left(\frac{3}{4} + \frac{3^2}{4^2} + \frac{3^3}{4^3} + \cdots \frac{3^x}{4^x}\right)$$

טור גיאומטרי יורד

$$T(n) = n + \frac{3n}{4} + \frac{9n}{16} + \frac{27n}{64} + \cdots = n\left(1 + \frac{3}{4} + \frac{9}{16} + \frac{27}{64} + \ \ldots\right) \le n\sum_{i=0}^{\infty}\left(\frac{3}{4}\right)^i = n \times \frac{1}{1-\frac{3}{4}} = 4n = O(n)$$

שיטת האיטרציה

- מפתחים את הנוסחה עד שמגיעים לסכום של איברים התלויים רק ב n ובתנאי ההתחלה.
- לבסוף חוסמים את הפתרון באמצעות סכומים
- דוגמא: $T(n) = 9T\left(\frac{n}{3}\right) + n$

$$* \ T\left(\frac{n}{3}\right) = 9T\left(\frac{n}{9}\right) + \frac{n}{3}$$

$$** \ T\left(\frac{n}{9}\right) = 9T\left(\frac{\frac{n}{9}}{3}\right) + \frac{n}{9} = 9T\left(\frac{n}{3^3}\right) + \frac{n}{3^2}$$

$$T(n) = 9T\left(\frac{n}{3}\right) + n = 9\left[9T\left(\frac{n}{9}\right) + \frac{n}{3}\right] + n = 3^4T\left(\frac{n}{9}\right) + 3n + n = 3^4\left[9T\left(\frac{n}{3^3}\right) + \frac{n}{3^2}\right] + 3n + n$$

$$= 3^6T\left(\frac{n}{3^3}\right) + 3^2n + 3n + n = 3^8T\left(\frac{n}{3^4}\right) + 3^3n + 3^2n + 3n + 3 = \cdots =$$

$$n(3 + 3^2 + 3^3 + 3^4 + \cdots + 3^x)$$

כמה הערך של X?

$$T\left(\frac{n}{3^x}\right) = T(1) \rightarrow \frac{n}{3^x} = 1 \rightarrow x = \log_3 n$$

סדרה הנדסית כאשר $q=3$, איברים: $x = \log_3 n$

סכום הסדרה:

$$S_x = \frac{a_1(q^x - 1)}{q - 1} = \frac{3(3^{\log_3 n} - 1)}{3 - 1} = \frac{3(n - 1)}{2} = 1.5n - 1.5 = O(n)$$

לא לשכוח להכפיל ב n!!!!

סה"כ $O(n^2)$

שיטת האיטרציה

• מגדלי האנוי:

```c
void hanoi(int n, char source, char dest, char help)
{
    if (n == 1)
        printf("move disc %d from %s to %s", n, source, dest);
    else {
        hanoi(n - 1, source, help, dest);
        printf("move disc %d from %s to %s", n, source, dest);
        hanoi(n - 1, help, dest, source);
    }
}
```

$$T(1) = 1$$

$$T(n) = 2T(n-1) + 1$$

$$* \, T(n-1) = 2T(n-2) + 1$$

$$** \, T(n-2) = 2T(n-3) + 1$$

$$\begin{aligned} T(n) &= 2T(n-1) + 1 = 2[2T(n-2) + 1] + 1 = 4T(n-2) + 2 + 1 \\ &= 4[2T(n-3) + 1] + 2 + 1 = 2^3T(n-3) + 2^2 + 2^1 + 2^0 \\ &= 2^4T(n-4) + 2^3 + 2^2 + 2^1 + 2^0 = \cdots = 2^xT(n-x) + 2^{x-1} + \cdots + 2^0 \end{aligned}$$

X=???

T(n-x)=T(1) ➔ x = n-1

$$T(n) = 2^{n-1}T(1) + 2^{n-2} + \cdots + 2^0 = 2^{n-1} + 2^{n-2} + \cdots + 2^0 = 1 + 2 + 4 + \cdots + 2^{n-1}$$

סדרה הנדסית רגילה

$$q = 2, a_1 = 1 \to S_n = \frac{a_1(q^n - 1)}{q - 1} = 2^n - 1$$

$$T(n) = O(2^n)$$

משפט המאסטר [משפט האב]

- בהינתן נוסחת נסיגה מהצורה $T(n) = aT\left(\frac{n}{b}\right) + f(n) \quad where \;\; a \geq 1, b > 1$
- ניתן למצוא אסימפטוטית את T(n) באחד משלושה מקרים:
- <u>מקרה א:</u> אם קיים קבוע $\varepsilon > 0$ כך ש $f(n) = O\left(n^{\log_b a - \varepsilon}\right)$,
  אז: $T(n) = O\left(n^{\log_b a}\right)$
- <u>מקרה ב:</u> אם $f(n) = \theta\left(n^{\log_b a}\right)$ אז $T(n) = \theta\left(n^{\log_b a} \times \log_2 n\right)$
- <u>מקרה ג:</u> אם קיים קבוע $\varepsilon > 0$ כך ש $f(n) = \Omega\left(n^{\log_b a + \varepsilon}\right)$ וקיים c < 1 כך ש $a \times f\left(\frac{n}{b}\right) \leq c \times f(n)$ עבור כל ה n-ים הגדולים, אז $T(n) = \theta(f(n))$

משפט האב
דוגמאות

דוגמאות:

1) $$T(n) = 9T(n/3) + n$$

בנוסחה זו, $a = 9$, $b = 3$, ו־ $f(n) = n$. נציב ונקבל: $n^{\log_b a} = n^{\log_3 9} = \Theta(n^2)$. כלומר, $f(n) = O(n^{\log_3 9 - \epsilon})$ עבור $\epsilon = 1$. הנוסחה מתאימה אפוא למקרה 1 של משפט שיטת האב, ולכן פתרונה הוא $T(n) = \Theta(n^2)$.

משפט האב
דוגמאות

2) $T(n) = T(2n/3) + 1$

בנוסחה זו, $a = 1$, $b = 3/2$, $f(n) = 1$, ו־ $n^{\log_b a} = n^{\log_{3/2} 1} = n^0 = 1$. הנוסחה מתאימה למקרה 2, שכן $f(n) = \Theta(n^{\log_b a}) = \Theta(1)$, ולכן הפתרון לנוסחת הנסיגה הוא $T(n) = \Theta(\lg n)$.

משפט האב
דוגמאות

3)
$$T(n) = 3T(n/4) + n\lg n$$

בנוסחה זו, $a = 3$, $b = 4$, $f(n) = n\lg n$, ו־ $n^{\log_b a} = n^{\log_4 3} = O(n^{0.793})$. מאחר
ש־ $f(n) = \Omega(n^{\log_4 3 + \epsilon})$ עבור $\epsilon = 0.2$, הנוסחה תתאים למקרה 3 אם נוכל להוכיח כי $f(n)$
מקיימת את תנאי הרגולריות. עבור n גדול דיו, מתקיים:

$$af(n/b) = 3(n/4)\lg(n/4) \le (3/4)n\lg n = cf(n)$$

עבור $c = 3/4$. ולכן, על פי מקרה 3, הפתרון לנוסחת הנסיגה הוא $T(n) = \Theta(n\lg n)$.

משפט האב
דוגמאות

4) $T(n) = 2T(n/2) + n\lg n$

אמנם צורתה הכללית מתאימה: $a = 2$, $b = 2$, $f(n) = n\lg n$, ו־ $n^{\log_b a} = n$. לכאורה נראה שהיא מתאימה למקרה 3, שכן $f(n) = n\lg n$ גדולה אסימפטוטית מ־ $n^{\log_b a} = n$; אולם היא איננה גדולה ממנה פולינומיאלית. עבור כל קבוע חיובי ϵ, היחס $f(n)/n^{\log_b a} = (n\lg n)/n = \lg n$ קטן אסימפטוטית מ־ n^{ϵ} עבור כל קבוע חיובי ϵ. כתוצאה מכך, נוסחת נסיגה זו נופלת בין מקרה 2 למקרה 3, ולכן אי אפשר לפתור אותה בשיטת האב.

זמן ריצה

- מהי הסיבוכיות של האלגוריתם הבא:

```
void Tr(int n)
{
    int k, x;
    if(n>1)
        {
          x=n;
          while(x>1)
              x=x- n/10;
          k=2;
          while(k<n)
              k=k*2;
          Tr(n-1);
        }
}
```

נוסחאות נסיגה
תרגיל

- פתרו את נוסחאות הנסיגה הבאות בהנחה ש $T(1)=1$

1. $T(n) = 4T(n/2) + n^3$
2. $T(n) = T(8n/9) + n$
3. $T(n) = 16T(n/4) + n^2$
4. $T(n) = 3T(n/9) + \sqrt{n}$
5. $T(n) = 6T(n/6) + n / \lg n$
6. $T(n) = T(n-2) + 2\lg n$

Partition

הפונקציה Partition: בהינתן מערך arr, ותחום עבודה, הפונקציה בוחרת איבר "ציר" ומחלקת את המערך לשני חלקים – איברים שקטנים או שווים לו ואיברים הגדולים ממנו

```
int partition (int arr[], int low, int high) {
    int pivot = arr[high];    // pivot
    int i = low - 1;  // Index of smaller element
    for (int j = low; j <= high- 1; j++)    {
        // If current element is smaller than or equal to pivot
        if (arr[j] <= pivot)        {
            i++;    // increment index of smaller element
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i + 1], &arr[high]);
    return (i+1);
}
```

מיון מהיר

```
/* The main function that implements QuickSort
 arr[] --> Array to be sorted,
  low  --> Starting index,
  high  --> Ending index */
void quickSort(int arr[], int low, int high) {
    if (low < high)    {
        /* pi is partitioning index, arr[p] is now at right place */
        int pi = partition(arr, low, high);
        // Separately sort elements before partition and after partition
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}
```

מיון מהיר
זמן ריצה

- **המקרה הגרוע-** המערך נתון כבר ממויין.

זמן הריצה: איבר הציר הוא תמיד האיבר הגדול ביותר במערך, לכן החלוקה לשני בכל שלב, n-1. תתי מערכים תיתן מערך "עליון" בגודל 1, ומערך "תחתון" בגודל החלוקה לוקחת זמן כגודל המערך. ולכן, זמן הריצה:

$(n-1)+(n-2)+\dots+1=O(n^2)$

- **המקרה הטוב-** איבר הציר הוא כזה שמחלק תמיד את המערך לשני חלקים שווים.

זמן הריצה: עומק הרקורסיה הוא logn – מספר הפעמים שניתן לחלק את **n** ל 2. בכל שלב החלוקה לוקחת זמן כגודל המערך, ולכן, **זמן הריצה O(nlogn)**

- **המקרה הממוצע-** הניתוח הוא יותר מורכב...

זמן הריצה O(nlogn)

- **המקרה הגרוע הוא נדיר, בפועל, אלגוריתם זה נותן ביצועים טובים מאוד!**

חציונים וערכי מיקום

- נתון מערך לא ממוין עם מספרים. עלינו למצוא מהו האיבר ה k בגודלו במערך
  - האיבר הקטן ביותר במערך
  - האיבר הגדול ביותר במערך
  - איבר החציון
- לדוגמא: נחפש את האיבר ה – k בגודלו
- arr[] = {7, 10, 4, 3, 20, 15} k = 3 Output: 7
- arr[] = {7, 10, 4, 3, 20, 15} k = 4 Output: 10
- אפשרות א': למיין את המערך ולגשת לתא ה k-1
  - זמן ריצה O(n*log(n))
- אפשרות ב': אלגוריתם Select

חציונים וערכי מיקום
Select Algorithm

- האפשרות הפשוטה ביותר של בחירה היא מציאת המינימום או המקסימום בזמן ריצה (O(n
- אפשרות יותר מורכבת היא מציאת איבר ה – k בגודלו במערך
  - נשתמש באלגוריתם partition לגלות את האינדקס של איבר רנדומאלי במערך
  - אם איבר הציר הוא גדול מ k, האיבר שלנו נמצא בצד שמאל
  - אחרת, נמצא בצד ימין

חציונים וערכי מיקום
Select Algorithm

```
int select(int  A[], int left, int right, int k)
{
//p is position of pivot in the partitioned array
int p = partition(A, left, right);
//k equals pivot got lucky
if (p == k - 1) { return A[p]; }
if (k - 1 < p) { //k less than pivot
     return select(A, left, p - 1, k); }
else { //k greater than pivot
     return select(A, p + 1, right, k); }
}
```

חציונים וערכי מיקום
Select Algorithm

זמן ריצה:

- הפונקציה משתמשת ב Partition רק על צד אחד של המערך ולא על שני הצדדים
- במקרה הממוצע:

$$T(n) = T(n/2) + n$$

לפי משפט האב מקרה ג', זמן הריצה הוא O(n)

הנך רשאי להשתמש בפונקציות האלה

שם הפונקציה	תיאור הפונקציה	סיבוכיות זמן הריצה של פונקציה
Sort(S)	פונקציה ממינת את איבריה של הקבוצה S ומחזירה אותה ממוינת על פי סדר עולה	O(n*logn)
Select(S,k)	פונקציה המוצאת ומחזירה את האיבר ה k הקטן ביותר מבין איברי הקבוצה S, כלומר היא מוצאת את ערך המיקום ה k בקבוצה S.	O(n)
Partition(S,z,S1,S2)	פונקציה המחלקת את הסדרה S בת n האיברים לשתי סדרות S1 ו-S2 באופן הבא: בסדרה S1 יהיו האיברים הקטנים מ z או השווים לו, ובסדרה S2 יהיו האיברים הגדולים מ z. **שים לב:** הסדרות S1 ו – S2 אינן בהכרח ממוינות	O(n)

נגדיר איבר רוב כאיבר שמופיע במערך בגודל n יותר מ n/2 פעמים. נתון מערך A המכיל n מספרים טבעיים כאשר n הוא מספר אי-זוגי.לפניך אלגוריתם יעיל אשר בודק האם קיים **איבר רוב** במערך A. אם קיים – האלגוריתם יחזיר את המספר שהוא איבר הרוב במערך A. אחרת – האלגוריתם יחזיר את הערך -1.

האלגוריתם:

צעד 1: ______(1)______

צעד 2: ______(2)______

צעד 3: ______(3)______

באלגוריתם חסרים שלושה ביטויים, המסומנים במספרים בין סוגריים עגולים. התשובה הנכונה עבור כל אחד מהביטויים החסרים מופיעה בסעיפים אלה:

הביטוי החסר (1) הוא:

1. $M = select(A, \frac{n+1}{2})$
2. $partition(A, \frac{n+1}{2}, A1, A2)$
3. $M = select(A, \frac{n}{2})$
4. $Sort(A)$

הביטוי החסר (2) הוא:

1. L מקבל את מספר ההופעות של M במערך A
2. L1 מקבל את מספר המספרים שבמערך A הקטנים מ־M או שווים לו
3. L2 מקבל את מספר המספרים שבמערך A הגדולים מ־M או שווים לו
4. L3 מקבל את מספר ההופעות של הערך השכיח במערך הממוין A

הביטוי החסר (3) הוא:

1. אם $L1 \geq \frac{n+1}{2}$ אזי החזר את המספר $\frac{L1}{2}$, אחרת – החזר את הערך 1-
2. אם $L \geq \frac{n+1}{2}$ אזי החזר את המספר M, אחרת – החזר את הערך 1-
3. אם $L2 \geq \frac{n+1}{2}$ אזי החזר את המספר L_2, אחרת – החזר את הערך 1-
4. אם $L3 \geq \frac{n+1}{2}$ אזי החזר את המספר השכיח במערך A, אחרת – החזר את הערך 1-

מהי סיבוכיות זמן הריצה של האלגוריתם הנתון?

1. $\Theta(n \log n)$
2. $\Theta(\log n)$
3. $\Theta(n)$
4. $\Theta(n^2)$

- נתונה קבוצה סופית S, המכילה n מספרים שלמים חיוביים ואיבריה משוכנים במערך A, החל באינדקס 1 ועד לאינדקס ה – n (כולל).

הנח כי כל הפעולות המתמטיות, כגון חיבור, כפל והשוואה, ניתנות לביצוע בזמן O(1).

לפניך אלגוריתם יעיל בשם TRI, אשר מחזיר את הערך "no" אם קיימת בקבוצה S תת-קבוצה כלשהי T, כך שסכום איבריה של T יהיה קטן מ $|T|^3$ [|T| - כמות האיברים בקבוצה T].

אחרת – אם בעבור **כל** תת-קבוצה T של S מתקיים $\sum_{t \in T} t \geq |T|^3$ האלגוריתם יחזיר את הערך "yes".

שים לב: |T| מציין את מספר האיברים שבקבוצה T.

האלגוריתם:

TR1(A, n)

צעד 1 : ______(1)______ .

צעד 2 : $Sum \leftarrow 0$.

צעד 3 : עבור k המקבל ערכים החל מ־1 ועד ______(2)______ בצע:

3.1 $Sum \leftarrow Sum +$ ______(3)______

3.2 אם ______(4)______ אז עצור והחזר "no"

צעד 4 : החזר "yes".

הביטוי החסר (1) הוא:

1. $x = select(A,k)$
2. $x = select\left(A, \left\lceil \frac{n+1}{2} \right\rceil\right)$
3. $x = select(A,1)$
4. $Sort(A)$

הביטוי החסר (2) הוא:

1. k
2. x
3. n
4. k^3

הביטוי החסר (3) הוא:

1. $x = select(A,k)$
2. $select\left(A, \left\lceil \frac{k+1}{2} \right\rceil\right)$
3. A[k]
4. A[Select(A,k)]

הביטוי החסר (4) הוא:

1. $Sum < k$
2. $Sum < k^3$
3. $Sum < k^3 + Select(S,k)$
4. $Sum < k^3 - Select(S,k)$

מהי סיבוכיות זמן הריצה של האלגוריתם TR1 ?

1. $\Theta(n \log n)$
2. $\Theta(\log n)$
3. $\Theta(1)$
4. $\Theta(n)$

פונקציית החזקה
Power Algorithm

```
int power(int x, int n) {
    if (n == 0)
        return 1;
    if (n % 2 == 0) {
        int a  = power(x, n / 2);
        return a * a;
    }
    if (n % 2 == 1) {
        int a = power(x, (n - 1) / 2);
        return a * a * x;
    }
}
```

פונקציית החזקה:

$$x^n = \begin{cases} x\left(x^2\right)^{\frac{n-1}{2}}, & \text{if } n \text{ is odd} \\ \left(x^2\right)^{\frac{n}{2}}, & \text{if } n \text{ is even.} \end{cases}$$

זמן ריצה: T(n) = T(n/2) + 1 = T(log(n))
לפי משפט האב

נוסחאות נסיגה
החלפת משתנים

$$T(n) = 2T(\sqrt{n}) + \log_2 n$$

$$m = \log_2 n \to 2^m = 2^{\log_2 n} \to n = 2^m \to \sqrt{n} = 2^{\frac{m}{2}}$$

עכשיו נציב

$$T(2^m) = 2T\left(2^{\frac{m}{2}}\right) + m$$

נסמן

$$S(m) = T(2^m)$$

$$S\left(\frac{m}{2}\right) = T\left(2^{\frac{m}{2}}\right)$$

עכשיו נציב

$$S(m) = 2S\left(\frac{m}{2}\right) + m$$

לפי משפט האב מקרה 2

$$S(m) = \theta(mlog_2 m)$$

עכשיו נחזיר הכל למשוואה עם n

$$S(m) = T(2^m) = T(n) = \theta(mlog_2 m) = \theta(\log_2 n \times \log_2 \log_2 n)$$

נוסחאות נסיגה
החלפת משתנים

שאלה רעה מאוד ממבחן 2017

$T(n) = 2T\left(3^{\sqrt{\log_3 n}}\right) + \log_2 \log_3 n$ היא פונקציית זמן הריצה של אלגוריתם מסוים, הפועל על קלט שגודלו **n** . מהי סיבוכיות זמן הריצה של האלגוריתם?

1. $\Theta((\log \log n) \cdot (\log \log \log n))$
2. $\Theta((\log \log n)^2)$
3. $\Theta(\sqrt{\log n} \cdot \log \log n)$
4. $\Theta(\sqrt{\log n} \cdot \log \log \log n)$

נסמן

$m = \log_2 \log_3 n \rightarrow 2^m = 2^{\log_2 \log_3 n} \rightarrow 2^m = \log_3 n \rightarrow n = 3^{2^m}$

נציב

$T\left(3^{2^m}\right) = 2T\left(3^{\sqrt{2^m}}\right) + m$

$* \sqrt{2^m} = (2^m)^{\frac{1}{2}} = 2^{m \times \left(\frac{1}{2}\right)}$

$T\left(3^{2^m}\right) = 2T\left(3^{2^{\frac{m}{2}}}\right) + m$

נסמן

$S(m) = T\left(3^{2^m}\right) \rightarrow s\left(\frac{m}{2}\right) = T\left(3^{2^{\frac{m}{2}}}\right)$

נציב

$S(m) = 2S\left(\frac{m}{2}\right) + m \rightarrow$ 2 לפי משפט האב מקרה $\rightarrow S(m) = \theta(m \times \log_2 m)$

עכשיו נחזיר הכל ל n

$S(m) = T\left(3^{2^m}\right) = T(n) = \theta(m \times \log_2 m) = \theta(\log_2 \log_3 n \times \log_2 \log_2 \log_3 n)$