© Itai June 9, 1999

מיון Sorting

קלט מערך בן n רשומות שלכל אחת מפתח

$a_1, \ldots, a_n$

משימה סדר מחדש את אותם המפתחות בסדר עולה.

שיטות נאיביות – זמן $\theta(n^2)$

דוגמא: החלפת איברים סמוכים BubbleSort

```
for (i=n-1; i>0; i--){
  for ( j=0; j<i; j++){
     if(a[j] > a[j+1])
        swap(&a[j], &a[j+1]);
  }
}
```

זמן $\Sigma\ n\text{-}i = n(n+1)/2 = O(n^2)$

הערות:

1. החסם $O(n^2)$ על האלגי הוא מדויק.
2. כל החלפה של איברים סמוכים מקטינה את מספר ה inversions ב-1 כיון שיש $\Theta(n^2)$ inversions נקבל חסם תחתון של $\Omega(n^2)$ על אלגי שמבצעים רק החלפות של איברים סמוכים.

חסם תחתון למיון

הנחה את אברי הקלט אפשר רק להשוות,
(לא לבצע עליהם פעולות אריתמטיות).

משפט כל אלגוריתם מיון דורש לפחות $\Omega(n \log n)$ השוואות.

הוכחה - Information Theory
מודל: משחק בין יריב לבין האלגוריתם.
מטרת האלגי - למצוא את פרמוטצית הקלט.
מטרת היריב - שבסוף עדיין תשאר יותר מפרמוטציה אחת
אפשרית שהאלגי לא יכול להבדיל ביניהן.

בכל השוואה, $a[i] < a[j]$, חלק מהפרמוטציות קונסיסטנטיות
עם תשובה חיובית וחלק עם שלילית.

אסטרטגית היריב: בחר את התשובה שמשאירה מספר מכסימלי של
פרמוטציות שקונסיסטנטיות עם כל התשובות עד כה.

בהתחלה יש $n!$ פרמטציות.
בכל השוואה היריב יכול להבטיח שמס' הפרמוטציות
הקונסיסטנטיות לא קטן יותר מאשר פי חצי .

מסקנה: אחרי k השוואות ליריב נותרו עדיין $\frac{n!}{2^k}$ פרמוטציות.
לכן מספר ההשוואות הוא לכל הפחות $\log_2 n!$.

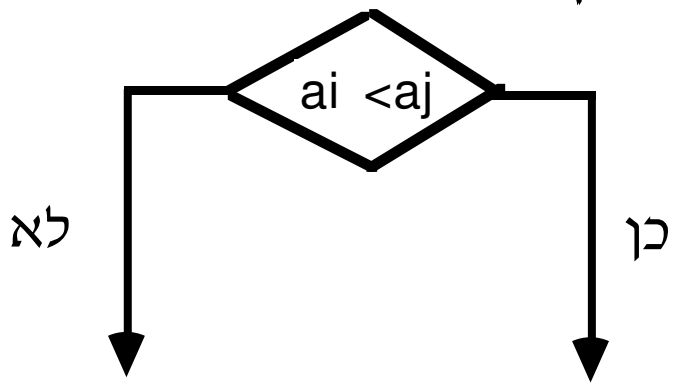
נוסחת סטירלינג $n! > \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$
ולכן מס' ההשוואות

$$\log_2 n! > \log_2\sqrt{2\pi n}\left(\frac{n}{e}\right)^n = \frac{1}{2}\log_2 2\pi n + n\log_2 n - n\log_2 e$$
$$> n\log_2 n - 1.4427n = n\log_2 n - O(n)$$

משייל

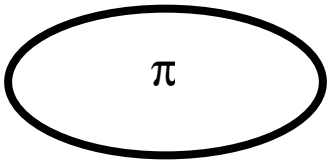
הוכחה אלטרנטיבית לחסם התחתון

יהי A אלגוריתם מיון.
נייצג את A בתור עץ החלטות – decision tree.



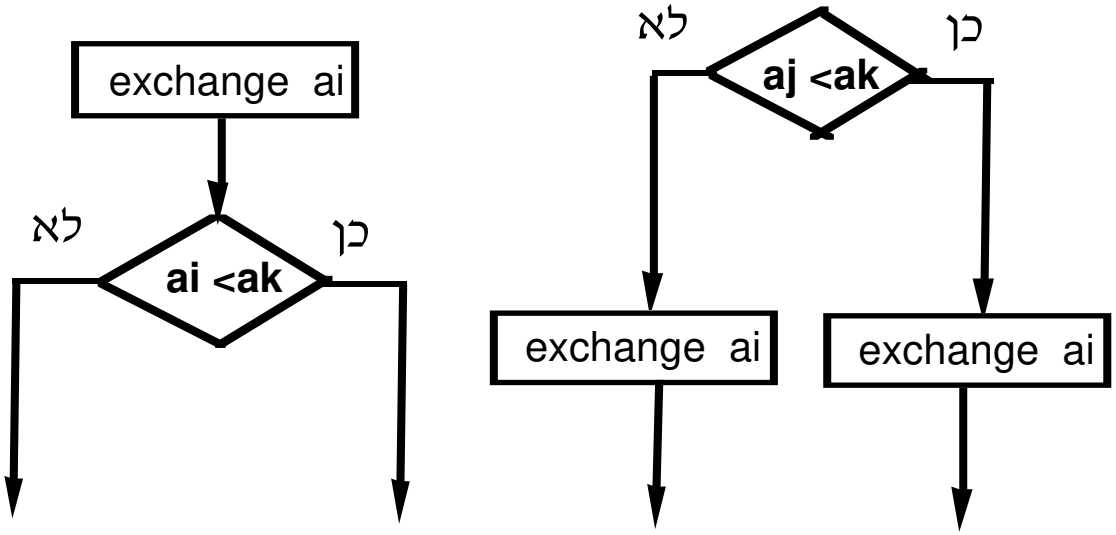
צמתי השוואה

צמתי פעולה



עלים

נוריד צמתי פעולה עד העלים:



דוגמא:

בכל עלה יש סדרת פעולות שמסדרת את הפרמוטציות שבעלה,
אותה סדרת פעולות יכולה לסדר רק פרמוטציה אחת,
לכן בכל עלה יש רק פרמוטציה אחת.
כיון שיש n! פרמוטציות,
לעץ ההשוואות יש n! עלים.

גובה העץ

$$h = \log(\text{מספר העלים}) = \log n! = n \log n + O(1).$$

בעץ יש מסלול באורך n log n.
קיים קלט, עבורו האלגוריתם מבצע n log n השוואות.

QuickSort

החלפת איברים (לאו דווקא סמוכים)

הפרד ומשול:

בשלב ביניים נמיין את המערך

a[first], a[first+1], ... , a[last]

1. בחר איבר ציר $\min_{first\le i\le last} a[i] \le pivot \le \max_{first\le i\le last} a[i]$

2. הפרד: סדר את אברי המערך כך ש-

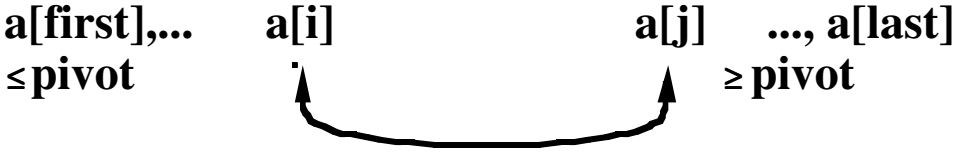
a[first], ... , a[i-1] ≤ pivot ≤ a[i], ... , a[last]

3. משול

a[first], ... , a[i-1] — **מיין רקורסיבית**

a[i], ... a[last] — **מיין רקורסיבית**

המפתח ליעילות נמצא ב-partition




```
choose pivot;
for(i=first,j=last; i≤j;){
  for( ; a[i]<pivot; i++)
       ;
  for( ; a[j]>pivot; j--)
       ;
  if(i≤j){
      exchange a[i] and  a[j];
      i++; j--;
  }
}
```

התכנית כולה

```
#include <stdio.h>
int *a;

swap(int i, int j)
{
  int temp;
  temp=a[i];
  a[i]=a[j];
  a[j]=temp;
}

KEY choose_pivot(KEY *a,
                 int left, int right)
{
  return a[left];
}

main()
{ int n,i;
  printf("enter integer\n");
  scanf("%d",&n);
  a=(int *)calloc(n+1,sizeof(int));
  for(i=1;i <= n; i++)
    scanf("%d",a+i);
  quick_sort(a,1,n);
  for(i=1;i <= n; i++)
    printf(" %4d",a[i]);
  printf("\n");
}
```

```
int partition(KEY *a,
              int left,right,
              KEY pivot){

     int i,j;

     for(i=left,j=right; i≤j;){
          for( ; a[i]<pivot; i++)
               ;
          for( ; a[j]>pivot; j--)
               ;
          if(i≤j){
             swap(i,j);
             i++; j--;
          }
     }
     return i;
}

void quick_sort(KEY *a,
                int left, int right){
     int i,j;
     KEY pivot;

     if(left>=right)
             return;

     pivot=choose_pivot(a,left,right);
     i=partition(a,left,right,pivot);


     quick_sort(a,left,i-1);

     quick_sort(a,i,right);
}
```

זמן — תלוי איך נבחר איבר הציר

מקרה אופטימלי -- חציון:

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

מקרה גרוע -- מינימום

$$T(n) = n + T(n-1)$$
$$T(n) = n + (n-1) + \ldots + 1 = n(n+1)/2$$
$$= O(n^2)$$

מקרה ממוצע -- (בחירה רנדומיאלית)

$$T(n) = n + (1/n)\Sigma\,(T(i) + T(n-i))$$
$$= O(n \log n)$$

עץ הקריאות -- עץ בינארי אקראי

הערות

מה קורה אם יש חזרות בקלט?

עבור $n \le 12$ נקרא ל BubbleSort .

שיטה יעילה ופופולארית.

עומק מחסנית הרקורסיה

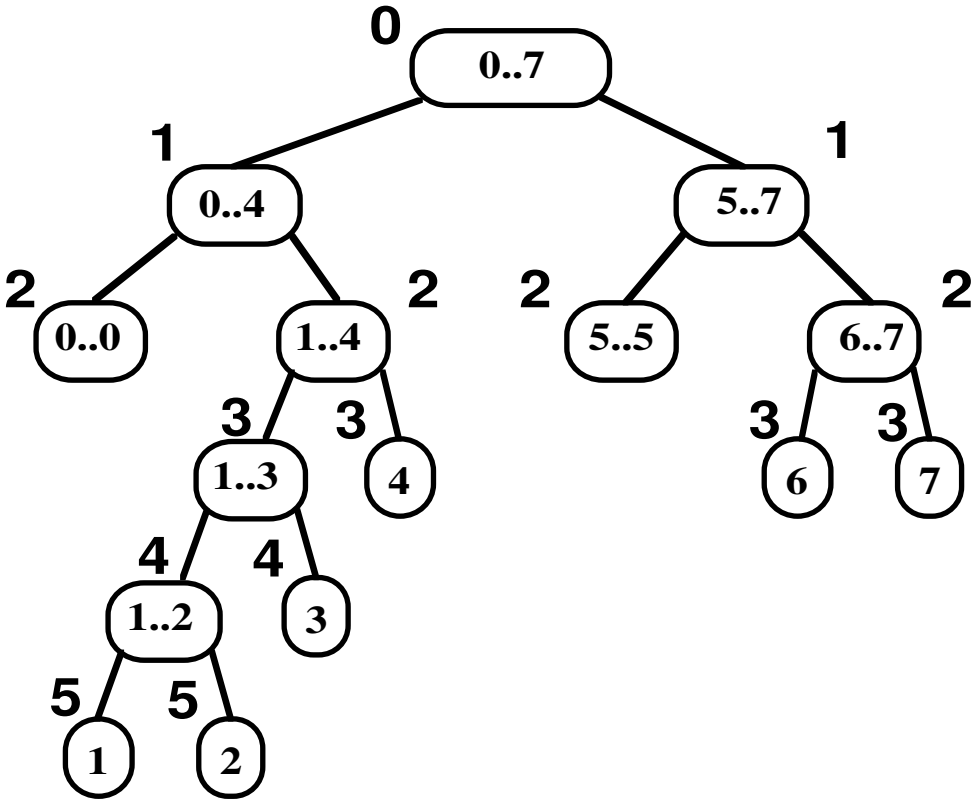
המקום שדורש QuickSort כולל

- מערך הקלט,
- משתנים מקומיים O(1),
- מחסנית הרקורסיה.

במקרה הגרוע עומק הרקורסיה הוא O(n).
נראה שיטה שגם במקרה הגרוע עומק הרקורסיה יוגבל ל- O(log n).

שלד התוכנית:

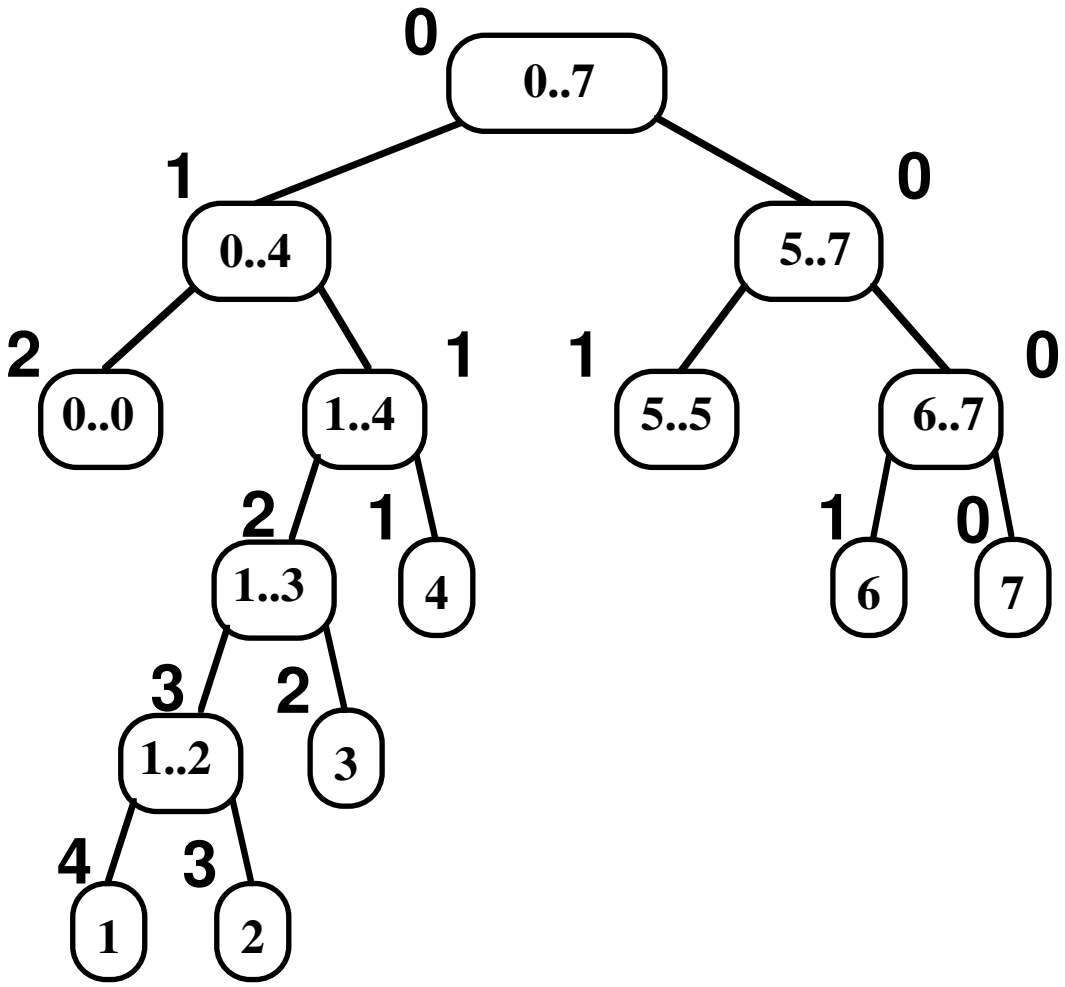
```
QS(KEY *a, int l, int r){
   int i,j,p;
   if(l ≥ r) return;
   p = choose_pivot(a,l,r);
   i=partition(a,l,r,p);
   QS(l, i-1);
   QS(i, r);
}
```



נחליף את הקריאה הרקוסיבית השניה באיטרציה

```
QS(int l, int r){
   int i,j,p;
   while(l < r){
      p=choose_pivot(a,l,r);
      i=partition(a,l,r,p);
      QS(l, i-1);
      l=i;
   }
}
```

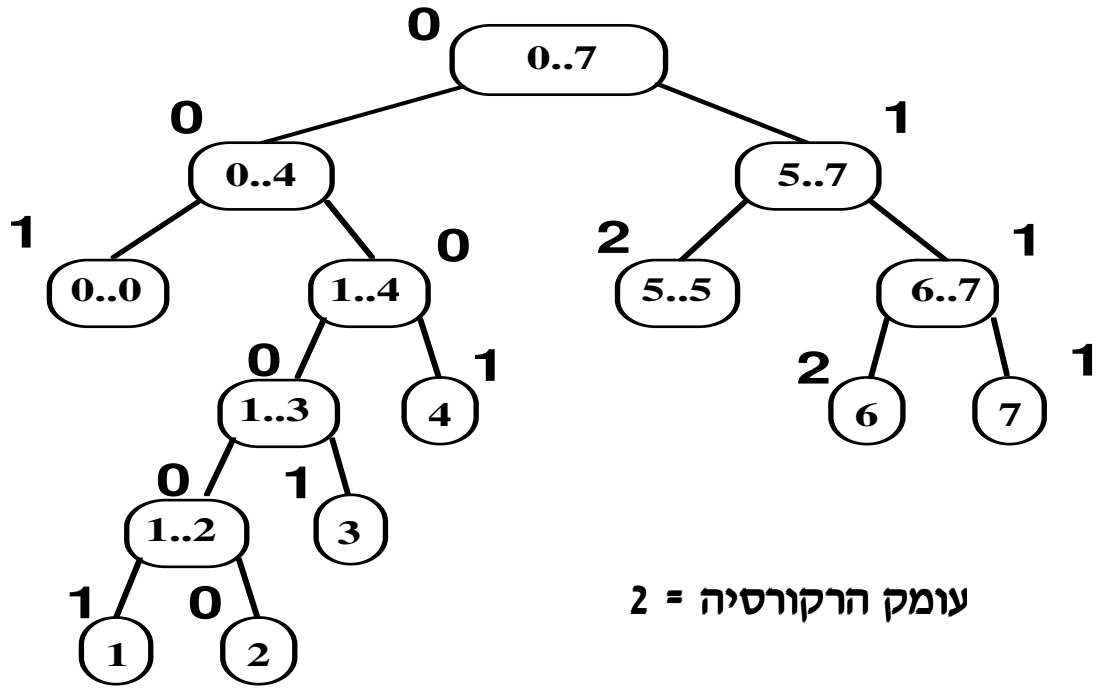
שימו לב, שנעשות אותן ההשוואות כמו קודם.



Tail recursion elimination

נקרא רקורסיבית לחלק הקטן:

```
QS(KEY *a, int l, int r){
   int i,j,p;
   while(l < r){
      p=choose_pivot(a,l,r)];
      i=partition(a,l,r,p);
      if(i-l ≤ r-i+1){
         QS(a, l, i-1);
         l=i;
      }
      else {
         QS(a, i, r);
         r=i-1;
      }
   }
}
```



בגירסה זו עומק הרקורסיה $\geq \log n$.

median -- חציון

הגדרה

יהי $X = \{ x_1, ..., x_n \}$.

x_i הוא חציון אם מחצית אברי X קטנים ממנו ומחציתם גדולים ממנו.

כלומר, האיבר ה- $\lfloor (n+1)/2 \rfloor$.

דוגמאות

א. $X = \{ 7\ \underline{4}\ 5\ 2\ 1 \}$

median = 4

ב. $X = \{ 7\ \underline{4}\ 5\ 2\ 1\ 8 \}$

median = 4

אלגוריתם רנדומאלי ל-Selection

לפי העקרון של QuickSort :

Selection(A, first, last, i)

מוצא את האיבר ה-i בגודלו מתוך A[first..last]

בחר באקראי איבר x ;

חלק את A לפי x (בעל אינדכס q) ;

המשך עם A[first..q] או עם A[q+1.. last] .

```
KEY rand_select(KEY *A, int first, last, i){
     int q, k;

     if(i < 1 || last - first +1 > i) error();
     if(first == last) return A[first];
     q = Rand_Partition(A, first, last);
     k = q - first; /* size of left partition */
     if(i <= k)
          return rand_select(A, first, q-1, i);
     return rand_select(A, q, last, i-k);
}
```

דוגמא

A = 7 3 2 1 5 9 8

איטרציה ראשונה

rand_select(A, 0, 6, 4)

נניח pivot = 3

1 2 ▲3 7 5 9 8

איטרציה שניה

rand_select(A, 2, 6, 2)

נניח pivot = 7

•• 3 5 ▲7 9 8

איטרציה שלישית

rand_select(A, 2, 3, 2)

נניח pivot = 5

**•• 3 ▲5 ••• **

איטרציה רביעית

rand_select(A, 3, 3, 1)

•• • 5 •••

מוחזר 5

חשבון זמנים

מקרה גרוע

$$T(n) = cn + T(n-1)$$
$$= cn + c(n-1) + \ldots + c1 = O(n^2)$$

מקרה טוב

$$T(n) = cn + T(n/2)$$
$$= cn + cn/2 + T(n/4)$$
$$= cn + cn/2 + cn/4 + cn/8 + \ldots + cn/n + T(1)$$
$$= O(n)$$

מקרה ממוצע

$$T(n) \le (1/n)\left(\sum_{k=1} T(\max(k,n-k))\right) + O(n)$$
$$\le (1/n)\left(T(n-1) + 2\sum_{k\ge\lceil n/2\rceil} T(k)\right) + O(n)$$
$$= (2/n)\left(\sum_{k\ge\lceil n/2\rceil} T(k)\right) + O(n)$$

יהי d הקונסטנטה של ה-O
נראה באינדוקציה שעבור $c > \max\{d/4, T(1)\}$ ו-$n>3(c-4d)$

$$T(n) \le c\,n$$

הוכחה
בסיס מתוך הגדרה
צעד האינדוקציה

$$T(n+1) = \frac{1}{n}\sum_{i=1}^{n} \max\{T(i), T(i+1-i)\} + d(n+1)$$

$$\frac{2}{n}\sum_{i=\lfloor (n+1)/2 \rfloor}^{n} T(i) + d\cdot(n+1) \le \frac{2c}{n}\sum_{i=\lfloor (n+1)/2 \rfloor}^{n} i + d\cdot(n+1)$$

$$\approx \frac{2c}{n}\frac{3}{2}n\frac{n}{4} + d\cdot(n+1)cn$$

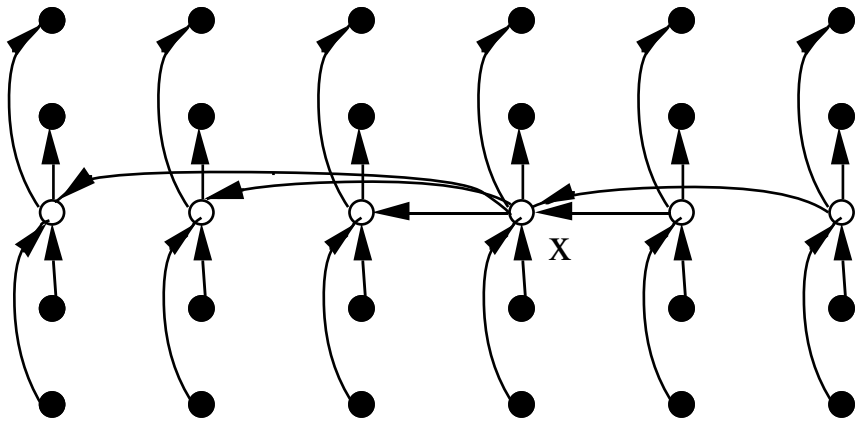
$$\frac{3}{4}cn + dn + d \le cn + d \le c\cdot(n+1).\ T(n) \le (2/n)$$

אלגוריתם דטרמיניסטי ליניארי

העקרון דומה לאלגי הרנדומאלי, אלא שבמקום לחלק ע"פ איבר אקראי נמצא איבר שנמצא במרחק αn מקצות המערך.

Select(A, first, last, i)

1. חלק את הקלט לחמישיות.
2. מצא חציון של כל חמישיה.
3. באופן רקורסיבי, מצא חציון של החציונים (נקרא לו x).
4. חלק את הקלט לפי x - יהי x האיבר ה- q.
5. אם q - first < i (x קטן מהאלמנט ה-i)
Select(A, q, last, i-q)
6. אחרת
Select(A, first, q, i)



חשבון זמנים (מקרה גרוע)

נניח ש-n מתחלק ב-5. והאינדכס של x קטן מ-i.
מחצית החציונים קטנים מ-x וכך גם האיברים שקטנים מהם.
כלומר לפחות 3n/10 איברים קטנים מ-x.

$$T(n) \le T(n/5) + T(7n/10) + O(n)$$

טענה אם עבור $a+b < 1$

$$T(n) = T(an) + T(bn) + cn$$

אזי $T(n) = O(n)$

מכאן לאלגוריתם זמן ליניארי.

מיון בעזרת תור עדיפויות / עריכה

תור עדיפויות מבנה נתונים המוגדר ע"י הפעולות:

1. **Empty (Q)**
2. **Insert (x, Q)**
3. **Min (Q)**
4. **Delete (y, Q)**

מיון בעזרת תור עדיפויות

1. **איתחול:** **Q = ∅**
2. **לכל** x **בקלט:** Insert (x, Q);
3. **כל עוד התור אינו ריק**

```
y = Min( Q );
output y;
Delete ( y, Q);
```

מימוש תור עדיפויות

1. **עץ חיפוש מאוזן.**
   כל פעולה בזמן O(log n)
   סה"כ זמן O(n log n)
   שיטת מיון זו נקראת TreeSort

ערימה Heap

ערימה היא מבנה נתונים זהה לתור עדיפויות,
פרט לכך שבמקום **Delete** מוגדר רק **Del_Min**

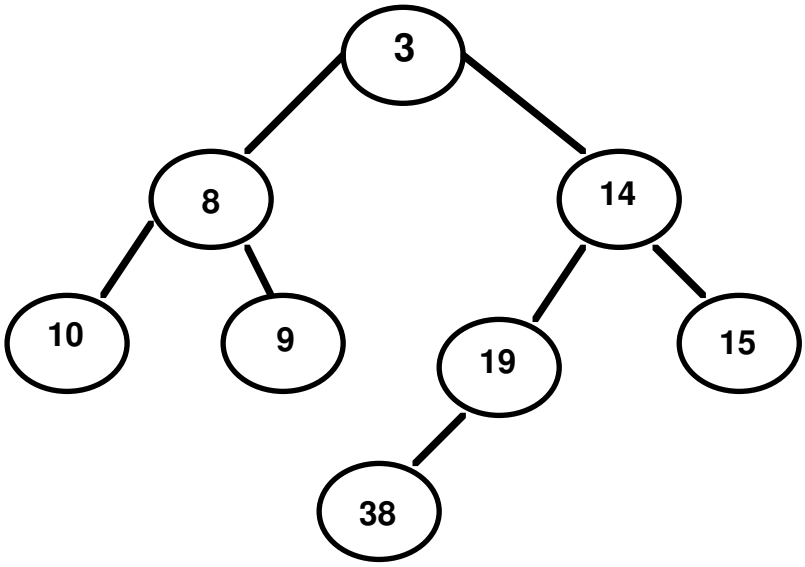
מיון בעזרת ערימה נקרא **HeapSort**

מימוש ערימה בעץ (לא עץ חיפוש)

סדר ערימה

המפתח של האבא ≥ מפתחות הבנים.

דוגמא:

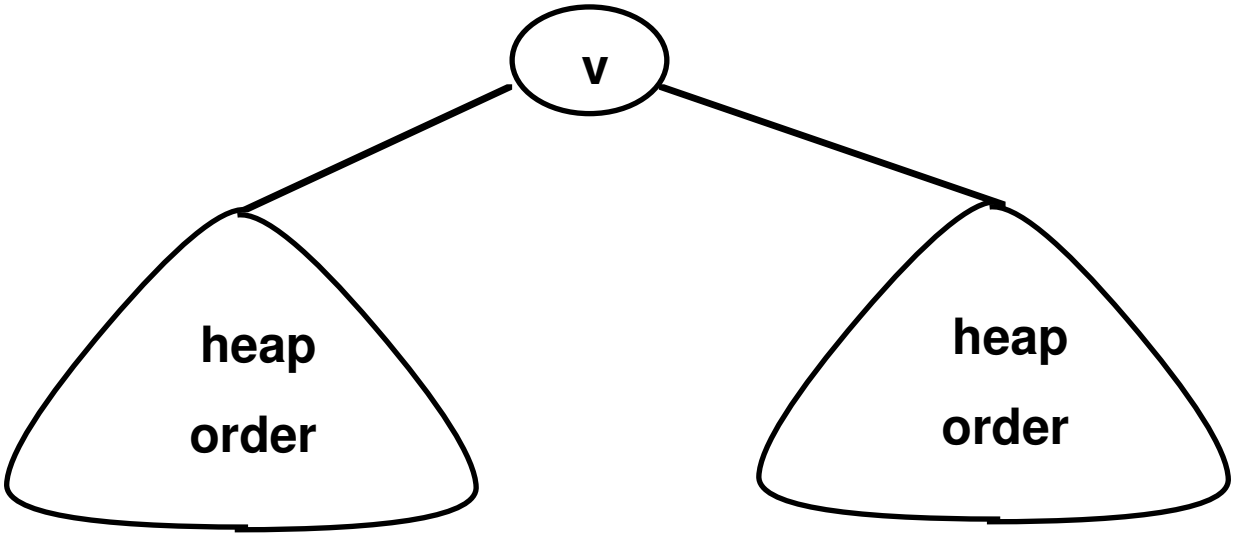


Sift_Down(v) פרוצדורת עזר

1. אם v עלה, או v->key קטן משני בניו -- גמרנו.
2. החלף את v->key עם הבן בעל מפתח מינימלי -- w,
ובצע Sift_Down(w) .

זמן O(h) .
בד"כ T עץ בינארי מלא ולכן זמן O(log n).

אם

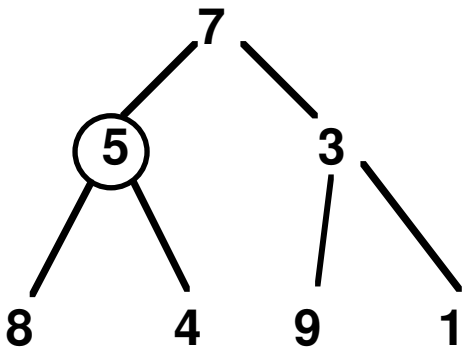


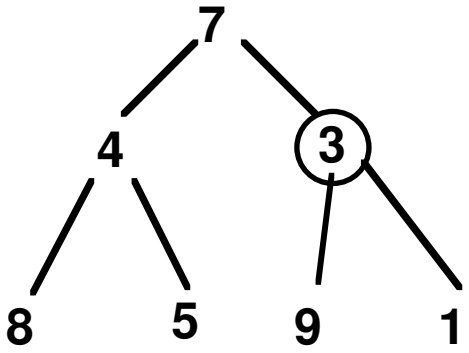
ביצוע Sift_Down(v) יהפוך את כל העץ לערימה.

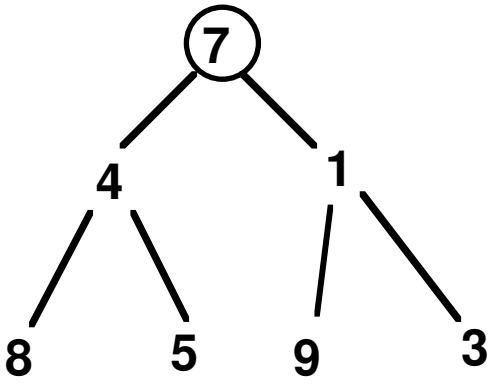
Make_Heap (T)

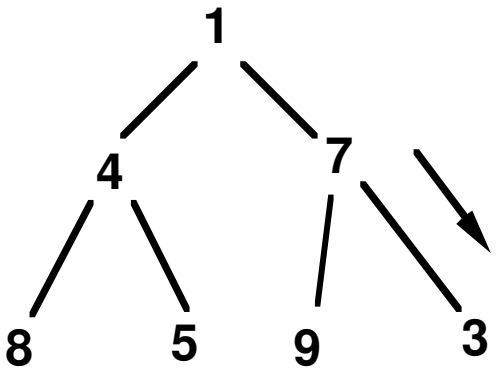
עבור על כל צמי העץ כך שנעבור על הבנים לפני האבא
(למשל Postorder).
לכל צומת בעץ בצע Sift_Down(v).

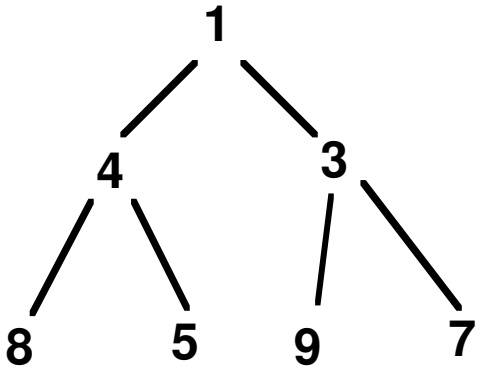
דוגמא











אנליזת זמנים של MakeHeap

גובה צומת: מרחק מכסימלי (בקשתות) בין הצומת לצאצא שלו (ברור שצאצא זה הוא עלה.)

זמן MakeHeap הוא $\text{Time(MakeHeap)} = \sum_{v \in T} h(v)$

$\text{Time(MakeHeap)} = O(n)$ **נראה שעבור עץ שלם**

למען הפשטות נניח ש $n = 2^{h+1} - 1$
כלומר, כל העלים הם באותה הרמה.

יותר מחצית הצמתים הם עלים, עבורם $h(v) = 0$
יותר מרבע מהצמתים הם אבות של עלים, עבורם $h(v) = 1$
...
יותר מ- $\frac{n}{2^{i+1}}$ **צמתים הם בגובה** i

על כן, $\sum_{v \in T} h(v) \le \sum_{i=0}^{h} i \frac{n}{2^{i+1}} < n \sum_{i=0}^{\infty} \frac{i}{2^{i+1}} = cn$

(הטור מתכנס, יהי c **= ערך הסכום.)**

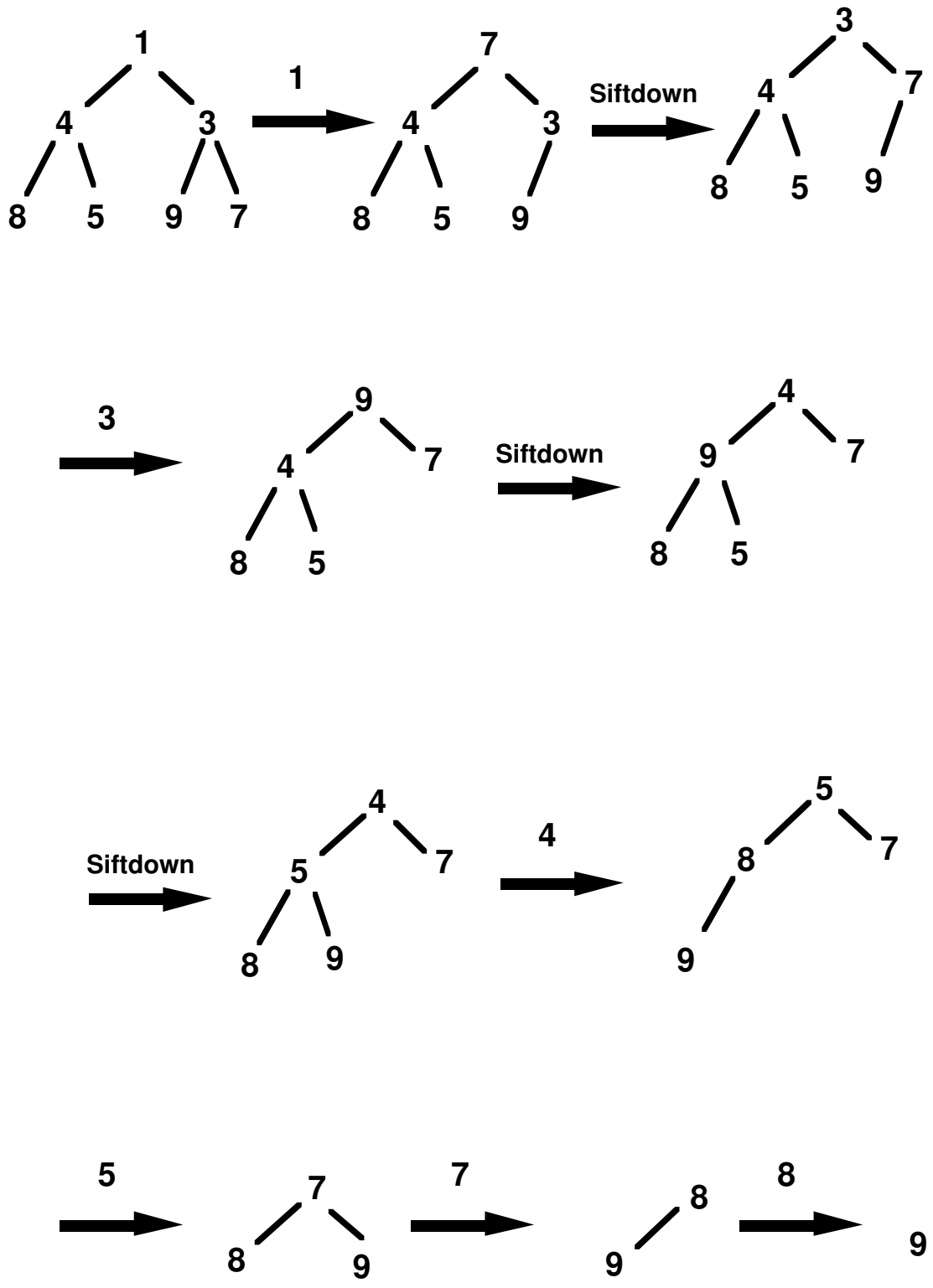
מש ל

Del_Min

שים עלה במקום השורש, r .
בצע Sift_Down (r) .

המיון:

```
Make_Heap(S);
while  (!Empty(S)){
       Output  Min(S);
       Del_Min(S);
}
```


1
4
3
8
5
9
7
1
7
4
3
8
5
9
Siftdown
3
4
7
8
5
9
3
9
4
7
8
5
Siftdown
4
9
7
8
5
Siftdown
4
4
5
7
8
9
4
5
8
7
9
5
7
8
9
7
8
9
8
9

אנליזת זמנים

אם התחלנו עם עץ שלם בן n צמתים
גובה h = log n
ולכן כל Del_Min זמן O(log n)

סה"כ

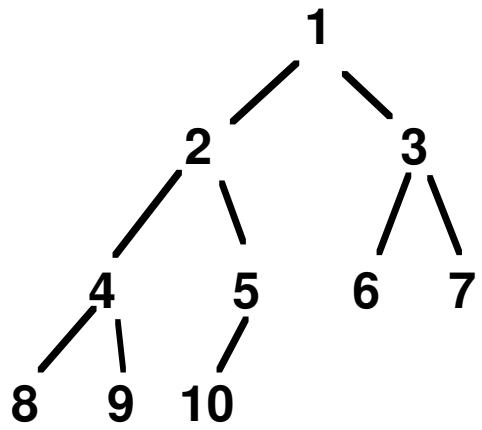
$$O(n) + O(n \log n) = O(n \log n)$$

Make_Heap Del_Min's

מקום:

בד"כ נאחסן את העץ במערך כך ש-
השורש במקום 1
הבנים של צמת v במקומות 2v, 2v+1

כלומר, לפי הסדר



1	2	3	4	5	6	7	8	9	10

התוכנית כולה

```
int *a;

swap(int i, int j)
{
    int temp;
    temp=a[i];
    a[i]=a[j];
    a[j]=temp;
}

sift_down(int first, int last)
{
    int r;
    for(r=first; r <= last/2;){
        if(2*r ==  last){
         /*  r has one child at 2*r */
            if(a[r]>a[2*r])
               swap(r,2*r);
            break;
        }
        else{/*r has two children at 2*r and
2*r+1 */
            if(a[r]>a[2*r]
                 && a[2*r] <= a[2*r+1]
){
                 swap(r,2*r);
                 r *= 2;
            }
            else if(a[r]>a[2*r+1]
                 && a[2*r+1]<=a[2*r]){
                swap(r,2*r+1);
                r = 2*r+1;
            }
            else break;
        }
    }
}
```

```
heap_sort(int n)

{
    int i;
    for(i=n/2;i>0;i--)
        sift_down(i,n);
    for(i=n;i>0;i--){
        swap(1,i);
        sift_down(1,i-1);
    }
}
main()
{ int n,i;
    printf("enter integer\n");
      scanf("%d",&n);
    a=(int*)calloc(n+1,
sizeof(int));
    for(i=1;i <= n; i++)
        scanf("%d",a+i);
    heap_sort(n);
    for(i=1;i <= n; i++)
        printf(" %4d",a[i]);
    printf("\n");
}
```

הערה: בתכנית הפלט ממוין מגדול לקטן.

יתרונות השיטה

1. זמן במקרה הגרוע O(n log n)
2. לא דורש מקום עזר.

שיטת מיון ליניארית BucketSort

המשימה מיין n מספרים בתחום 1..m .

השיטה נשתמש במערך בוליאני A באורך m.

1. אפס את A .
2. לכל x בקלט:

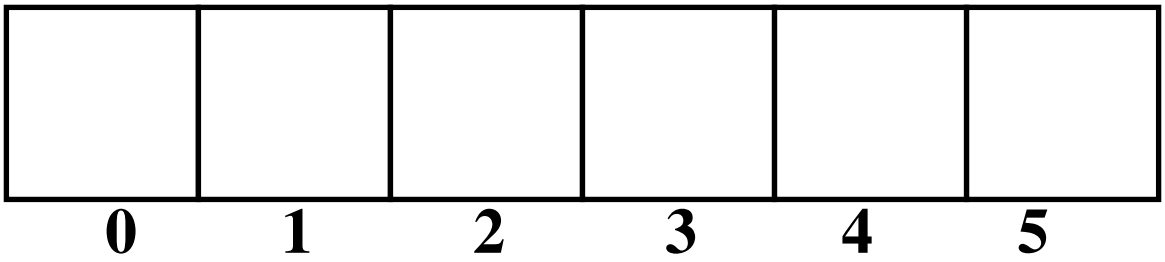
```
A[x] = 1
```

3. איסוף

```
for(i=0; i<m; i++){
   if(A[i] == 1)
         output i;
}
```

דוגמא m=6, n=4

קלט 5 3 2 1



פלט

זמן

איפוס המערך m

מעבר על הקלט n

איסוף m

סה"כ O(n+m)

אם m = O(n) נקבל שיטת מיון ליניארית

ריבוי רשומות אם יש מספר רשומות עם אותו המפתח

בנה מערך של m תורים.
בזמן האיסוף: שרשר את התורים.

הגדרה שיטת מיון תיקרא יציבה (stable) אם
`a[i].key == a[j].key` ו- `i < j`
אזי בפלט `a[i]` יקדים את `a[j]`.

טענה BucketSort היא שיטת מיון יציבה.

Radix Sort

BucketSort אינה יעילה כאשר $n << m$.
נניח כל מפתח מורכב מ-d ספרות (עשרוניות)
נמיין ספרה ספרה.

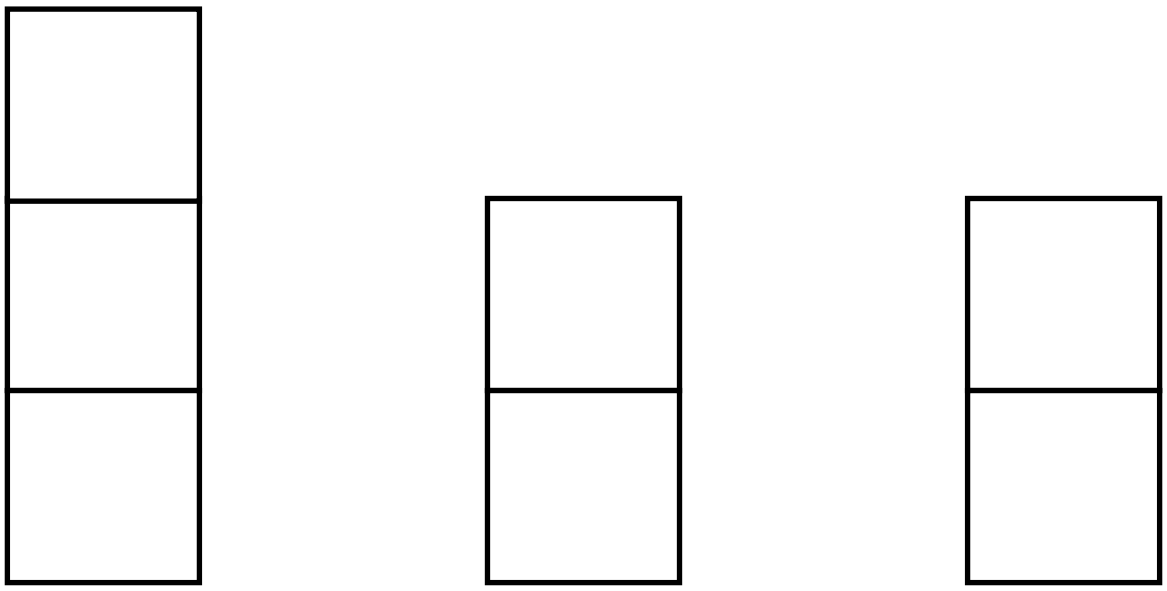
מאיזה כיוון להתחיל?

☆ LSD - Least Significant Digit ☆☆☆☆
MSD - Most Significant Digit

דוגמא

קלט (משמאל לימין)21 61 63 58 68 51 53

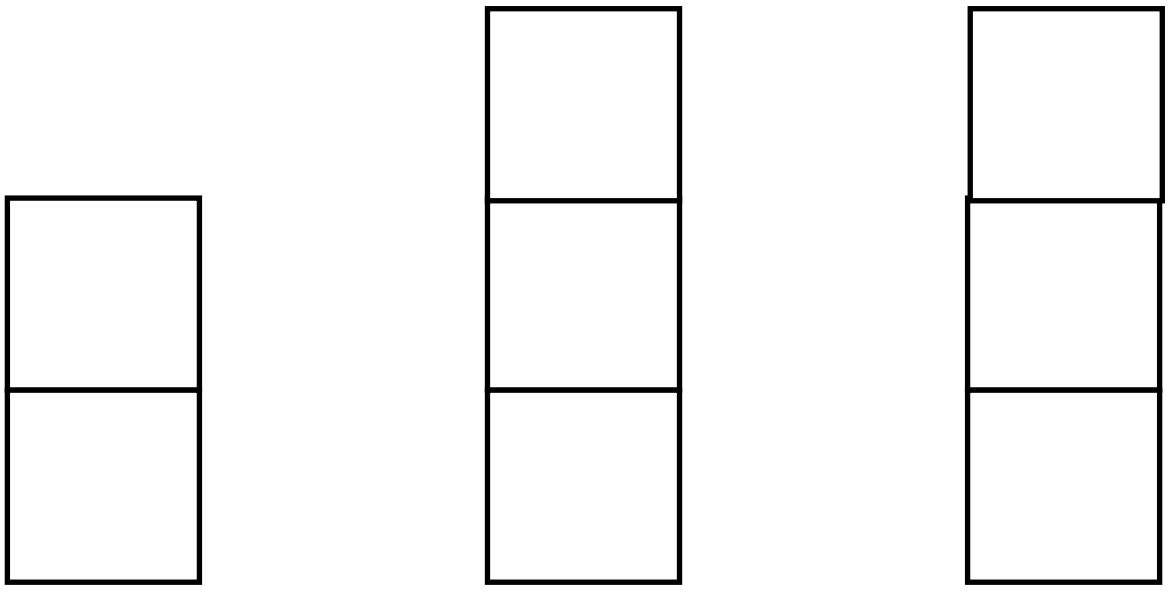
איטרציה ראשונה (ספרת היחידות)



1 3 8

איסוף 58 68 63 53 21 61 51

איטרציה שניה (ספרת העשרות)



2 5 6

איסוף 68 63 61 58 53 51 21

חשבון זמנים:

עבור בסיס (radix) 10 : d(10+n)

עבור בסיס (radix) r : d(r + n)

$$d(r + n) = (\log_r m)\,(r+n)$$
$$= (\log_2 m)\,(r+n) / (\log_2 r)$$

כאשר m הוא האיבר המכסימאלי בקלט.

עבור m = O(n) נקבל

$$O(n \log n / \log r)$$