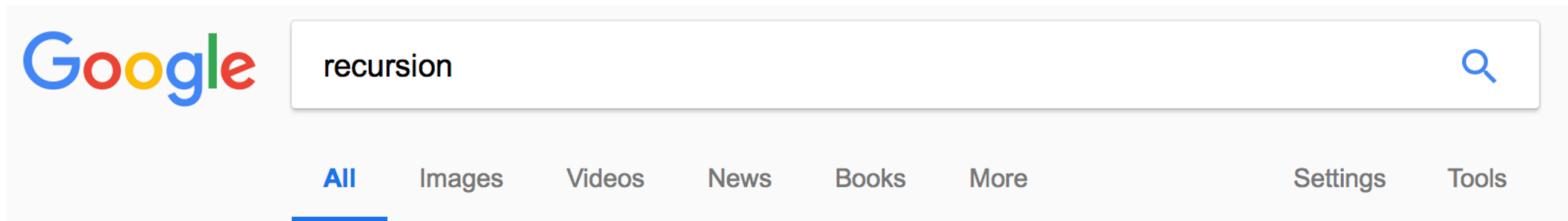


# מבוא לתכנות מתמטי

רקורסיה

# רקורסיה

**רִקּוּרְסִיָּה** (בעברית: [נסיגה](#)) היא תופעה שכל מופע שלה מכיל מופע נוסף שלה, כך שהיא מתרחשת ומשתקפת בשלמותה בתוך עצמה שוב ושוב. (ויקיפדיה)



About 10,300,000 results (0.49 seconds)

Did you mean: ***recursion***

# רקורסיה

- פונקציה שקוראת לעצמה
- סכימת הרקורסיה:
  - תנאי עצירה: זה מקרה פשוט בו הפתרון ידוע.
  - קריאה רקורסיבית עבור בעיה קטנה יותר: נניח שהפונקציה יודעת לפתור את הבעיה המוקטנת.
  - צעד הרקורסיה נפתור את הבעיה הגדולה על ידי שימוש בפתרון של הבעיה הקטנה.

```
% ex_1
% write a function that, given a number n,
% the function prints the n-th Fibonacci Number
%  $F(n) = F(n-1) + f(n-2)$ 
```

```
function fib = fib_7(n)
    if n <= 2
        fib = 1;
        return
    end
    fib = fib_7(n-1) + fib_7(n-2);
end
```

```
% calculate  
% m*n recursively, n > 0
```

```
function m_times_n = nult_7(m, n)  
if n == 0  
    m_times_n = 0;  
    return  
end  
m_times_n = m + nult_7(m, n-1);  
end
```

% write a function that receives two numbers a,b. and calculate a^b

```
function answer = power_7(num, pow)
% stop condition!
if pow == 0
    answer = 1;
    return
end
% recursive call
answer = num * power_7(num, pow - 1);
end
```

---

```
% GCD
% goal: finding gcd of two numbers a,b
%  $r + qb = a$ 
%  $\text{gcd}(b, r) = \text{gcd}(b, a)$ 
%  $\text{gcd}(b, \text{mod}(a,b)) = \text{gcd}(a,b)$ 
%  $\text{gcd}(a,0)=0$ 
% the recursive call will stop when  $r == 0$ 
```

```
function gcd_val = gcd_7(small, big)
% stop condition!
if small == 0
    gcd_val = big;
    return
end
gcd_val = gcd_7(mod(big, small), small)
end
```

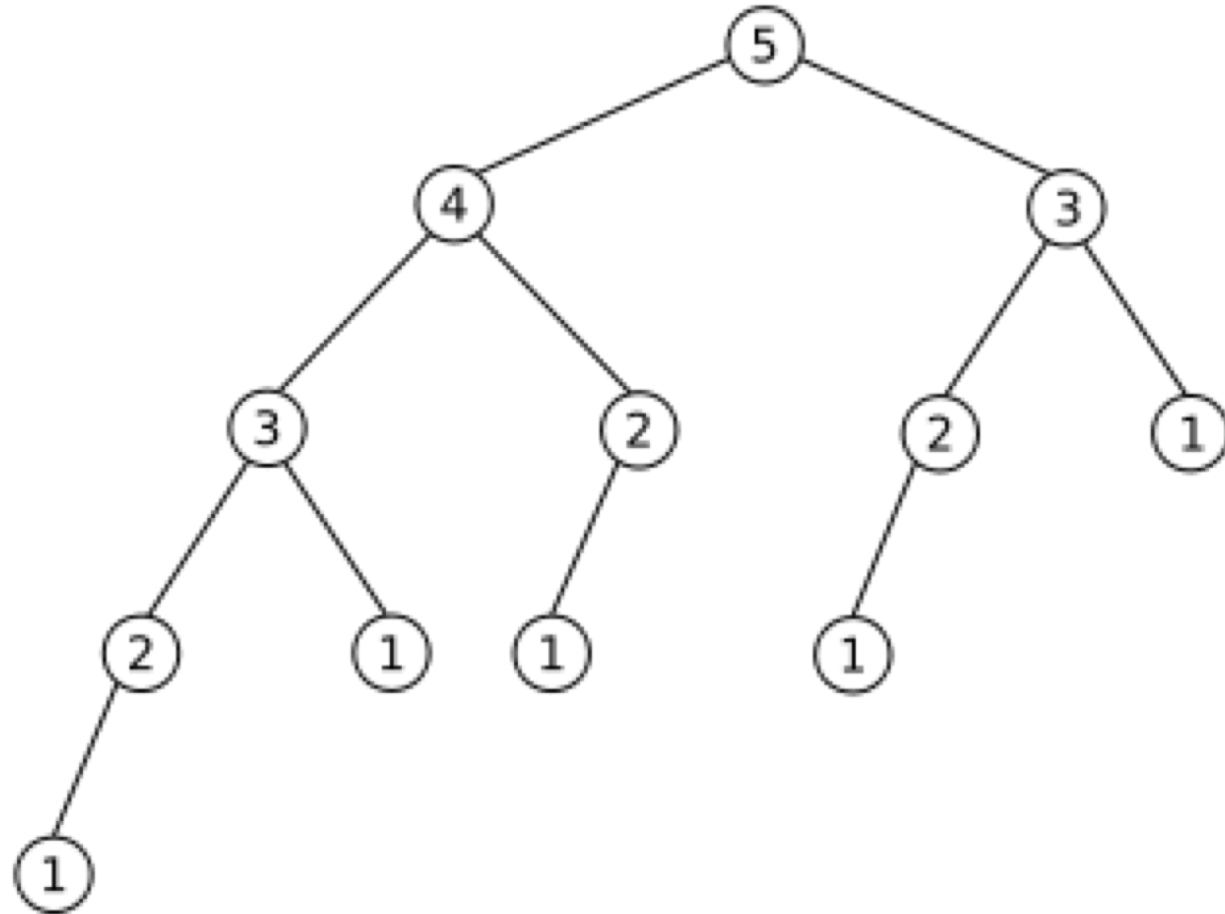
```
function found = binary_search_7(vec, num)
found=0;
% stop condition!
if length(vec)>=1

    % recursive step
    % search in only one side of the vec
    middle = ceil(length(vec) / 2);
    % search left side
    if num == vec(middle)
        found = 1;
        return
    % search left side
    elseif num < vec(middle)
        found = binary_search_7(vec(1:middle), num);
    % search right side
    else
        found = binary_search_7(vec(middle + 1:end), num);

    end
end
end
```



# בחזרה לסדרת פיבונאצ'י



```
function fib = fub_better(num)
```

```
    n=zeros(1,num)
```

```
    n(1) = 1;
```

```
    n(2) = 1;
```

```
    k=3;
```

```
    while k <= num
```

```
        n(k) = n(k-1)+n(k-2);
```

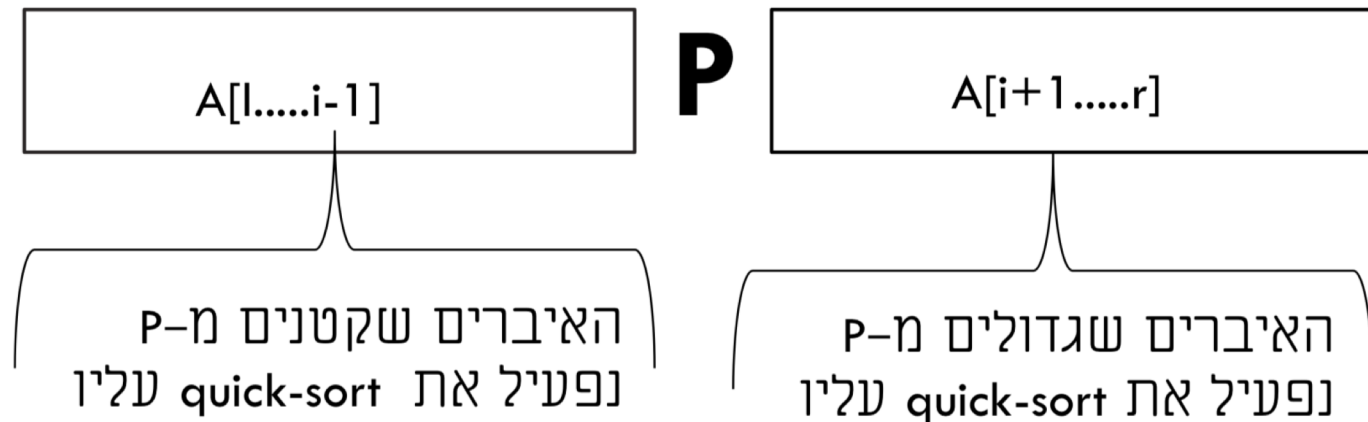
```
        k=k+1;
```

```
        fib=n
```

```
    end
```

# QUICK-SORT – הרעיון הבסיסי

- עובד באופן רקורסיבי – גישה של “הפרד ומשול”.
- הרעיון של האלגוריתם:
  - בכל שלב ממיינים מערך  $A[0..r]$
  - נבחר איבר מהמערך שאנחנו מימנים  $A[p]$  שיהיה איבר הציר (pivot)
  - נחלק את המערך ל 3 חלקים: האיבר שקטנים מאיבר הציר שהם בעצם  $A[0..i-1]$ , איבר הציר יהיה במקום ה  $n$ , האיברים שגדולים מאיבר הציר  $A[i+1..r]$ . (הפרד)
  - נמיין רקורסיבית את  $A[0..i-1]$  ואת  $A[i+1..r]$  (משול)



# QUICK-SORT – נשים לב

מצאנו את המיקום של הערך במערך הסופי!



**P**



האיברים שקטנים מ-P  
נפעיל את quick-sort עליו



האיברים שגדולים מ-P  
נפעיל את quick-sort עליו

3	1	4	5	9	2	8	6
---	---	---	---	---	---	---	---

1	2	3	4	5	9	8	6
---	---	---	---	---	---	---	---

1	2	3	4	5	9	8	6
---	---	---	---	---	---	---	---

1	2	3	4	5	9	8	6
---	---	---	---	---	---	---	---

1	2	3	4	5	8	6	9
---	---	---	---	---	---	---	---

1	2	3	4	5	6	8	9
---	---	---	---	---	---	---	---

1	2	3	4	5	6	8	9
---	---	---	---	---	---	---	---

```
function sortedArray = quickSort(array)

    if numel(array) <= 1 %If the array has 1 element then it can't be sorted
        sortedArray = array;
        return
    end

    pivot = array(end);
    array(end) = [];

    %Create two new arrays which contain the elements that are less than or
    %equal to the pivot called "less" and greater than the pivot called
    %"greater"
    less = array( array <= pivot );
    greater = array( array > pivot );

    %The sorted array is the concatenation of the sorted "less" array, the
    %pivot and the sorted "greater" array in that order
    sortedArray = [quickSort(less) pivot quickSort(greater)];

end
```

9. הדפס את כל הפרמוטציות של מערך

```
function [ ] = permutations( v, tail )
%print all permutations of v followed by tail
% example: permutations(1:5, []);

n=length(v);
if n==1
    disp([v tail]);
else
    for i=1:n
        %add v(i) to the beginning of tail
        permutations([v(1:i-1) v(i+1:n)], [v(i) tail]);
    end
end
end
```