CSC 250: Foundations of Computer Science I

Fall 2023 - Lecture 3

Amitabha Dey

Department of Computer Science
University of North Carolina at Greensboro

Iteration

Selection Sort: An Iterative Sorting Algorithm

Selection sort is a simple sorting algorithm that repeatedly selects the smallest (or largest) element from the unsorted portion of the list and swaps it with the element at the beginning of the sorted portion. This process is repeated until the entire list is sorted.

Initialization:

- Start with the first element of the list as the "sorted portion."
- The rest of the list is the "unsorted portion."

Find the Minimum:

- Scan through the unsorted portion to find the smallest element.

Swap:

- Swap the found minimum element with the first element of the unsorted portion.

Expand the Sorted Portion:

- Now, the first element of the list is a part of the sorted portion.
- The rest of the sorted portion is one element larger.

Repeat:

- Repeat steps 2 to 4 for the remaining unsorted portion until the entire list is sorted.

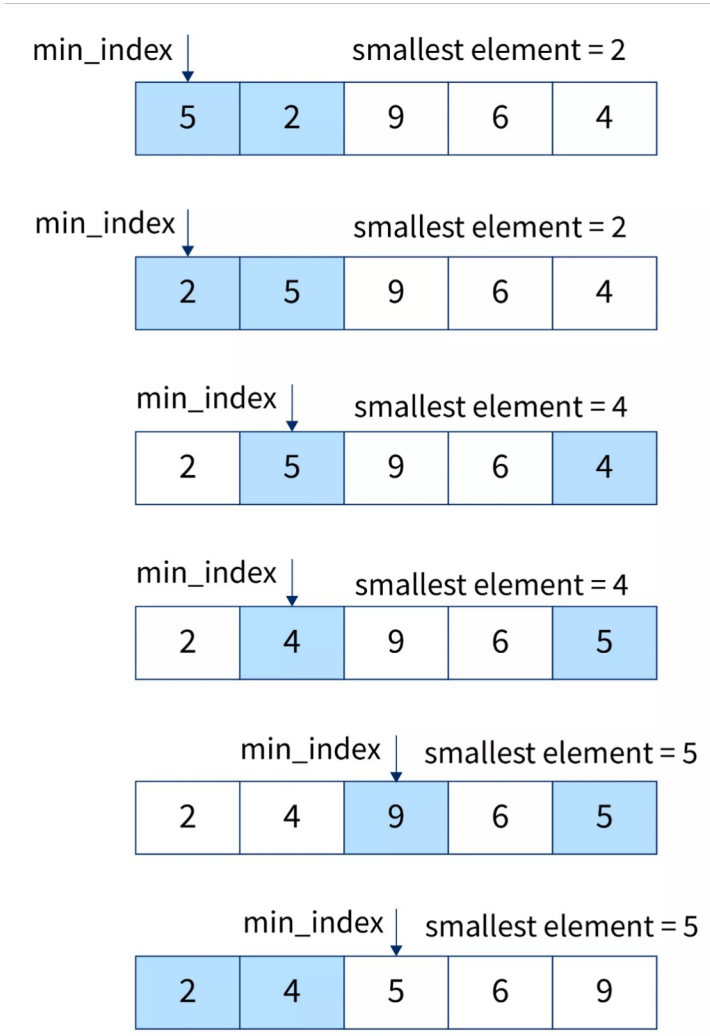


Image credit - https://www.scaler.com/

Iteration

Selection Sort: An Iterative Sorting Algorithm

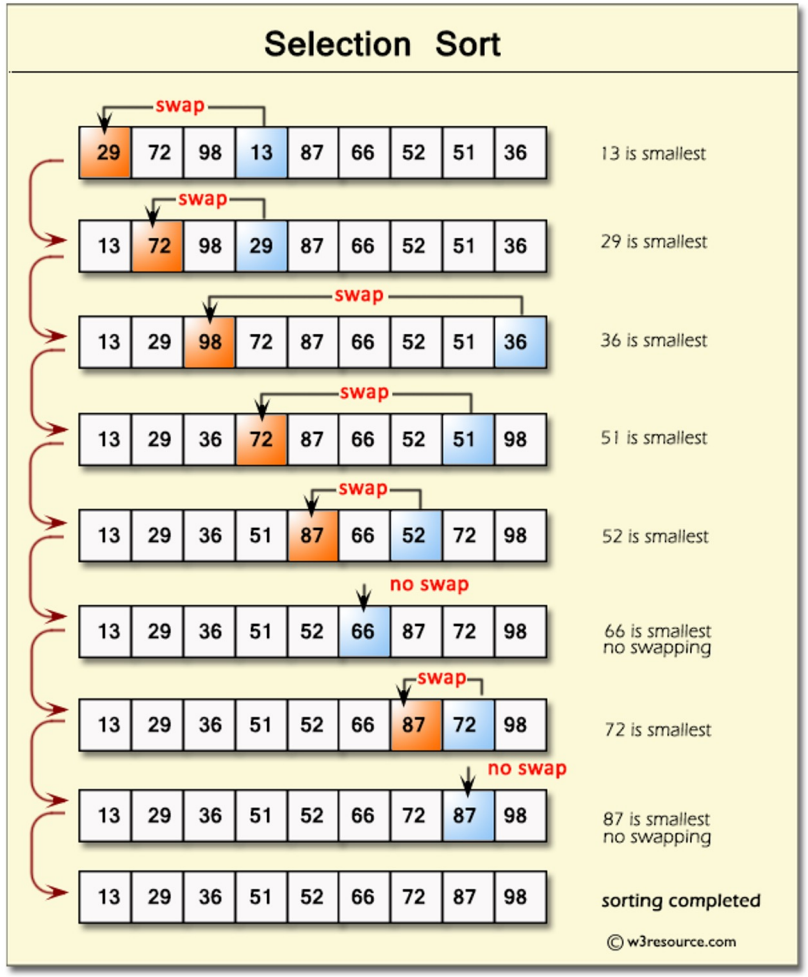


Image credit - https://www.w3resource.com/

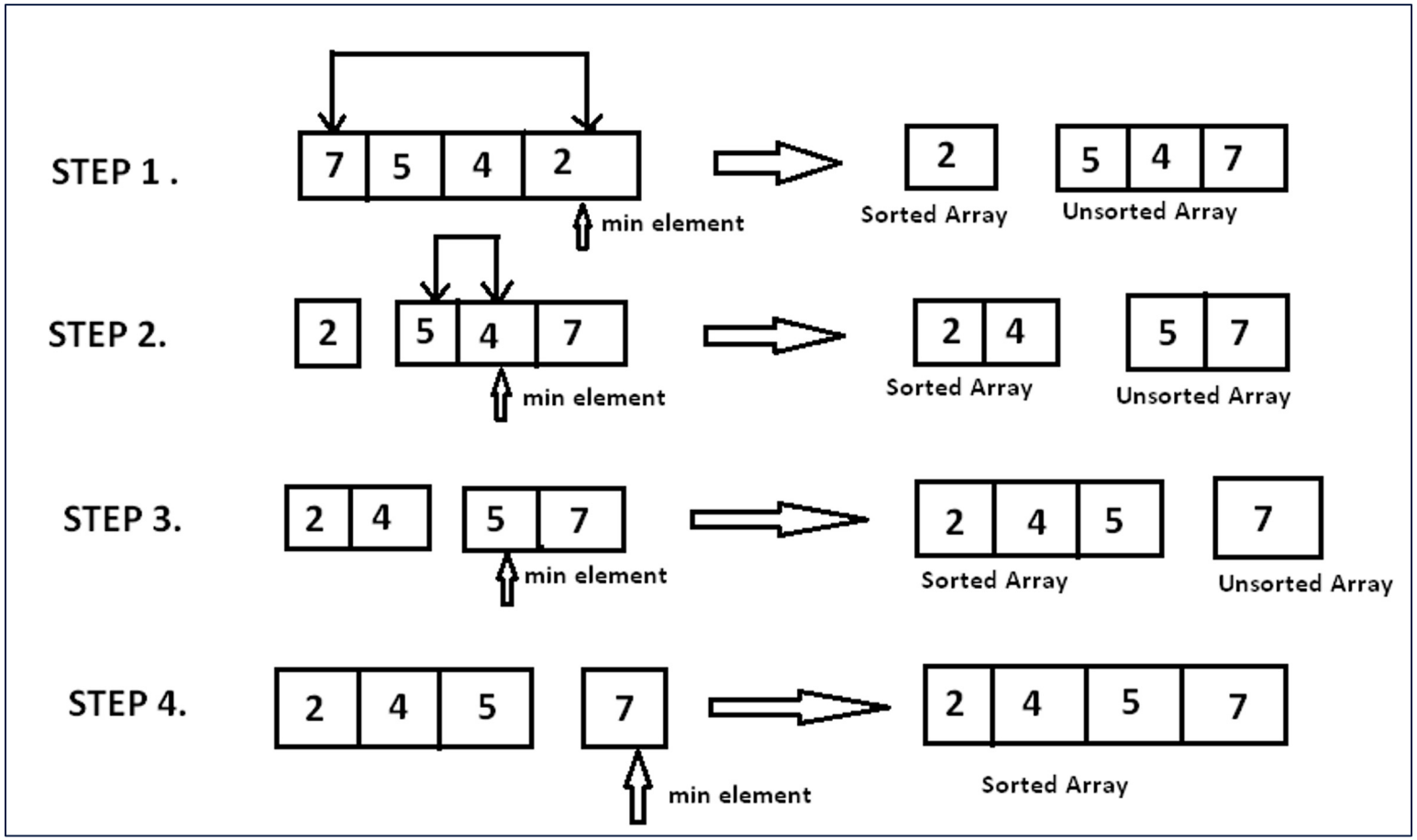


Image credit - https://www.hackerearth.com/

Induction

To understand how induction works, suppose there is a professor who brings to class a bottomless bag of assorted miniature candy bars. She offers to share the candy in the following way. First, she lines the students up in order. Next she states two rules:

1. The student at the beginning of the line gets a candy bar.
2. If a student gets a candy bar, then the following student in line also gets a candy bar.

Let's number the students by their order in line, starting the count with 0, as usual in computer science. Now we can understand the second rule as a short description of a whole sequence of statements:

- If student 0 gets a candy bar, then student 1 also gets one.
- If student 1 gets a candy bar, then student 2 also gets one.
- If student 2 gets a candy bar, then student 3 also gets one.

$\vdots$

Of course this sequence has a more concise mathematical description:

> If student n gets a candy bar, then student $n+1$ gets a candy bar, for all nonnegative integers n.

So suppose you are student 17. By these rules, are you entitled to a miniature candy bar? Well, student 0 gets a candy bar by the first rule. Therefore, by the second rule, student 1 also gets one, which means student 2 gets one, which means student 3 gets one as well, and so on. By 17 applications of the professor's second rule, you get your candy bar! Of course the rules actually guarantee a candy bar to *every* student, no matter how far back in line they may be.

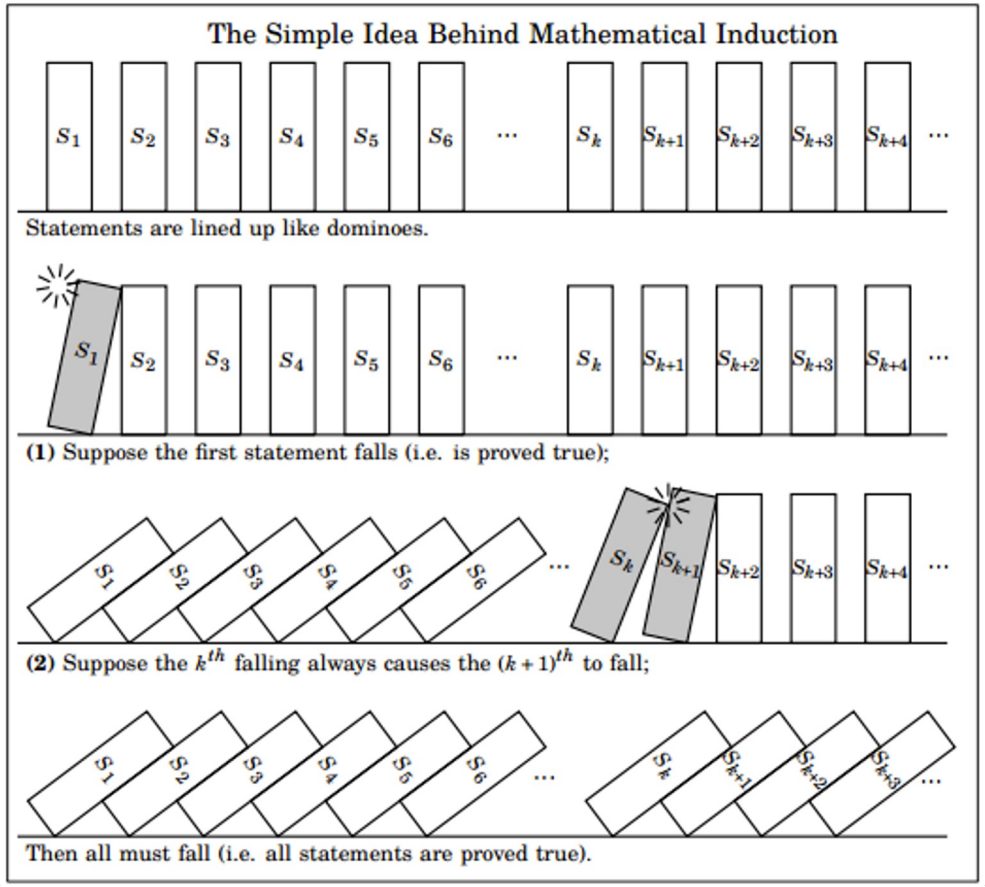


Image credit - Book of Proof by Richard Hammack

Induction
Proof by Mathematical Induction

Mathematical induction is a useful technique for proving that a statement S(n) is true for all nonnegative integers n, or, more generally, for all integers at or above some lower limit.
Steps to perform proof by Mathematical Induction:

- Step 1 Basis (base case): Prove that the statement holds for the first natural number n. Usually, n = 0 or n = 1.
- Step 2 Inductive Hypothesis: Assume that the statement is true for
- n = k
- Step 3 Inductive Step: If it is true for n=k, prove that the statement is true for n = k+1. Plug in k+1, and then simplify or rearrange RHS such that LHS = RHS. When both sides are equal, the inductive step is completed.
- Step 4 Conclusion: Invoke the Induction. By mathematical induction, it is proved that the statement is true for all nonnegative n.

❑ Show it is true for the first one
❑ Show that if any one is true then the next one is true
Then all are true

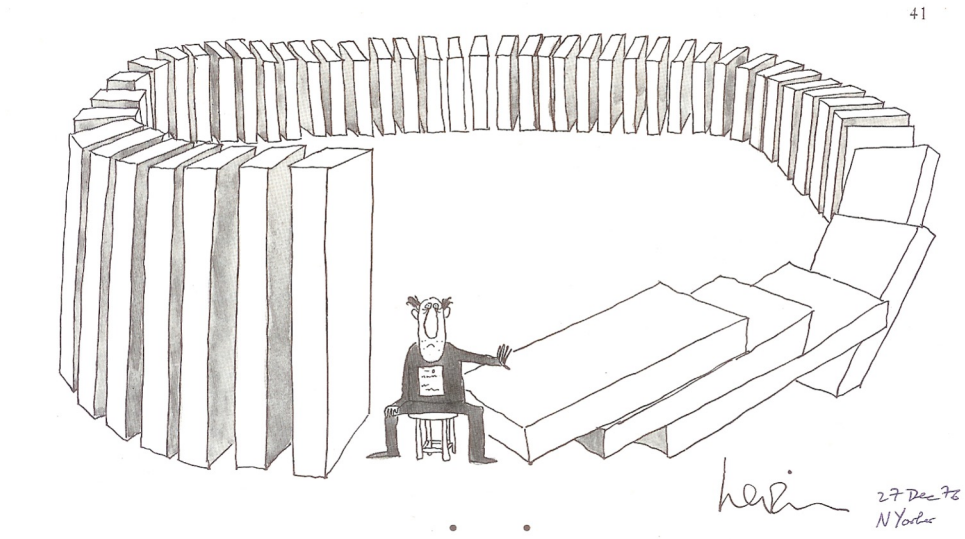
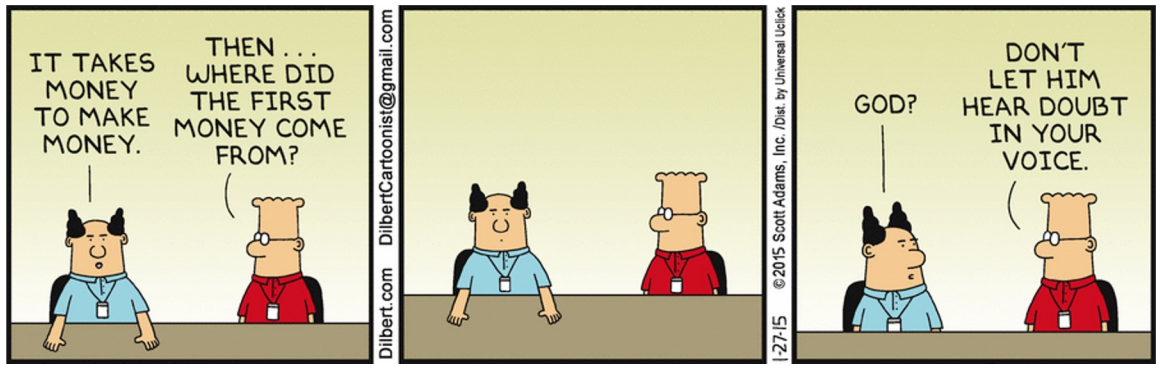
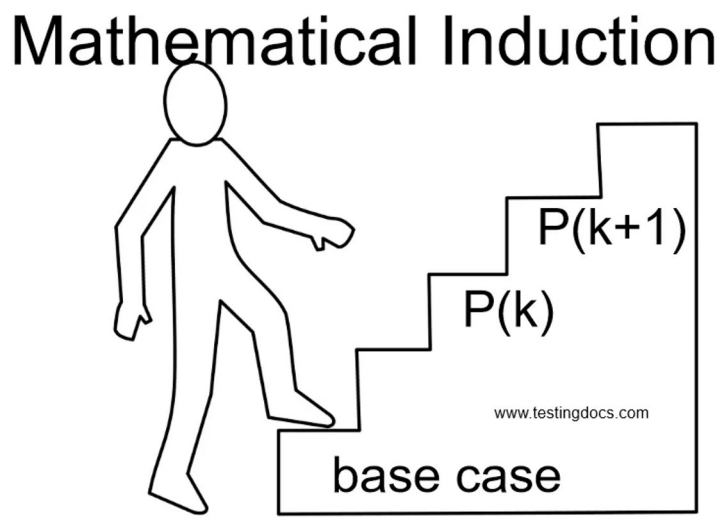


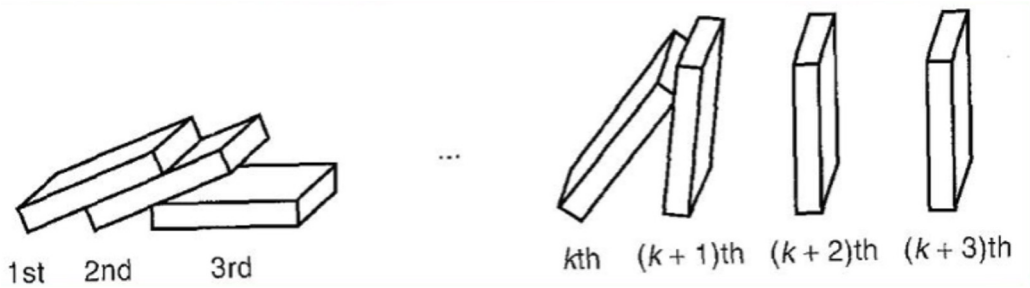
Image credit - https://cse.buffalo.edu/



Induction

Induction Proofs Template





1. **State that the proof uses induction.** This immediately conveys the overall structure of the proof, which helps your reader follow your argument.

2. **Define an appropriate predicate $P(n)$.** The predicate $P(n)$ is called the *induction hypothesis*. The eventual conclusion of the induction argument will be that $P(n)$ is true for all nonnegative n. Clearly stating the induction hypothesis is often the most important part of an induction proof, and omitting it is the largest source of confused proofs by students.

   In the simplest cases, the induction hypothesis can be lifted straight from the proposition you are trying to prove, as we did with equation (6.1). Sometimes the induction hypothesis will involve several variables, in which case you should indicate which variable serves as n.

3. **Prove that $P(0)$ is true.** This is usually easy, as in the example above. This part of the proof is called the *base case* or *basis step*.

4. **Prove that $P(n)$ implies $P(n+1)$ for every nonnegative integer n.** This is called the *inductive step*. The basic plan is always the same: assume that $P(n)$ is true and then use this assumption to prove that $P(n+1)$ is true. These two statements should be fairly similar, but bridging the gap may require some ingenuity. Whatever argument you give must be valid for every nonnegative integer n, since the goal is to prove the implications $P(0) \rightarrow P(1)$, $P(1) \rightarrow P(2)$, $P(2) \rightarrow P(3)$, etc. all at once.

5. **Invoke induction.** Given these facts, the induction principle allows you to conclude that $P(n)$ is true for all nonnegative n. This is the logical capstone to the whole argument, but it is so standard that it's usual not to mention it explicitly.

Always be sure to explicitly label the *base case* and the *inductive step*. It will make your proofs clearer, and it will decrease the chance that you forget a key step (such as checking the base case).

Image credit - Mathematics for Computer Science, Eric Lehman (Google Inc.)

Induction

Proof by Mathematical Induction

Shows $5+10+15+\cdots+5n=\frac{5n(n+1)}{2}$ is true for all positive integers.

(Basis) When $n=1$ then $5(1)=\frac{5(1)((1)+1)}{2}$, which follows that $5=\frac{5(2)}{2}=5$ is true

(Hypothesis) Assuming $n=k$ such that $5+10+15+\cdots+5k=\frac{5k(k+1)}{2}$ true when $k\geq 1$

(Inductive) We want to show that $n=k+1$ such that $5+10+15+\cdots+5(k+1)=\frac{5(k+1)((k+1)+1)}{2}$ is also true when $k\geq 1$

$$\begin{aligned}5+10+15+\cdots+5(k+1) &= 5+10+15+\cdots+5k+5(k+1)\\ &= \frac{5k(k+1)}{2}+5(k+1), \text{ using the inductive hypothesis}\\ &= \frac{5k(k+1)}{2}+\frac{10(k+1)}{2}=\frac{5k(k+1)+10(k+1)}{2}\\ &= \frac{5(k+1)[k+2]}{2}\\ &= \frac{5(k+1)((k+1)+1)}{2}\end{aligned}$$

By induction we have shown $5+10+15+\cdots+5n=\frac{5n(n+1)}{2}$ is true for all positive intergers

Show $n<2^n$ for $n\geq 1$ using mathematical induction

(Basis) When $n=1$ then $(1)<2^{(1)}$ which follows that $1<2$ is true

(Hypothesis) Assuming $n=k$ such that $k<2^k$ true when $k\geq 1$

(Inductive) We want to show that $n=k+1$ such that $(k+1)<2^{k+1}$ is also true when $k\geq 1$.

$$\begin{aligned}k+1 &< k+k\\ &< \left(2^k\right)+\left(2^k\right), \text{ using the inductive hypothesis}\\ &< 2\left(2^k\right)\\ &< 2^{k+1}\end{aligned}$$

By induction we have shown $n<2^n$ for $n\geq 1$

Image credit - https://calcworkshop.com/

Induction

Proof by Mathematical Induction

Theorem
For any $n \in \mathbb{N}$,

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

Proof

Base case $n = 1$: If $n = 1$, the left side is 1 and the right side is $\frac{1(2)}{2} = 1$.

So, the theorem holds when $n = 1$.

Inductive hypothesis: Suppose the theorem holds for all values of n up to some k, $k \geq 1$.

Inductive step: Let $n = k + 1$. Then our left side is

$$\begin{aligned}
\sum_{i=1}^{k+1} i &= (k+1) + \sum_{i=1}^{k} i \\
&= (k+1) + \frac{k(k+1)}{2}, \text{ by our inductive hypothesis} \\
&= \frac{2(k+1)}{2} + \frac{k(k+1)}{2} \\
&= \frac{2(k+1) + k(k+1)}{2} \\
&= \frac{(k+1)(k+2)}{2}
\end{aligned}$$

which is our right side. So, the theorem holds for $n = k + 1$. By the principle of mathematical induction, the theorem holds for all $n \in \mathbb{N}$.

Image credit - https://bgrasley.wordpress.com/

Question 1. *Prove using mathematical induction that for all $n \geq 1$,*

$$1 + 4 + 7 + \cdots + (3n - 2) = \frac{n(3n-1)}{2}.$$

Solution.

For any integer $n \geq 1$, let P_n be the statement that

$$1 + 4 + 7 + \cdots + (3n - 2) = \frac{n(3n-1)}{2}.$$

Base Case. The statement P_1 says that

$$1 = \frac{1(3-1)}{2},$$

which is true.

Inductive Step. Fix $k \geq 1$, and suppose that P_k holds, that is,

$$1 + 4 + 7 + \cdots + (3k - 2) = \frac{k(3k-1)}{2}.$$

It remains to show that P_{k+1} holds, that is,

$$1 + 4 + 7 + \cdots + (3(k+1) - 2) = \frac{(k+1)(3(k+1)-1)}{2}.$$

$$\begin{aligned}
1 + 4 + 7 + \cdots + (3(k+1) - 2) &= 1 + 4 + 7 + \cdots + (3(k+1) - 2) \\
&= 1 + 4 + 7 + \cdots + (3k+1) \\
&= 1 + 4 + 7 + \cdots + (3k-2) + (3k+1) \\
&= \frac{k(3k-1)}{2} + (3k+1) \\
&= \frac{k(3k-1) + 2(3k+1)}{2} \\
&= \frac{3k^2 - k + 6k + 2)}{2} \\
&= \frac{3k^2 + 5k + 2)}{2} \\
&= \frac{(k+1)(3k+2)}{2} \\
&= \frac{(k+1)(3(k+1)-1)}{2}.
\end{aligned}$$

Therefore P_{k+1} holds.

Thus, by the principle of mathematical induction, for all $n \geq 1$, P_n holds.

Image credit - University of Manitoba

Induction

Proof by Mathematical Induction

Use mathematical induction to prove that

$1 + 2 + 3 + \ldots + n = n (n + 1) / 2$

for all positive integers n.

Solution to Problem 1:

Let the statement P (n) be

$1 + 2 + 3 + \ldots + n = n (n + 1) / 2$

STEP 1: We first show that p (1) is true.

Left Side = 1

Right Side = $1 (1 + 1) / 2 = 1$

Both sides of the statement are equal hence p (1) is true.

STEP 2: We now assume that p (k) is true

$1 + 2 + 3 + \ldots + k = k (k + 1) / 2$

and show that p (k + 1) is true by adding k + 1 to both sides of the above statement

$1 + 2 + 3 + \ldots + k + (k + 1) = k (k + 1) / 2 + (k + 1)$

$= (k + 1)(k / 2 + 1)$

$= (k + 1)(k + 2) / 2$

The last statement may be written as

$1 + 2 + 3 + \ldots + k + (k + 1) = (k + 1)(k + 2) / 2$

Which is the statement p(k + 1).

Prove that for any positive integer number n , **$n^3 + 2 n$** is divisible by 3

Solution to Problem 4:

Statement P (n) is defined by

$n^3 + 2 n$ is divisible by 3

STEP 1: We first show that p (1) is true. Let n = 1 and calculate $n^3 + 2n$

$1^3 + 2(1) = 3$

3 is divisible by 3

hence p (1) is true.

STEP 2: We now assume that p (k) is true

$k^3 + 2 k$ is divisible by 3

is equivalent to

$k^3 + 2 k = 3 M$, where M is a positive integer.

We now consider the algebraic expression $(k + 1)^3 + 2 (k + 1)$; expand it and group like terms

$(k + 1)^3 + 2 (k + 1) = k^3 + 3 k^2 + 5 k + 3$

$= [k^3 + 2 k] + [3 k^2 + 3 k + 3]$

$= 3 M + 3 [k^2 + k + 1] = 3 [M + k^2 + k + 1]$

Hence $(k + 1)^3 + 2 (k + 1)$ is also divisible by 3 and therefore statement P(k + 1) is true.

1. Using the principle of mathematical induction, prove that $n(n + 1)(n + 5)$ is a multiple of 3 for all $n \in N$.

Solution:

Let P(n): $n(n + 1)(n + 5)$ is a multiple of 3.

For n = 1, the given expression becomes $(1 \times 2 \times 6) = 12$, which is a multiple of 3.

So, the given statement is true for n = 1, i.e. P(1) is true.

Let P(k) be true. Then,

P(k): $k(k + 1)(k + 5)$ is a multiple of 3

$\Rightarrow K(k + 1)(k + 5) = 3m$ for some natural number m, ... (i)

Now, $(k + 1l)(k + 2)(k + 6) = (k + 1)(k + 2)k + 6(k + 1)(k + 2)$

$= k(k + 1)(k + 2) + 6(k + 1)(k + 2)$

$= k(k + 1)(k + 5 - 3) + 6(k + 1)(k + 2)$

$= k(k + 1)(k + 5) - 3k(k + 1) + 6(k + 1)(k + 2)$

$= k(k + 1)(k + 5) + 3(k + 1)(k +4)$ [on simplification]

$= 3m + 3(k + 1)(k + 4)$ [using (i)]

$= 3[m + (k + 1)(k + 4)]$, which is a multiple of 3

$\Rightarrow$ P(k + 1): $(k + 1)(k + 2)(k + 6)$ is a multiple of 3

$\Rightarrow$ P(k + 1) is true, whenever P(k) is true.

Thus, P(1) is true and P(k + 1) is true, whenever P(k) is true.

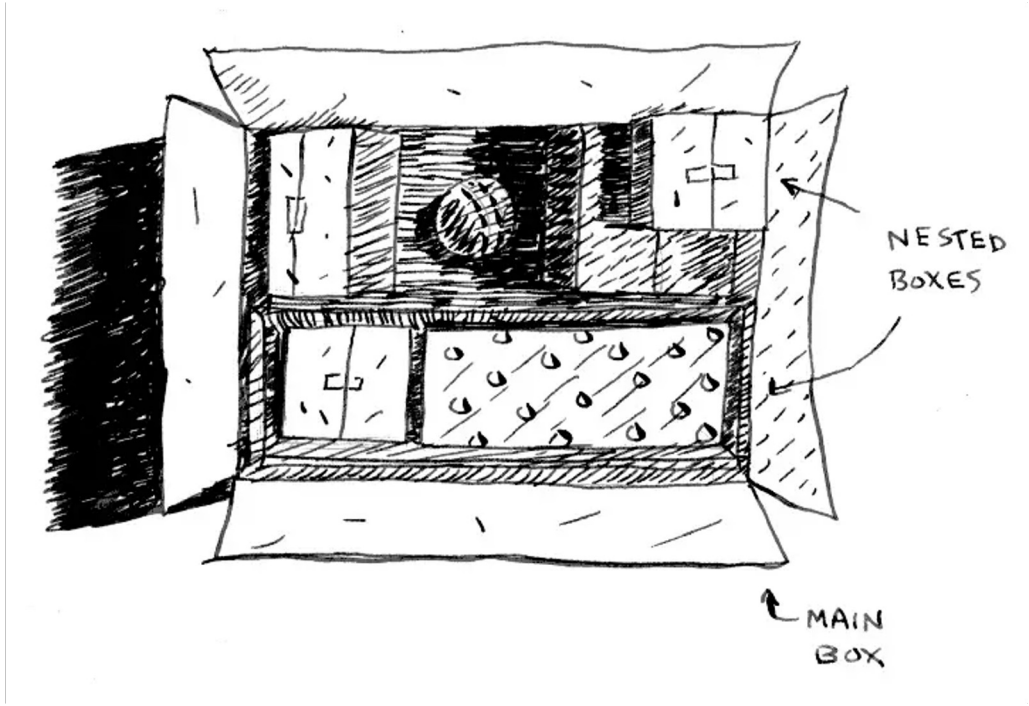
Hence, by the principle of mathematical induction, P(n) is true for all $n \in N$.

Image credit - https://www.analyzemath.com/

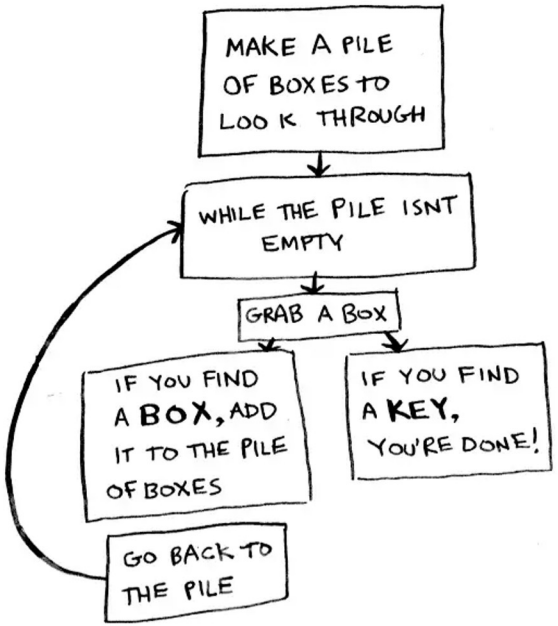
Image credit - https://www.math-only-math.com/

Recursion

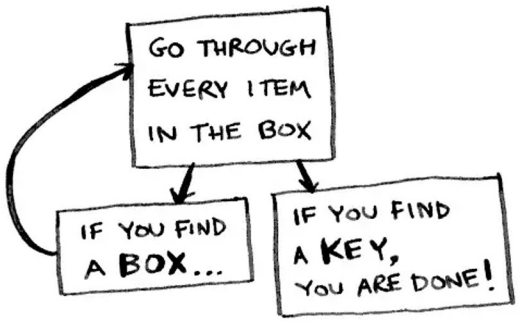
Imagine you go to open your bedroom door and it's locked.
Your three-year-old son pops in from around the corner and let's you know he hid the only key in a box. You open the box only to find… more boxes. Boxes inside of boxes. And you don't know which one has the key!



Iterative Approach



Recursive Approach



All illustrations by Adit Bhargava (Beau Carnes @ Medium)

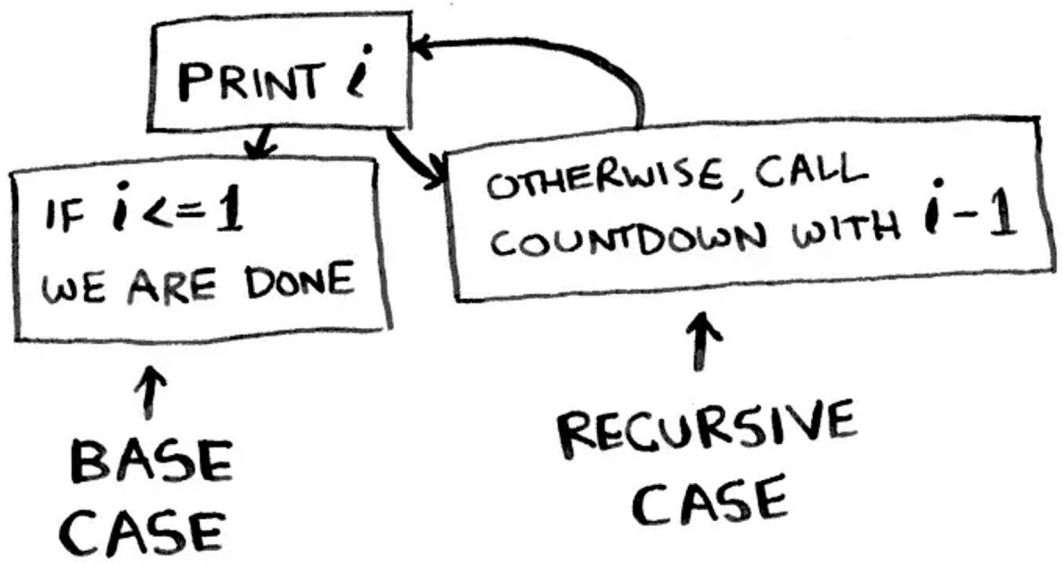
Recursion

Something you have to look out for when writing a recursive function is an infinite loop. This is when the function keeps calling itself… and never stops calling itself!

For instance, you may want to write a countdown function. This function will keep counting down forever.

A recursive function always has to say when to stop repeating itself. There should always be two parts to a recursive function: the **recursive case** and the **base case**. The recursive case is when the function calls itself. The base case is when the function stops calling itself. This prevents infinite loops.

Here is the countdown function again, with a base case:



All illustrations by Adit Bhargava (Beau Carnes @ Medium)

Recursion

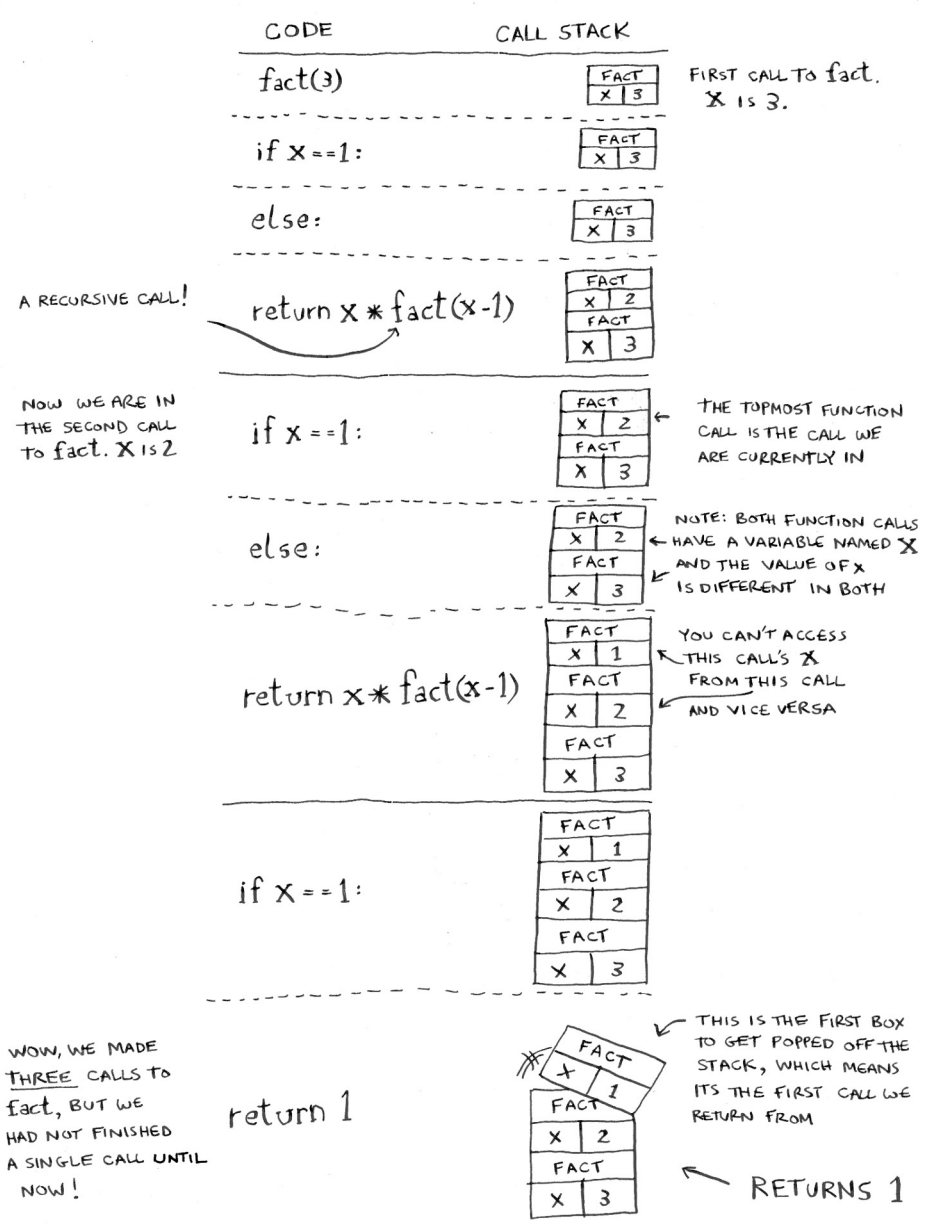
Recursive functions use something called "**the call stack**." When a program calls a function, that function goes on top of the call stack. This similar to a stack of books. You add things one at a time. Then, when you are ready to take something off, you always take off the top item.

Here is a recursive function to calculate the factorial of a number:

```
function fact(x) {
  if (x == 1) {
    return 1;
  } else {
    return x * fact(x-1);
  }
}
```

Now let's see what happens if you call fact(3). The topmost box in the stack tells you what call to fact you're currently on.

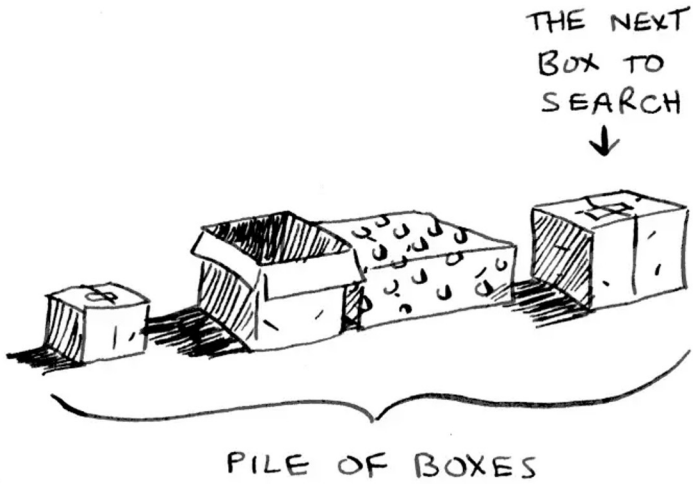
Notice how each call to fact has its own copy of x. This is very important to making recursion work. You can't access a different function's copy of x.



All illustrations by Adit Bhargava (Beau Carnes @ Medium)

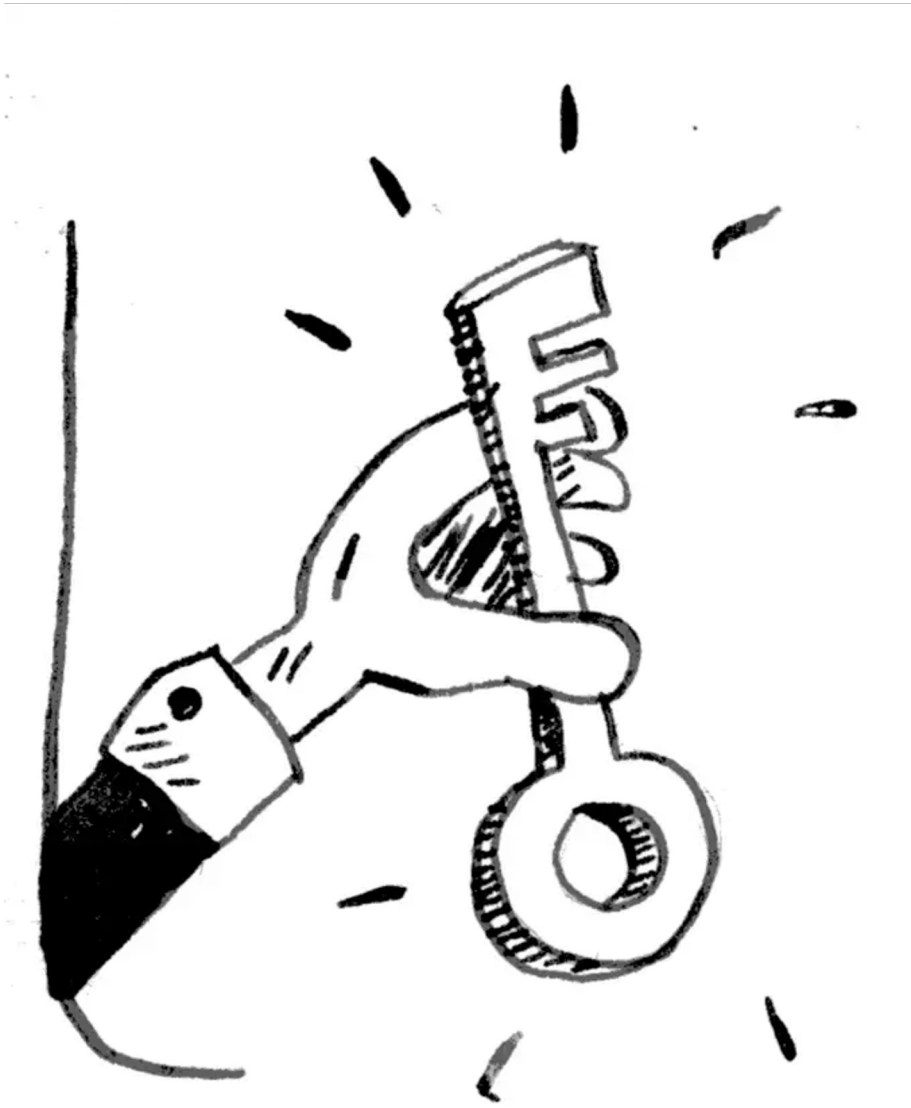
Recursion

With an iterative approach, you make a pile of boxes to search through, so you always know what boxes you still need to search.



But there is no pile in the recursive approach. How does your algorithm know which boxes you still have to look though? The “pile of boxes” is saved on the stack.

This is a stack of half-completed function calls, each with its own half-complete list of boxes to look through. The stack keeps track of the pile of boxes for you!



All illustrations by Adit Bhargava (Beau Carnes @ Medium)

Recursion

Recursive Algorithm Template

```
function countdown(num) {
    if (num <=0) {
        console.log("You've reached the end");
        return;
    }

    console.log(num);
    num--

    countdown(num)
}
```

Base Case

Recursive function

Image credit - https://dev.to/

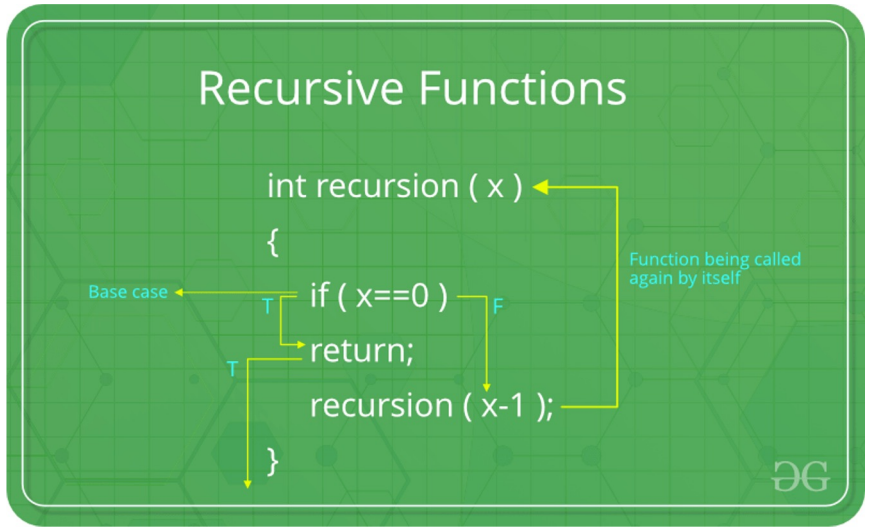


Image credit - https://www.geeksforgeeks.org/

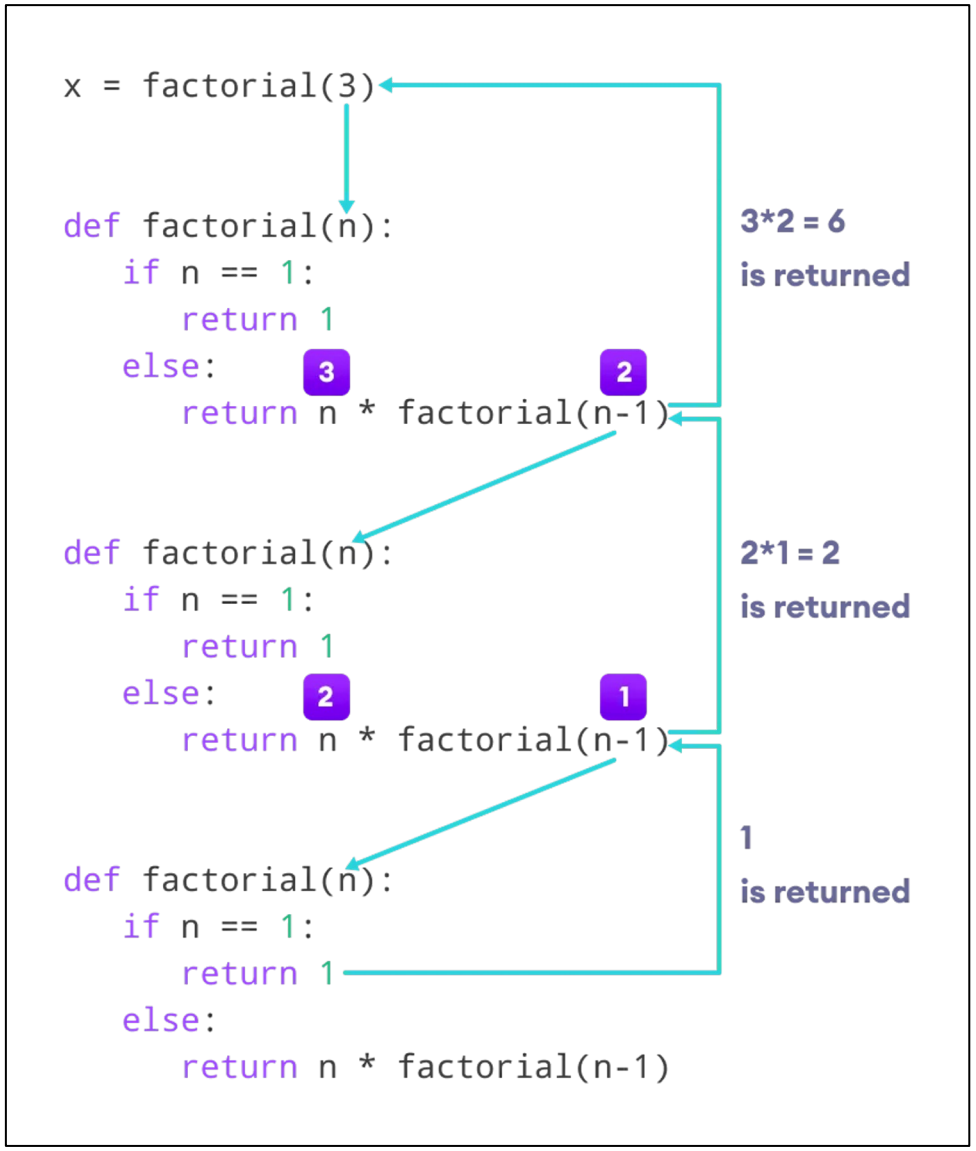


Image credit - https://www.programiz.com/

Recursion
Finding a Palindrome

A **palindrome** is a word that is spelled the same forward and backward. For example, *rotor* is a palindrome, but *motor* is not.

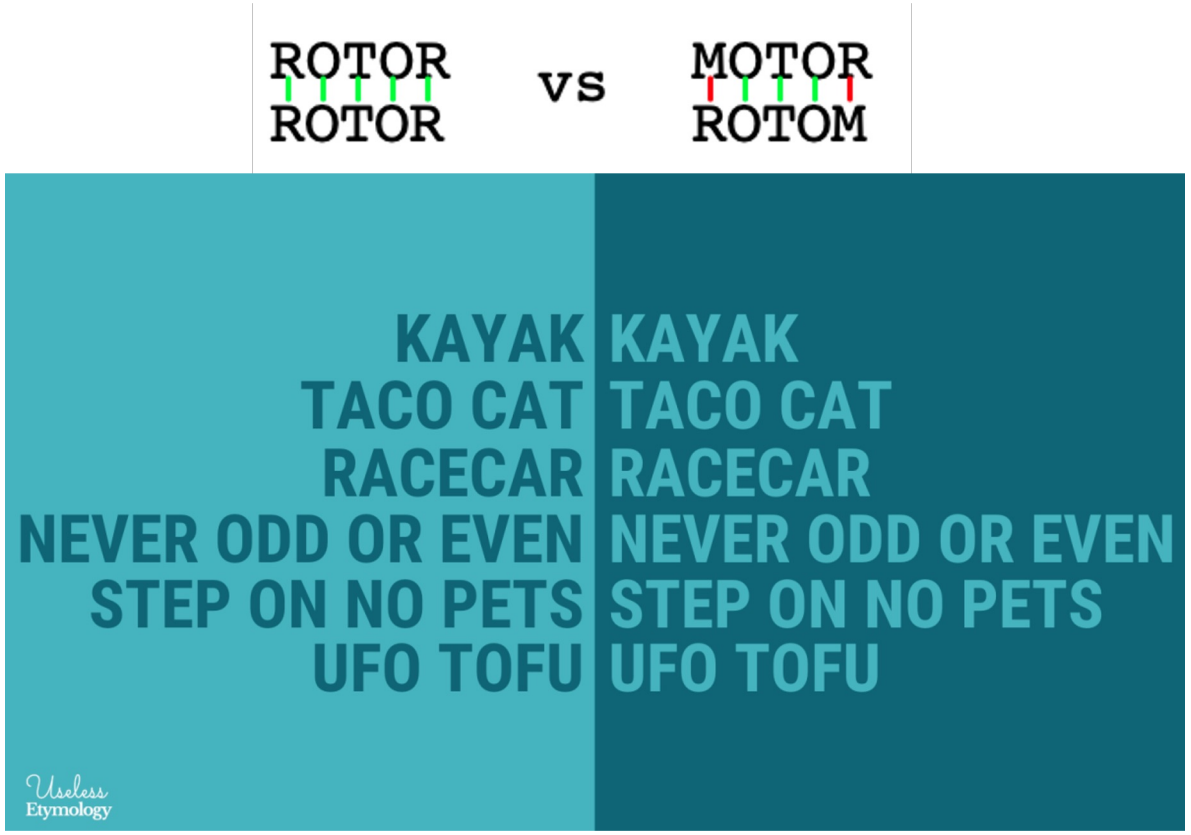
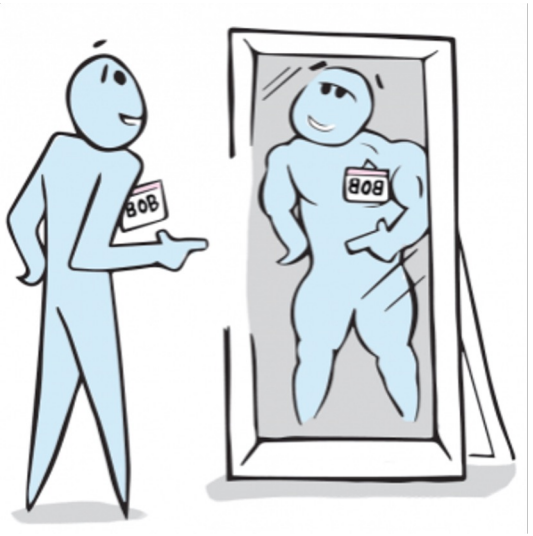


Image credit - https://uselessetymology.com/



Step 1: Establish a Base Case

Consider the word *a*. It's a palindrome. We can say that any string containing just one letter is by default a palindrome.

Now, a string can contain no letters; we call a string of zero letters an empty string. An empty string is also a palindrome, since it "reads" the same forward and backward.

So now let's say that any string containing at most one letter is a palindrome. That's our base case: a string with exactly zero letters or one letter is a palindrome.

Recursion
Finding a Palindrome

Step 2: Build the Recursive Case

What if the string contains two or more letters? Consider the palindrome *rotor*. Its first and last letters are the same, and this trait must hold for any palindrome.

On the other hand, if the first and last letters are not the same, as in *motor*, then the string cannot possibly be a palindrome. So now we have a way to declare that a string is not a palindrome: when its first and last letters are different. We can think of this situation as another base case, since we have the answer immediately.

Going back to when the first and last letters are the same, what does that tell us? The string might be a palindrome. Then again, it might not be. In the string *rater*, the first and last letters are the same, but the string is not a palindrome.

Suppose we strip off the first and last letters, leaving the string *ate*. Then the first and last letters of this remaining string are not the same, and so we know that rater is not a palindrome.

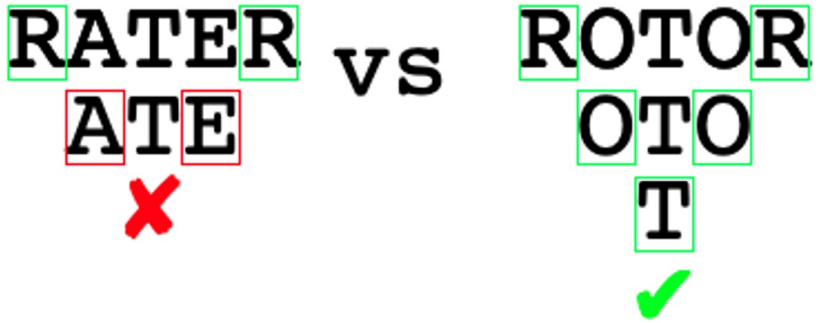


Image credit - https://www.khanacademy.org/

If the first and last letters differ, then declare that the string is not a palindrome. Otherwise, strip off the first and last letters, and determine whether the string that remains—the subproblem—is a palindrome.

Declare the answer for the shorter string to be the answer for the original string. Once you get down to a string with no letters or just one letter, declare it to be a palindrome.

Recursion
Finding a Palindrome

```
function palindromeCheck(word) {
  if (word.length === 1 || 0) {
    return true;
  } else {
    if (word[0] === word[word.length - 1]) {
      let newWord = word.slice(1, -1);
      return palindromeCheck(newWord);
    } else {
      return false;
    }
  }
}
```

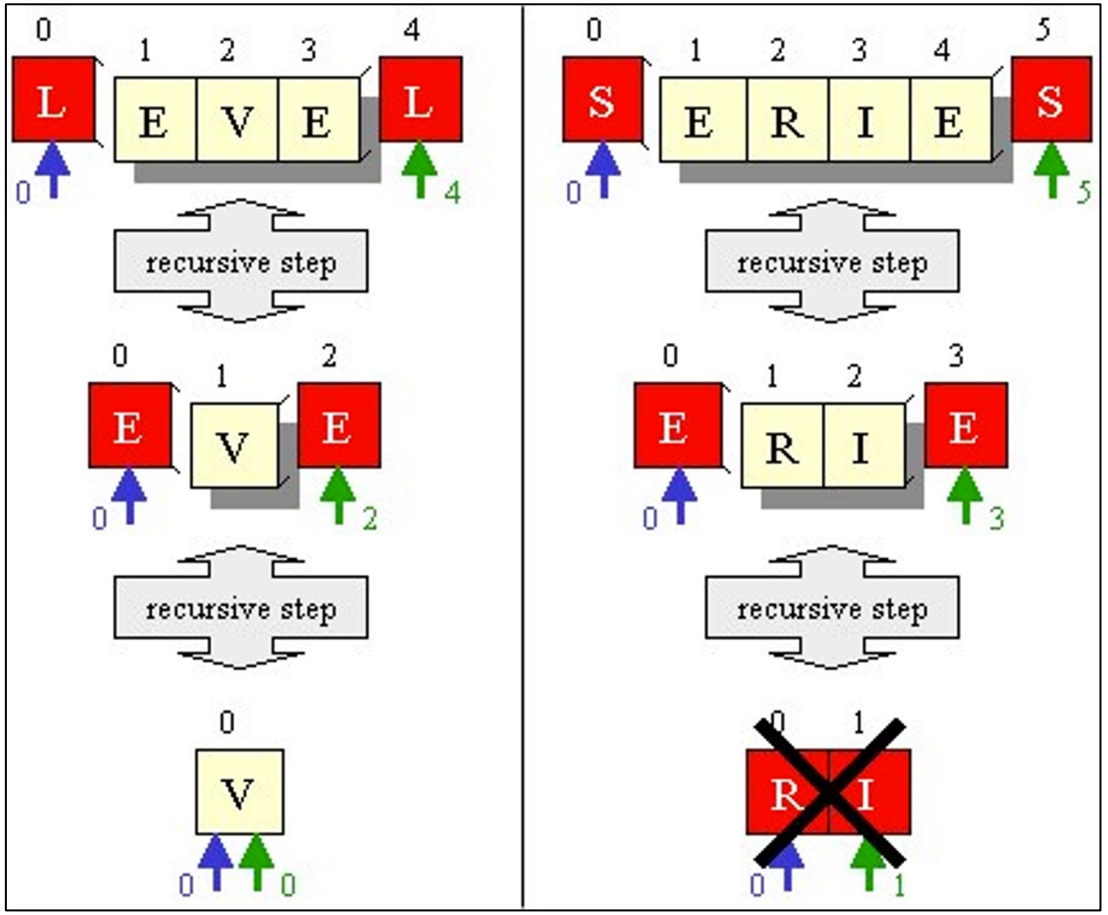


Image credit - https://levelup.gitconnected.com/

Image credit - https://stackoverflow.com/

Recursion

Merge Sort: A Recursive Sorting Algorithm

Merge sort follows the **"divide and conquer"** strategy to sort a list by repeatedly dividing it into smaller parts, sorting those parts, and then merging them to form a sorted result.

Divide:
Divide the unsorted list into two equal sublists (or nearly equal if the list has an odd number of elements).
This step is recursively applied until the sublists have only one element each.

Conquer (Sort):
Sort each of the smaller sublists (base case of recursion).
For sublists with only one element, they are already considered sorted.

Merge:
Merge the sorted sublists created in the previous step to produce a new sorted sublist.
The merging process involves comparing the first elements of each sublist and placing them in order into a new merged sublist.
Continue this process until all elements are merged.

Repeat:
Repeat steps 1 to 3 until there's only one sorted list remaining, which is the fully sorted list.

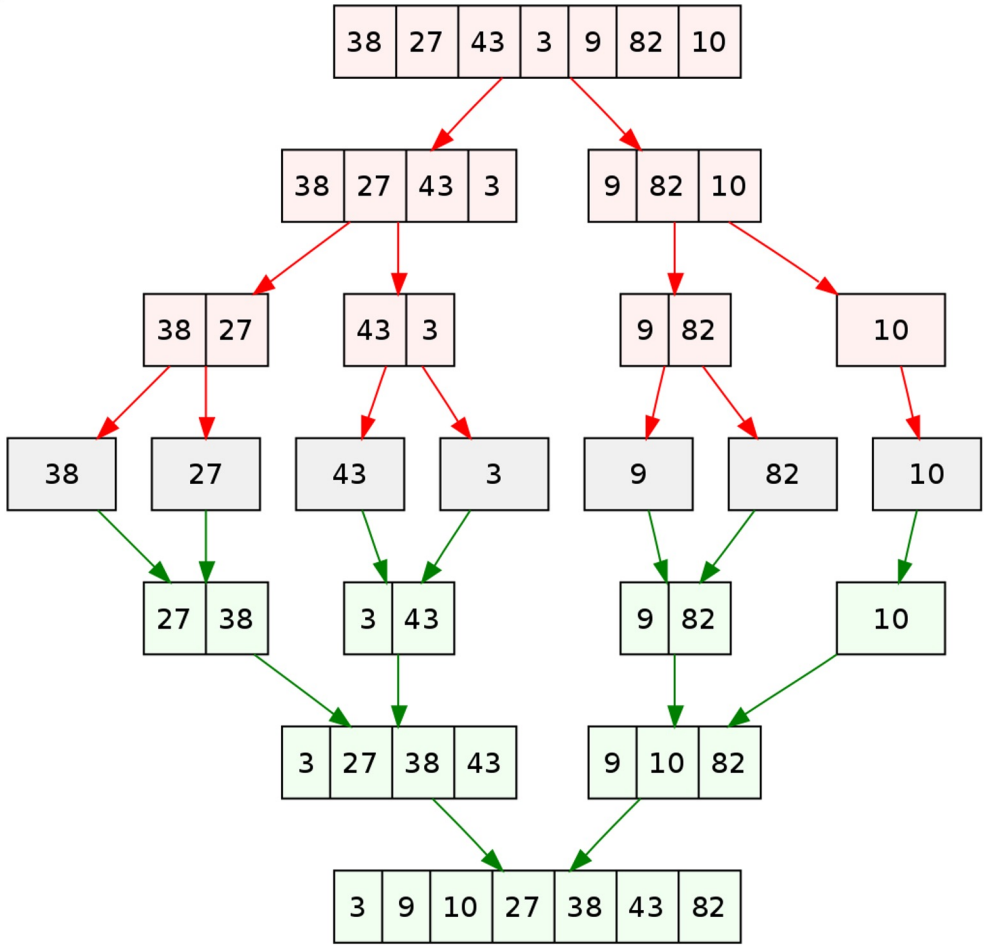


Image credit - https://cs.stackexchange.com/