

Recursion & Recurrence Relations

I. Factorial

$$n! = n(n-1)(n-2) \cdots 1$$

```
def factorial(n):  
    if n == 0:  
        return 1  
    else:  
        return n * factorial(n - 1)
```

Cost: Let $M(n)$ be the number of multiplications. Each call does one multiplication and then recurses on $n - 1$. Unrolling gives:

$$\begin{aligned} M(n) &= 1 + M(n-1) \\ &= 1 + 1 + M(n-2) \\ &= 1 + 1 + 1 + M(n-3) \\ &\vdots \\ &= n \end{aligned}$$

2. Selection Sort (Recursive)

Idea: Find the smallest element in `A[i:]`, swap it into position `i`, then sort the rest.

```
def vec_sort(A, i):  
    if i == len(A):  
        return  
    else:  
        min_val = inf  
        loc = i  
        for j = i to len(A) - 1:  
            if A[j] < min_val:  
                min_val = A[j]  
                loc = j
```

```
swap A[i], A[loc]
vec_sort(A, i + 1)
```

Cost: Scanning n elements costs n , then we recurse on $n - 1$ elements.

$$\begin{aligned}T(n) &= T(n - 1) + n \\&= T(n - 2) + (n - 1) + n \\&= T(n - 3) + (n - 2) + (n - 1) + n \\&\vdots \\&= 0 + \frac{n(n + 1)}{2} \approx n^2\end{aligned}$$

3. Why Divide and Conquer?

Recursion is useful when a problem can be **divided into smaller subproblems**.

$$T(n) = 2T\left(\frac{n}{2}\right)$$

The smaller each subproblem is (relative to n), the more we save:

$$2T\left(\frac{n}{3}\right) < 2T\left(\frac{n}{2}\right)$$

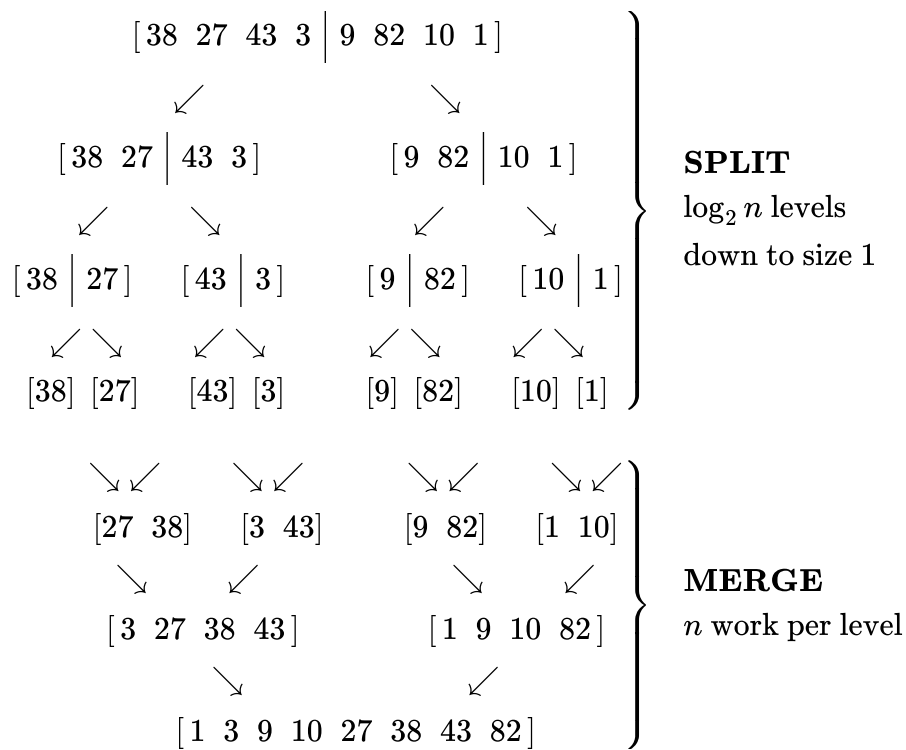
Shrinking by a **fraction** ($n \rightarrow n/2$) is far better than shrinking by a **constant** ($n \rightarrow n - 1$), as in selection sort.

4. Merge Sort

Idea: Split the array in half, sort each half recursively, then merge the two sorted halves.

```
def merge_sort(A):
    if len(A) <= 1:
        return A
    left = merge_sort(A[0 : n/2])
    right = merge_sort(A[n/2 : n])
    return join(left, right) # merge two sorted lists in O(n)
```

Split / Merge Diagram



Every level touches all n elements once, and there are $\log_2 n$ levels, so the total is about $n \log n$.

Cost: Two half-size subproblems, plus n work to merge.

$$\begin{aligned}
 T(n) &= 2T\left(\frac{n}{2}\right) + n \\
 &= 2\left[2T\left(\frac{n}{4}\right) + \frac{n}{2}\right] + n = 2^2 T\left(\frac{n}{4}\right) + 2n \\
 &= 2^2 \left[2T\left(\frac{n}{8}\right) + \frac{n}{4}\right] + 2n = 2^3 T\left(\frac{n}{8}\right) + 3n \\
 &\vdots \\
 &= 2^{\log_2 n} T(1) + n \log_2 n \\
 &= n + n \log n \\
 &\approx n \log n
 \end{aligned}$$

5. Quick Select

The recurrence splits into unequal pieces, so guess a bound and verify it.

$$T(n) \leq T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10}\right) + O(n) = T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10}\right) + c_1 n$$

Claim: $T(n) \leq 10c_1 n$

$$\begin{aligned}
 T(n) &\leq T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10}\right) + c_1 n \\
 &\leq 10c_1 \left(\frac{n}{5}\right) + 10c_1 \left(\frac{7n}{10}\right) + c_1 n \\
 &= 2c_1 n + 7c_1 n + c_1 n \\
 &= 10c_1 n \checkmark
 \end{aligned}$$

Works because $\frac{1}{5} + \frac{7}{10} = \frac{9}{10} < 1$: the pieces together are smaller than the original, so the work shrinks each level.

6. Power Set

Idea: For each element, branch two ways: **pick** it or **don't pick** it.

```
def pow_set(A, sub):
    if A is empty:
        return {sub}
    pick    = pow_set(A[1:], sub ∪ {A[0]})
    no_pick = pow_set(A[1:], sub)
    return pick ∪ no_pick
```

Cost: Each call makes two calls on a list one shorter.

$$\begin{aligned} T(n) &= 2T(n-1) \\ &= 2 \cdot 2T(n-2) = 2^2 T(n-2) \\ &\vdots \\ &= 2^n T(0) \\ &= 2^n \end{aligned}$$

Makes sense: a set of size n has exactly 2^n subsets.

7. Fibonacci Sequence

$$\begin{aligned} a_0 &= 1, & a_1 &= 2, & a_2 &= 3, & a_3 &= 5, & a_4 &= 8, & a_5 &= 13, \dots \\ a_n &= a_{n-1} + a_{n-2} \end{aligned}$$

```
def fib(n):
    if n == 0:
        return 1
    if n == 1:
        return 2
    else:
        return fib(n - 1) + fib(n - 2)
```

Cost:

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

Since $T(n-2) \leq T(n-1)$, we get the upper bound

$$T(n) \leq 2T(n-1) \Rightarrow T(n) = O(2^n)$$

To get the exact growth rate, we need to solve the recurrence. Before that, a simpler example.

8. Solving Linear Recurrences

Simpler Example

$$f_n = 5f_{n-1} - 6f_{n-2}, \quad f_0 = 3, \quad f_1 = 8, \quad f_n = ?$$

Ansatz (guess): $f_n = x^n$. Substitute and factor:

$$\begin{aligned} x^n &= 5x^{n-1} - 6x^{n-2} \\ x^n - 5x^{n-1} + 6x^{n-2} &= 0 \\ x^{n-2}(x^2 - 5x + 6) &= 0 \\ x^{n-2}(x^2 - 3x - 2x + 6) &= 0 \\ x^{n-2}(x(x-3) - 2(x-3)) &= 0 \\ x^{n-2}(x-2)(x-3) &= 0 \end{aligned}$$

Roots: $r_1 = 2, r_2 = 3$. The general solution is a combination of both:

$$f_n = k_1 r_1^n + k_2 r_2^n = k_1 2^n + k_2 3^n$$

Plug in initial conditions f_0, f_1 :

$$\begin{aligned} 3 &= k_1 + k_2 & \Rightarrow 6 &= 2k_1 + 2k_2 \\ 8 &= 2k_1 + 3k_2 \\ 8 - 6 &= (2k_1 + 3k_2) - (2k_1 + 2k_2) = k_2 \\ k_2 &= 2, \quad k_1 &= 1 \end{aligned}$$

$$\boxed{f_n = 2^n + 2 \cdot 3^n}$$

Repeated root: if both roots equal r , the solution becomes

$$f_n = (k_1 n + k_2) r^n$$

Back to Fibonacci

$$\begin{aligned} x^n &= x^{n-1} + x^{n-2} \\ x^n - x^{n-1} - x^{n-2} &= 0 \\ x^{n-2}(x^2 - x - 1) &= 0 \end{aligned}$$

Quadratic formula:

$$r = \frac{1 \pm \sqrt{1 + 4(1)}}{2(1)} = \frac{1 \pm \sqrt{5}}{2} = \phi, -\frac{1}{\phi}$$

$$a_n = k_1 r_1^n + k_2 r_2^n$$

Solve the system using $a_0 = 1$, $a_1 = 2$:

$$k_1 = \frac{5 + 3\sqrt{5}}{10}, \quad k_2 = \frac{5 - 3\sqrt{5}}{10}$$

$$a_n = \left(\frac{5 + 3\sqrt{5}}{10} \right) \left(\frac{1 + \sqrt{5}}{2} \right)^n + \left(\frac{5 - 3\sqrt{5}}{10} \right) \left(\frac{1 - \sqrt{5}}{2} \right)^n$$

Check with $n = 2$ (should give $a_2 = 3$), using $\left(\frac{1 \pm \sqrt{5}}{2} \right)^2 = \frac{6 \pm 2\sqrt{5}}{4}$:

$$\begin{aligned} a_2 &= \left(\frac{5 + 3\sqrt{5}}{10} \right) \left(\frac{6 + 2\sqrt{5}}{4} \right) + \left(\frac{5 - 3\sqrt{5}}{10} \right) \left(\frac{6 - 2\sqrt{5}}{4} \right) \\ &= \frac{(30 + 10\sqrt{5} + 18\sqrt{5} + 30) + (30 - 10\sqrt{5} - 18\sqrt{5} + 30)}{40} \\ &= \frac{120}{40} = 3 \quad \checkmark \end{aligned}$$

Since $|-1/\phi| < 1$, the second term vanishes as n grows, so $a_n \approx k_1 \phi^n$. Naive recursive `fib` therefore runs in about $\phi^n \approx 1.618^n$ time.

Practice Questions

1. Unroll $T(n) = T(\frac{n}{2}) + n$ with $T(1) = 1$. What is $T(n)$ approximately?

✓ Answer >

$$T(n) = n + \frac{n}{2} + \frac{n}{4} + \dots + 1 \approx 2n, \text{ so } T(n) = O(n).$$

2. Unroll $T(n) = 2T(\frac{n}{2}) + 1$ with $T(1) = 1$. Compare it to merge sort.

✓ Answer >

$T(n) = 2^k T(\frac{n}{2^k}) + (1 + 2 + \dots + 2^{k-1})$. At $k = \log_2 n$: $T(n) = n + (n - 1) \approx 2n = O(n)$. Merge sort is $n \log n$ because it does n work per call instead of 1.

3. Using substitution, prove $T(n) \leq T\left(\frac{n}{4}\right) + T\left(\frac{n}{2}\right) + cn$ satisfies $T(n) \leq 4cn$.

✓ Answer >

$$T(n) \leq 4c\frac{n}{4} + 4c\frac{n}{2} + cn = cn + 2cn + cn = 4cn \checkmark$$

4. Solve $f_n = 7f_{n-1} - 10f_{n-2}$ with $f_0 = 2, f_1 = 7$.

✓ Answer >

$$x^2 - 7x + 10 = (x - 2)(x - 5) = 0, \text{ roots } 2, 5.$$

$$k_1 + k_2 = 2, 2k_1 + 5k_2 = 7 \Rightarrow k_2 = 1, k_1 = 1.$$

$$f_n = 2^n + 5^n$$

5. Solve $f_n = 4f_{n-1} - 4f_{n-2}$ with $f_0 = 1, f_1 = 4$.

✓ Answer >

$$x^2 - 4x + 4 = (x - 2)^2 = 0, \text{ repeated root } r = 2, \text{ so } f_n = (k_1n + k_2)2^n.$$

$$f_0: k_2 = 1. f_1: (k_1 + 1) \cdot 2 = 4 \Rightarrow k_1 = 1.$$

$$f_n = (n + 1)2^n \text{ (check: } f_2 = 4 \cdot 4 - 4 \cdot 1 = 12 = 3 \cdot 4 \checkmark)$$
