

Lecture 11: Recurrence Relations

Lecturer and scribe: Karthik Gajulapalli

1 Algorithms and Their Underlying Recurrence

1.1 Factorial and Expansion

For a nonnegative integer n , consider:

```
Factorial(n):
    if n <= 1:
        return 1
    return n * Factorial(n - 1)
```

Let $T(n)$ be the number of *multiplications*. The base call does none. Every other call does one multiplication after its recursive call:

$$T(1) = 0, \quad T(n) = T(n - 1) + 1 \quad (n \geq 2).$$

Expand the smaller term repeatedly:

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

After k expansions, the expression is $T(n - k) + k$. We stop at the *base case*, so $n - k = 1$ and $k = n - 1$. Thus

$$\boxed{T(n) = n - 1 \quad \text{for } n \geq 1.}$$

For example, `Factorial(5)` multiplies by 2, 3, 4, 5: four multiplications. Also $T(0) = 0$; the formula $n - 1$ was only claimed for $n \geq 1$.

This illustrates the first method: **expand until reaching a known value, then evaluate the accumulated work**. Here the accumulated work is a sum of ones, so the running time is linear: $O(n)$.

1.2 Recursive Selection Sort

Selection sort finds the smallest remaining item, moves it to the next position, and sorts the remaining suffix. In `rec_sort(A, i)`, i increases, so the number of unsorted entries decreases.

```

rec_sort(A, i):
    if i >= len(A) - 1:
        return
    m = i
    for j = i + 1, ..., len(A) - 1:
        if A[j] < A[m]:
            m = j
    swap(A[i], A[m])
    rec_sort(A, i + 1)

```

The first call is `rec_sort(A, 0)`. If the current suffix has length n , the next suffix has length $n - 1$.

There are $n - 1$ comparisons and one execution of `swap` in this call. If we charge one unit for each of these operations, the local charge is exactly n , and our lecture recurrence is

$$T(1) = 0, \quad \boxed{T(n) = T(n - 1) + n \quad (n \geq 2).}$$

Here a swap counts once even if its two positions coincide. This is an exact count for the chosen operations, and a useful model of the total time, since the remaining work has the same asymptotic order.

Expanding gives a sum of progressively shorter scans:

$$\begin{aligned}
 T(n) &= T(n - 1) + n \\
 &= T(n - 2) + (n - 1) + n \\
 &= T(1) + 2 + 3 + \cdots + n \\
 &= \frac{n(n + 1)}{2} - 1.
 \end{aligned}$$

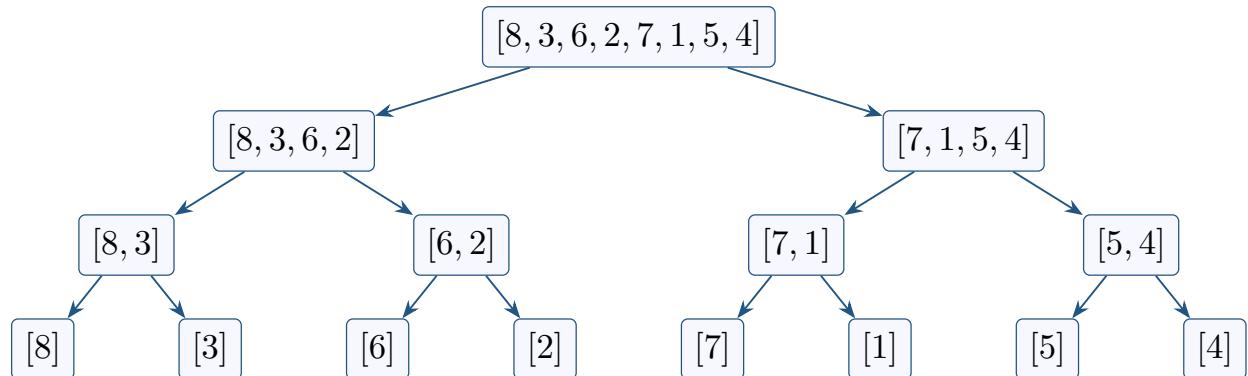
This is a quadratic function of n , so $T(n) = O(n^2)$.

1.3 Merge Sort

Merge sort sorts both halves and then combines them. Let `join(L,R)` merge two *sorted* arrays into one sorted array in linear time, $O(|L| + |R|)$. We use it as a helper without implementing it here.

```
merge_sort(A):
    if len(A) <= 1:
        return A
    mid = floor(len(A) / 2)
    L = merge_sort(A[0:mid])
    R = merge_sort(A[mid:len(A)])
    return join(L, R)
```

The picture follows the *splitting* phase. On the way back up, `join` merges the sorted results of each pair of children.



1.3.1 Analysis by Levels

For the exact model, let $T(n)$ count entries written by all joins; a join writes exactly n entries. Treat slices as views. If n is a power of 2,

$$T(1) = 0, \quad \boxed{T(n) = 2T(n/2) + n.}$$

At level j , there are 2^j subproblems of size $n/2^j$, so the total join work on that level is $2^j(n/2^j) = n$. There are $\log_2 n$ levels with joins, giving

$$\boxed{T(n) = n \log_2 n.}$$

Equivalently, after k expansions, $T(n) = 2^k T(n/2^k) + kn$; take $k = \log_2 n$. Including all overhead still gives $O(n \log n)$ time. For other input sizes, use $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + n$; the same asymptotic bound holds.

1.4 Deterministic Selection

The selection problem. Given an array of n values and an integer $1 \leq k \leq n$, return the k th smallest value, counting duplicates with their multiplicities. For example, rank 3 in $[9, 1, 7, 3, 5]$ is 5. Finding a median is a special case.

There is a deterministic selection algorithm with the following worst-case recurrence. We treat the algorithm as a black box; we do not need its implementation to analyze the recurrence:

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

1.4.1 Analysis by Guess and Check

Temporarily suppress integer rounding. Choose $c_1 > 0$ so the nonrecursive work is at most $c_1 n$. The two recursive input sizes add to $9n/10$, suggesting that a linear bound might work.

Guess $T(n) \leq Cn$. Use *strong induction*: assume this bound for every smaller input on the right-hand side. Then

$$\begin{aligned} T(n) &\leq C(n/5) + C(7n/10) + c_1 n \\ &= \left(\frac{9}{10}C + c_1 \right) n. \end{aligned}$$

For this to be at most Cn , we need

$$\frac{9}{10}C + c_1 \leq C \iff C \geq 10c_1.$$

Taking $C = 10c_1$ gives the bound from class:

$$T(n) \leq 10c_1 n = O(n).$$

The missing tenth of the guessed budget pays for the linear work outside the recursive calls.

Base cases are part of the proof. Enlarge c_1 , if necessary, so that every positive base input m also satisfies $T(m) \leq 10c_1 m$. Increasing a valid upper-bound constant preserves the bound on local work. This supplies the induction's starting point.

Rounding. The displayed constant $10c_1$ belongs to the simplified recurrence. With rounded sizes and bounded additive terms, their sum is at most $0.9n + b$ for some fixed $b \geq 0$. For $n \geq 20b$, this is at most $0.95n$. Taking $C \geq 20c_1$ and large enough for the finitely many smaller inputs proves $T(n) \leq Cn$. Thus rounding changes the constants, but preserves the linear bound.

2 The Power Set and Two Choices per Element

Let A list the distinct elements of a set. To generate all subsets, decide for each element whether to pick it. Here `chosen` is mutable: `append` and `pop` undo one another, and `visit` processes the current subset.

```
power_set(A, i, chosen):
    if i == len(A):
        visit(chosen)
        return
    power_set(A, i + 1, chosen)      # don't pick A[i]
    chosen.append(A[i])
    power_set(A, i + 1, chosen)      # pick A[i]
    chosen.pop()
```

Start with `power_set(A, 0, [])`. Each nonbase call makes *two* calls with one fewer undecided element.

Let $T(n)$ count completed subsets, or equivalently the leaves in the recursion tree, when n elements remain. Then

$$T(0) = 1, \quad \boxed{T(n) = 2T(n-1)}.$$

The base case is 1, since even an empty set has one subset: the empty set. Expansion gives

$$T(n) = 2^k T(n-k) = 2^n T(0) = \boxed{2^n}.$$

For two elements a, b , the four root-to-leaf decisions are

Decision for a	Decision for b	Subset
don't pick	don't pick	$\emptyset$
don't pick	pick	$\{b\}$
pick	don't pick	$\{a\}$
pick	pick	$\{a, b\}$

2.1 Leaves, Calls, and Output Cost

There are 2^j calls at depth j , for $j = 0, \dots, n$. The total number of calls is therefore

$$C(n) = 1 + 2 + \dots + 2^n = 2^{n+1} - 1.$$

Equivalently, $C(0) = 1$ and $C(n) = 2C(n-1) + 1$. With constant work per call and per visit, the running time is $O(2^n)$.

If `visit` copies or prints every element of each subset, the output itself contains $n2^{n-1}$ element occurrences for $n \geq 1$: each of the n elements occurs in half the subsets. That explicit-output version takes $O(n2^n)$ time. The recurrence $T(n) = 2T(n-1)$ counts the subsets, not their total length.

4 Homogeneous Recurrences with Two Roots

Theorem 1 (Two root rule). *Consider a recurrence of the form*

$$f_n = af_{n-1} + bf_{n-2} \quad (n \geq 2),$$

*where a, b are constants and $b \neq 0$. Its **characteristic equation** is*

$$r^2 - ar - b = 0.$$

We focus on real roots, which cover all the examples in these notes. There are two cases:

Two different roots r_1, r_2 . *Every solution has the form*

$$f_n = Ar_1^n + Br_2^n.$$

A repeated root r . *If both roots are equal, every solution has the form*

$$f_n = (A + Bn)r^n.$$

In either case, the initial values f_0 and f_1 determine the two constants A and B uniquely.

4.1 How to Use the Rule

1. Solve $r^2 - ar - b = 0$ by factoring or using the quadratic formula.
2. Write the appropriate formula above, depending on whether the two roots are different or equal.
3. Substitute $n = 0$ and $n = 1$, then solve the two equations for A and B .

For example, with two different roots, the equations for the constants are

$$A + B = f_0, \quad Ar_1 + Br_2 = f_1.$$

With a repeated root, they are $A = f_0$ and $(A + B)r = f_1$.

4.2 Why Do We Look for Roots?

Try a solution of the form $f_n = r^n$. Substitution gives

$$r^n = ar^{n-1} + br^{n-2} \implies r^2 - ar - b = 0.$$

Here we divided by r^{n-2} ; the condition $b \neq 0$ ensures that neither root is zero. This explains where the characteristic equation comes from.

When the rule applies. Each term uses the previous *two* terms with fixed coefficients. “Homogeneous” means there is no extra term such as $+1$ or $+n$. Thus the rule applies to Fibonacci numbers, but not directly to $C(n) = C(n-1) + C(n-2) + 1$ or to a recurrence involving $T(n/2)$.

5 Distinct Roots and Repeated Roots

5.1 The Example from Class

Solve $f_n = 5f_{n-1} - 6f_{n-2}$, with $f_0 = 3$ and $f_1 = 8$. The characteristic equation is

$$r^2 - 5r + 6 = (r - 2)(r - 3) = 0.$$

The roots are distinct, so $f_n = A2^n + B3^n$. The initial values give

$$A + B = 3, \quad 2A + 3B = 8.$$

Subtract twice the first equation from the second to get $B = 2$, and then $A = 1$. Therefore

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

As a quick check, the recurrence gives $f_2 = 5(8) - 6(3) = 22$, and the formula gives $4 + 18 = 22$.

5.2 The Same Roots with Different Initial Values

For the same recurrence, take $g_0 = 2$ and $g_1 = 5$. The roots remain 2 and 3, but now

$$A + B = 2, \quad 2A + 3B = 5, \quad \boxed{g_n = 2^n + 3^n.}$$

Initial values affect the coefficients, not the characteristic roots. They can even remove a root's contribution: $h_0 = 1$, $h_1 = 2$ give $h_n = 2^n$, despite 3 being a characteristic root. Therefore the largest root controls growth only when its coefficient is nonzero and other contributions do not cause relevant cancellation.

5.3 When the Two Roots Are Equal

Now solve $u_n = 4u_{n-1} - 4u_{n-2}$, with $u_0 = 1$, $u_1 = 4$. Its characteristic polynomial is

$$r^2 - 4r + 4 = (r - 2)^2.$$

Writing $A2^n + B2^n$ would only give one adjustable coefficient. The repeated-root rule instead gives $u_n = (A + Bn)2^n$. Thus

$$A = 1, \quad 2(A + B) = 4 \implies B = 1, \quad \boxed{u_n = (n + 1)2^n.}$$

Check: $u_2 = 4(4) - 4(1) = 12 = (2 + 1)2^2$. The exact formula includes a factor of $n + 1$ multiplying 2^n ; the repeated root contributes this extra factor. In particular, $u_n = O(n2^n)$.

6 Returning to Fibonacci and the Golden Ratio

The Fibonacci recurrence has characteristic equation

$$r^2 - r - 1 = 0.$$

Its two roots are

$$\varphi = \frac{1 + \sqrt{5}}{2} \approx 1.618034, \quad \psi = \frac{1 - \sqrt{5}}{2} \approx -0.618034.$$

The positive root φ is the **golden ratio**. Notice $\varphi^2 = \varphi + 1$, $\psi = -1/\varphi$, and $|\psi| < 1$. The two root rule gives

$$F_n = A\varphi^n + B\psi^n.$$

The initial values $F_0 = 0$, $F_1 = 1$ imply

$$A + B = 0, \quad A\varphi + B\psi = 1.$$

Since $\varphi - \psi = \sqrt{5}$, we get $A = 1/\sqrt{5}$ and $B = -1/\sqrt{5}$. Thus **Binet's formula** is

$$F_n = \frac{\varphi^n - \psi^n}{\sqrt{5}}.$$

6.1 Turning the Formula into a Growth Bound

The second term decreases in magnitude, giving the precise error bound

$$\left| F_n - \frac{\varphi^n}{\sqrt{5}} \right| = \frac{|\psi|^n}{\sqrt{5}} < \frac{1}{2} \quad (n \geq 0).$$

So F_n is the nearest integer to $\varphi^n/\sqrt{5}$. In particular, $F_n = O(\varphi^n)$.

We can also prove useful two-sided bounds *without* the closed form. For every $n \geq 1$,

$$\varphi^{n-2} \leq F_n \leq \varphi^{n-1}.$$

The claims hold at $n = 1, 2$. For $n \geq 3$, add the two lower induction bounds and use $1 + \varphi = \varphi^2$:

$$F_n \geq \varphi^{n-3} + \varphi^{n-4} = \varphi^{n-4}(\varphi + 1) = \varphi^{n-2}.$$

Similarly, the upper bounds give $F_n \leq \varphi^{n-2} + \varphi^{n-3} = \varphi^{n-1}$.

Why the base φ appears. Guessing an exponential r^n for Fibonacci makes the induction close exactly when $r^{n-1} + r^{n-2} = r^n$, or $r^2 = r + 1$. The golden ratio is the positive solution of precisely that equation.

7 The Exact Size of the Fibonacci Call Tree

The number returned by `fib` and the number of calls it makes are different sequences. Recall

$$C(0) = C(1) = 1, \quad C(n) = C(n-1) + C(n-2) + 1.$$

Remove the extra $+1$ by defining $D(n) = C(n) + 1$. Then

$$\begin{aligned} D(n) &= C(n-1) + C(n-2) + 2 \\ &= D(n-1) + D(n-2), \quad D(0) = D(1) = 2. \end{aligned}$$

The sequence $2F_{n+1}$ has the same recurrence and initial values, so it equals $D(n)$. Therefore

$$C(n) = 2F_{n+1} - 1 = \frac{2}{\sqrt{5}}(\varphi^{n+1} - \psi^{n+1}) - 1.$$

Using the Fibonacci bounds from the previous page, for $n \geq 0$,

$$2\varphi^{n-1} - 1 \leq C(n) \leq 2\varphi^n - 1.$$

In particular, $C(n) = O(\varphi^n)$, and the lower bound also grows exponentially with base φ . The direct recursive algorithm does constant local work per node in the unit-cost model, so its running time is also $O(\varphi^n)$.

7.1 A Second Proof by Counting Leaves

Let $L(n)$ count base calls. Then $L(0) = L(1) = 1$ and $L(n) = L(n-1) + L(n-2)$, so $L(n) = F_{n+1}$. Every internal node has exactly two children. If there are I internal nodes and L leaves, counting edges gives $2I = I + L - 1$, hence $I = L - 1$. The total number of nodes is $I + L = 2L - 1 = 2F_{n+1} - 1$. There is one addition per internal node, so the addition count is $F_{n+1} - 1$.

n	0	1	2	3	4	5
F_n	0	1	1	2	3	5
$L(n)$	1	1	2	3	5	8
$C(n)$	1	1	3	5	9	15

Since $\varphi < 2$, this is sharper than $O(2^n)$; indeed $C(n)/2^n \rightarrow 0$. The repeated subproblems explain the expense. Storing each answer once (memoization) reduces the computation to $O(n)$ arithmetic operations; this is a different algorithm.

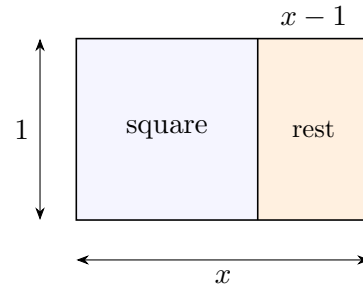
8 Some Places the Golden Ratio Appears

8.1 A Rectangle That Repeats Its Shape

A rectangle has long side x and short side 1. Remove a unit square. If the remaining rectangle has the same aspect ratio after rotation, then

$$x = \frac{1}{x-1} \implies x^2 = x + 1.$$

Since $x > 1$, we obtain $x = \varphi$. This is a *golden rectangle*.



8.2 Fibonacci Ratios and a Continued Fraction

Binet's formula gives

$$\frac{F_{n+1}}{F_n} = \varphi \frac{1 - (\psi/\varphi)^{n+1}}{1 - (\psi/\varphi)^n} \longrightarrow \varphi \quad (n \rightarrow \infty).$$

For example, $8/5 = 1.6$, $13/8 = 1.625$, and $21/13 \approx 1.61538$. Repeatedly substituting $\varphi = 1 + 1/\varphi$ produces

$$\varphi = 1 + \frac{1}{1 + \frac{1}{1 + \frac{1}{1 + \dots}}}$$

The finite truncations are $1, 2, 3/2, 5/3, 8/5, \dots$: ratios of consecutive Fibonacci numbers. Their convergence follows from the ratio limit above.

8.3 Staircases and Tilings

Let a_n count ways to climb n stairs using steps of size 1 or 2. Split by the last step: $a_n = a_{n-1} + a_{n-2}$, with $a_0 = a_1 = 1$. Hence $a_n = F_{n+1}$, so $a_n = O(\varphi^n)$. The same recurrence counts tilings of a $1 \times n$ strip with unit squares and 1×2 dominoes. The golden ratio describes how the number of possibilities grows.

8.4 Sunflower Spirals and the Golden Angle

A golden-ratio division of a full turn gives the smaller angle $360^\circ/\varphi^2 \approx 137.508^\circ$, called the *golden angle*. It appears in mathematical models of plant arrangements (Okabe, 2015). Sunflower seedheads often have Fibonacci spiral counts, but there are also non-Fibonacci patterns: this is an observed tendency, not a rule for every plant. A study of 657 sunflowers documented both types.

Further reading: J. Swinton, E. Ochu, and the MSI Turing's Sunflower Consortium (2016), *Novel Fibonacci and non-Fibonacci structure in the sunflower: results of a citizen science experiment*, Royal Society Open Science 3:160091.