You basically have two layers where you can prioritize:

1. **At spec level** → which spec files run first
2. **At test level** → which tests inside a spec are treated as more important

I'll focus on **practical ways you can actually implement today**, especially with Cypress + CI + Cypress Cloud.

---

1. Create a dedicated "critical" (smoke) suite

Most teams do this first, because it's simple and very clear.

a) Put critical flows in separate specs or folder

Example structure:

```
cypress/
  e2e/
    critical/
      login.cy.js
      checkout.cy.js
      payments.cy.js
    regression/
      profile.cy.js
      reports.cy.js
      ...
```

All "business must work" tests (login, add to cart, payment, etc.) go into `critical/`.

---

2. In CI, run "critical first, everything else later"

In your GitHub Actions (or any CI), you can:

1. Run **critical specs** first (fast feedback).

2. Run **full suite** afterward (maybe in parallel).

Example GitHub Actions (2 jobs)

```
name: Cypress Prioritized

on: [push]

jobs:
  critical-tests:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: cypress-io/github-action@v6
        with:
          start: npm start
          wait-on: 'http://localhost:3000'
          record: true
          # Only run critical tests
          spec: cypress/e2e/critical/**/*.cy.js
        env:
          CYPRESS_RECORD_KEY: ${{ secrets.CYPRESS_RECORD_KEY }}
          GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}

  full-regression:
    needs: critical-tests   # 👈 run only *after* critical passes
    runs-on: ubuntu-latest
    strategy:
      fail-fast: false
      matrix:
        containers: [1, 2, 3]
    steps:
      - uses: actions/checkout@v4
      - uses: cypress-io/github-action@v6
        with:
          start: npm start
          wait-on: 'http://localhost:3000'
          record: true
          parallel: true
          group: regression-${{ matrix.containers }}
          ci-build-id: '${{ github.run_id }}'
          # All e2e tests
          spec: cypress/e2e/**/*.cy.js
        env:
```

CYPRESS_RECORD_KEY: ${{ secrets.CYPRESS_RECORD_KEY }}
GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}

Idea you can explain in training:

> “Critical specs run first in a small, fast job. If they fail, we know quickly. If they pass, a second job kicks off the heavier regression suite (optionally in parallel with Cypress Cloud).”

---

3. Tag-based prioritization

Instead of folder-based, you can **tag tests/specs** as critical and filter with `--env grepTags=...` if you use `cypress-grep`, or by grep-like libraries.

a) Mark important tests/specs

Using `cypress-grep` style:

```
// login.cy.js
describe('Login', { tags: ['critical'] }, () => {
  it('allows valid user to log in', { tags: ['critical'] }, () => {
    // test logic
  });
});
```

b) In CI, run only `critical` first

```
# Critical-only run
npx cypress run --env grepTags=critical
```

Then you can have another run for `regression` tag, `noncritical`, etc.

In GitHub Action:

```
with:
  record: true
  env: grepTags=critical
```

---

4. Manual spec ordering (single machine)

If you're *not* using parallelization and just want **spec order**, you can:

1. Name critical spec files so they sort first:
    - `01-login.cy.js`
    - `02-checkout.cy.js`
    - `99-reports.cy.js`
2. Or explicitly list specs via `--spec`:

```
npx cypress run \
  --spec "cypress/e2e/critical/login.cy.js,cypress/e2e/critical/checkout.cy.js,cypress/e2e/**/*.cy.js"
```

But for serious projects, **separate jobs (critical vs full)** is cleaner.

---

5. Cypress Cloud "smart" prioritization (optional topic)

For advanced/paid tiers, Cypress Cloud offers features like:

- Running **only impacted tests** based on changed files
- Prioritizing which tests to run first for a given commit

But even without that, **you already have good control** via:

- Folder structure
- Tags (`critical` vs `regression`)
- Separate CI jobs + `--spec` / `--env` filters
