Simplicity to Control Complexity

Based on Slides by Professor Lui Sha

Reliability

- Reliability for a giving mission duration t, $R(t)$, is the probability of the system working as specified (i.e., probability of no failures) for a duration that is at least as long as t.
- The most commonly used reliability function is the exponential reliability function:

$$R(t) = e^{-\lambda t}$$

where λ is the failure rate.

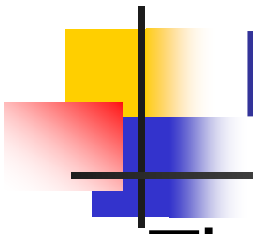
Reliability

- Reliability for a giving mission duration t, $R(t)$, is the probability of the system working as specified (i.e., probability of no failures) for a duration that is at least as long as t.
- The most commonly used reliability function is the exponential reliability function:

$$R(t) = e^{-\lambda t}$$

From queueing theory: Probability of zero independent arrivals in t time units (Poisson arrival process)

where λ is the failure rate.



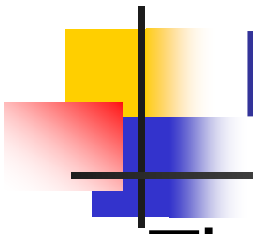
Reliability

- The most commonly used reliability function is the exponential reliability function:

$$R(t) = e^{-\lambda t}$$

where λ is the failure rate.

- Mean time to failure (MTTF) ?



Reliability

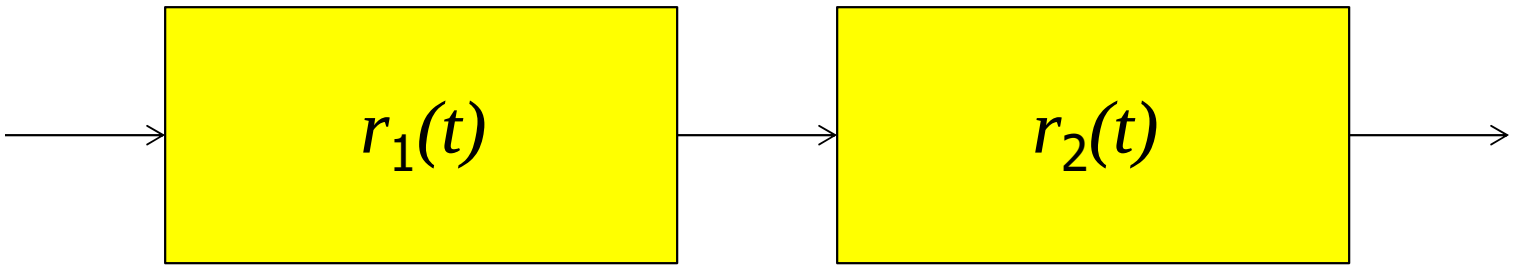
- The most commonly used reliability function is the exponential reliability function:

$$R(t) = e^{-\lambda t}$$

where λ is the failure rate.

- Mean time to failure (MTTF): $1/\lambda$

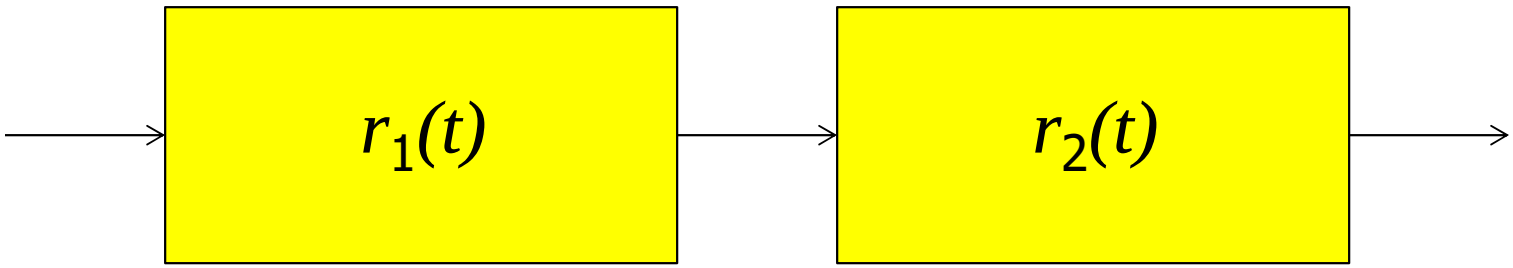
Simple Reliability Modeling



Note: This system needs both components to function.

- What is the reliability of a system that is made of the above two components?
  - Failure rate of first component: λ_1
  - Failure rate of second component: λ_2

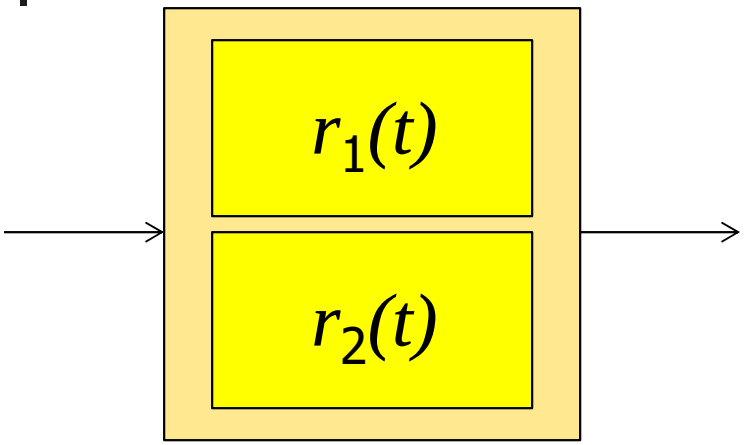
Simple Reliability Modeling



- Total failure rate = $\lambda_1 + \lambda_2$
- Mean time to failure = $1/(\lambda_1 + \lambda_2)$
- Total reliability:

$$R(t) = r_1(t)r_2(t) = e^{-(\lambda_1+\lambda_2)t}$$

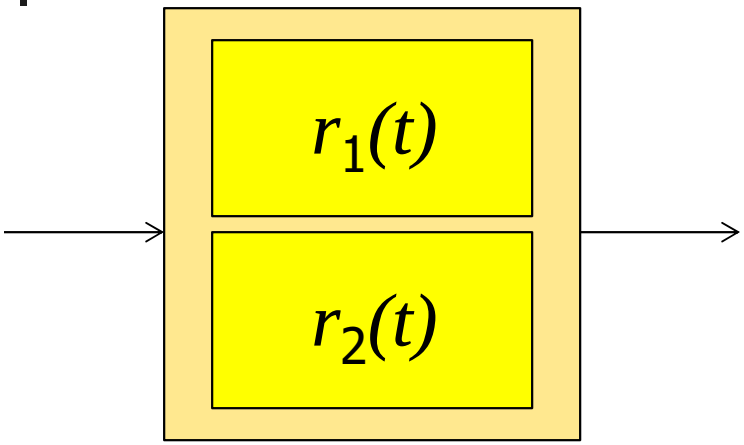
Simple Reliability Modeling



Note: This system needs at least one of the two components to function.

- Total reliability?

Simple Reliability Modeling

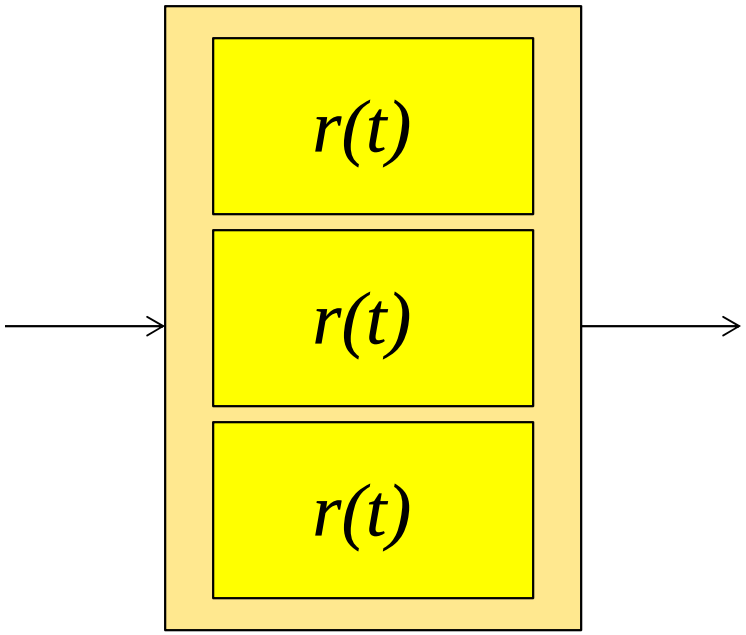


Note: This system needs at least one of the two components to function.

- Total reliability:

$$R(t) = 1 - (1 - r_1(t))(1 - r_2(t))$$

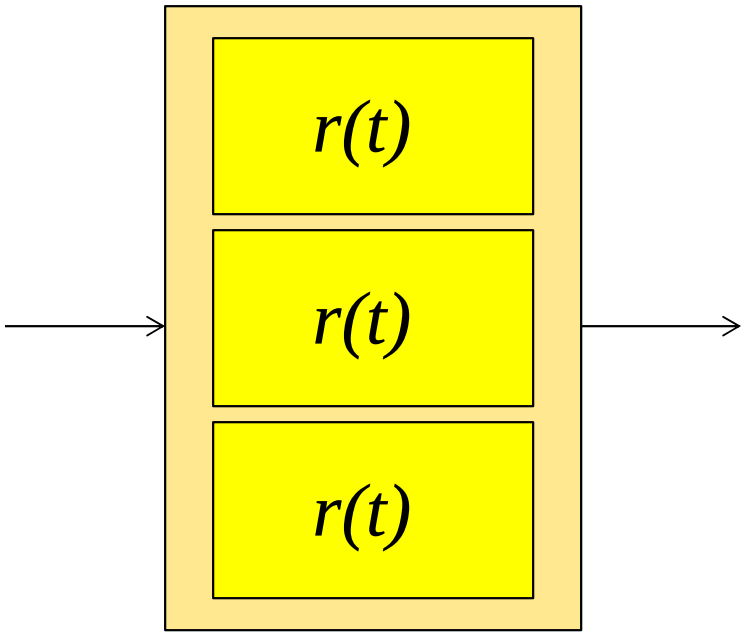
Triple Modular Redundancy



Note: This system needs at least two of the three components to function.

- Total reliability?

Triple Modular Redundancy



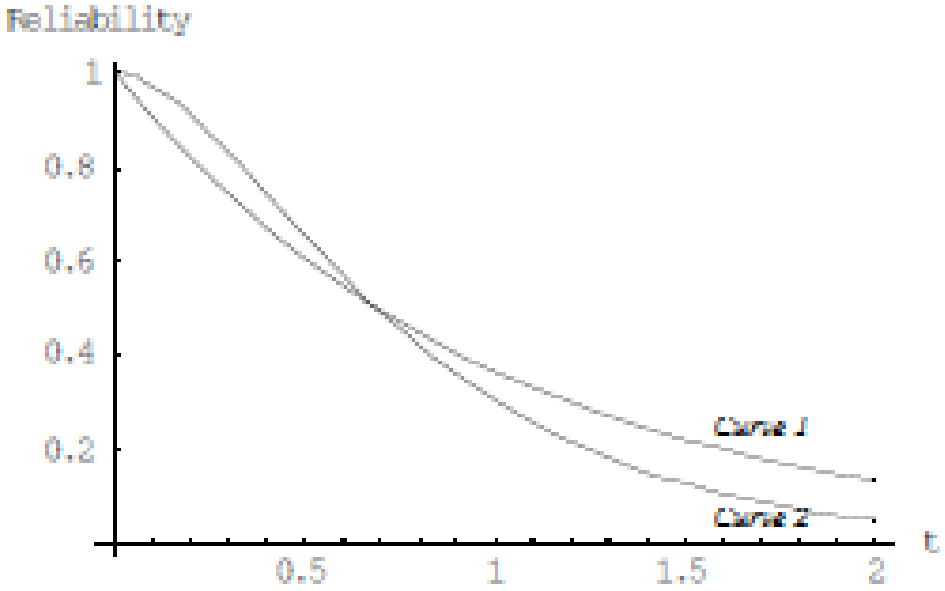
Note: This system needs at least two of the three components to function.

- Total reliability:

$$R(t) = r^3(t) + 3r^2(t)(1 - r(t))$$

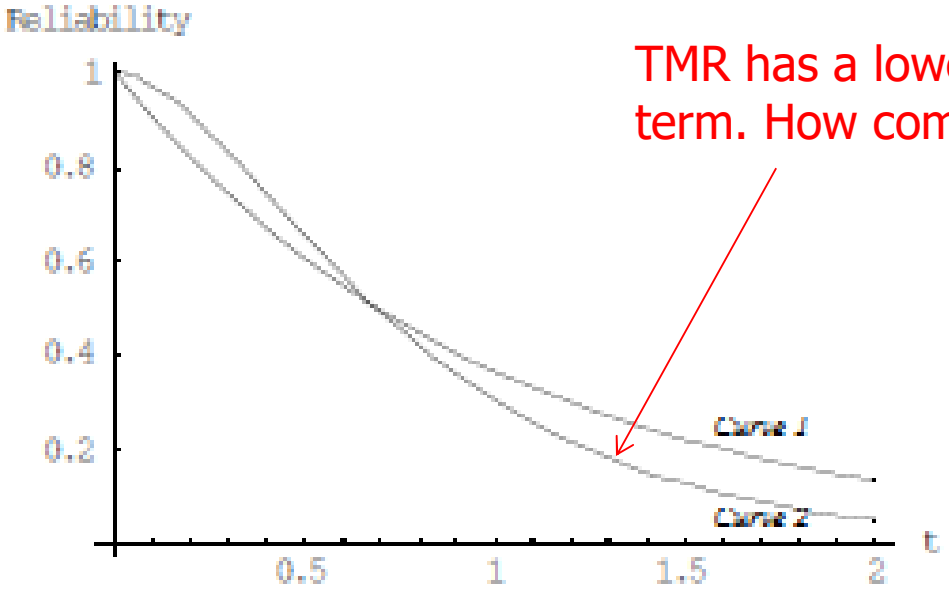
Triple Modular Redundancy

- Which case is TMR?

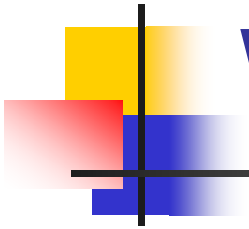


Triple Modular Redundancy

- Which case is TMR?



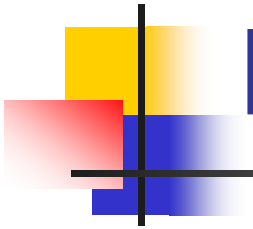
TMR has a lower reliability in the long term. How come?



Which Side Would You Take?

- Improving the reliability of increasingly complex software is a serious challenge. There are two philosophical positions:
  - *The diversity camp:* Diversity in crops resists diseases… diversity in software improves reliability. The likelihood of making the same mistakes decreases as the degree of diversity increases. Don't put all your eggs in one basket.
  - *The bullet-proof your basket camp:* Concentrate all the available resource to one version and do it right. Do-it-right-the-first-time is the time honored approach to quality products.

Software Development Postulates



- In science we rely on facts and logic. Let's begin with well known observations in software development. We make three postulates:
  - *P1: Complexity Breeds Bugs*. Everything else being equal, the more complex the software project is, the harder it is to make it reliable.
  - *P2: All Bugs are Not Equal*. You fix a bunch of obvious bugs quickly, but finding and fixing the last few bugs is much harder, if you can ever hunt them down.
  - *P3: All Budgets are Finite*. There is only a finite amount of effort (budget) that we can spend on any project.

Implications of the Postulates

- A reliability function in the form:

  $R(Effort, Complexity, t) = e^{-kC\ t/E}$

  satisfies P1 and P2
- The Finite Budget Assumption implies that diversity is not free. If we go for n version diversity, we must divide the available effort n-ways. This allows us to compare different approaches fairly.

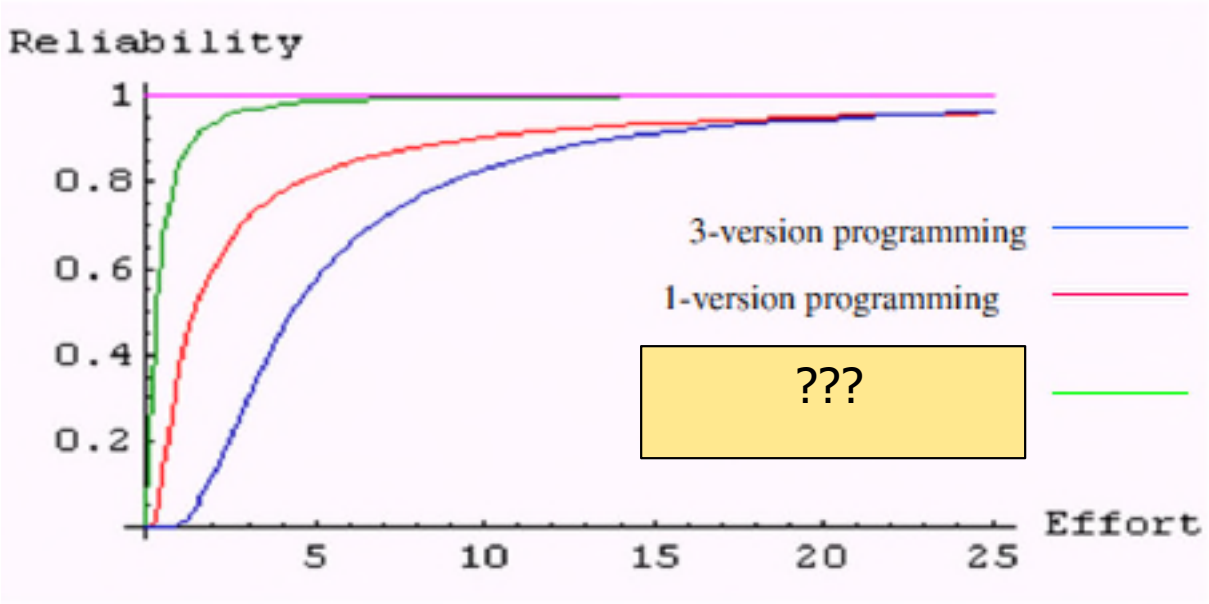
Implications of the Postulates

$$R(Effort, Complexity, t) = e^{-kC\ t/E}$$

- Note: splitting the effort greatly reduces reliability.

Analysis

Analysis shows that redundancy/diversity does not win. What are we going to do??



$$R(Effort, Complexity, t) = e^{-kC\ t/E}$$

Another Look at Redundancy: Complexity Reduction

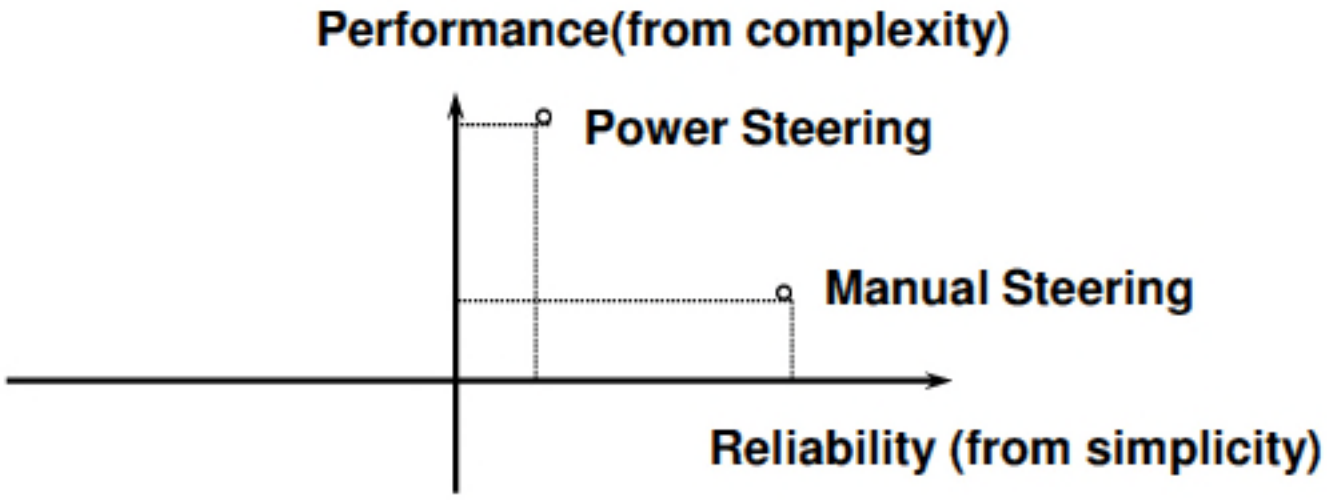
- Safety-critical versus performance requirements
- Example: power steering

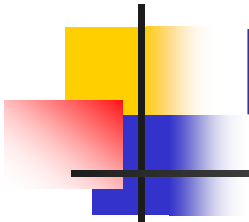
Another Look at Redundancy: Complexity Reduction

- Power steering:
  - Safety requirements: cannot lose control over steering even when power is lost (must have mechanical backup)
  - Performance requirements: ease of steering

Analytic Redundancy and Complexity Reduction

- Partial redundancy via simple backup that meets only safety-critical requirements

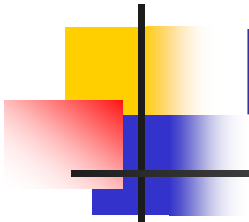




Example: A Sorting Exercise

- Sorting:
  - Bubble sort: easy to write but slower, $O(n^2)$
  - Quick sort: faster, $O(n \log(n))$, but more complicated to write
- Joe remembers how to do bubble sort, but is not perfectly sure of quick sort (has a 50% chance of getting it right).
- Joe is asked to write a sorting routine:
  - Correct and fast: A
  - Correct but slow: B
  - Incorrect: F

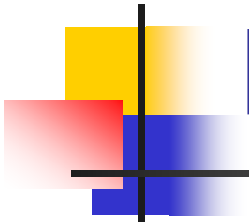
Example: A Sorting Exercise



- Sorting:
  - Bubble sort: easy to write but slower, $O(n^2)$
  - Quick sort: faster, $O(n \log(n))$, but more complicated to write
- Joe remembers how to do bubble sort, but is not perfectly sure of quick sort (has a 50% chance of getting it right).
- Joe is asked to write a sorting routine:
  - Correct and fast: A
  - Correct but slow: B
  - Incorrect: F

What is Joe's optimal strategy?

Example: A Sorting Exercise



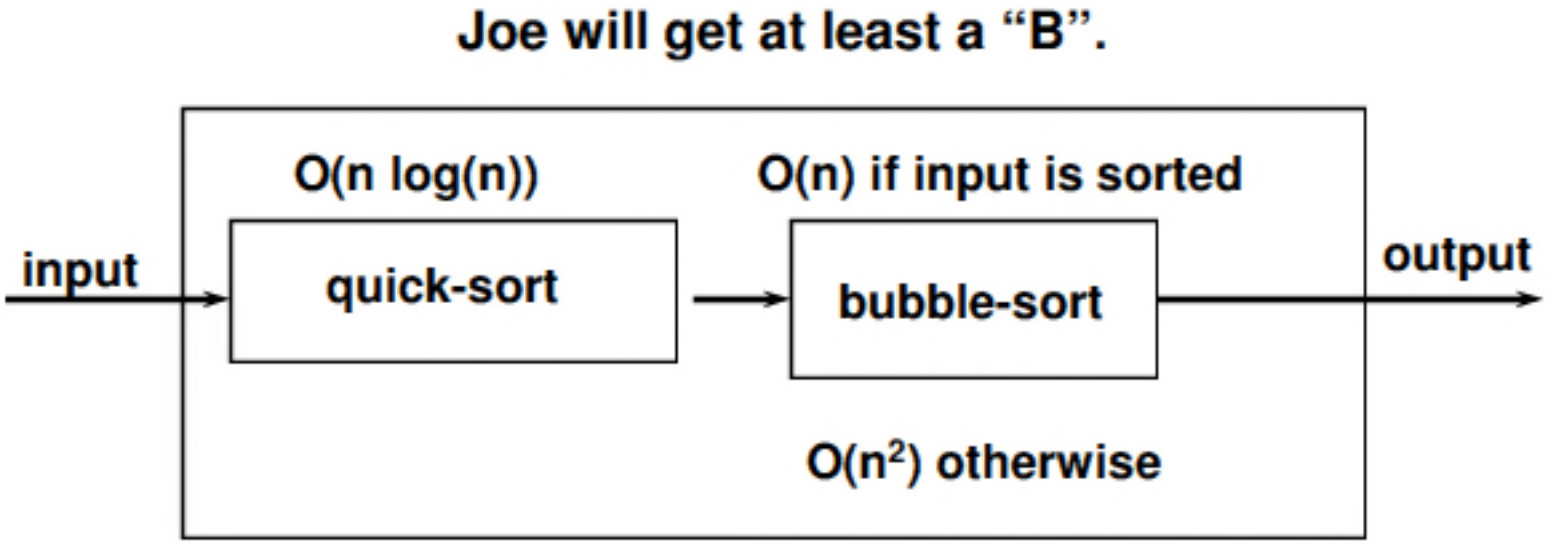
- Sorting:
  - Bubble sort: easy to write but slower, $O(n^2)$
  - Quick sort: faster, $O(n \log(n))$, but more complicated to write
- Joe remembers how to do bubble sort, but is not perfectly sure of quick sort (has a 50% chance of getting it right).
- Joe is asked to write a sorting routine:
  - Correct and fast: A
  - Correct but slow: B
  - Incorrect: F

What is Joe's optimal strategy?

Critical requirement:
Must pass!

Solution

- Simplicity to "control" complexity



Solution

- Key property
  - Use complex but efficient solution in the common case
  - If the complex solution fails, catch the failure and switch to the simple (less efficient) but safe option