Algorithm Design and Analysis

The Nuts and Bolts of Computer Science

Julián Mestre

School of Information Technologies
The University of Sydney

Computational model

In Theoretical Computer Science we study what a computer *can* and *cannot* do.

Rather than working with real computers we work with abstract *computational models* that allow us to make useful predictions about the real world.

The main objective of this lecture is to give you flavor of what computational models look like, and how to design and analyze algorithms for them.

Nuts and bolts

Imagine you are given n nuts and n bolts.

Every nut has a unique perfectly matching bolt, and vice-versa.

You can compare any nut-bolt pair. The outcome can be:

- Nut is too small for the bolt
- Nut is a perfect match for the bolt
- Nut is too large for the bolt

Out goal is to find the n pairs of nuts and bolts.

A cleaner version

Input:

- Set of n nuts and n bolts

Output:

- List of matching nut-bolt pairs

Operations:

- We can compare any nut-bolt pair (three-way outcome)
- We can keep any other record we want

Analysis:

- We count the number of nut-bolt comparison
- We express this count as a function of n

A simple algorithm

For every nut-bolt pair, check if they match.

If they do, output the pair.

Analysis:

- Number of comparisons is n^2

A more sophisticated algorithm

Pick a nut x and find its matching bolt y

Use x to split the remaining bolts into smaller and larger than x

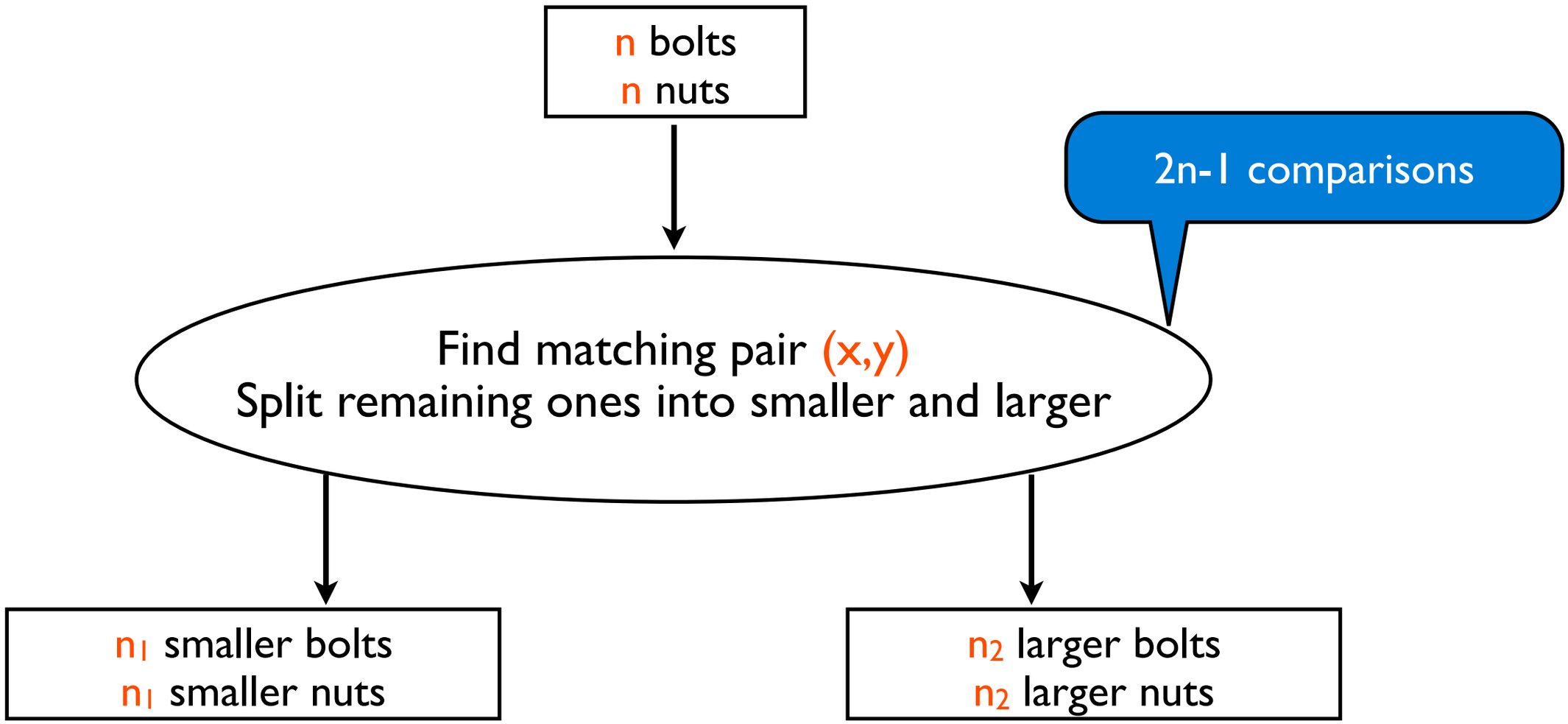
Use y to split the remaining nuts into smaller and larger than y

Output (x,y)

Match up the smaller nuts and bolts

Match up the larger nuts and bolts

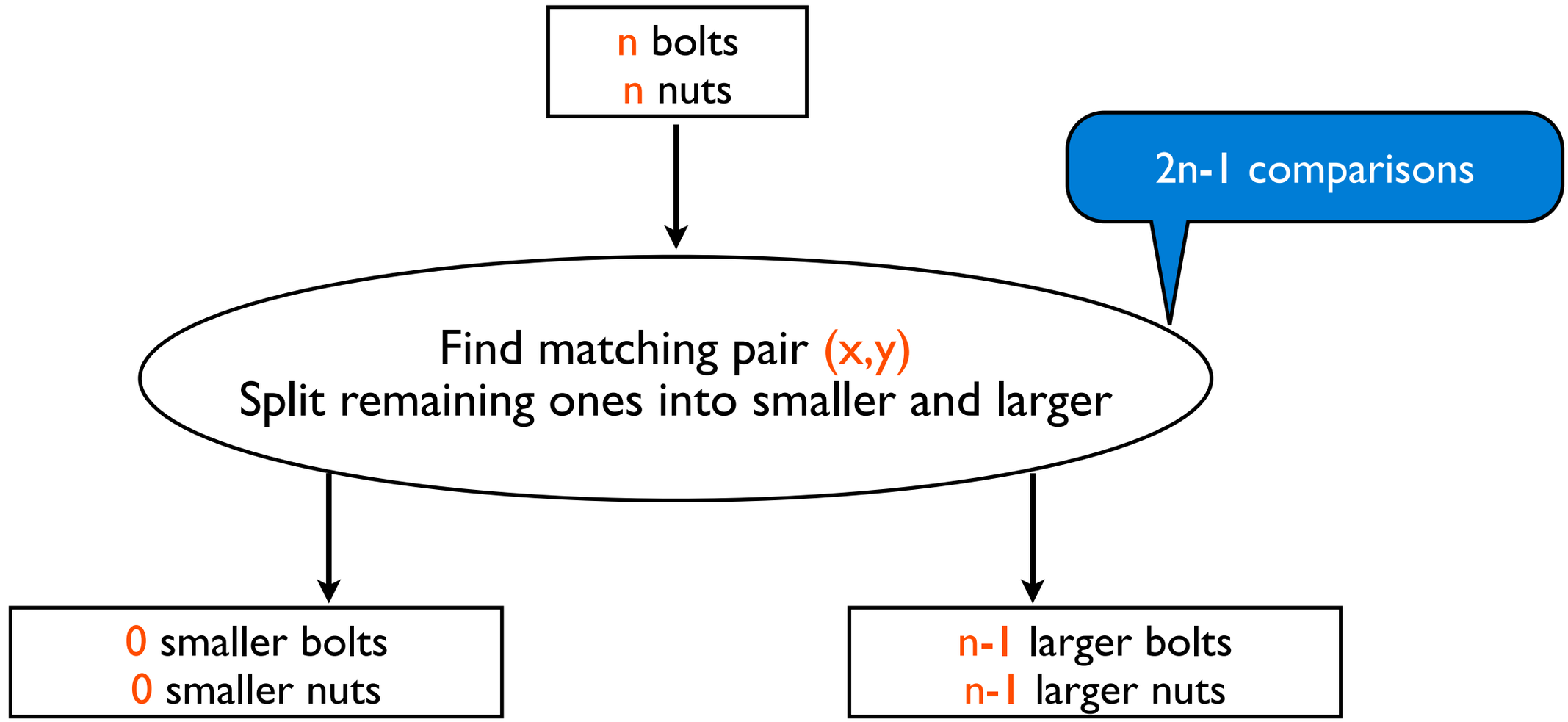
Number of comparisons



Let $f(n)$ be the number of comparisons for n bolts.

Then $f(n) = 2n - 1 + f(n_1) + f(n_2)$

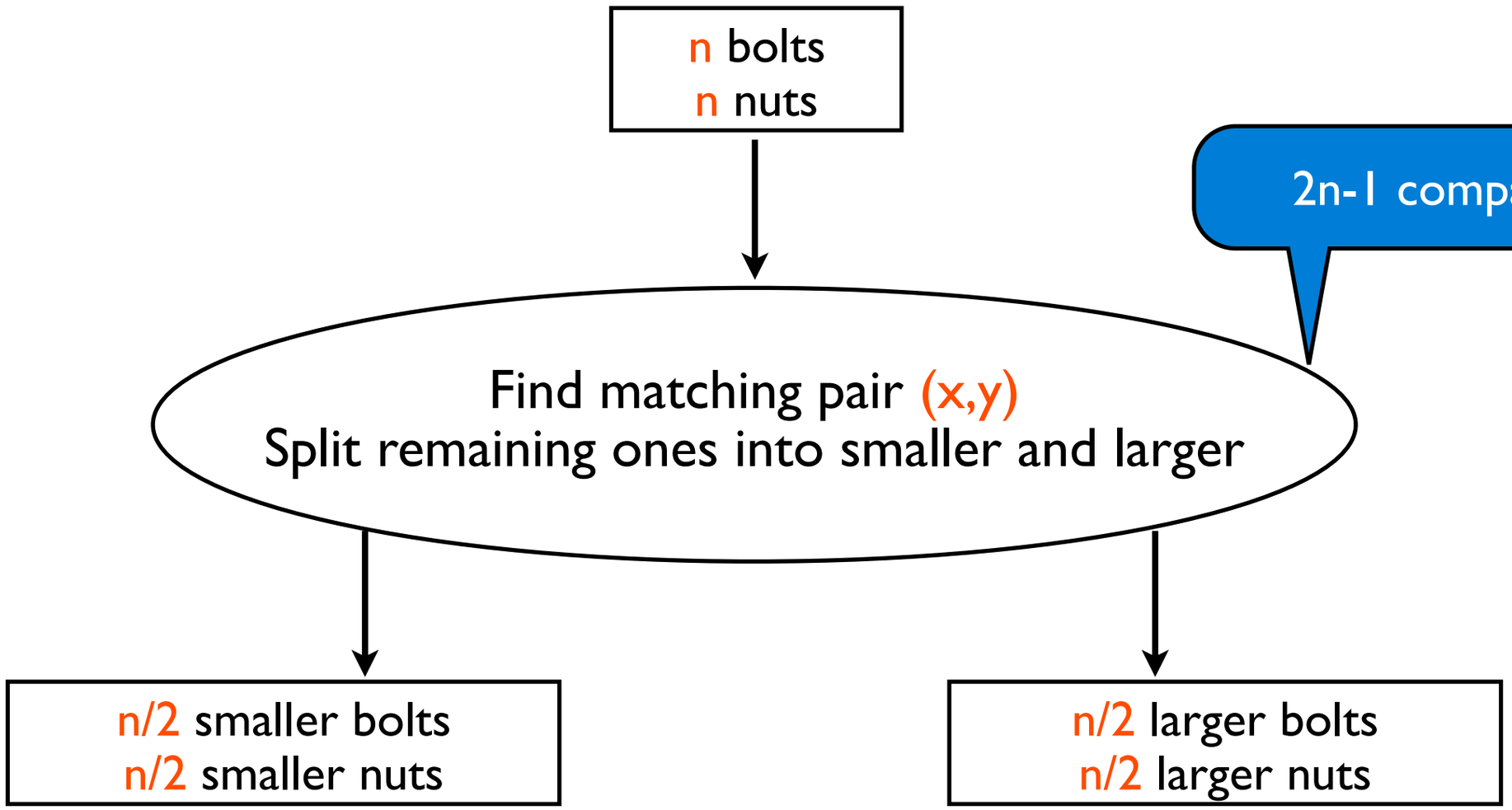
Extreme case: biased partition



Let f(n) be the number of comparisons for n bolts.

Then f(n) = 2n -1+ f(n-1)

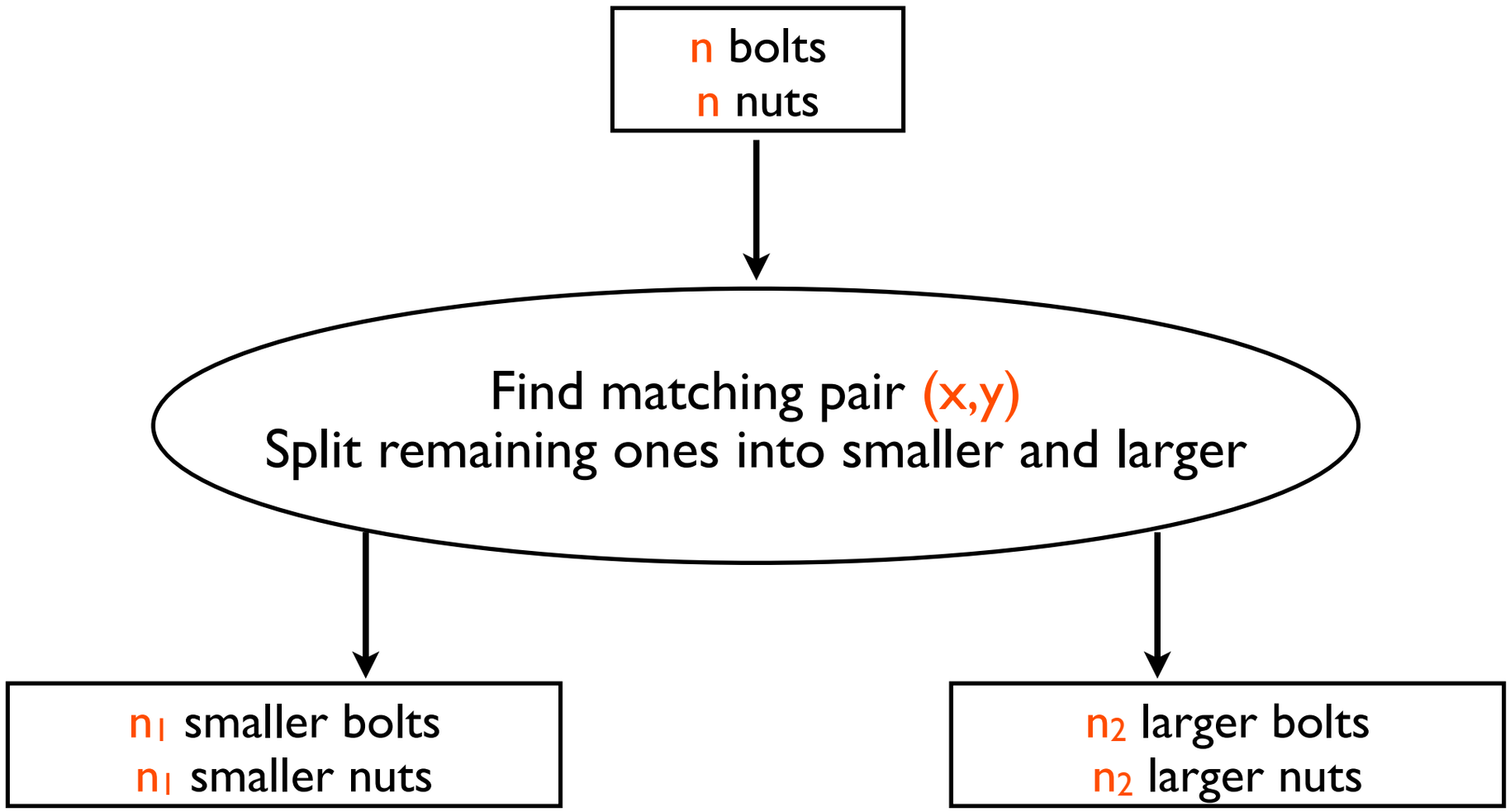
Extreme case: balanced partition



Let f(n) be the number of comparisons for n bolts.

Then $f(n) = 2n - 1 + 2\,f(n/2)$

Likely case: sort of balanced



If we pick the pair (x,y) at random then
the probability that n_1 and n_2 are <u>both</u> less than (2/3)n is 1/3

A better algorithm

Pick a nut x at random and find its matching bolt y

Use x to split the remaining bolts into smaller and larger than x

Use y to split the remaining nuts into smaller and larger than y

Output (x,y)

Match up the smaller nuts and bolts

Match up the larger nuts and bolts

Analysis wrap-up

If we are not careful how to pick the pair (x,y) the number of comparisons can be as bad as n^2

If we pick the pair (x,y) uniformly at random, then the expected number of comparisons is $c\ n \log n$ for some constant c

Back to reality

Even though computers do not compare nuts and bolts, they do compare numbers all the time:

- Google sorts its search results by relevance
- Amazon allows you to sort their products by price, review scores, etc.

The same recursive principle we saw today can be used to sort a list of numbers. This is called the *quicksort* algorithm.

Number of comparisons is a fairly good proxy for actual performance but in practice other factors need to be considered.

Key concepts

Computational model:

- Input
- Output
- Operations

Algorithms:

- Step-by-step sequence of operations
- Take us from input to output

Analysis:

- Measures a single parameter
- Gives predictions of actual performance

Questions?

Thank you for your attention!