Entropy

Entropy

- (Information) Entropy aka *Shannon Entropy* is defined as: the average amount of information produced by a stochastic* source of data.
- What does that mean? Well, let's take an example of a die (singular of *dice*). Each time we roll it, we get a number from 1 to 6.
- How much *information* is that? Well, the fact that we rolled a 6 tells us that we didn't roll a 1, 2, 3, 4, or 5. In particular, we have been given one of six equally-likely possible outcomes. There's no information value to the fact that we didn't roll 17—we knew that was impossible going in.

* *stochastic* is just a fancy word for randomized

Entropy (continued)

- If we have *n* possible outcomes, each with probability p_i, then the average information (i.e. the entropy) is:

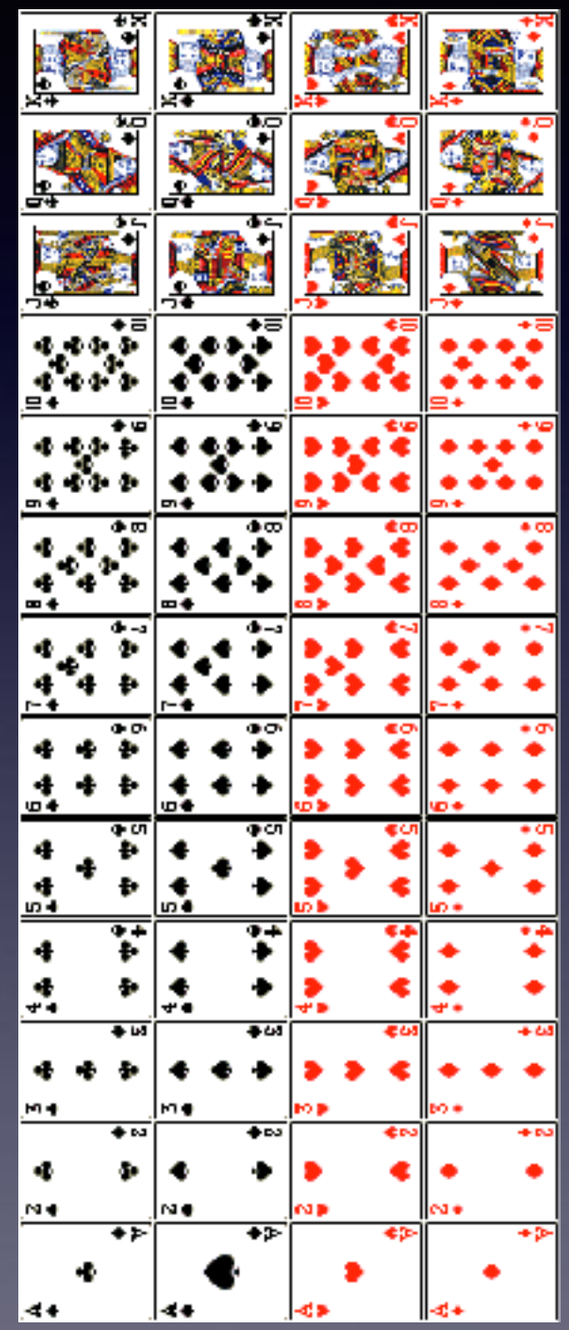
  h = $-\sum_i p_i \log p_i.$

- We normally use log to the base 2 for entropy (which implies that the value can be expressed in *bits*).
- For the die with six possible outcomes,

  h = 2.585 bits where $2^h = 6$

Another example

- Drawing a card from a shuffled deck (52 cards):
  - h = 5.7
- You take the card but don't show me. I ask you several questions which you must answer truthfully:
- Is it a red or black card? Now, h = 4.7
- Is it a round suit ♥♣ or a pointy suit ♠♦? h = 3.7
- Is it an "honor" (T, J, Q, K, or A)
  - If yes, then h = 2.32
  - If no, then h = 3
- Assume no, then is *k* an odd number? h = 2
- Is *k/2* an odd number? h = 1
- Is *k/4* an odd number? h = 0
  - (Now I **know** which card it is because there is zero entropy—i.e. nothing is yet unknown—no surprises to come).



What about a shuffled deck?

- The entropy (average information per card) is 5.7 so one estimate for the total entropy in the deck is 51 * 5.7 = 290 [Actually this is a high estimate because as we go through the deck, cards aren't returned and so the entropy for each drawing is successively smaller].
- There are 52! (8 followed by 67 zeros) possible orderings of the deck and we don't know which one it's in until we've drawn 51 cards (we get the last by process of elimination). So another way to calculate the total entropy of the deck is simply to calculate:

  $lg\ (52!) = 225.6$

- Suppose we were trying to sort the deck so that it was in some pre-defined order? Let's say each operation we perform (like compare + swap) removes 1 bit of entropy.
- **We would need *at least* 226 operations to sort the deck.** We cannot do any better than that, unless the deck happens to be partially ordered, however smart we are!

Generalizing our ideal sort

- We showed that to sort a deck of cards, we'll need at least 226 operations. But what about a list of *n* elements?
- What we did for the deck of cards was to calculate 52! and take its log to base 2.
- What about *n!* ? Is there a way we can approximate this so that we don't have to multiply it out (hard to do when we don't know the actual number!)
- It's called Stirling's Approximation:

$$n! \sim \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$

- What we actually want is the *lg,* i.e. $\log_2$:
  - *~ n(lg n - lg e) + 1/2 lg (2πn)*
  - *~ n(lg n - 1.44)* [as *n* increases, the importance of the 2nd/3rd terms reduces]
- **This expression represents the theoretical minimum number of operations to sort our list of length *n*.**

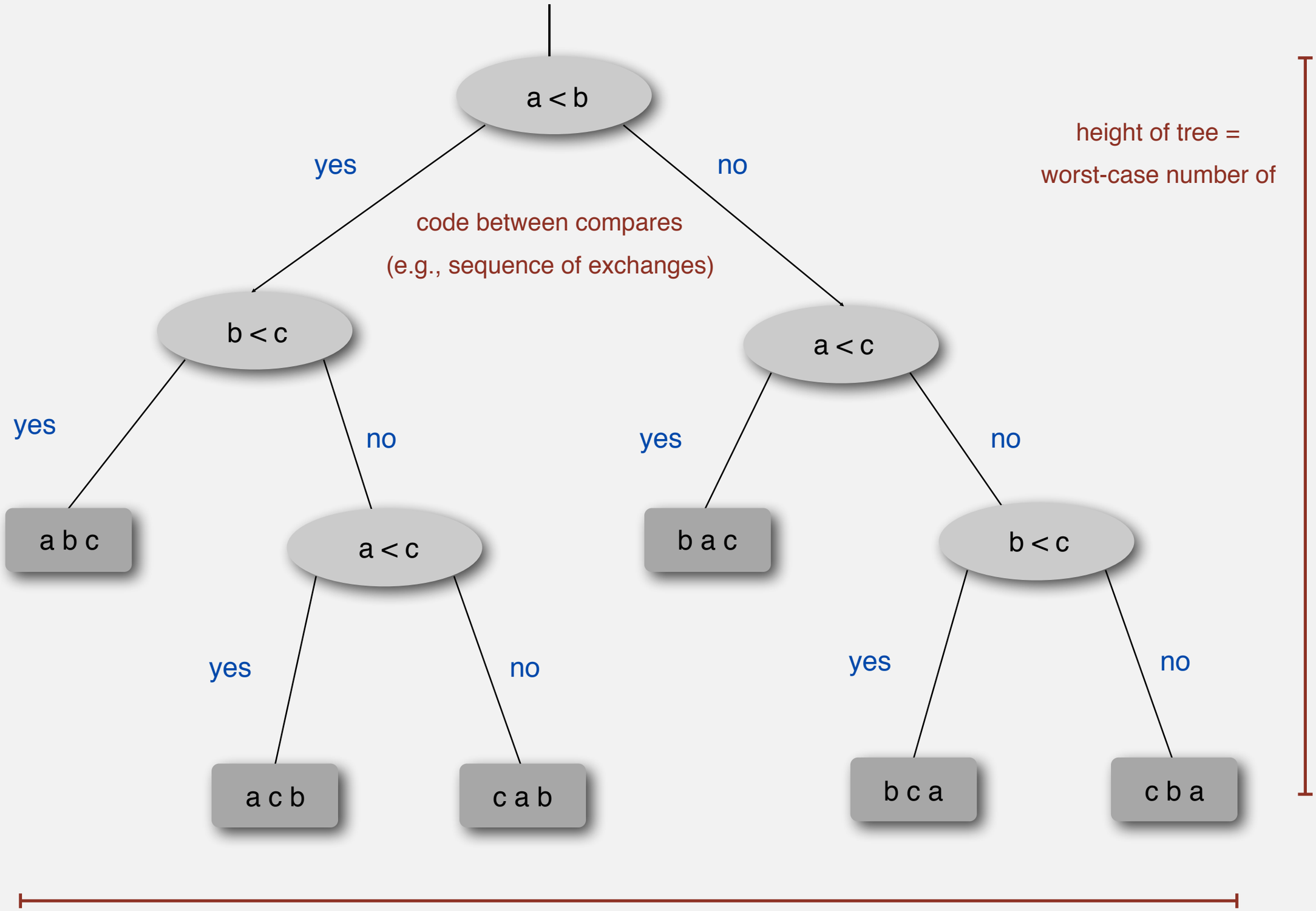
Another look at Stirling's Approximation

- lg (N!) = lg 1 + lg 2 + lg 3 … lg N ~ N lg N
- But this obviously over-estimates the value.
- It was de Moivre who first wrote that:
  - $lg\ N! = N\ lg\ N + c_1\ N + c_2\ lg\ N$
  - Stirling actually derived the values of c_1 & c_2

Another look at compare-based sorting

- Sorting N elements:
  - If we're thinking about compare-based sorting where all the elements are distinct, then we can postulate a (binary) decision tree where the number of leaves is the number of permutations (N!) and the depth of the tree (the longest path) is the minimum number of comparisons which we must make.

Decision tree (for 3 distinct keys a, b, and c)



Going back to our deck of cards

- ~ n(lg n - 1.44)
- ~ 52 (5.7 - 1.44)
- ~ 221.5
- = 225.6
- Without the 1.44 correction, we would estimate: 296.4 which is too high but is commonly used. Remember it arises from the difference between natural logs and log to base 2.

Because of entropy, the best we can do when sorting a randomly ordered list of *n** elements is:
n lg n operations.

* where n is large

Entropy of non-distinct lists

- What about when some of the *n* elements are the same?
  - Let's say we have 10 marbles: three red, three green and four blue.
  - How many possible orderings are there?
  - If we could distinguish between them (by serial number, perhaps), there would be 10! ways of arranging them, i.e. 3,628,800.
- But now, we realize that there are actually 6 ways of rearranging the three reds in such a way that, at a distance, we can't tell the difference.
- Similarly, there are 6 ways to rearrange the greens and 24 ways to rearrange the blues.
- That's 4,200 (that's 864 times fewer arrangements).
- In general, our formula is $n! / m_1! / m_2! / \ldots / m_r!$ where r is the number of distinct classes.
- The entropy, therefore, is $h = lg\,(n!) - \sum (lg\,(m_i!))$.

h(non-distinct lists) part 2

- Let's say there are *r* classes and that a object is equally likely to belong to any class
  - So, we have *n* marbles with *r* different colors
  - $m_i \sim n/r$ for each color *i*
  - So, $h = lg\ (n!) - r\ lg\ ((n/r)!)$
  - And by Stirling's approximation this yields:
    - $h = \sim n(lg\ n - lg\ e) + 1/2\ lg\ (2\pi n) - n\ (lg\ (n/r) - lg\ e) - r/2\ lg\ (2\pi n/r)$
    - $\sim n\ (lg\ n - lg\ n + lg\ r)$
    - $\sim n\ lg\ r$ *

h(non-distinct lists) part 3

- Now, let's think about some real-life examples:
  - Decimal digits (*r* = 10); *h ~ 10 n / 3*
    - Notice that *r* is the "radix" of decimal numbers (a.k.a. the "base")
  - Extended (8-bit) ASCII (*r* = 256); *h ~ 8 n*
  - DNA bases (*r* = 4); *h ~ 2 n*
- If we can perform a <u>stable</u> sort on successive digits (characters) then we can sort numbers and strings in *linear* time!
  - This is the basis of so-called *string sorts;*
  - Because *r* is also known as the <u>radix</u>, we alternatively call these "radix" sorts.