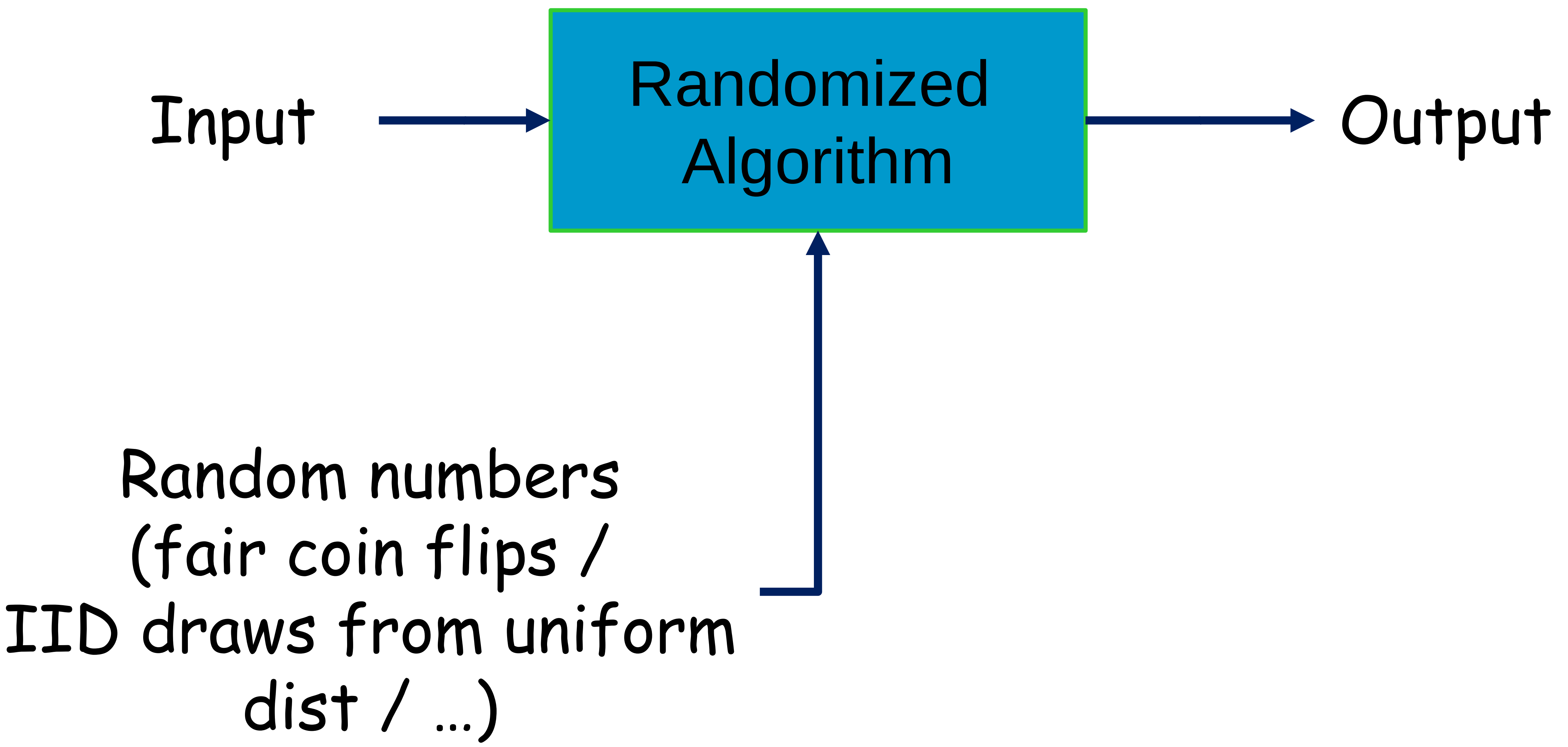
Randomized Algorithms



Order of evaluation of conditions

- Consider the following conditional statement:
- If x>y and z>3 then return x-y;
- If not x>y then there is no need to check the second condition.
- This will save computation time: testing one condition instead of 2.
- What about the case that x>y but not z>3? In this case we need to reverse the order of evaluation to save time.
- But how can the computer decide which order to use?
- Randomize!

Randomized checking of conditions

- Rand()=a random bit with prob ½ for 1, prob ½ for 0.
- If Rand()==1:
  (a) if x>y and z>3: return x-y;
  else:
  (b) if z>3 and x>y: return x-y;
- What can we say about the running time of this algorithm segment?

The possible outcomes

	$x > y$ $z > 3$	$x \leq y$ $z > 3$	$x > y$ $z \leq 3$	$x \leq y$ $z \leq 3$
If x>y and z>3	2	1	2	1
If z>3 and x>y	2	2	1	1
Expected time	2	1.5	1.5	1

Performance bound

- We are comparing three choices:
  1. If x>y and z>3
  2. If z>3 and x>y
  3. Random: Choose 1 or 2 by flipping a fair coin.
- In two cases it does not matter:
  1. If both conditions fail, all choices stop after one test.
  2. If both conditions succeed, all choices stop after two tests.
- In two cases the randomized version has an advantage:
  - If x>y but not z>3: choice 1 takes 2 steps, choice 2 takes 1 step and random choice takes 1.5 steps in expectation.
  - If z>3 but not x>y: choice 2 takes two steps, choice 1 takes 1 step, and random choice takes 1.5 steps in expectation.

Single occupant hashing

- Suppose we have an array A with N entries. In each entry we can store a single (key,value) pair.
- Initially A is empty. We add new elements to it one by one.
- Suppose N/2 of the N entries are filled. What is a good strategy for finding an empty slot?
  - Search through 1,2,3,4,…
  - Search through N,N-1,N-2,….
  - Search using some deterministic order.
- It is not hard to see that, in the worst case, any deterministic method will check **N/2+1** entries before finding an empty entry.
- Randomized algorithm:
  Repeat until empty slot found:
  - choose a random number uniformly at random from 1,…,N

Expected search time for randomized algorithm

- Let n be the RV corresponding to the number of locations checked until an empty location is found.
- The distribution of n is geometric:
  - n=1 with probability ½
  - n=2 with probability ¼
  - n=3 with probability 1/8
  - …
  - n=k with probability $\left(\frac{1}{2}\right)^k$
- The expected number of checks is : $E(n) = \sum_{i=1}^{\infty} i \left(\frac{1}{2}\right)^i = 2$

Excercise

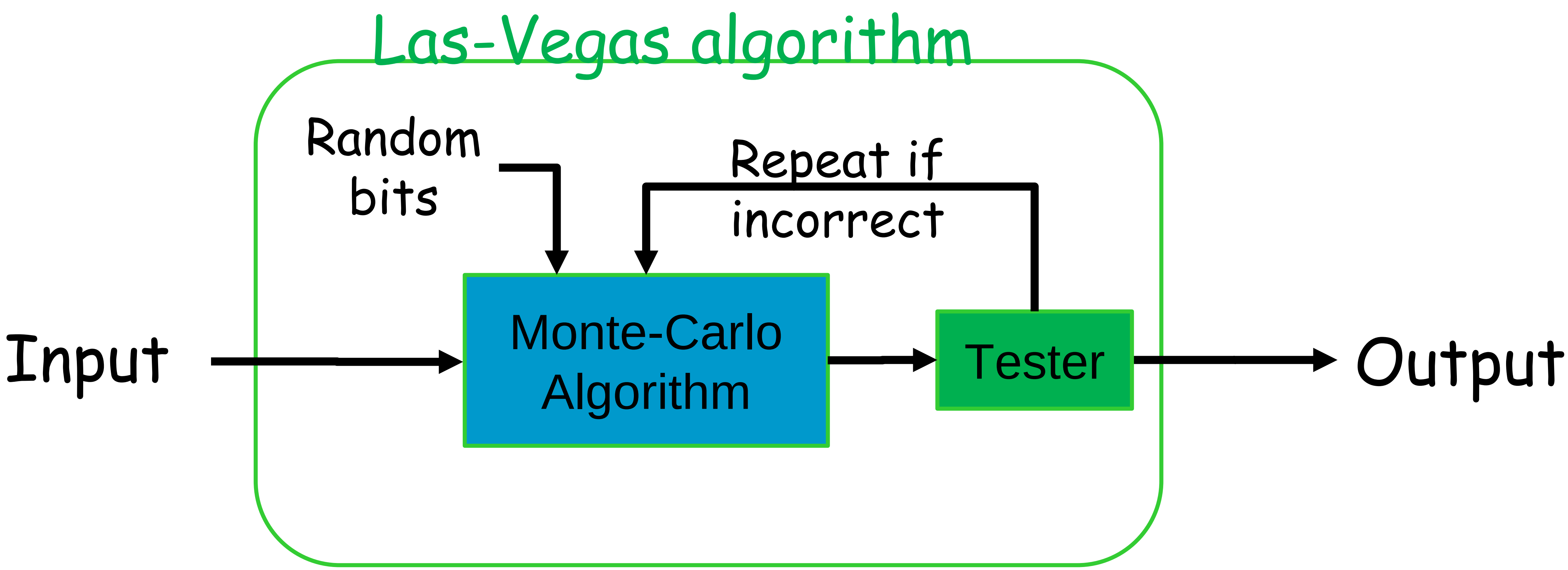
- Suppose the table has N=200 cells and 191 of them are full.
- What is the expected number of random pokes until the algorithm finds an empty cell?

Las Vegas vs. Monte-Carlo

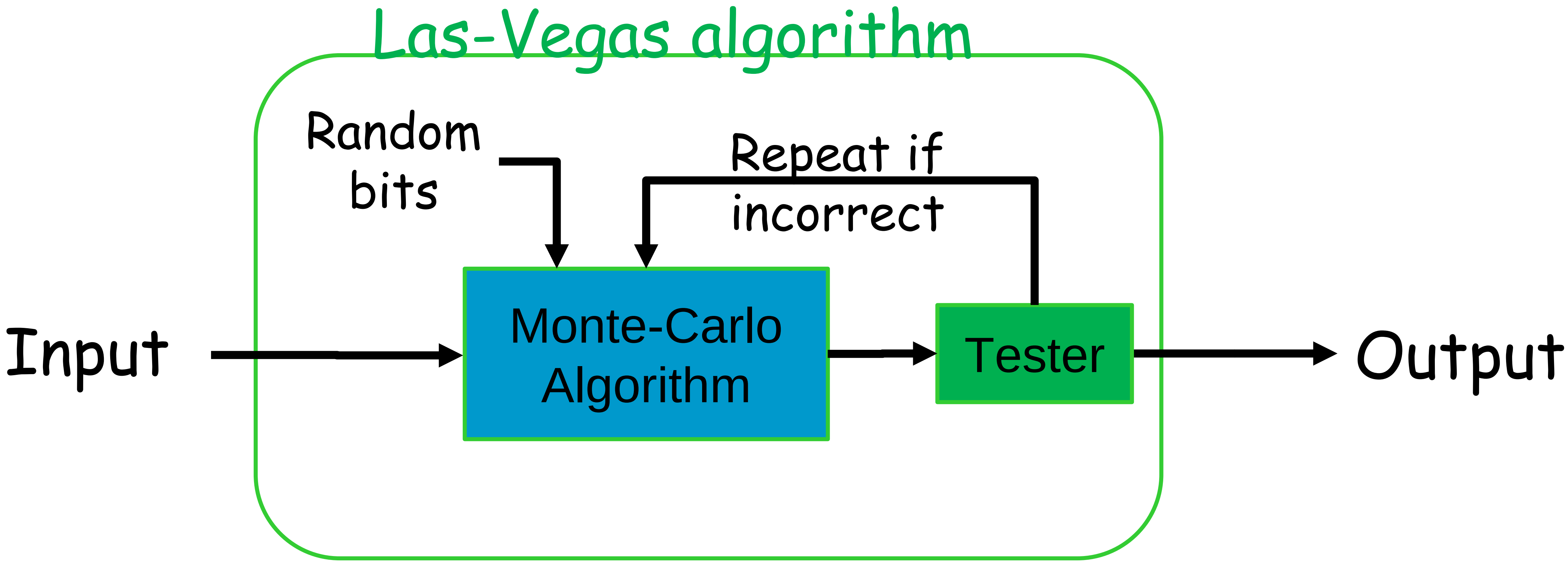
- A las-vegas algorithm:
  - Always finds a correct answer
  - Expected running time is bounded.
  - Example: The randomized algorithm for finding an empty cell.
- A monte-carlo algorithm:
  - Finds the correct answer with non-zero probability.
  - Running time is bounded.
  - Example: A single poke into the array, checking whether the cell is empty.
- From las vegas to monte carlo: early stopping.
- From monte-carlo to las-vegas: test and repeat.

From Monte-Carlo to Las Vegas (1)

- We often want a las-vegas style algorithm:
  - Find an empty cell in an array
  - Find out whether a conjunction is true.
- We can transform a Monte-Carlo algorithm into Las Vegas IF there is an efficient algorithm for testing to see if an answer is correct



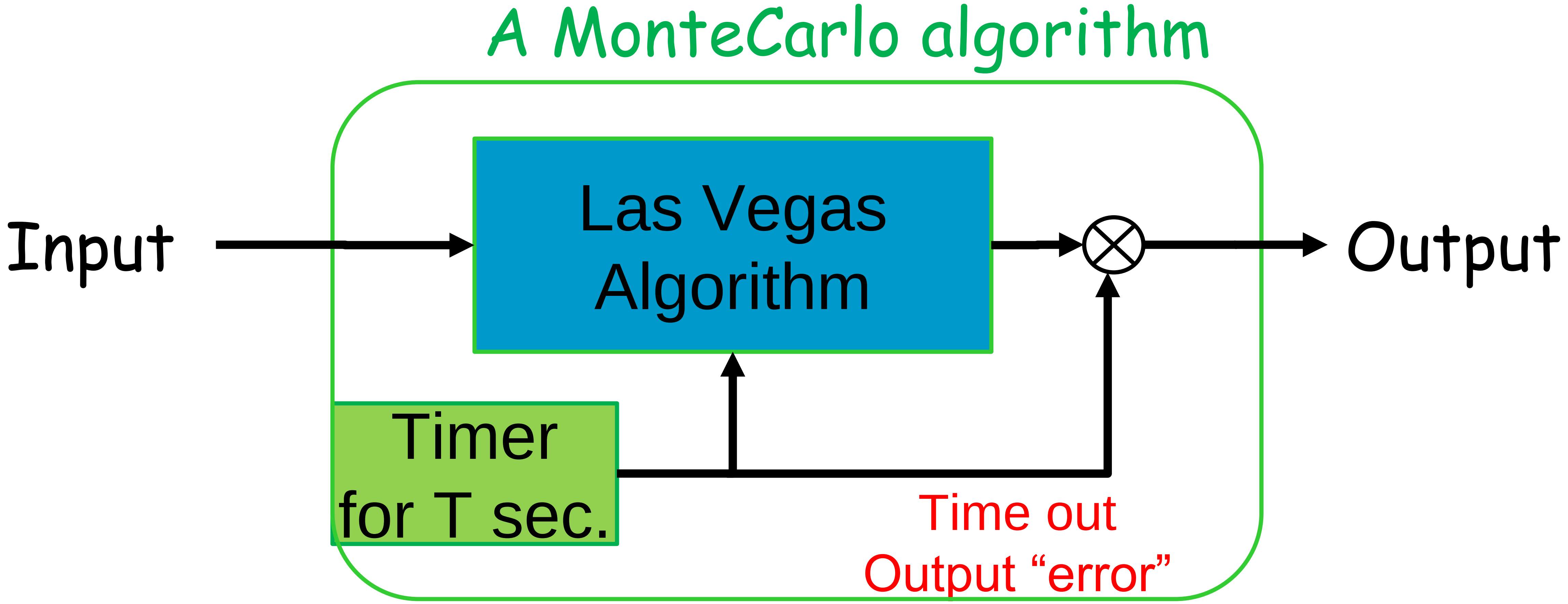
From Monte-Carlo to Las Vegas (2)



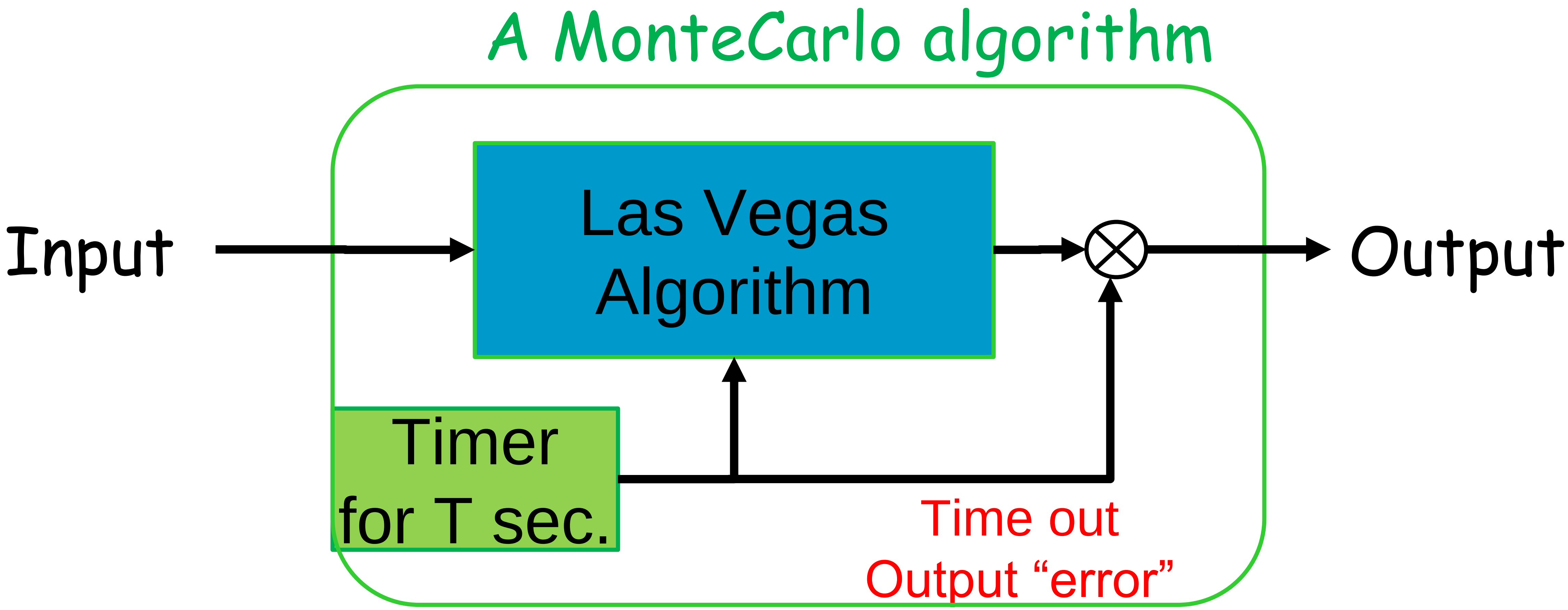
- Suppose the probability that the Monte Carlo algorithm generates a correct answer is p
- Let n be the number of times the Monte Carlo runs until it is successful.
- What is expected number of tries until success?
- Same as expected number of coin flips with $P(\text{heads})=p$ until we get the first heads. $E(n)=1/p$

From las vegas to monte carlo (1)

- Situations where we need the result within a time limit, does not have to always be correct:
  - Robotics
  - Communication protocols.
- We can transform a las-vegas algorithm to monte-carlo:



From las vegas to monte carlo (2)



- The monte-carlo always output a correct answer
- The transformed algorithm is incorrect iff the running time is at least T
- $E(t)$ is the expected running time of the algorithm.
- From Markov Inequality we get that $\mathrm{P(error)} = P(t \geq T) \leq \frac{E(t)}{T}$

Computing Percentiles

A problem with the average

$$Average(X_1,\ldots,X_n) \doteq \frac{1}{n}\sum_{i=1}^{n} X_i$$

The average is the most common estimator of the "center" of a distribution.
It takes linear time to compute.

However, the average is "sensitive to outliers" :

Suppose that you have a company in which 1000 employees earn 1$/day and one employee earns 1000$/day. The average dayly pay is 2000/1001 ~ 2$/day, but that is double what most people earn.

Using the Median instead of the average

To compute the median sort all n elements from smallest to largest and take the value of the element that is the middle of the list (position $n/2$) (take the average of the two middle elements if the list length is even).

In the earlier example, the median will be 1$ regardless of how big is the largest salary - outliers are ignored.

The median is a special case of percentiles

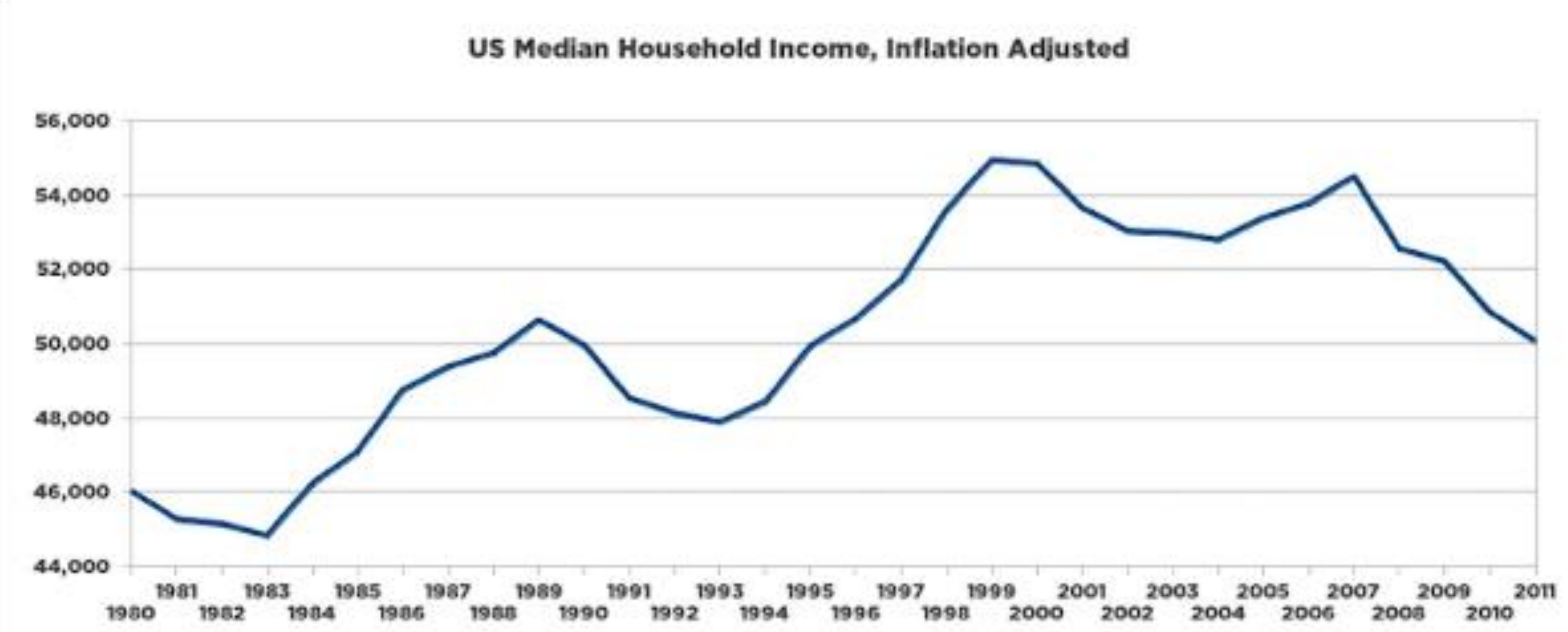
To compute the P-percentile sort all n elements from smallest to largest and take the value of the element that is in the floor(Pn) position.

The Median is the 1/2-percentile

Table 1: Income, net worth, and financial worth in the U.S. by percentile, in 2010 dollars

Wealth or income class	Mean household income	Mean household net worth	Mean household financial (non-home) wealth
Top 1 percent	$1,318,200	$16,439,400	$15,171,600
Top 20 percent	$226,200	$2,061,600	$1,719,800
60th-80th percentile	$72,000	$216,900	$100,700
40th-60th percentile	$41,700	$61,000	$12,200
Bottom 40 percent	$17,300	-$10,600	-$14,800

From Wolff (2012); only mean figures are available, not medians. Note that income and wealth are separate measures; so, for example, the top 1% of income-earners is not exactly the same group of people as the top 1% of wealth-holders, although there is considerable overlap.



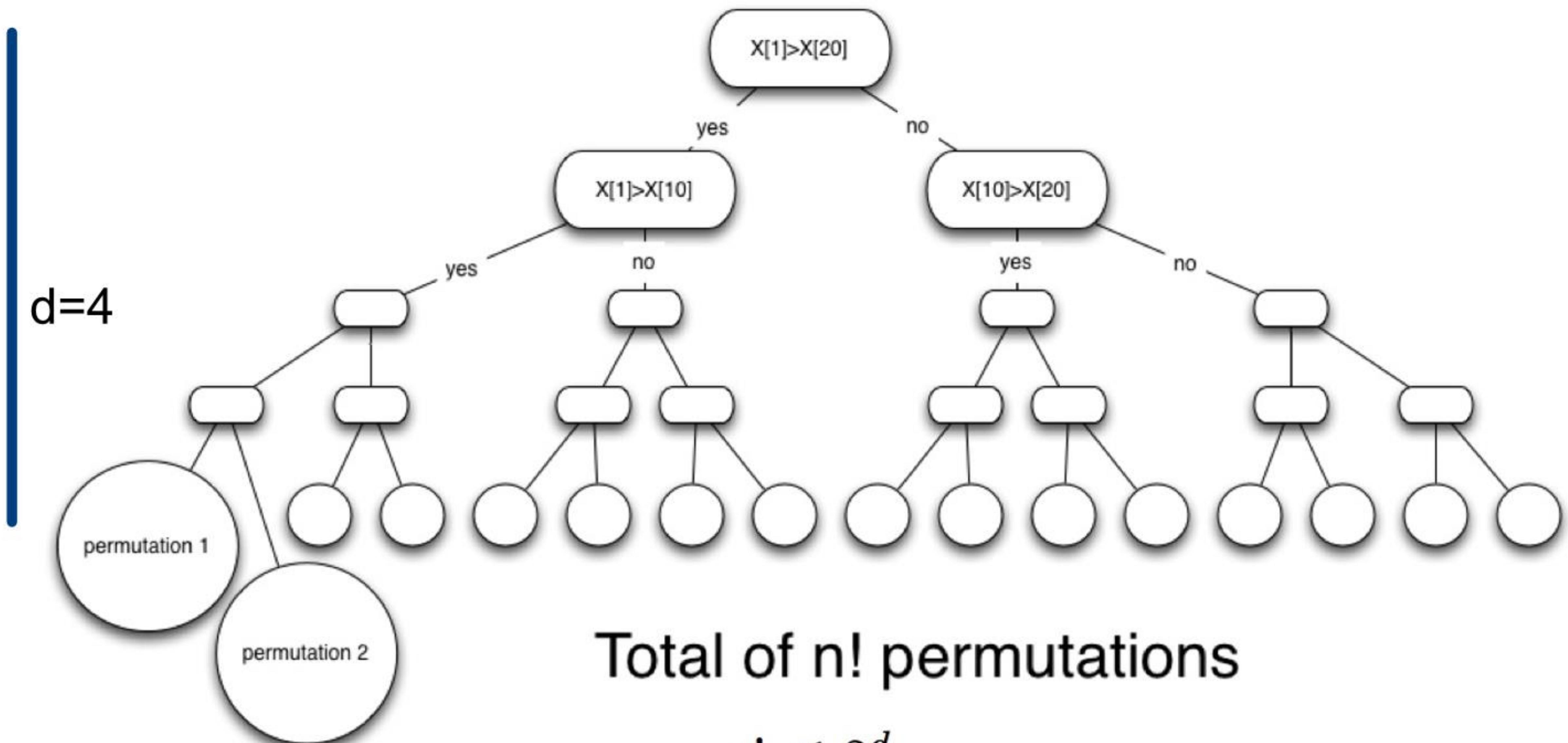
The median American family is not doing very well…

A linear time algorithm for computing percentiles

We can calculate P-percentile by sorting and then picking the element in location Pn. But this requires time

We will now describe a randomized algorithm whose expected running time is O(n)

sorting requires n log(n) time in the worst case



Total of n! permutations

$$n! \leq 2^d$$

$$d \ln 2 \geq \ln n! \geq (n-1) \ln n$$

Find the 5^{th} smallest element in the following table:

7	**10**	**3**	**20**	**15**	**16**	**70**	**33**	**30**	**7**

Solution 1, sort and locate ---- takes worst case time O(n log n):

3	**7**	**7**	**10**	**15**	**16**	**20**	**30**	**33**	**70**

Expected time is also Omega(n log n):

Find the 5th smallest element in the following table:

7	**10**	**3**	**20**	**15**	**16**	**70**	**33**	**30**	**7**

Solution 2, randomized algorithm ---- takes ***expected*** time O(n):

Find the 5^{th} smallest element in the following table:

7	**10**	**3**	**20**	**15**	**16**	**70**	**33**	**30**	**7**

Choose a random element as pivot

7	**10**	**3**	**20**	**15**	**16**	**70**	**33**	**30**	**7**

Find the 5th smallest element in the following table:

7	10	3	20	15	16	70	33	30	7

Choose a random element as pivot

7	10	3	20	15	16	70	33	30	7

Partition list into 3 lists: <7 , =7 , >7

Find the 5th smallest element in the following table:

7	10	3	20	15	16	70	33	30	7

Choose a random element as pivot

7	10	3	20	15	16	70	33	30	7

Partition list into 3 lists: <7 , =7 , >7

3

S_L

Find the 5th smallest element in the following table:

7	10	3	20	15	16	70	33	30	7

Choose a random element as pivot

7	10	3	20	15	16	70	33	30	7

Partition list into 3 lists: <7 , =7 , >7

3

S_L

7	7

S_V

Find the 5th smallest element in the following table:

7	10	3	20	15	16	70	33	30	7

Choose a random element as pivot

7	10	3	20	15	16	70	33	30	7

Partition list into 3 lists: <7 , =7 , >7

3

S_L

7	7

S_V

10	20	15	16	70	33	30

S_R

Find the 5th smallest element in the following table:

7	10	3	20	15	16	70	33	30	7

Choose a random element as pivot

7	10	3	20	15	16	70	33	30	7

Partition list into 3 lists: <7 , =7 , >7

3

S_L

7	7

S_V

10	20	15	16	70	33	30

S_R

Find the 2nd smallest element in the following table:

10	20	15	16	70	33	30

Find the 5th smallest element in the following table:

7	10	3	20	15	16	70	33	30	7

Choose a random element as pivot

7	10	3	20	15	16	70	33	30	7

Partition list into 3 lists: <7 , $=7$, >7

3

S_L

7	7

S_V

10	20	15	16	70	33	30

S_R

Find the 2nd smallest element in the following table:

10	20	15	16	70	33	30

recurse

The split operation

- Choosing a split value v we divide the set S into three subsets:
  - $S_L=\{x \in S | x<v\}$
  - $S_v=\{x \in S | x=v\}$
  - $S_R=\{x \in S | x>v\}$

After we split, we know how to continue

- We know the size of the three sets: $|S_L|$, $|S_v|$, $|S_R|$.
- If median is in S_v, then we are done.
- Otherwise we continue with either S_L or S_R

Randomized algorithm

- Let S be a set of N different numbers. Suppose we pick a random element of S to be the pivot v. What is the probability that the size of S_L is equal to 10 ?
  - 1/10
  - 9/10
  - 1/N

Randomized algorithm

- Let S be a set of N different numbers. Suppose we pick a random element of S to be the pivot v. What is the probability that the size of S_L is between $\lceil N/4 \rceil$ and $\lceil 3N/4 \rceil$?

A. About ¼

B. About ½

C. About ¾

D. About 1/N

Lucky splits

- We say that the split of a set S of size N is lucky if
- $\frac{1}{4}N \leq |S_L| \leq (3/4)N$
- Which implies also that $\frac{1}{4}N \leq |S_u| \leq (3/4)N$
- If the split is lucky then the size of the set we operate on decreases by a factor of (3/4)
- In order to reduce the set to all-equal elements we need at most k lucky splits:

$$\left(\frac{3}{4}\right)^k N \leq 1 \;\Rightarrow\; k \log\frac{3}{4} + \log N \leq 0 \;\Rightarrow\; k \geq \frac{\log N}{\log(3/4)}$$

Expected time to first lucky split

- What is the expected number of random splits until we get a lucky split?
  A. 1
  B. 2
  C. 1/2

Expected Running time

$n =$ The number of elements in the input array.

$T(n) =$ The expected running time of the algorithm

$$T(n) \le n + \frac{1}{2}T(n) + \frac{1}{2}T\left(\frac{3}{4}n\right)$$

Multiply both sides by 2 and rearrange:

$$2T(n) \le 2n + T(n) + T\left(\frac{3}{4}n\right); \qquad T(n) \le 2n + T\left(\frac{3}{4}n\right)$$

$$T(n) \le 2n + T\left(\frac{3}{4}n\right)$$

$$T(n) \le 2n + \frac{3}{4}2n + T\left(\left(\frac{3}{4}\right)^2 n\right)$$

$$T(n) \le 2n + \left(\frac{3}{4}\right)2n + \left(\frac{3}{4}\right)^2 2n + T\left(\left(\frac{3}{4}\right)^2 n\right)$$

$$T(n) \le 2n\left(1 + \left(\frac{3}{4}\right) + \left(\frac{3}{4}\right)^2 + \left(\frac{3}{4}\right)^3 + \cdots\right) + T(\le 1)$$

$$T(n) \le 2n\frac{1}{1-(3/4)} = 8n$$

This is an upper bound - the actual constant is smaller.