Data Structures & Algorithms

Randomized Algorithms

- Introduction
- Random Variable
- Approximate Median
- Hiring Problem
- Selection
- Quicksort

Randomization

Algorithmic design patterns.

- Greedy.
- Divide-and-conquer.
- Dynamic programming.
- Network flow.
- Randomization.

Randomized Algorithms. A randomized algorithm is an algorithm whose working not only depends on the input but also on certain random choices made by the algorithm.

Assumption. We have a random number generator Random(a, b) that generates for two integers a, b with $a < b$ an integer r with a $a \leq r \leq b$ uniformly at random. We assume that Random(a, b) runs in O(1) time or precisely fair coin flip is done in unit time.

Why randomize? Can lead to simplest, fastest, or only known algorithm for a particular problem.

Ex. graph algorithms, quicksort, hashing, load balancing, cryptography.

Random Variable

Finite Probability Space

- An Experiment is a procedure that yields one of a given set of possible outcomes.
- The sample space of the experiment is the set of possible outcomes.
- An event is an subset of the sample space.
- A (finite) probability space is a pair (Ω, P) where Ω is a finite set (the sample space) and P is an additive measure on subsets of Ω with $P(\Omega) = 1$. Any subset of Ω is called an event and each element of Ω is called an elementary event.

Finite Probability Space

- The probability measure (probability distribution) is determined by its value on elementary events: in other words, by specifying a function $P: \Omega \to [0,1]$ with $\sum_{\omega \in \Omega} P(\omega) = 1$. Then, the probability measure on an event A is given by $\mathrm{P[A]} = \sum_{a \in A} P[a]$

- The basic example of a probability measure is the uniform distribution on Ω, where $P[A] = \frac{|A|}{|\Omega|}$ for any $A \subseteq \Omega$. Such a distribution represents the situation where any outcome of an experiment (such as rolling a dice) is equally likely.

Examples

Example: Consider the experiment of tossing a fair coin. We have $\Omega = \{H, T\}$ and $P[H] = P[T] = 1/2$.

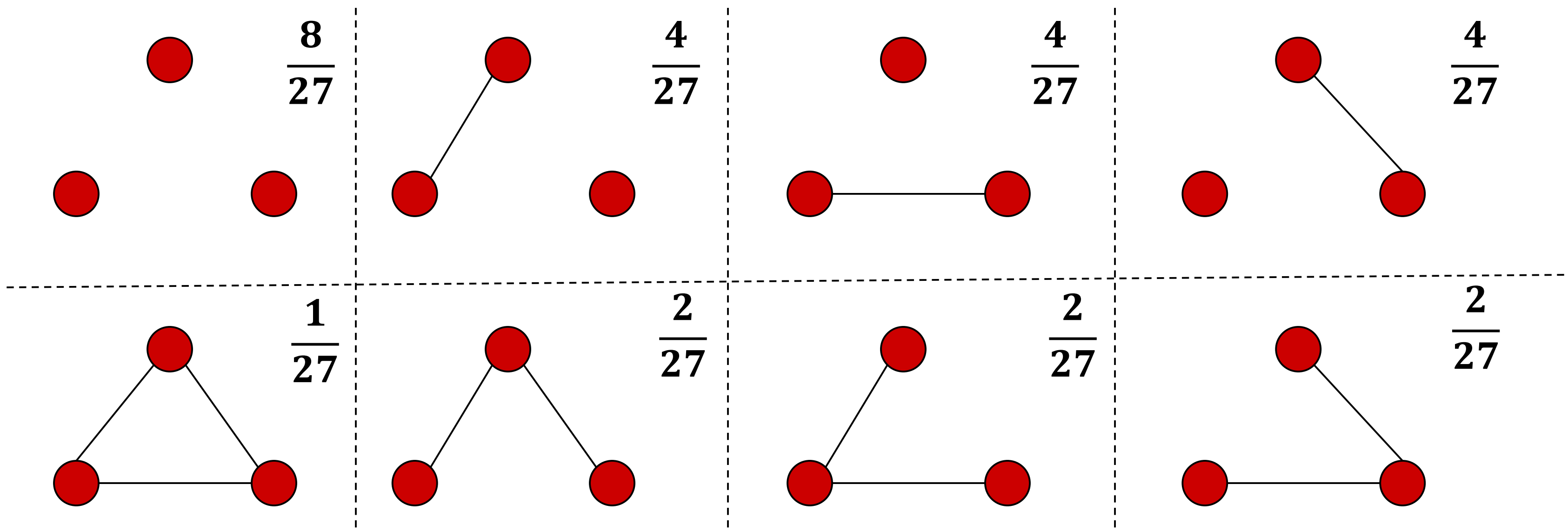
Example: Consider the experiment of tossing a fair coin twice. We have $\Omega = \{(H,H),(H,T),(T,H),(T,T)\}$ and $P[(H,H)] = P[(T,T)] = P[(H,T)] = P[(T,H)] = 1/4$.

Example: Consider the experiment of rolling a fair dice. We have $\Omega = \{1,2,3,4,5,6\}$ and $P[1] = \cdots = P[6] = 1/6$.

Example: Consider the experiment of rolling a fair dice first and then tossing a fair coin. We have $\Omega = \{(1,H), \ldots, (6,H), (1,T), \ldots, (6,T)\}$ and $P[(i,H)] = P[(i,T)\ ] = 1/12$ for any i

The Probability Space $G(n,p)$

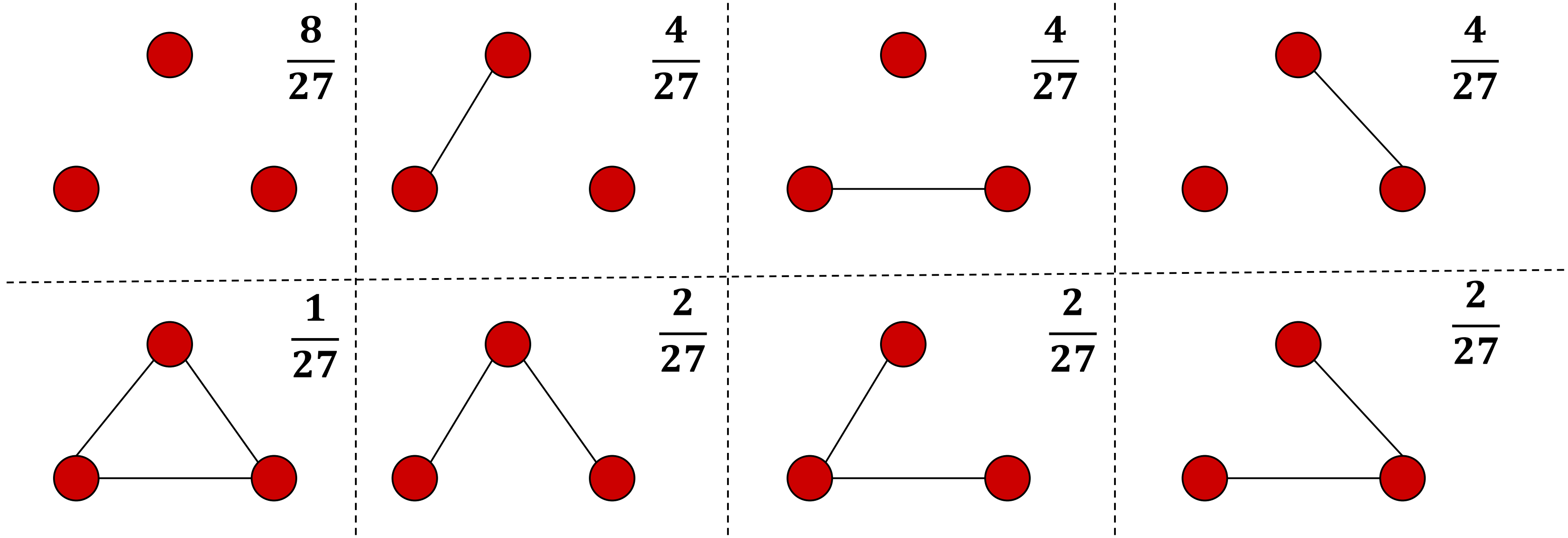
P.S. $G(n,p)$: Consider the experiment of constructing a graph with n vertices where each edge appears in the graph with probability p. The sample space Ω of $G(n,p)$ is the set of all graphs on a fix set of n vertices. The probability of graph with m edges is $p^m(1-p)^{\binom{n}{2}-m}$. For $n=3$ and $p=1/3$ elements of Ω with their probability are illustrated below.



The Probability Space $G(n, p)$

How to do the experiment (construct a random graph)

- Method 1: For each edge toss a coin where $P[H] = p, P[T] = 1 - p$
- Method 2: Produce all elements of the sample space with their probability like below. Then u.a.r. select an integer number in $[1,27]$. If it is between 1 and 8, select top-left one, if it is between 9 to 12, select the one next to top-left one, and so on.



Independent Sets

Given a graph G. Consider a probability space whose sample space Ω is the set of independent sets of G (an independent set is a subset of vertices that there is no edge between any two vertices of the subset) and whose probability measure is an uniform distribution.

How to do the experiment (select a random independent set)

One simple way is to produce all independent set and number them from 1 to m where m is the number of independent set of G. Then select an integer k in $[1, m]$ u.a.r. and report the independent set whose number is k. This method is not efficient (takes too much time as m can be exponential). Any faster method?

Random Variable

- A Random Variable X is a function from the sample space to real numbers (i.e X: $\Omega \rightarrow R$). A random variable assign a real number to each possible outcome. procedure that yields one of a given set of possible outcomes.
- Note that a random variable is a function. It is not a variable, and it is not random.

The distribution of a random variable X:

- $P[X = r] = \sum_{\omega: X(\omega)=r} P[\omega]$
- Consider event $A_r = \{\omega: X(\omega) = r\}$. So $P[X = r] = P[A_r]$
- Since the input of X is a random phenomena, the output of X is a random number in the codomain of X with the above distribution.

Examples

Problem: Suppose that a fair coin is flipped three times. Let $X(\omega)$ be the random variable that equal to the number of heads that appear when ω is the outcome. Then $X(\omega)$ takes on the following values.

$X(HHH) = 3, X(HHT) = X(HTH) = X(THH) = 2, X(TTH) = X(THT) = X(HTT)$
$= 1, X(TTT) = 0$
$P[X = 0] = \frac{1}{8}, P[X = 1] = \frac{3}{8}, P[X = 2] = \frac{3}{8}, P[X = 3] = \frac{1}{8}$

Examples

Problem: Let X be the sum of the numbers that appear when a pair of dice is rolled. What are the values of this random variable for the 36 possible outcome (i, j)?

Solution:

$$
\begin{aligned}
&X((1,1)) = 2,\\
&X((1,2)) = X((2,1)) = 3,\\
&X((1,3)) = X((2,2)) = X((3,1)) = 4,\\
&X((1,4)) = X((2,3)) = X((3,2)) = X((4,1)) = 5,\\
&X((1,5)) = X((2,4)) = X((3,3)) = X((4,2)) = X((5,1)) = 6,\\
&X((1,6)) = X((2,5)) = X((3,4)) = X((4,3)) = X((5,2)) = X((6,1)) = 7,\\
&X((2,6)) = X((3,5)) = X((4,4)) = X((5,3)) = X((6,2)) = 8,\\
&X((3,6)) = X((4,5)) = X((5,4)) = X((6,3)) = 9,\\
&X((4,6)) = X((5,5)) = X((6,4)) = 10,\\
&X((5,6)) = X((6,5)) = 11,\\
&X((6,6)) = 12.
\end{aligned}
$$

The Expected Value

The Expected Value of a random variable X on the sample space Ω is equal to $E(X) = \sum_{\omega \in \Omega} P[\omega] X(\omega)$

Problem: what is the expected edges of a random graph in $G(3,1/3)$.

Solution:

Let $X(\omega)$ be the number of edges of ω where ω is a graph in the sample space of $G(3,1/3)$. We should compute $\mathrm{E}(X)$ which is

$$\sum_{\omega \in \Omega} P[\omega] X(\omega) = \frac{8}{27} \times 0 + \frac{4}{27} \times 1 + \frac{4}{27} \times 1 + \frac{4}{27} \times 1 + \frac{2}{27} \times 2 + \frac{2}{27} \times 2 + \frac{2}{27} \times 2 + \frac{1}{27} \times 3 = \frac{8}{27} \times 0 + \left(\frac{4}{27} + \frac{4}{27} + \frac{4}{27}\right) \times 1 + \left(\frac{2}{27} + \frac{2}{27} + \frac{2}{27}\right) \times 2 + \frac{1}{27} \times 3$$

$$= \sum_{r \in R} P[X = r] r$$

$$E(X) = \sum_{\omega \in \Omega} P[\omega] X(\omega) = \sum_{r \in R} P[X = r] r$$

Linearity of Expectations

Theorem: Let X_i $(i = 1, \dots, n)$ be random variables on the sample space Ω, and let a and b be two real numbers, then

(i) $E(\sum_{i=1}^{n} X_i) = \sum_{i=1}^{n} E(X_i)$

(ii) $E(aX + b) = aE(X) + b$

Proof:

(i) $E(X_1 + X_2) = \sum_{\omega\in\Omega} P[\omega](X_1(\omega) + X_2(\omega)) = \sum_{\omega\in\Omega} P[\omega] X_1(\omega) + \sum_{\omega\in\Omega} P[\omega] X_2(\omega) = E(X_1) + E(X_2)$

(ii) $E(aX + b) = \sum_{\omega\in\Omega} P[\omega](aX(\omega) + b) = a \sum_{\omega\in\Omega} P[\omega] X(\omega) + b \sum_{\omega\in\Omega} P[\omega] = aE(X) + b$

Linearity of Expectations

Problem: Compute the expected value of the sum of the numbers that appear when a pair of fair dice is rolled.

Solution:

One way is to list 36 outcomes and compute the value of X and its probability like below.

$$p(X = 2) = p(X = 12) = 1/36,$$
$$p(X = 3) = p(X = 11) = 2/36 = 1/18,$$
$$p(X = 4) = p(X = 10) = 3/36 = 1/12,$$
$$p(X = 5) = p(X = 9) = 4/36 = 1/9,$$
$$p(X = 6) = p(X = 8) = 5/36,$$
$$p(X = 7) = 6/36 = 1/6.$$

$$\begin{aligned} E(X) &= 2 \cdot \frac{1}{36} + 3 \cdot \frac{1}{18} + 4 \cdot \frac{1}{12} + 5 \cdot \frac{1}{9} + 6 \cdot \frac{5}{36} + 7 \cdot \frac{1}{6} \\ &\quad + 8 \cdot \frac{5}{36} + 9 \cdot \frac{1}{9} + 10 \cdot \frac{1}{12} + 11 \cdot \frac{1}{18} + 12 \cdot \frac{1}{36} \\ &= 7. \end{aligned}$$

$X_1(i,j) = i, X_2(i,j) = j$. We have

$E(X_1) = \mathrm{E}(X_2) = \frac{1+2+3+4+5+6}{6} = \frac{7}{2}$. So $E(X_1 + X_2) = E(X_1) + \mathrm{E}(X_2) = 7$

Random Permutation

Definition: For an event A, we define the indicator variable I_A:

- $I_A(\omega) = 1$ if $\omega \in A$, and
- $I_A(\omega) = 0$ if $\omega \notin A$

Problem: For given n produce a random permutation σ of 1, 2, …,n with uniform distribution.

Solution: As usual, the simple way is to produce all $n!$ permutation and number them from 1 to $n!$ and then select a number k u.a.r. and report the permutation whose number is k. A more efficient way is to select $\sigma(1)$, the leftmost item of the permutation, u.a.r. from set $\{1, 2, \ldots, n\}$, the select $\sigma(2)$ u.a.r from set $\{1, 2, \ldots, n\} - \{\sigma(1)\}$, and so on. How to program this?

Problem: Let σ be random permutation and let X be the number of i s.t. $\sigma(i) = i$. Show that $E(X) = 1$

Solution: Let A_i be the event $\sigma(i) = i$ and let X_i be its I.R.V. So we have $\mathrm{X} = X_1 + \cdots + X_n$ and $E(X) = E(X_1) + \cdots + E(X_n) = 1/\mathrm{n} + \cdots + 1/\mathrm{n} = 1$ as $E(X_i) = \mathrm{P}[A_i] = 1/n$

Randomized Approximate Median

Randomized Approximate Median

Input. A set S of n numbers. Assume for simplicity that all numbers are distinct.

Rank. The rank of a number x in S is 1 plus the number of elements in S that are smaller than x.

Median. A median of S is a number of rank $\lfloor (n+1)/2 \rfloor$.

Approximate Median. A δ-approximate median is an element of rank k with $\left(\frac{1}{2}-\delta\right)(n+1) \le k \le \left(\frac{1}{2}+\delta\right)(n+1)$ for some given constant $0 \le \delta \le \frac{1}{2}$.

Problem. Report a δ-approximate median

Algorithm 1

```
ApproxMedian1(S, δ)
    r = Random(1,n)
    x* = S[r]
    k = 1
    for  i = 1 to n do
        if S[i] < x* then
            k = k+1
    if (1/2 − δ)(n + 1) ≤ k ≤ (1/2 + δ)(n + 1)  then
        return x*
     else
        return "error"
```

Running time. O(n)

Success probability. $\frac{\left(\frac{1}{2}+\delta\right)(n+1)-\left(\frac{1}{2}-\delta\right)(n+1)}{n} \approx 2\delta$

Ex. For $\delta = \frac{1}{4}$, the success probability is $\frac{1}{2}$ and for $\delta = \frac{1}{10}$ where we are looking for an element that is closer to the median, the success probability is getting worse.

Algorithm 2

```
ApproxMedian2(S, δ,c)
    j = 1
    repeat
        result = ApproxMedian1(S, δ)
        j = j+1
    until (result ≠ error) or (j = c+1)
    return result
```

Running time. O(cn)
Success probability. $1-(1-2\delta)^c$

Ex. For $\delta=\frac{1}{4}$ and c=10, we get a $\frac{1}{4}$-approximate median with success rate 99.9%. And For $\delta=\frac{1}{10}$ and c=10, we get a $\frac{1}{10}$-approximate median with success rate 89.2%.

Algorithm 3

```
ApproxMedian3(S, δ)
   repeat
       result = ApproxMedian1(S, δ)
   until result ≠ error
   return result
```

Success probability. 1

Running time.

E(running time of ApproxMedian3)=E((#calls to ApproxMedian1).O(n)) =O(n). E(#calls to ApproxMedian1)=O(n). $(1/2\delta)$=O(n/δ)

Remark. when we will talk about "expected running time" we actually mean "worst-case expected running time" (for different inputs the expected running time may be different —this is not the case in the ApproxMedian3).

Running Time

Deterministic Algorithms.

- $T_{worst\text{-}case}(n) = \max_{|X|=n} T(X)$
- $T_{best\text{-}case}(n) = \min_{|X|=n} T(X)$
- $T_{average\text{-}case}(n) = E_{|X|=n}(T(X)) = \sum T(x).\Pr(X = x)$

Randomized Algorithms.

- $T_{worst\text{-}case\ expected}(n) = \max_{|X|=n} E(T(X))$

Monte Carlo vs. Las Vegas Algorithms

Monte Carlo algorithm. Guaranteed to run in poly-time, likely to find correct answer.
Ex: ApproxMedian1

Las Vegas algorithm. Guaranteed to find correct answer, likely to run in poly-time.
Ex: ApproxMedian3

	Running time	Correctness
Las Vegas Algorithm	probabilistic	certain
Monte Carlo Algorithm	certain	probabilistic

Remark. ApproxMedian2 is mixture: the random choices both impact the running time and the correctness. Sometimes this is also called a Monte Carlo algorithm.

Random Permutation

Random Permutation

Problem. Produce a Random Permutiton of items stored in array A.

```
RandomPermutation(A)
for i = 1 to n do
      r = rand(i, n)
      swap (A[i], A[r])
```

Question: Does the algorithm work if if we replace random(i, n) by random(1, n)?

Hiring Problem

Definition

- Suppose you are doing a project, for which you need the best assistant you can get.
- You contact an employment agency that promises to send you some candidates, one per day, for interviewing.
- Since you really want to have the best assistant available, you decide on the following strategy:

Strategy: whenever you interview a candidate and the candidate turns out to be better than your current assistant, you fire your current assistant and hire the new candidate.

Deterministic Algorithm

```
HireAssistant(A)
Current-assist = nil
for i = 1 to n do
      interview candidate i
      if he is better than current-assist then
         current-assist = candidate i
```

Cost: You have to pay f.n in the worse case where f is the cost of changing your assistant and n is the number of candidates.

Randomized Algorithm

```
HireAssistant(A)
Compute a R.P. of candidats
Current-assist = nil
for i = 1 to n do
      interview candidate i
      if he is better than current-assist then
         current-assist = candidate i
```

Cost: You have to pay f.n in the worse case where f is the cost of changing your assistant and n is the number of candidates.

Analysis

Let X_i be an I.R.V. which is 1 iff the candidate i is better than candidate 1, 2, ..., i-1

$$E(cost) = E(\sum_i cost\ to\ be\ paid\ for\ candidate\ i) =$$

$$\sum_i E(cost\ to\ be\ paid\ for\ candidate\ i) =$$

$$\sum_i E(f.X_i) = f\sum_i E(X_i) = f\sum_i Pr(X_i = 1) = f\sum_i \frac{1}{i} \approx f.\ln n$$

Randomized Selection

Randomized Selection

Selection. Given a set S of n distinct elements and an integer i, we want to find the element of rank i in S

```
Selection(S,i)
   if |S| = 1 return the only element of S

   choose a splitter aj ∈ S uniformly at random
   foreach (a ∈ S) {
      if      (a < aj) put a in S-
      else if (a > aj) put a in S+
   }
   k = |S-|
   if k = i-1 then return aj
   else if k > i-1 then
        Selection(S-,i)
   else
        Selection(S+,i-k-1)
```

Randomized Selection: Analysis

Running time.

- [Best case.] Select the median element as the splitter: Selection makes $\Theta(n)$ comparisons ($T_{best}(n)=O(n)+T_{best}(n/2)$).
- [Worst case.] Select the smallest element as the splitter: Selection makes $\Theta(n^2)$ comparisons ($T_{worst}(n)=O(n)+T_{worst}(n-1)$).

Randomize. Protect against worst case by choosing splitter at random.

Intuition. If we always select an element that is bigger than 25% of the elements and smaller than 25% of the elements, then Selection makes $\Theta(n)$ comparisons.

Randomized Selection: Analysis

Running time.

$T_{exp}(n) = O(n) + \sum_{j=1}^{n} \Pr($element of rank j is splitter$). T_{exp}(\max(j-1, n-j))$

$= O(n) + 1/\mathrm{n} \sum_{j=1}^{n} \mathrm{T_{exp}}(\max(\mathrm{j}, \mathrm{n}-\mathrm{j}))$

It can be shown than $T_{exp}(n) = \mathrm{O}(\mathrm{n})$

Easier method. With probability 1/2 we recurse on at most 3n/4 elements. So

$$T_{exp}(n) \leq O(n) + \frac{1}{2} T_{exp}(3n/4) + \frac{1}{2} T_{exp}(n-1)$$

This recurrence is pretty easy to solve by induction.

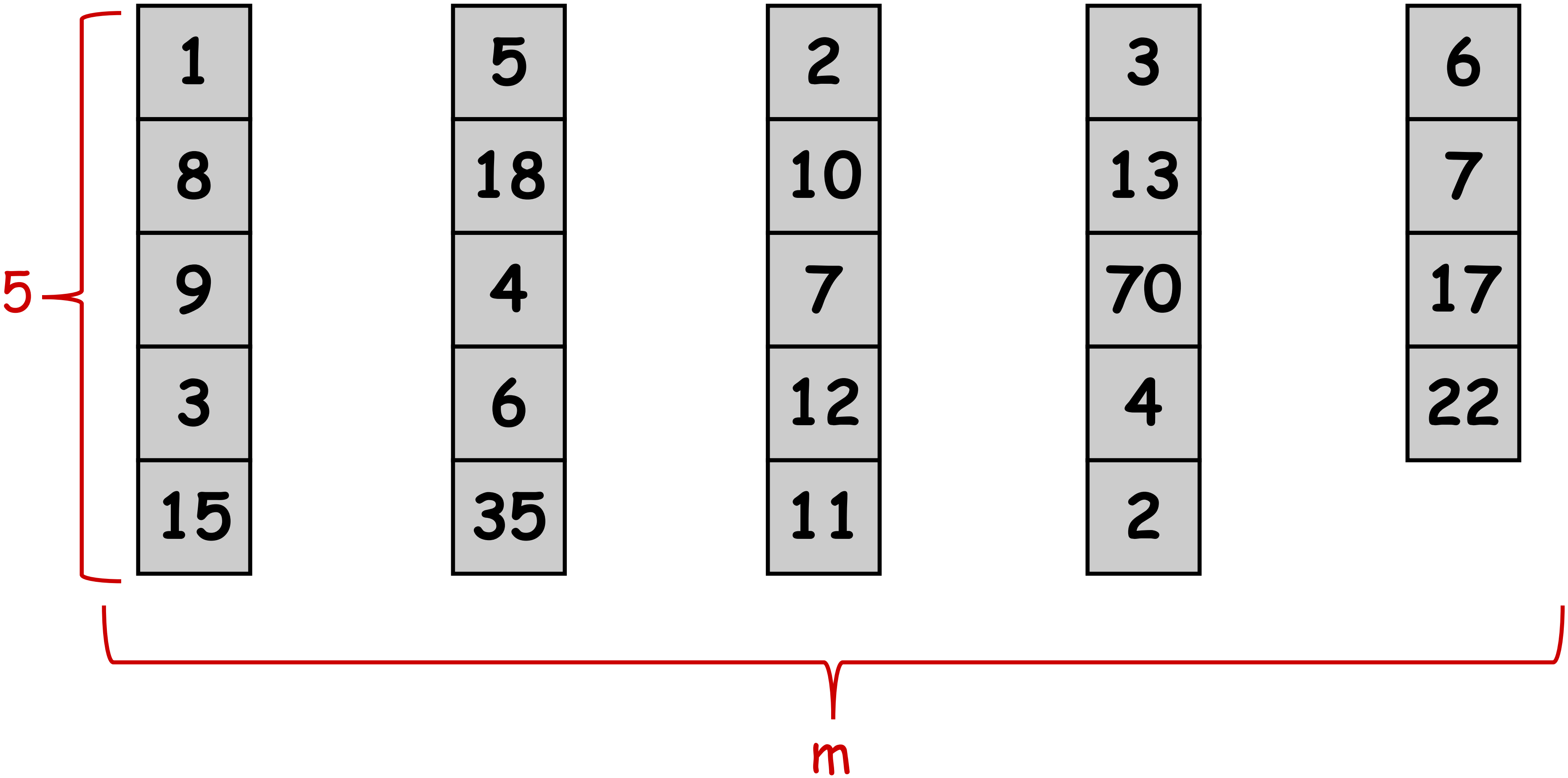
Deterministic Selection

Selection. Given a set S of n distinct elements and an integer i, we want to find the element of rank i in S

```
Selection(S,i)
   if |S| = 1 return the only element of S
   foreach i from 0 to |S|/5 do
       B[i] = the median of S[5i+1],...,S[5i+5]
   x = Selection(B, n/10)
   foreach (a ∈ S)
       if a < x then then  put a to S⁻
       else if put a to S⁺
   k = |S⁻|
   if k = i-1 then return x
   else if k > i-1 then
        Selection(S⁻,i)
   else
        Selection(S⁺,i-k-1)
```

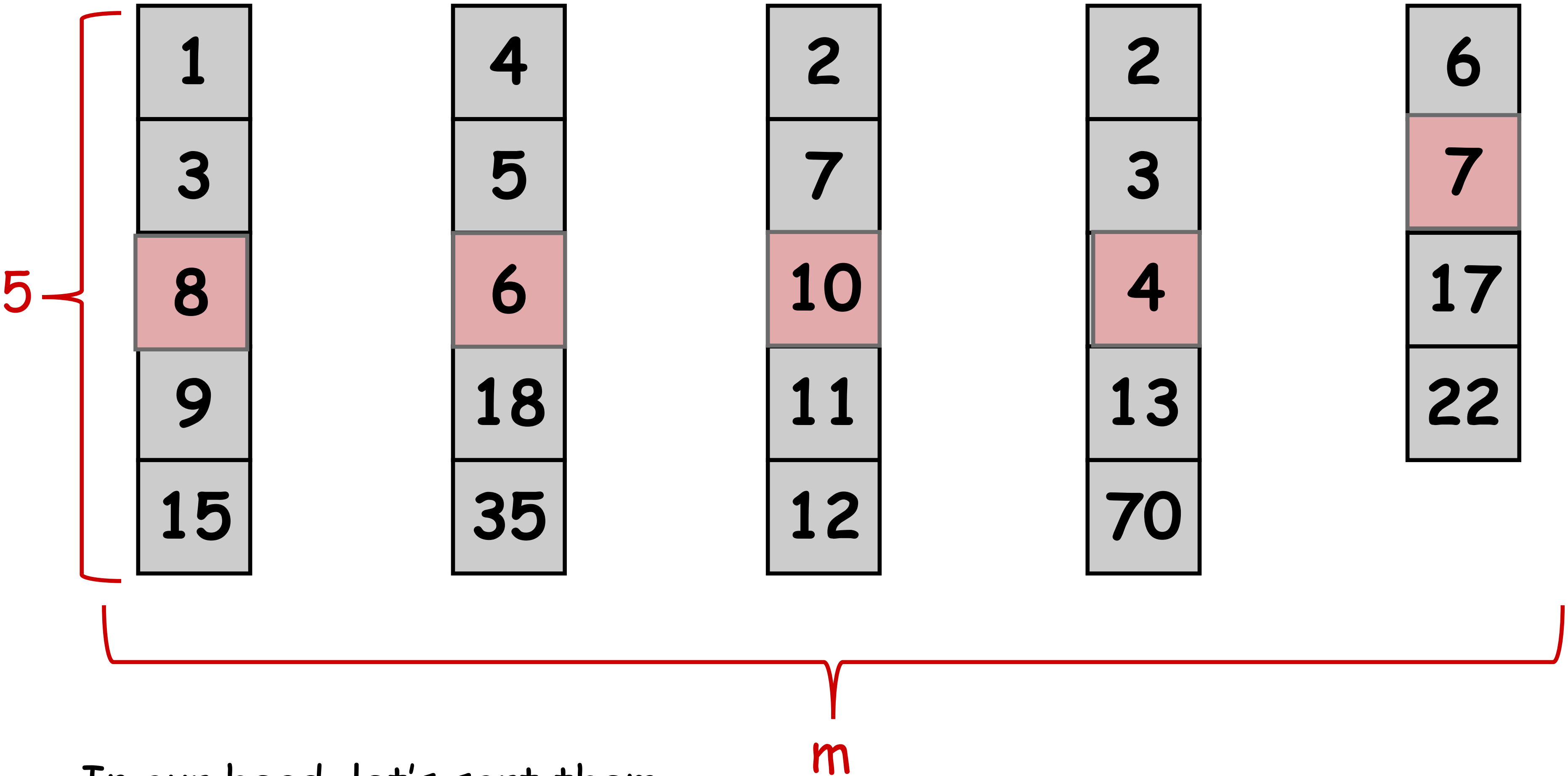
$3/10n < |S^-|, |S^+| < 7/10n \Rightarrow T(n)=O(n)+T(2n/10)+T(7/10n) \Rightarrow T(n)=O(n)$

Proof by picture



Say these are our $m = \lceil n/5 \rceil$ sub-arrays of size at most 5.

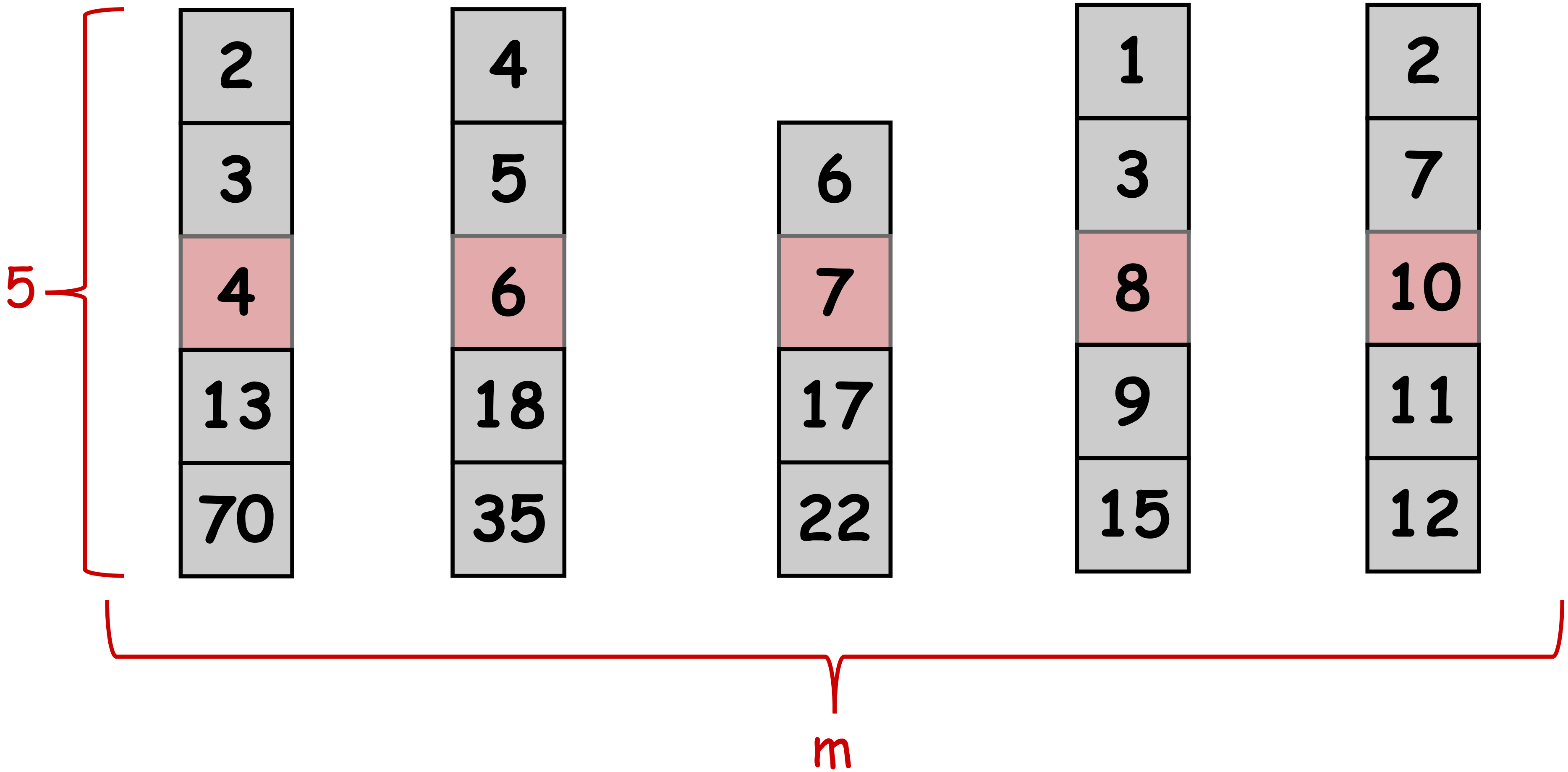
Proof by picture



In our head, let's sort them.

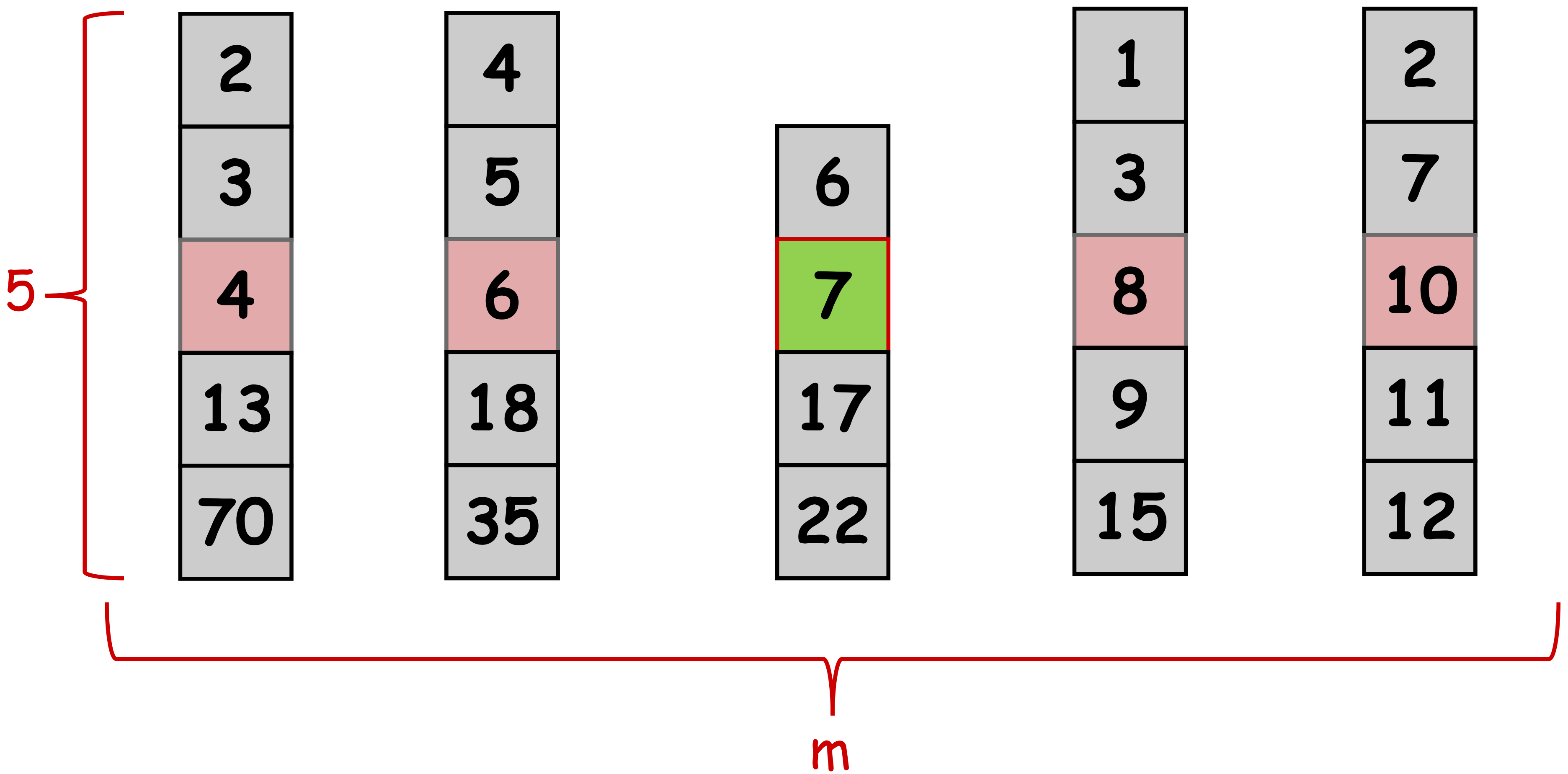
Then find medians.

Proof by picture



Then let's sort them by the median

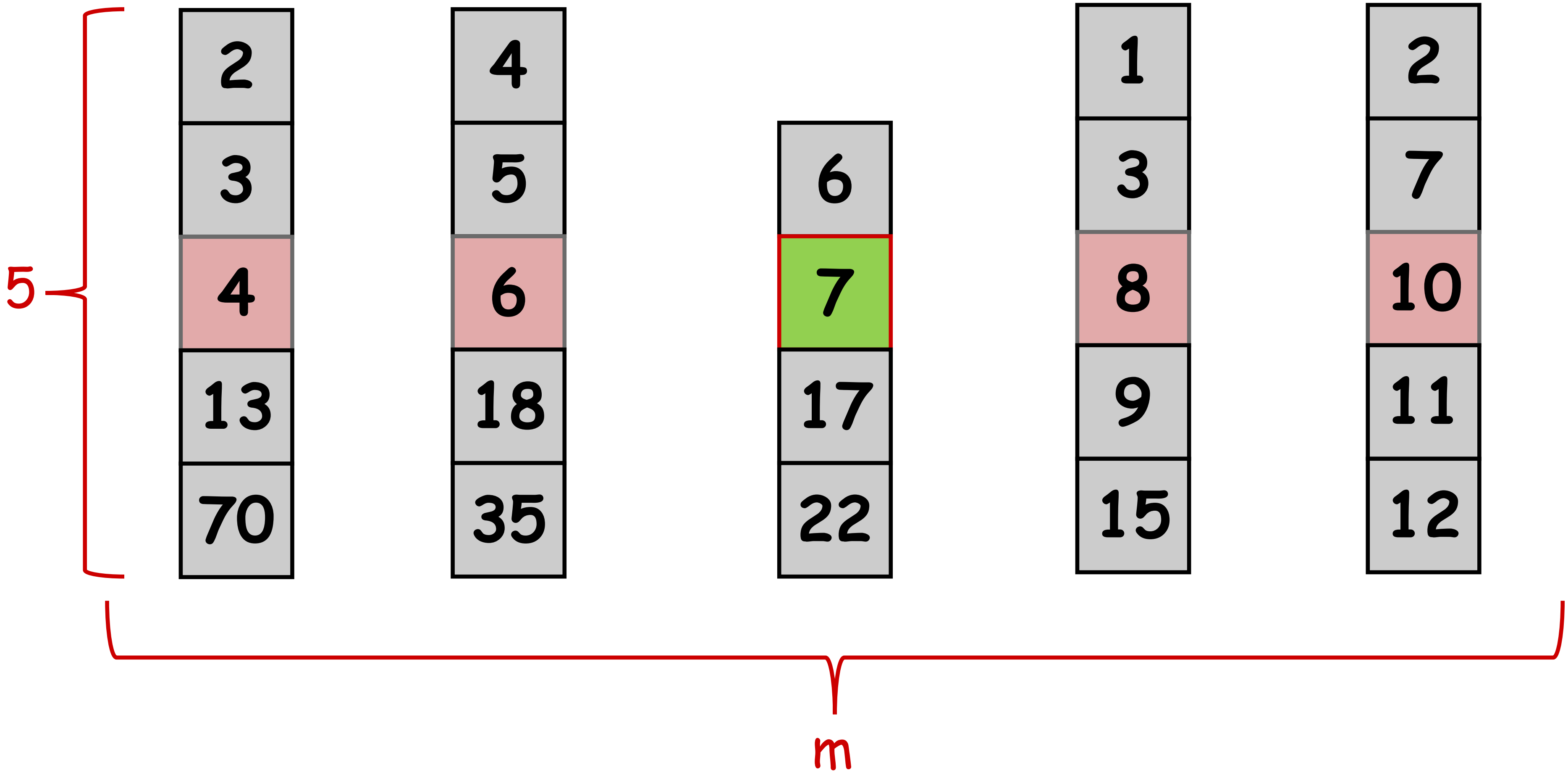
Proof by picture



The median of the medians is 7. That's our pivot!

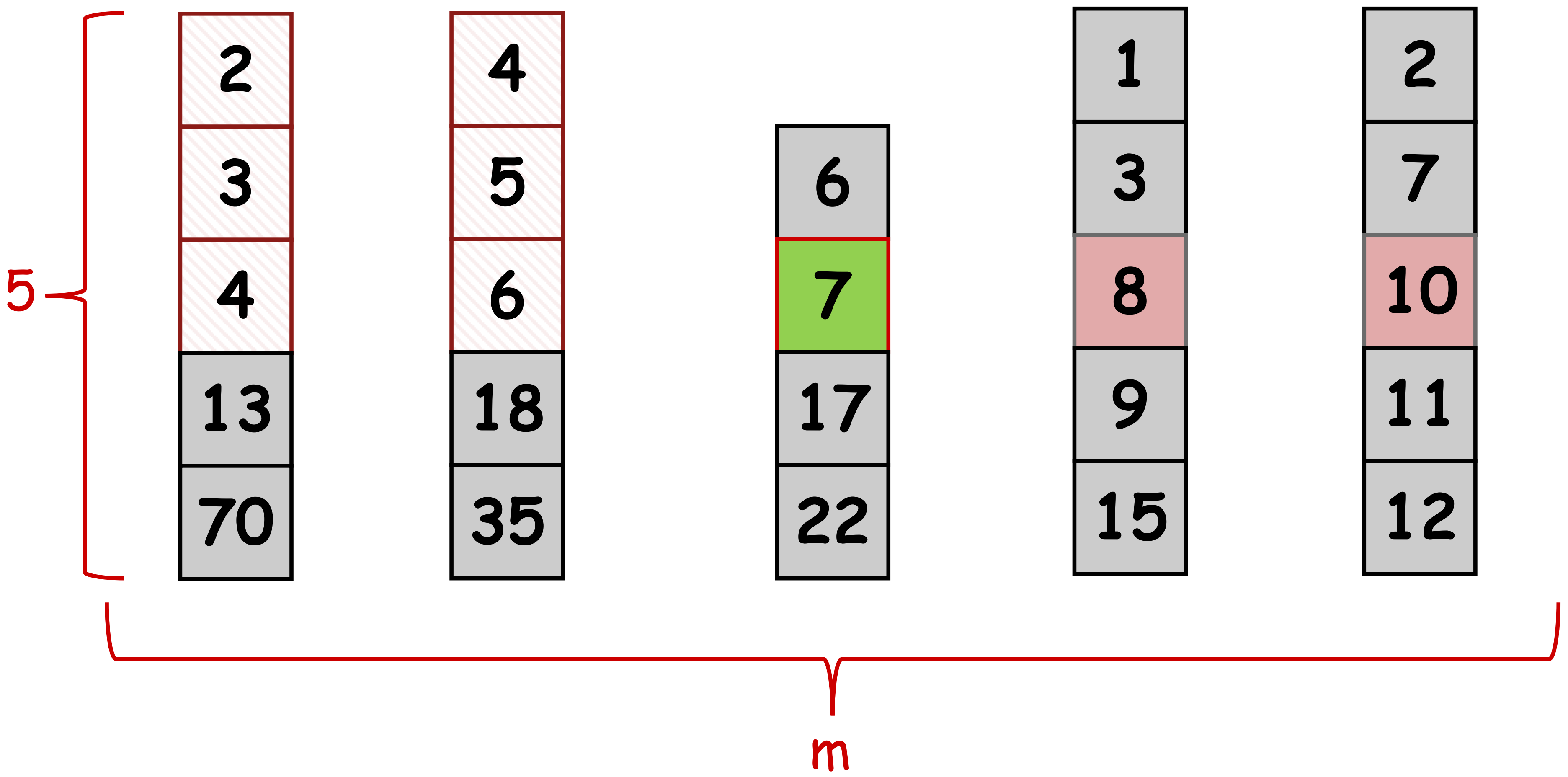
We will show that lots of elements are smaller than the pivot, hence not too many are larger than the pivot.

Proof by picture



How many elements are SMALLER than the pivot?

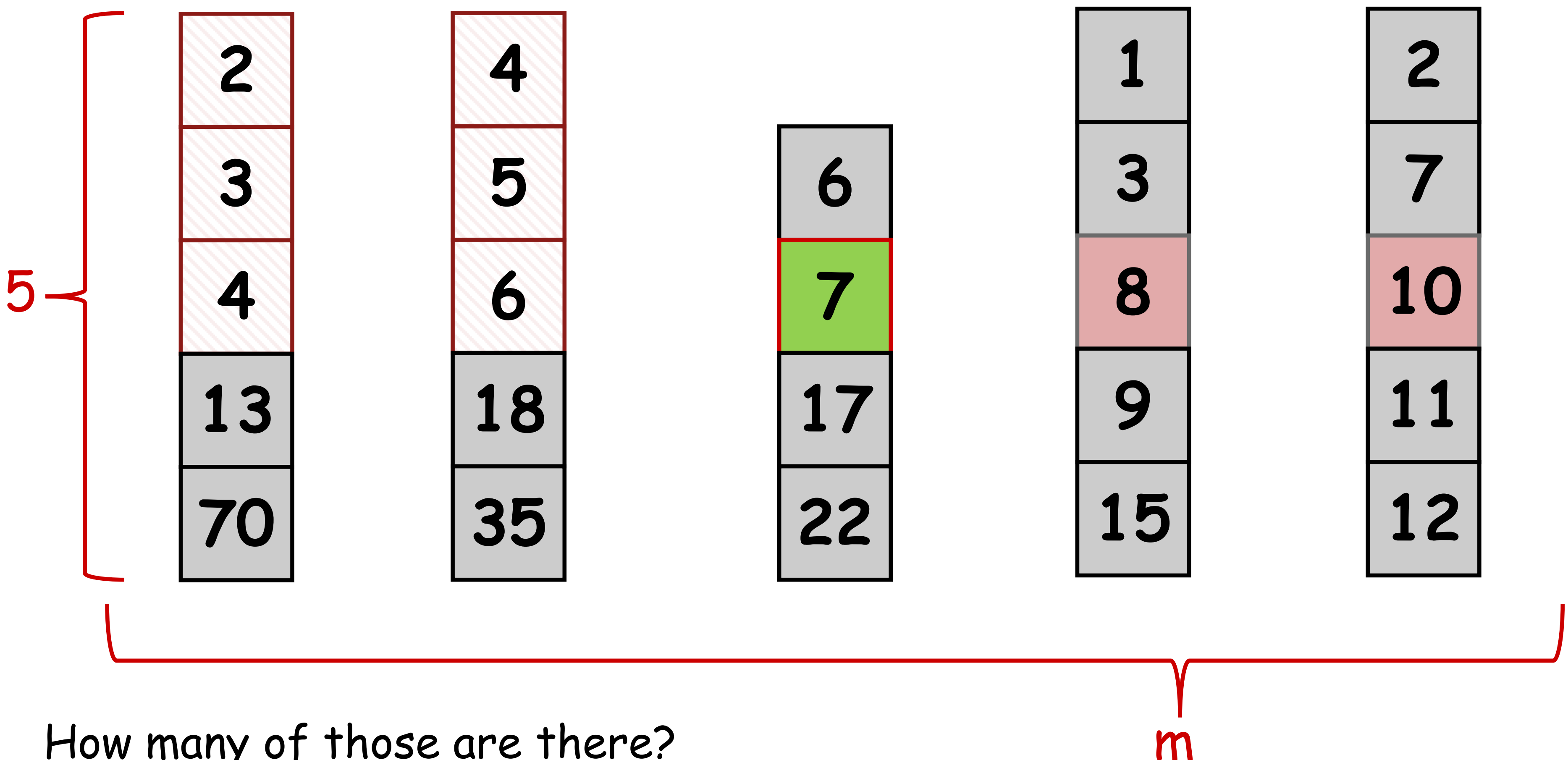
Proof by picture



At least these ones: everything above and to the left.

Proof by picture

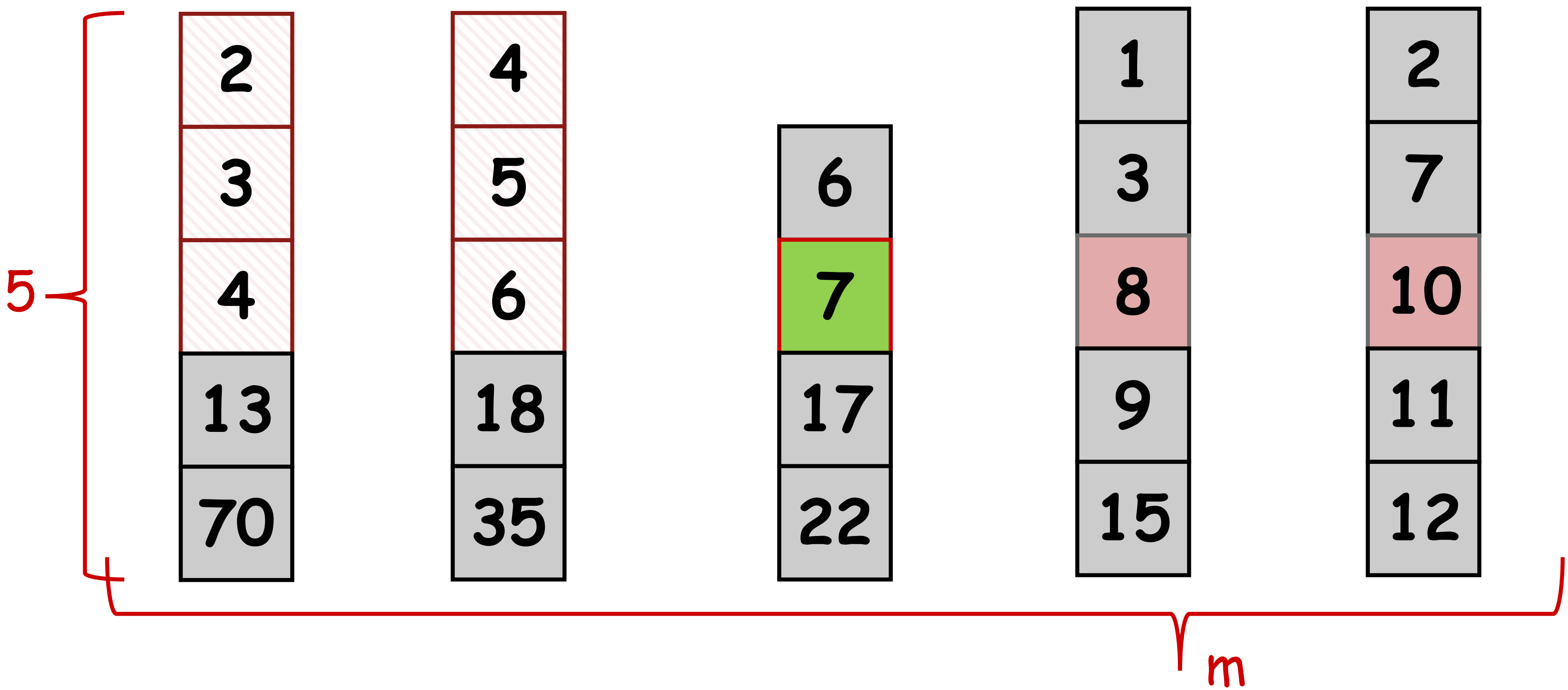
$3 \cdot \left(\left\lceil \frac{m}{2} \right\rceil - 1\right)$ of these, but then one of them could have been the "leftovers" group.



How many of those are there?

at least $3 \cdot \left(\left\lceil \frac{m}{2} \right\rceil - 2\right)$

Proof by picture



So how many are LARGER than the pivot? At most…

$$n \ - 1 \ - 3\left(\left\lceil\frac{m}{2}\right\rceil - 2\right) \leq \frac{7n}{10} + 5$$

Remember

$$m = \left\lceil\frac{n}{5}\right\rceil$$

One Part of the Lemma

Lemma: If $|S^-|$, and $|S^+|$, are as in the algorithm SELECT given above, then

$$|S^-| \le \frac{7n}{10} + 5$$

and

$$|S^+| \le \frac{7n}{10} + 5$$

The other part is exactly the same.

Randomized Quicksort

Sorting. Given a set of n distinct elements S, rearrange them in ascending order.

```
RandomizedQuicksort(S) {
   if |S| = 0 return

   choose a splitter a_i ∈ S uniformly at random
   foreach (a ∈ S) {
      if      (a < a_i) put a in S⁻
      else if (a > a_i) put a in S⁺
   }
   RandomizedQuicksort(S⁻)
   output a_i
   RandomizedQuicksort(S⁺)
}
```

Quicksort: Analysis

Running time.

- [Best case.] Select the median element as the splitter: quicksort makes $\Theta(n \log n)$ comparisons.
- [Worst case.] Select the smallest element as the splitter: quicksort makes $\Theta(n^2)$ comparisons.

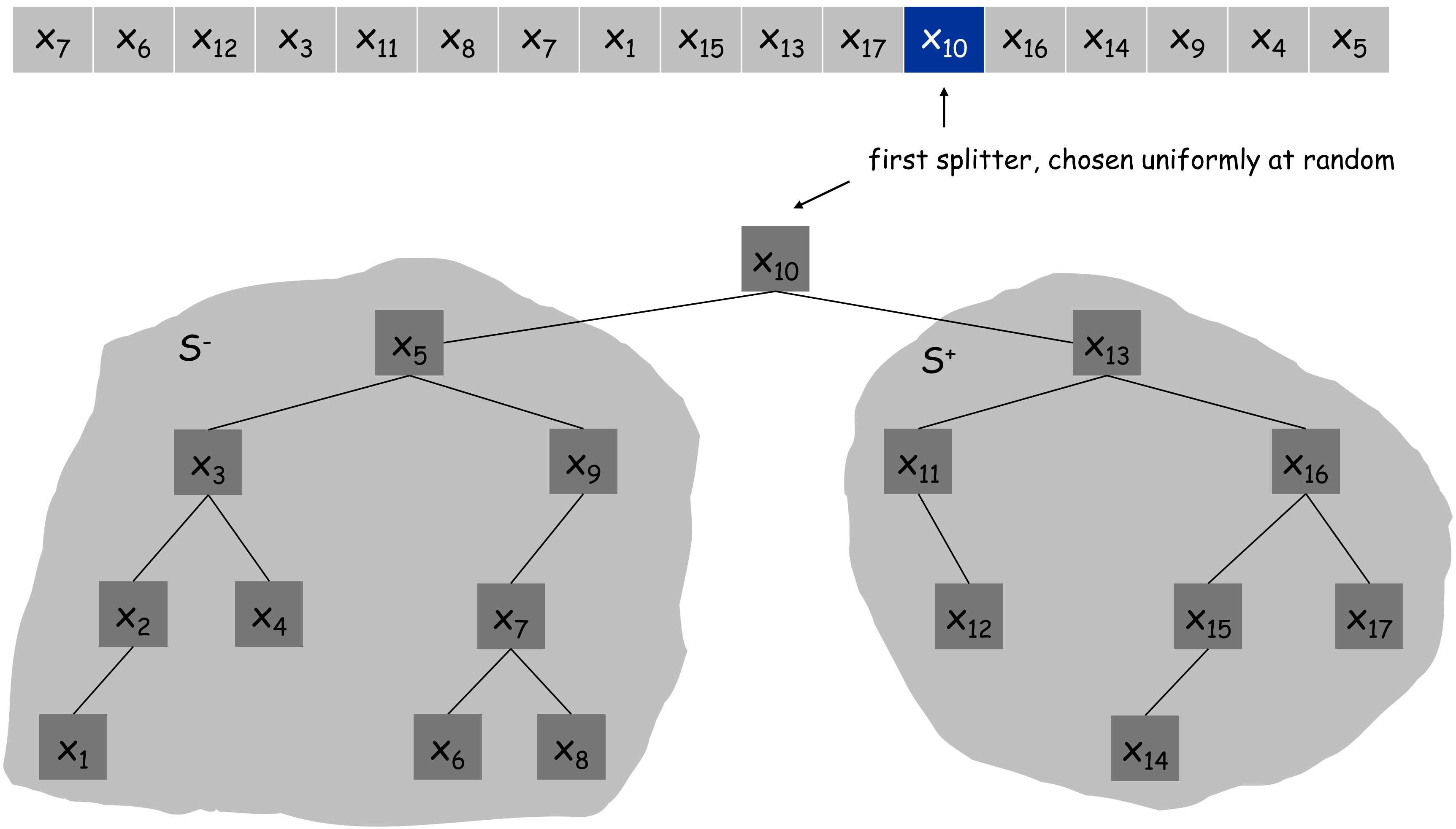
Randomize. Protect against worst case by choosing splitter at random.

Intuition. If we always select an element that is bigger than 25% of the elements and smaller than 25% of the elements, then quicksort makes $\Theta(n \log n)$ comparisons.

Notation. Label elements so that $x_1 < x_2 < \ldots < x_n$.

Quicksort: BST Representation of Splitters

BST representation. Draw recursive BST of splitters.

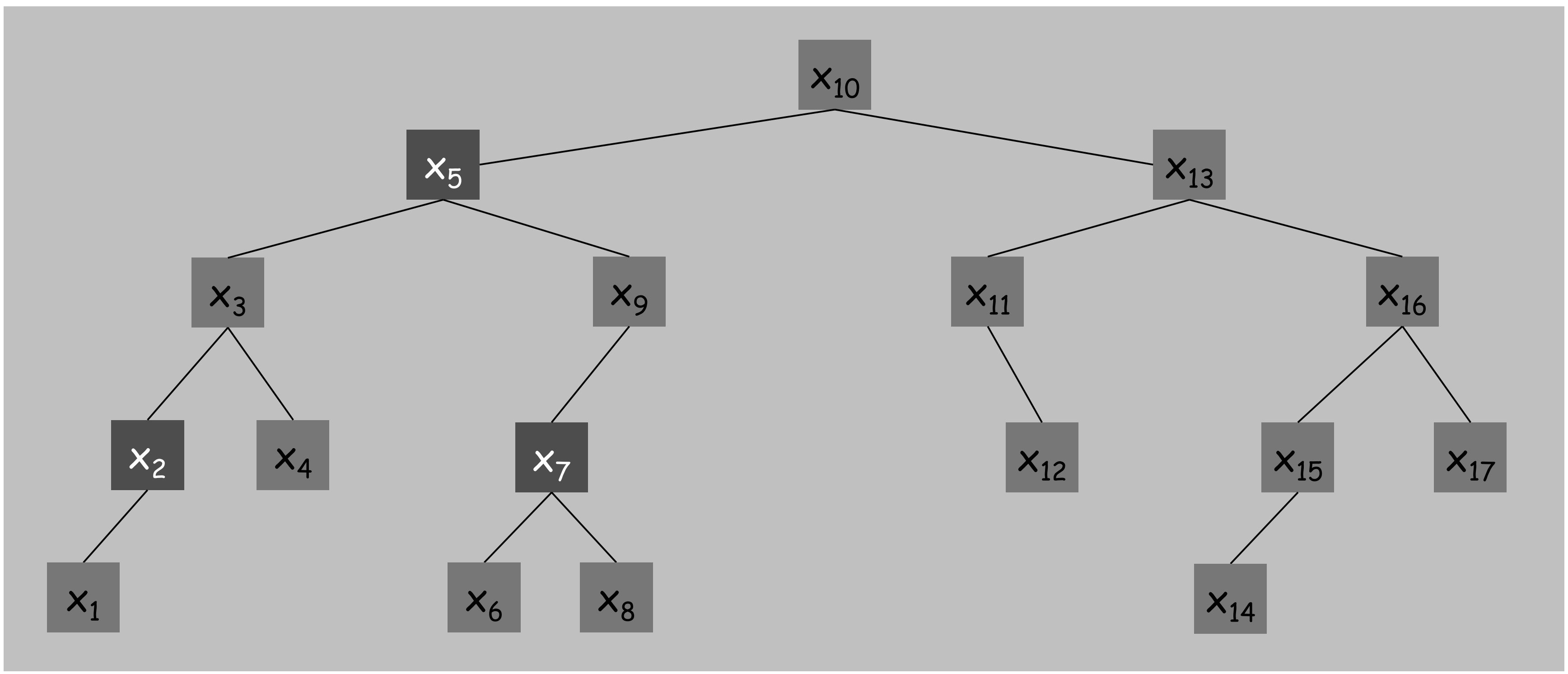


Quicksort: BST Representation of Splitters

Observation. Element only compared with its ancestors and descendants.

- x_2 and x_7 are compared if their lca = x_2 or x_7.
- x_2 and x_7 are not compared if their lca = x_3 or x_4 or x_5 or x_6.

Claim. $\Pr[x_i \text{ and } x_j \text{ are compared}] = 2 / |j - i + 1|$.



Quicksort: BST Representation of Splitters

Claim Proof.

- Consider $S_{ij} = \{x_i, \ldots, x_j\}$
- If splitter does not belong to S_{ij}, all elements of S_{ij} stay together and no comparison is made between x_i and x_j.
- This continues until at some point one of the elements in S_{ij} is chosen as the splitter.
- If x_i or x_j is selected to be the splitter, x_i and x_j are compared. Otherwise, x_i and x_j are never compared.
- Since each element of S_{ij} has equal probability of being chosen as splitter, we therefore find

$$\Pr[x_i \text{ and } x_j \text{ are compared}] = 2 / |j - i + 1|.$$

Quicksort: Expected Number of Comparisons

Theorem. Expected # of comparisons is O(n log n).

Pf.

- $X_{ij} = 1$ if x_i and x_j are compared. Otherwise, $X_{ij} = 0$
- $X = \sum X_{ij}$ is the #comparisons and $E(X) = \sum E(X_{ij})$

$$\sum_{1 \le i < j \le n} E(X_{ij}) = \sum_{1 \le i < j \le n} \frac{2}{j-i+1} = 2\sum_{i=1}^{n} \sum_{j=2}^{i} \frac{1}{j} \le 2n \sum_{j=1}^{n} \frac{1}{j} \approx 2n \int_{x=1}^{n} \frac{1}{x} dx = 2n \ln n$$

Ex. If n = 1 million, the probability that randomized quicksort takes less than 4n ln n comparisons is at least 99.94%.

Chebyshev's inequality. $\Pr[|X - \mu| \ge k\sigma] \le 1 / k^2$.

Quicksort: Another Approach

Use approximate median. Instead of picking the pivot uniformly at random from S, we could also insist in picking a good pivot. An easy way to do this is to use algorithm ApproxMedian3 to find a (1/4)-approximate median. Now the expected running time is bounded by

E[(running time of ApproxMedian3 with δ = 1/4) + (time for recursive calls)] = O(n)+E[time for recursive calls]

$$T_{exp}(n) \leq O(n) + T_{exp}(3n/4) + T_{exp}(n/4)$$

Then,

$$T_{exp}(n) = O(n \ \log n)$$

.

References

References

- Lecture notes of advanced algorithms by Mark de berg
- The slides were prepared by Kevin Wayne. The slides are distributed by Pearson Addison-Wesley.