Algorithms and Data Structures

Average Case Analysis

Recall: The Quicksort algorithm

Quicksort first divides the array into two parts, such that the first part is “smaller” than the second part.

This is done via the Partition procedure.

Then it calls itself recursively.

The two parts are joined, but this is trivial.

The Partition procedure

```
Procedure Partition(A[i,...,j])

    Choose a pivot element x of A

    k = i

    For h = i to j do

        If A[h] < x

            Swap A[k] with A[h]
            k = k + 1

       Swap A[k] with A[h]

Return k
```

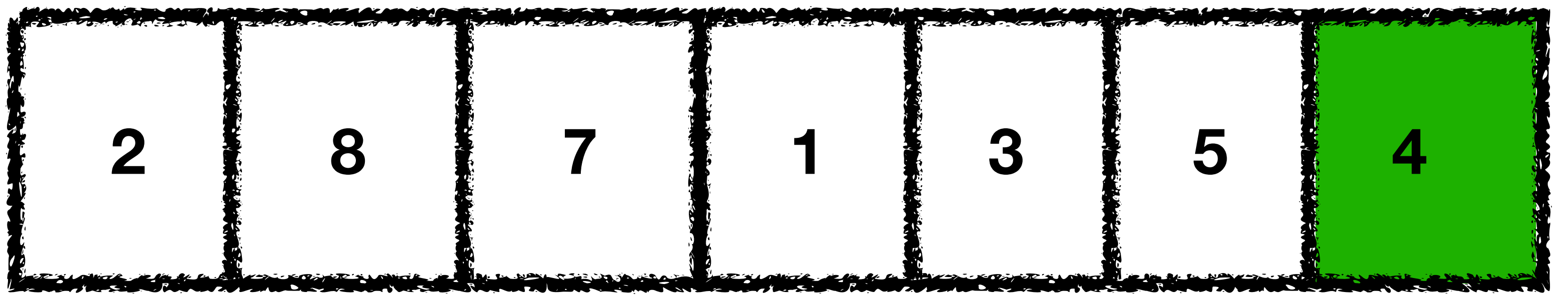
Correctness of Partition:
(CLRS p. 171-173)

Running time **O(n)**

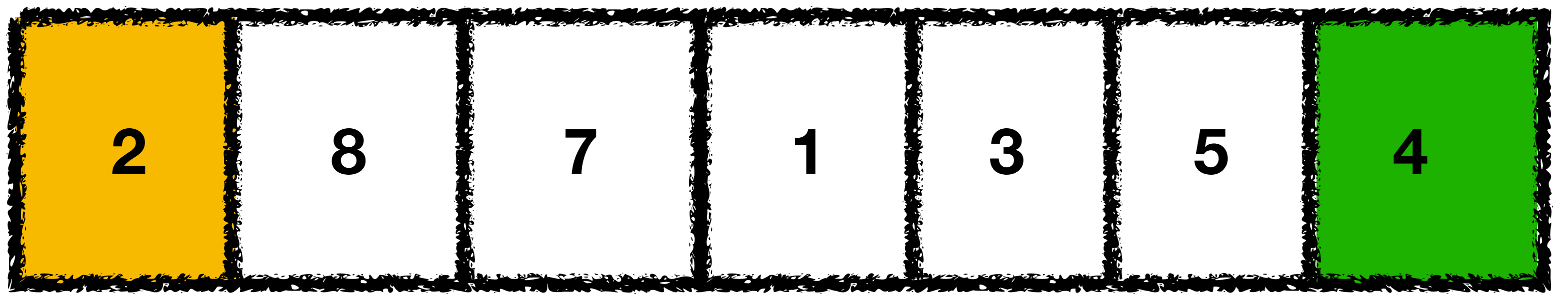
The Quicksort algorithm

2	8	7	1	3	5	4

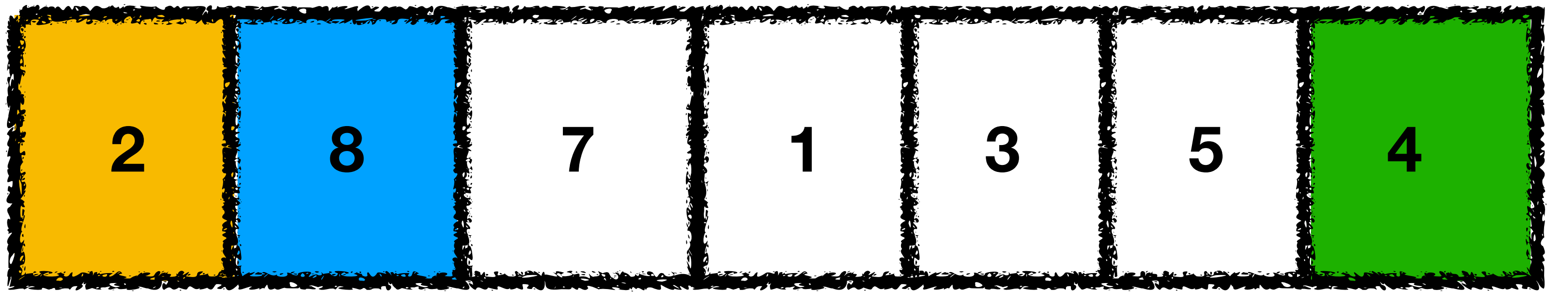
The Quicksort algorithm



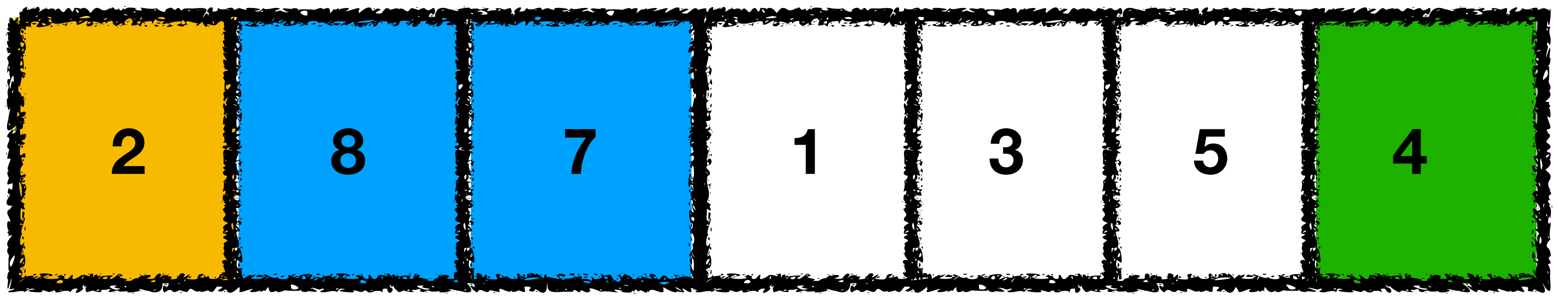
The Quicksort algorithm



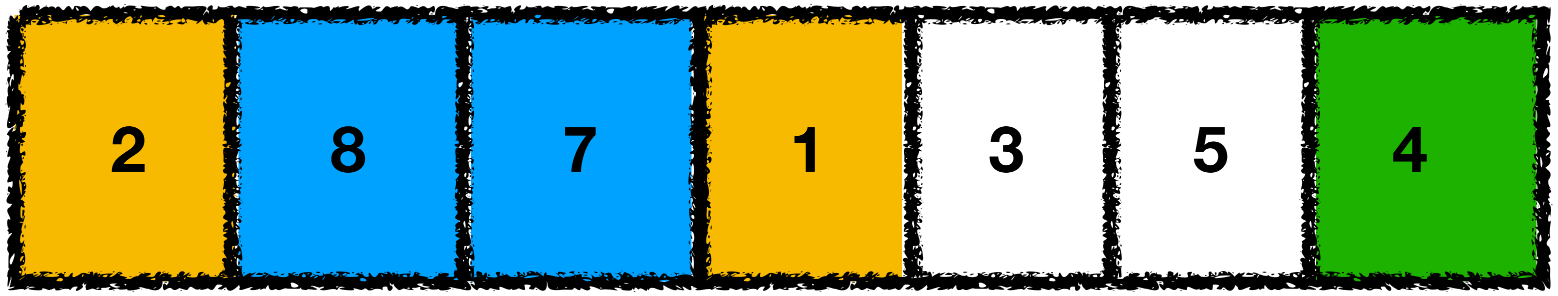
The Quicksort algorithm



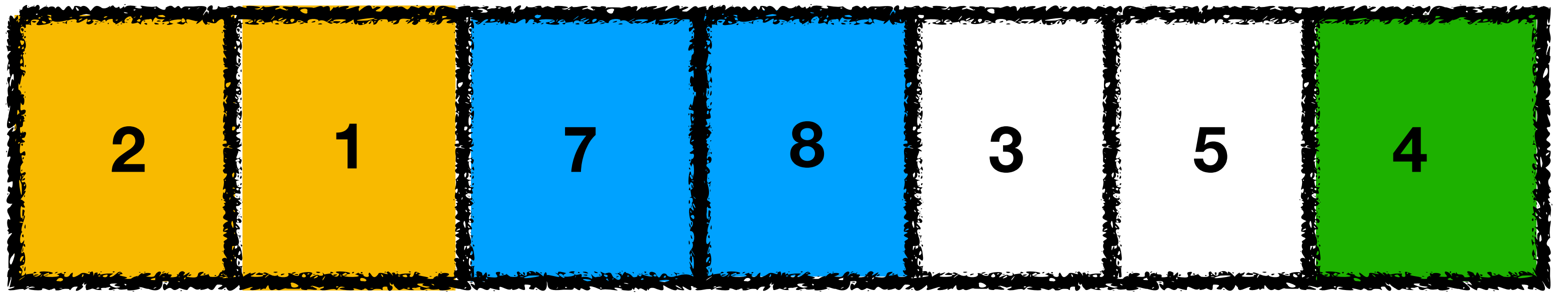
The Quicksort algorithm



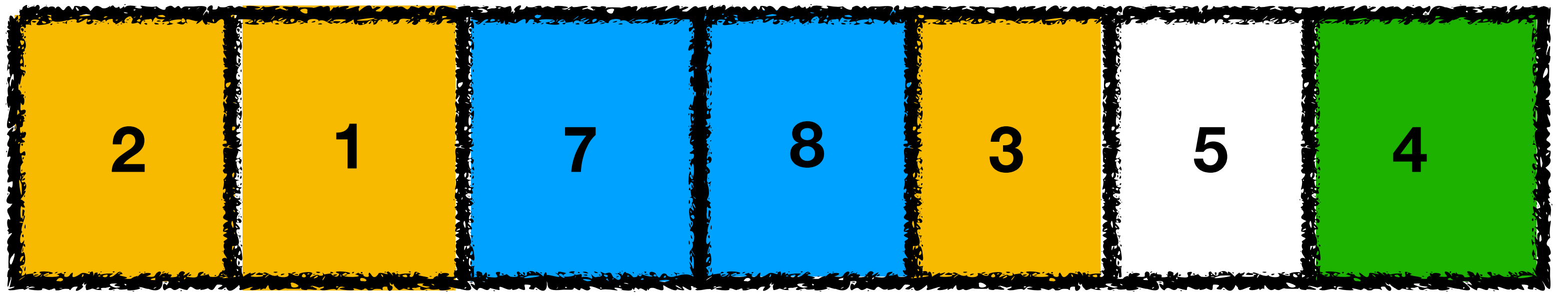
The Quicksort algorithm



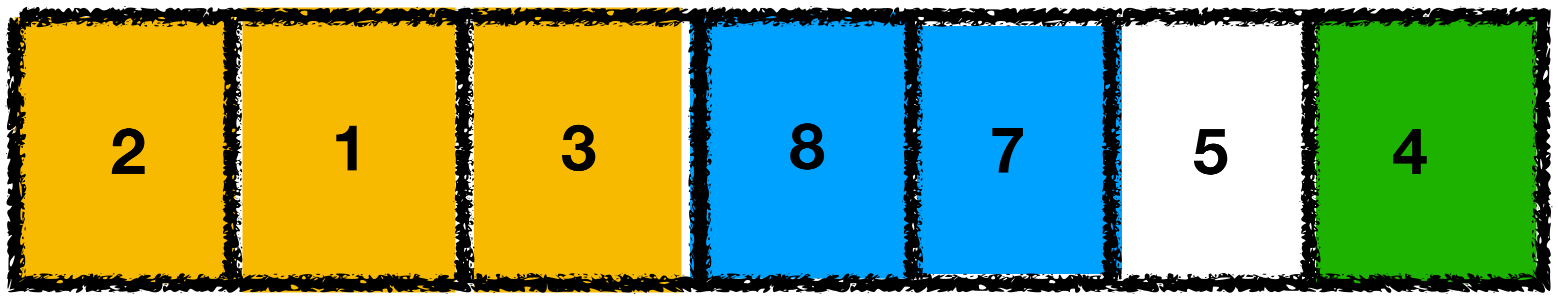
The Quicksort algorithm



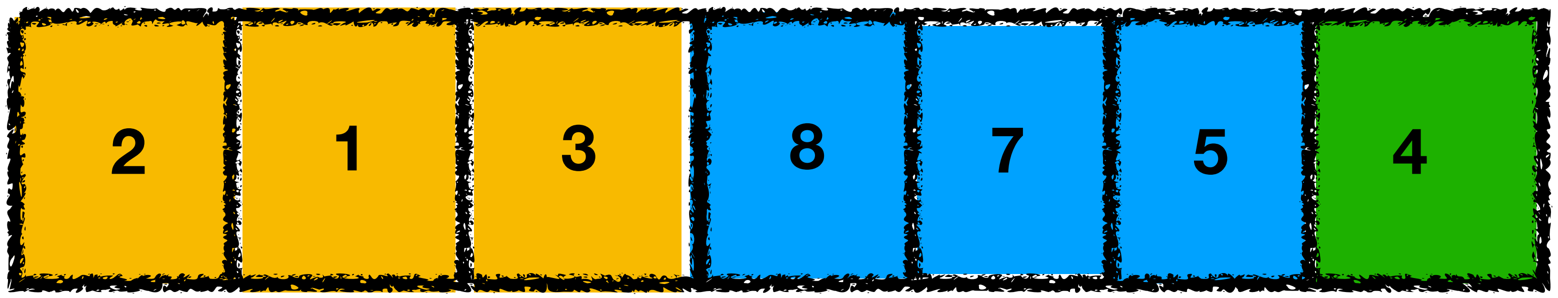
The Quicksort algorithm



The Quicksort algorithm



The Quicksort algorithm



The Quicksort algorithm

The Quicksort algorithm

Sort this using
Quicksort

The Quicksort algorithm

Sort this using **Quicksort**

Sort this using **Quicksort**

The Quicksort algorithm

Sort this using **Quicksort**

Sort this using **Quicksort**

Algorithm **Quicksort**(**A**[$i,\ldots,j$])

y = **Partition**(**A**[$i,\ldots,j$])

Quicksort(**A**[$i,\ldots,y-1$])

Quicksort(**A**[$y+1,\ldots,j$])

Running time of Quicksort

Quicksort: $T(n) \leq T(n_1) + T(n_2) + cn$

Running time of Quicksort

Quicksort: $T(n) \leq T(n_1) + T(n_2) + cn$

When $n_1 = n_2$, we get $T(n) \leq 2T(n) + cn$ and the running time is $O(n \log n)$.

This is the best-case running time.

Running time of Quicksort

Quicksort: $T(n) \leq T(n_1) + T(n_2) + cn$

When $n_1 = n_2$, we get $T(n) \leq 2T(n) + cn$ and the running time is $O(n \log n)$.

This is the best-case running time.

When $n_1 = n - 1$ $n_2 = 0$, we get $T(n) \leq T(n-1) + cn$ and the running time is $O(n^2)$.

This is the worst-case running time.

Running time of Quicksort

Quicksort: $T(n) \leq T(n_1) + T(n_2) + cn$

What about the average-case running time?

Worst vs Best vs Average Case

Convention: When we say "the running time of Algorithm A", we mean the worst-case running time, over all possible inputs to the algorithm.

We can also measure the best-case running time, over all possible inputs to the problem.

In between: average-case running time.

Running time of the algorithm on inputs which are chosen at random from some distribution.

The appropriate distribution depends on the application (usually the uniform distribution - all inputs equally likely).

Running time of Quicksort

Quicksort: $T(n) \leq T(n_1) + T(n_2) + cn$

What about the average-case running time?

Assume that the input sequence of n numbers is drawn uniformly at random from a distribution over all $n!$ possible inputs.

Unbalanced Partitions

Quicksort: $T(n) \leq T(n_1) + T(n_2) + cn$

Assume that we use a pivot element that results in a 9-to-1 split, i.e., $n_1 = 9n/10$ and $n_2 = n/10$.

Unbalanced Partitions

Quicksort: $T(n) \leq T(n_1) + T(n_2) + cn$

Assume that we use a pivot element that results in a 9-to-1 split, i.e., $n_1 = 9n/10$ and $n_2 = n/10$.

Q: Can you work out what the recurrence relation evaluates to? Use the unrolling technique.

Unbalanced Partitions

Quicksort: $T(n) \leq T(n_1) + T(n_2) + cn$

Assume that we use a pivot element that results in a 99-to-1 split, i.e., $n_1 = 99n/100$ and $n_2 = n/100$.

Q: Can you work out what the recurrence relation evaluates to? Use the unrolling technique.

Unbalanced Partitions

Quicksort: $T(n) \leq T(n_1) + T(n_2) + cn$

Assume that we use a pivot element that results in a 99-to-1 split, i.e., $n_1 = 99n/100$ and $n_2 = n/100$.

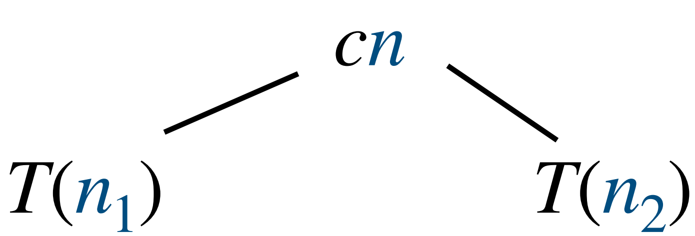
Q: Can you work out what the recurrence relation evaluates to? Use the unrolling technique.

Main message: Bad partitions are rather unlikely to happen. Most partitions are good partitions.

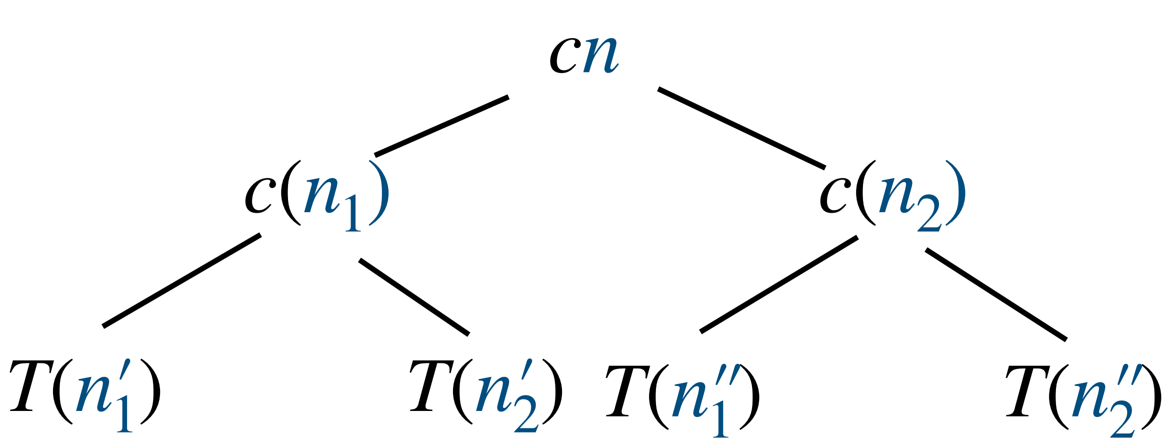
For the sake of intuition

Consider the recursion tree of **Quicksort**.

First iteration



Second iteration

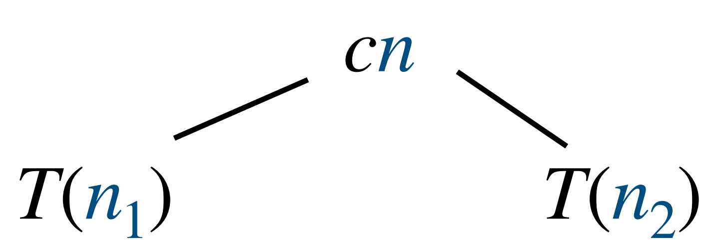


For the sake of intuition

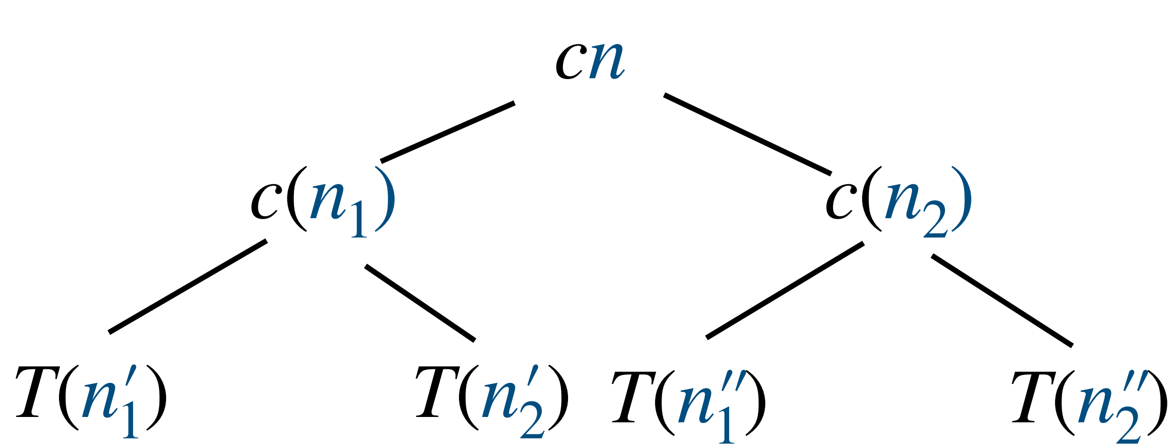
Consider the recursion tree of **Quicksort**.

Assume bad and good levels *alternate*.

First iteration



Second iteration

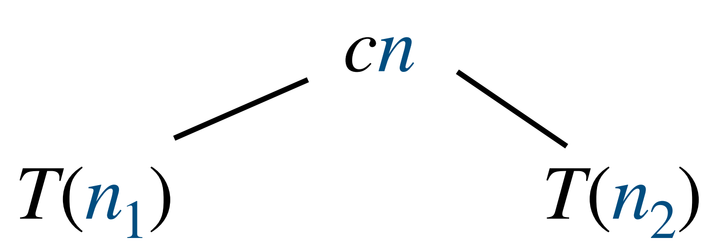


For the sake of intuition

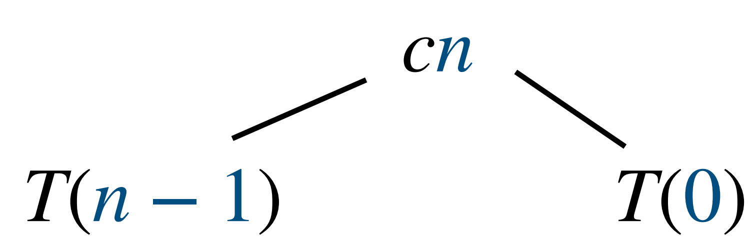
Consider the recursion tree of **Quicksort**.

Assume bad and good levels *alternate*.

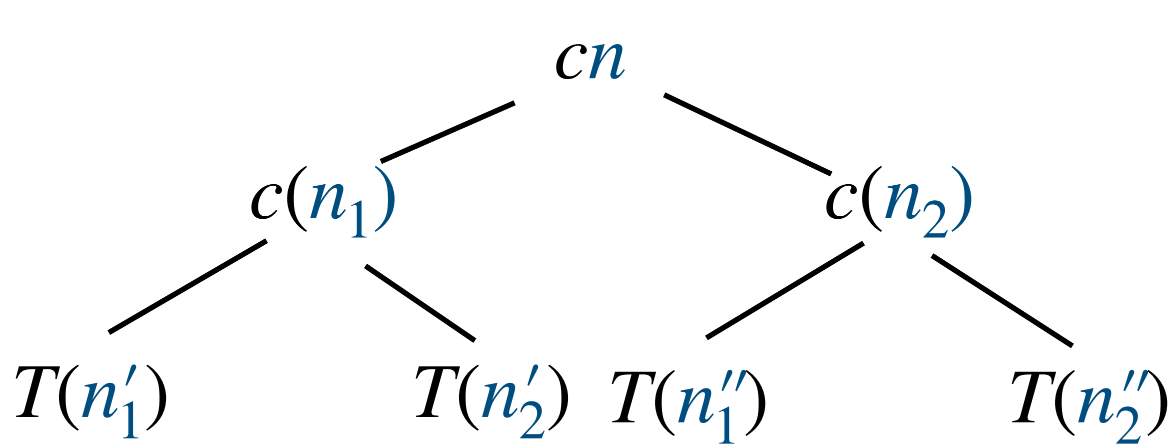
First iteration



First iteration



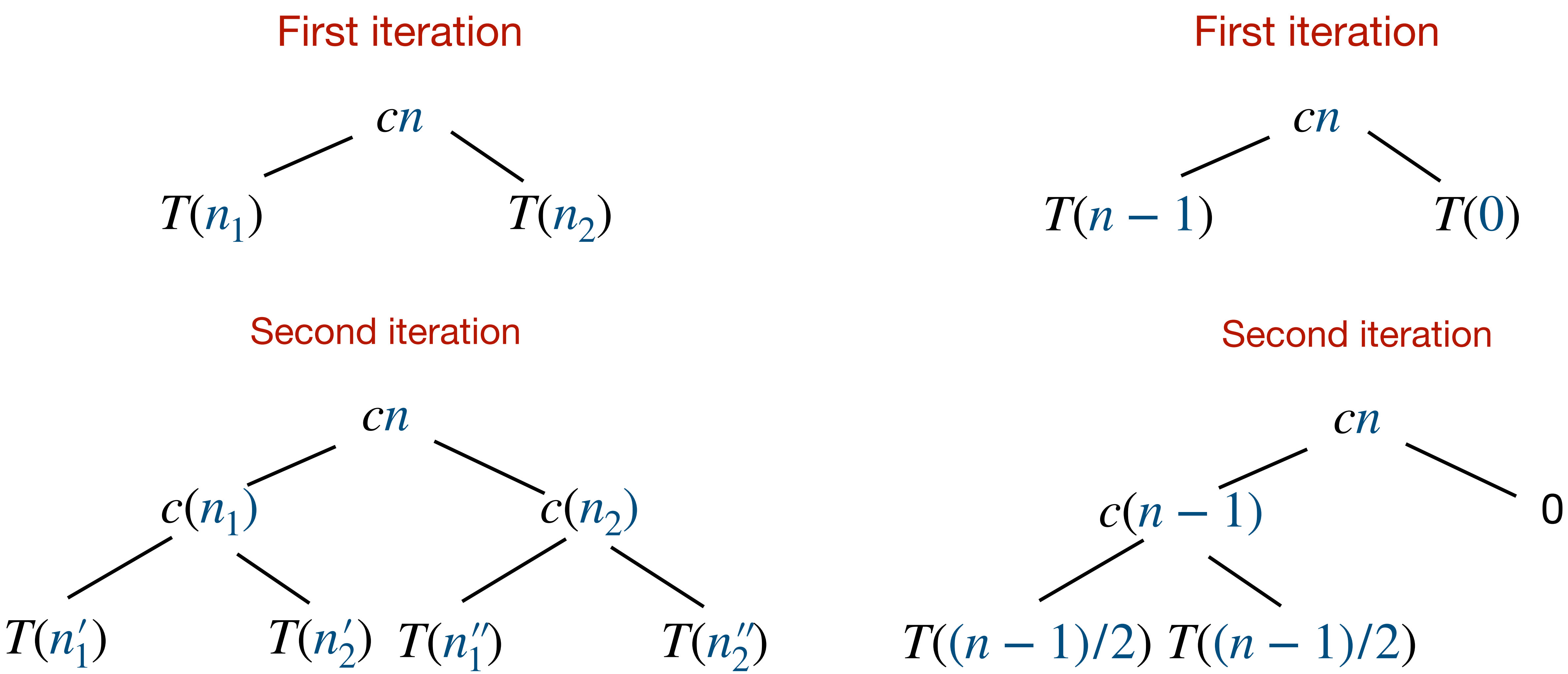
Second iteration



For the sake of intuition

Consider the recursion tree of **Quicksort**.

Assume bad and good levels *alternate*.



For the sake of intuition

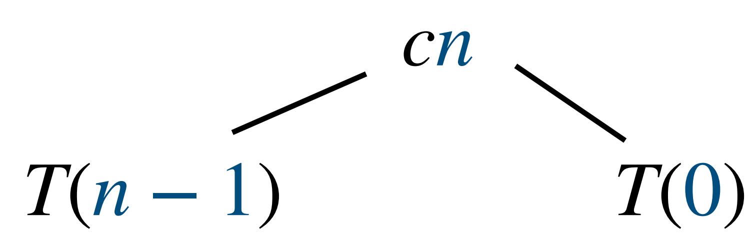
Consider the recursion tree of **Quicksort**.

Assume bad and good levels *alternate*.

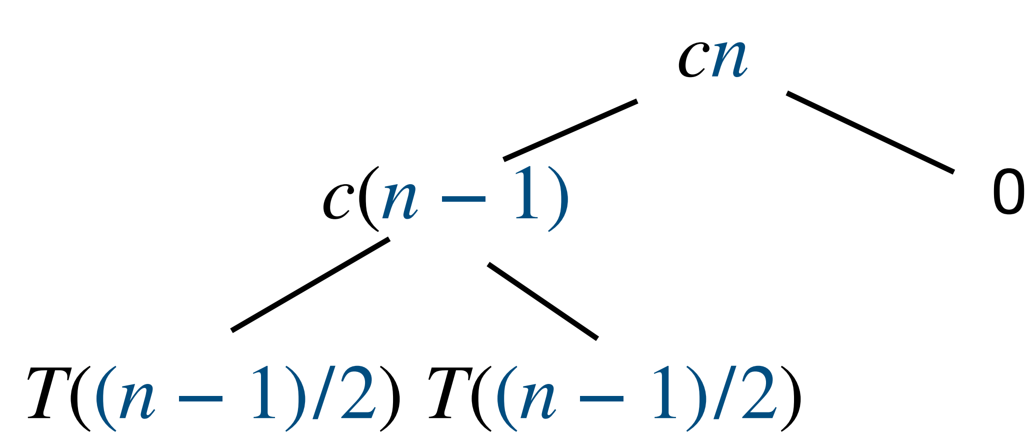
Recurrence:

$$T(n) \leq T(n-1) + cn \leq 2T(n-1/2) + c(2n-1)$$

First iteration



Second iteration



For the sake of intuition

Consider the recursion tree of **Quicksort**.

Assume bad and good levels *alternate*.

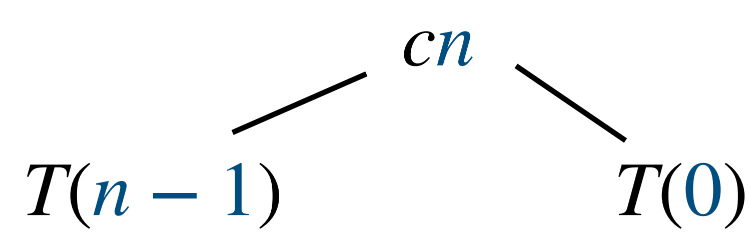
Recurrence:

$$T(n) \leq T(n-1) + cn \leq 2T(n-1/2) + c(2n-1)$$

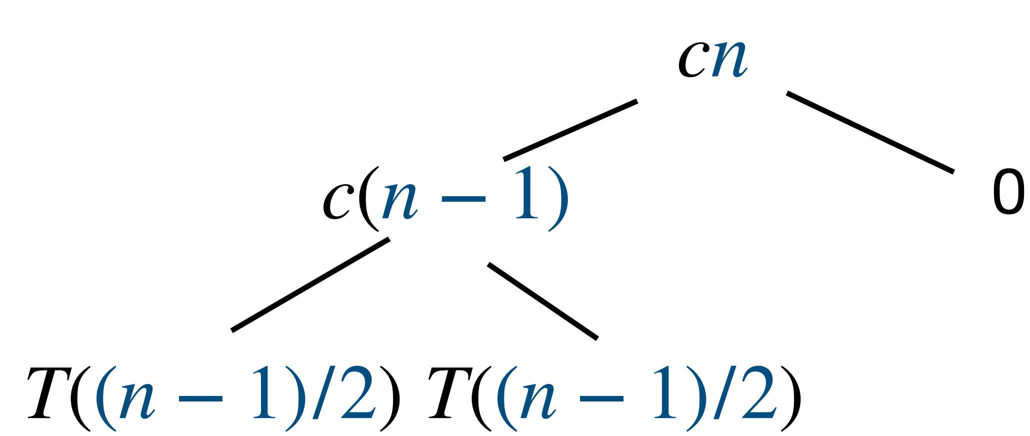
Almost the same as:

$$T(n) \leq 2T(n/2) + cn$$

First iteration



Second iteration



For the sake of intuition

Consider the recursion tree of **Quicksort**.

Assume bad and good levels *alternate*.

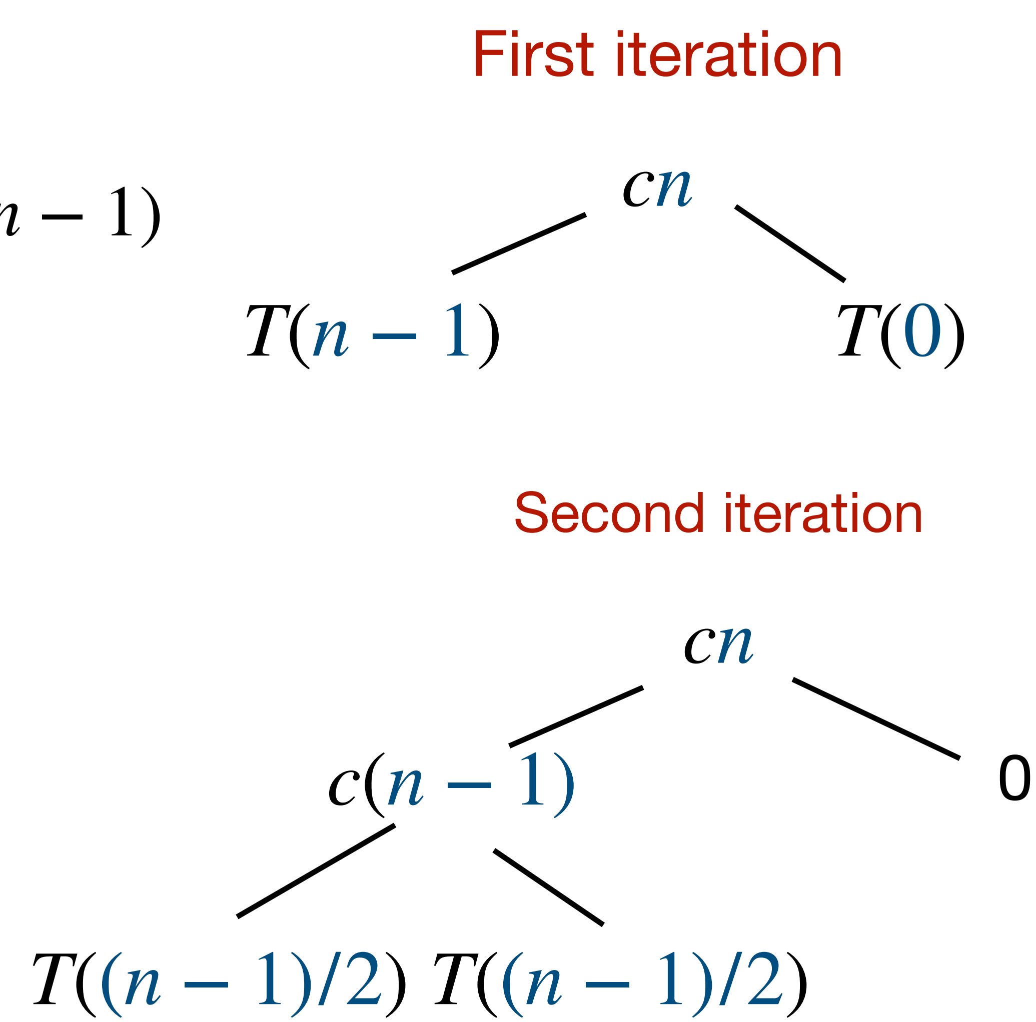
Recurrence:

$$T(n) \leq T(n-1) + cn \leq 2T(n-1/2) + c(2n-1)$$

Almost the same as:

$$T(n) \leq 2T(n/2) + cn$$

The cost of the unbalanced partition is “*absorbed*” in the cost of the balanced partition.



For the sake of intuition

Consider the recursion tree of **Quicksort**.

Assume bad and good levels *alternate*.

Recurrence:

$$T(n) \leq T(n-1) + cn \leq 2T(n-1/2) + c(2n-1)$$

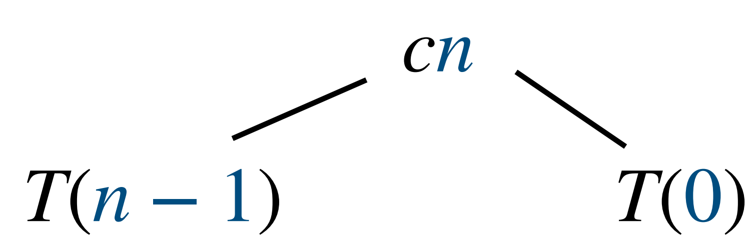
Almost the same as:

$$T(n) \leq 2T(n/2) + cn$$

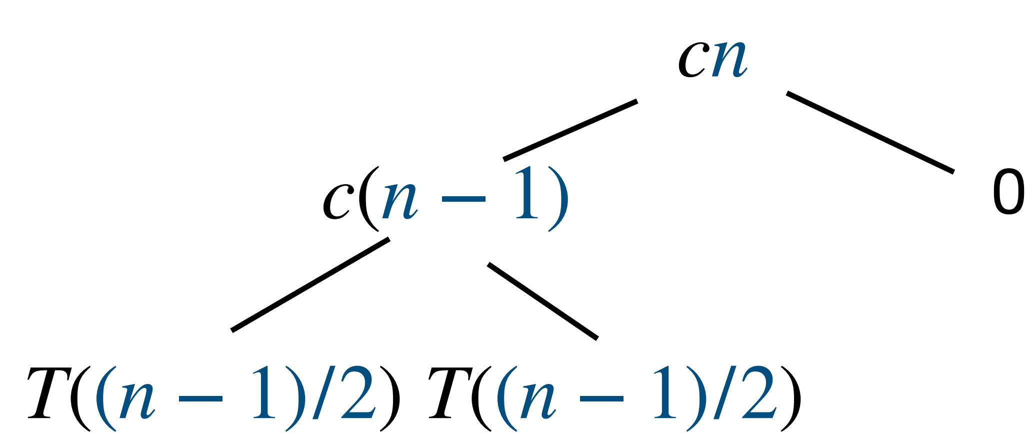
The cost of the unbalanced partition is “*absorbed*” in the cost of the balanced partition.

We only pay extra in constants.

First iteration



Second iteration



The Quicksort algorithm

Procedure **Partition**(**A**[*i*,...,*j*])

Choose a **pivot element** **x** of **A**

$k = i$

For $h = i$ to j do

If **A**[*h*] < **x**

Swap **A**[*k*] with **A**[*h*]
$k = k + 1$

Swap **A**[*k*] with **A**[*h*]

Return *k*

Algorithm **Quicksort**(**A**[*i*,...,*j*])

y = **Partition**(**A**[*i*,...,*j*])
Quicksort(**A**[*i*,...,*y*-1])
Quicksort(**A**[*y*+1,...,*j*])

The Quicksort algorithm

Procedure Partition(**A**[$i,\dots,j$])

Choose a **pivot element** **x** of **A**

$k = i$

For $h = i$ to j do

If **A**[h] < **x**

Swap **A**[k] with **A**[h]
$k = k + 1$

Swap **A**[k] with **A**[h]

Return k

Algorithm Quicksort(**A**[$i,\dots,j$])

y = Partition(**A**[$i,\dots,j$])
Quicksort(**A**[$i,\dots,y-1$])
Quicksort(**A**[$y+1,\dots,j$])

How many calls to Partition?

The Quicksort algorithm

Procedure **Partition**(**A**[$i,...,j$])

Choose a **pivot element** **x** of **A**

$k = i$

For $h = i$ to j do

If **A**[h] < **x**

Swap **A**[k] with **A**[h]
$k = k + 1$

Swap **A**[k] with **A**[h]

Return k

Algorithm **Quicksort**(**A**[$i,...,j$])

y = **Partition**(**A**[$i,...,j$])
Quicksort(**A**[$i,...,y-1$])
Quicksort(**A**[$y+1,...,j$])

How many calls to **Partition**? at most n

The Quicksort algorithm

Procedure **Partition**(**A**[$i,\ldots,j$])

Choose a **pivot element** **x** of **A**

$k = i$

For $h = i$ to j do

If **A**[h] < **x**

Swap **A**[k] with **A**[h]
$k = k + 1$

Swap **A**[k] with **A**[h]

Return k

Algorithm **Quicksort**(**A**[$i,\ldots,j$])

y = **Partition**(**A**[$i,\ldots,j$])
Quicksort(**A**[$i,\ldots,y-1$])
Quicksort(**A**[$y+1,\ldots,j$])

How many calls to **Partition**? at most n

How many calls to **Quicksort**?

The Quicksort algorithm

Procedure **Partition**(**A**[$i,\ldots,j$])

Choose a **pivot element** **x** of **A**

$k = i$

For $h = i$ to j do

If **A**[h] < **x**

Swap **A**[k] with **A**[h]
$k = k + 1$

Swap **A**[k] with **A**[h]

Return k

Algorithm **Quicksort**(**A**[$i,\ldots,j$])

y = **Partition**(**A**[$i,\ldots,j$])
Quicksort(**A**[$i,\ldots,y-1$])
Quicksort(**A**[$y+1,\ldots,j$])

How many calls to **Partition**? at most n

How many calls to **Quicksort**? at most $2n$

The Quicksort algorithm

Procedure **Partition**(**A**[$i,\ldots,j$])

Choose a **pivot element** **x** of **A**

$k = i$

For $h = i$ to j do

If **A**[h] < **x**

Swap **A**[k] with **A**[h]
$k = k + 1$

Swap **A**[k] with **A**[h]

Return k

Algorithm **Quicksort**(**A**[$i,\ldots,j$])

y = **Partition**(**A**[$i,\ldots,j$])
Quicksort(**A**[$i,\ldots,y-1$])
Quicksort(**A**[$y+1,\ldots,j$])

How many calls to **Partition**? at most n

How many calls to **Quicksort**? at most $2n$

How many operations, excluding those in the for loop?

The Quicksort algorithm

Procedure **Partition**(**A**[$i,\ldots,j$])

Choose a **pivot element** **x** of **A**

$k = i$

For $h = i$ to j do

If **A**[h] < **x**

Swap **A**[k] with **A**[h]
$k = k + 1$

Swap **A**[k] with **A**[h]

Return k

Algorithm **Quicksort**(**A**[$i,\ldots,j$])

y = **Partition**(**A**[$i,\ldots,j$])
Quicksort(**A**[$i,\ldots,y-1$])
Quicksort(**A**[$y+1,\ldots,j$])

How many calls to **Partition**? at most n

How many calls to **Quicksort**? at most $2n$

How many operations, excluding those in the for loop? $O(n)$

The Quicksort algorithm

Procedure **Partition**(**A**[$i,\ldots,j$])

Choose a **pivot element x** of **A**

$k = i$

For $h = i$ to j do

If **A**[h] < **x**

Swap **A**[k] with **A**[h]
$k = k + 1$

Swap **A**[k] with **A**[h]

Return k

Let X_k be the number of comparisons in the kth execution of the loop. Let $X = \sum_k X_k$

Algorithm **Quicksort**(**A**[$i,\ldots,j$])

y = **Partition**(**A**[$i,\ldots,j$])
Quicksort(**A**[$i,\ldots,y-1$])
Quicksort(**A**[$y+1,\ldots,j$])

How many calls to **Partition**? at most n

How many calls to **Quicksort**? at most $2n$

How many operations, excluding those in the for loop? $O(n)$

The Quicksort algorithm

Procedure **Partition**(**A**[*i*,...,*j*])

Choose a **pivot element** **x** of **A**

$k = i$

For $h = i$ to j do

If **A**[*h*] < **x**

Swap **A**[*k*] with **A**[*h*]
$k = k + 1$

Swap **A**[*k*] with **A**[*h*]

Return k

Let X_k be the number of comparisons in the kth execution of the loop. Let $X = \sum_k X_k$

Algorithm **Quicksort**(**A**[*i*,...,*j*])

y = **Partition**(**A**[*i*,...,*j*])
Quicksort(**A**[*i*,...,*y*-1])
Quicksort(**A**[*y*+1,...,*j*])

How many calls to **Partition**? at most n

How many calls to **Quicksort**? at most $2n$

How many operations, excluding those in the for loop? $O(n)$

How many operations in each execution of the loop?

The Quicksort algorithm

Procedure **Partition**(**A**[*i*,...,*j*])

Choose a **pivot element** **x** of **A**

$k = i$

For $h = i$ to j do

If **A**[*h*] < **x**

Swap **A**[*k*] with **A**[*h*]
$k = k + 1$

Swap **A**[*k*] with **A**[*h*]

Return *k*

Let X_k be the number of comparisons in the kth execution of the loop. Let $X = \sum_k X_k$

Algorithm **Quicksort**(**A**[*i*,...,*j*])

y = **Partition**(**A**[*i*,...,*j*])
Quicksort(**A**[*i*,...,*y*-1])
Quicksort(**A**[*y*+1,...,*j*])

How many calls to **Partition**? at most n

How many calls to **Quicksort**? at most $2n$

How many operations, excluding those in the for loop? $O(n)$

How many operations in each execution of the loop? $O(X_k)$

The Quicksort algorithm

Procedure **Partition**(**A**[*i*,...,*j*])

Choose a **pivot element** **x** of **A**

$k = i$

For $h = i$ to j do

If **A**[*h*] < **x**

Swap **A**[*k*] with **A**[*h*]
$k = k + 1$

Swap **A**[*k*] with **A**[*h*]

Return k

Let X_k be the number of comparisons in the kth execution of the loop. Let $X = \sum_k X_k$

Algorithm **Quicksort**(**A**[*i*,...,*j*])

y = **Partition**(**A**[*i*,...,*j*])
Quicksort(**A**[*i*,...,*y*-1])
Quicksort(**A**[*y*+1,...,*j*])

How many calls to **Partition**? at most n

How many calls to **Quicksort**? at most $2n$

How many operations, excluding those in the for loop? $O(n)$

How many operations in each execution of the loop? $O(X_k)$

How many operations in total?

The Quicksort algorithm

Procedure **Partition**(**A**[*i*,...,*j*])

Choose a **pivot element** **x** of **A**

k = *i*

For *h* = *i* to *j* do

If **A**[*h*] < **x**

Swap **A**[*k*] with **A**[*h*]
k = *k* + 1

Swap **A**[*k*] with **A**[*h*]

Return *k*

Let X_k be the number of comparisons in the kth execution of the loop. Let $X = \sum_k X_k$

Algorithm **Quicksort**(**A**[*i*,...,*j*])

y = **Partition**(**A**[*i*,...,*j*])

Quicksort(**A**[*i*,...,*y*-1])

Quicksort(**A**[*y*+1,...,*j*])

How many calls to **Partition**? at most n

How many calls to **Quicksort**? at most $2n$

How many operations, excluding those in the for loop? $O(n)$

How many operations in each execution of the loop? $O(X_k)$

How many operations in total? $O(n + X)$

Running time of Quicksort

Running time of Quicksort

The running time of the algorithm is $O(n + X)$ where X is the total number of comparisons.

Running time of Quicksort

The running time of the algorithm is $O(n + X)$ where X is the total number of comparisons.

When assuming that the input is drawn from a distribution, X is a *random variable*.

Running time of Quicksort

The running time of the algorithm is $O(n + X)$ where X is the total number of comparisons.

When assuming that the input is drawn from a distribution, X is a *random variable*.

We need to compute its expectation $\mathbb{E}[X]$.

Notation

Notation

Let $z_1, z_2, \ldots, z_n$ be the elements of the input array A *after they have been sorted.*

Notation

Let $z_1, z_2, \ldots, z_n$ be the elements of the input array A *after they have been sorted.*

This is for ease of reference: we might start with something like $z_3 \;\; z_5 \;\; z_1 \;\; z_8 \;\; \cdots \;\; z_2$

Notation

Let $z_1, z_2, \ldots, z_n$ be the elements of the input array A *after they have been sorted.*

This is for ease of reference: we might start with something like $z_3 \;\; z_5 \;\; z_1 \;\; z_8 \;\; \cdots \;\; z_2$

Let $Z_{ij} = \{z_i, z_{i+1}, \ldots, z_j\}$ contain the elements of a subsequence of the sorted array.

Useful Lemma

Useful Lemma

Lemma: During the execution of the algorithm, an element z_i is compared with an element z_j, where $i < j$ iff one of them is chosen as the pivot before any other element in the set Z_{ij}. Moreover, no two elements are ever compared more than once.

Useful Lemma

Lemma: During the execution of the algorithm, an element z_i is compared with an element z_j, where $i < j$ iff one of them is chosen as the pivot before any other element in the set Z_{ij}. Moreover, no two elements are ever compared more than once.

Proof:

Useful Lemma

Lemma: During the execution of the algorithm, an element z_i is compared with an element z_j, where $i < j$ iff one of them is chosen as the pivot before any other element in the set Z_{ij}. Moreover, no two elements are ever compared more than once.

Proof:

$\Leftarrow$

Useful Lemma

Lemma: During the execution of the algorithm, an element z_i is compared with an element z_j, where $i < j$ iff one of them is chosen as the pivot before any other element in the set Z_{ij}. Moreover, no two elements are ever compared more than once.

Proof:

$\Leftarrow$ If none of z_i and z_j is chosen as the pivot before any other element $z \in Z_{ij}$, then they are not compared with each other.

Useful Lemma

Lemma: During the execution of the algorithm, an element z_i is compared with an element z_j, where $i < j$ iff one of them is chosen as the pivot before any other element in the set Z_{ij}. Moreover, no two elements are ever compared more than once.

Proof:

$\Leftarrow$ If none of z_i and z_j is chosen as the pivot before any other element $z \in Z_{ij}$, then they are not compared with each other.

$$Z_{ij} = \{z_i, z_{i+1}, \ldots, z_k, \ldots, z_{j-1}, z_j\}$$

Useful Lemma

Lemma: During the execution of the algorithm, an element z_i is compared with an element z_j, where $i < j$ iff one of them is chosen as the pivot before any other element in the set Z_{ij}. Moreover, no two elements are ever compared more than once.

Proof:

$\Leftarrow$ If none of z_i and z_j is chosen as the pivot before any other element $z \in Z_{ij}$, then they are not compared with each other.

$$Z_{ij} = \{z_i, z_{i+1}, \ldots, z_k, \ldots, z_{j-1}, z_j\}$$

first pivot element from Z

Useful Lemma

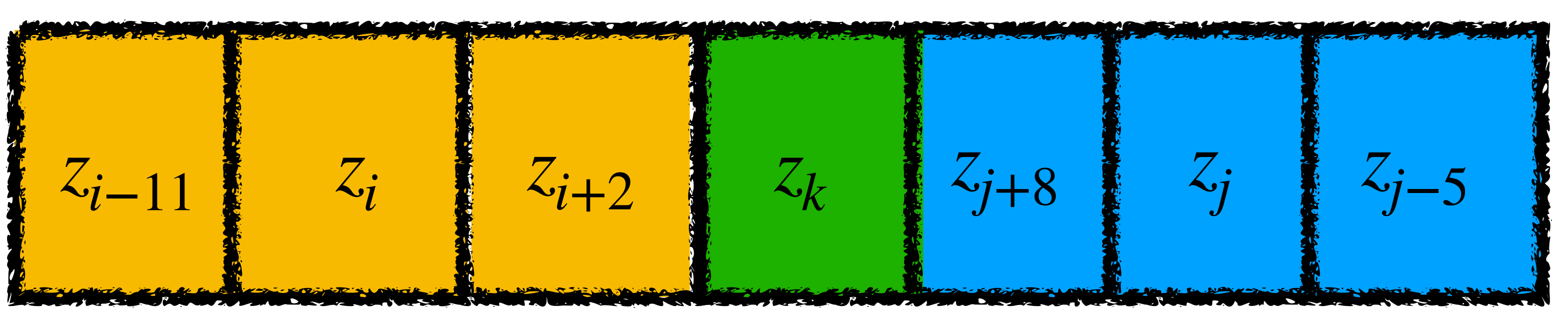
Lemma: During the execution of the algorithm, an element z_i is compared with an element z_j, where $i < j$ iff one of them is chosen as the pivot before any other element in the set Z_{ij}. Moreover, no two elements are ever compared more than once.

Proof:

$\Leftarrow$ If none of z_i and z_j is chosen as the pivot before any other element $z \in Z_{ij}$, then they are not compared with each other.

$$Z_{ij} = \{z_i, z_{i+1}, \ldots, z_k, \ldots, z_{j-1}, z_j\}$$

first pivot element from Z



Useful Lemma

Lemma: During the execution of the algorithm, an element z_i is compared with an element z_j, where $i < j$ iff one of them is chosen as the pivot before any other element in the set Z_{ij}. Moreover, no two elements are ever compared more than once.

Proof:

Useful Lemma

Lemma: During the execution of the algorithm, an element z_i is compared with an element z_j, where $i < j$ iff one of them is chosen as the pivot before any other element in the set Z_{ij}. Moreover, no two elements are ever compared more than once.

Proof:

$\Rightarrow$

Useful Lemma

Lemma: During the execution of the algorithm, an element z_i is compared with an element z_j, where $i < j$ iff one of them is chosen as the pivot before any other element in the set Z_{ij}. Moreover, no two elements are ever compared more than once.

Proof:

$\Rightarrow$ If one of z_i and z_j is chosen as the pivot before any other element $z \in Z_{ij}$, then they are compared with each other.

Useful Lemma

Lemma: During the execution of the algorithm, an element z_i is compared with an element z_j, where $i < j$ iff one of them is chosen as the pivot before any other element in the set Z_{ij}. Moreover, no two elements are ever compared more than once.

Proof:

$\Rightarrow$ If one of z_i and z_j is chosen as the pivot before any other element $z \in Z_{ij}$, then they are compared with each other.

$$Z_{ij} = \{z_i, z_{i+1}, \ldots, z_k, \ldots, z_{j-1}, z_j\}$$

Useful Lemma

Lemma: During the execution of the algorithm, an element z_i is compared with an element z_j, where $i < j$ iff one of them is chosen as the pivot before any other element in the set Z_{ij}. Moreover, no two elements are ever compared more than once.

Proof:

$\Rightarrow$ If one of z_i and z_j is chosen as the pivot before any other element $z \in Z_{ij}$, then they are compared with each other.

$$Z_{ij} = \{z_i, z_{i+1}, \ldots, z_k, \ldots, z_{j-1}, z_j\}$$

first pivot element from Z

Useful Lemma

Lemma: During the execution of the algorithm, an element z_i is compared with an element z_j, where $i < j$ iff one of them is chosen as the pivot before any other element in the set Z_{ij}. Moreover, no two elements are ever compared more than once.

Proof:

$\Rightarrow$ If one of z_i and z_j is chosen as the pivot before any other element $z \in Z_{ij}$, then they are compared with each other.

$$Z_{ij} = \{z_i, z_{i+1}, \ldots, z_k, \ldots, z_{j-1}, z_j\}$$

first pivot element from Z

z_i will be compared with every element $z \in Z$

Useful Lemma

Lemma: During the execution of the algorithm, an element z_i is compared with an element z_j, where $i < j$ iff one of them is chosen as the pivot before any other element in the set Z_{ij}. Moreover, no two elements are ever compared more than once.

Proof:

$\Rightarrow$ If one of z_i and z_j is chosen as the pivot before any other element $z \in Z_{ij}$, then they are compared with each other.

$$Z_{ij} = \{z_i, z_{i+1}, \ldots, z_k, \ldots, z_{j-1}, z_j\}$$

first pivot element from Z

z_j will be compared with every element $z \in Z$

Probability of comparison

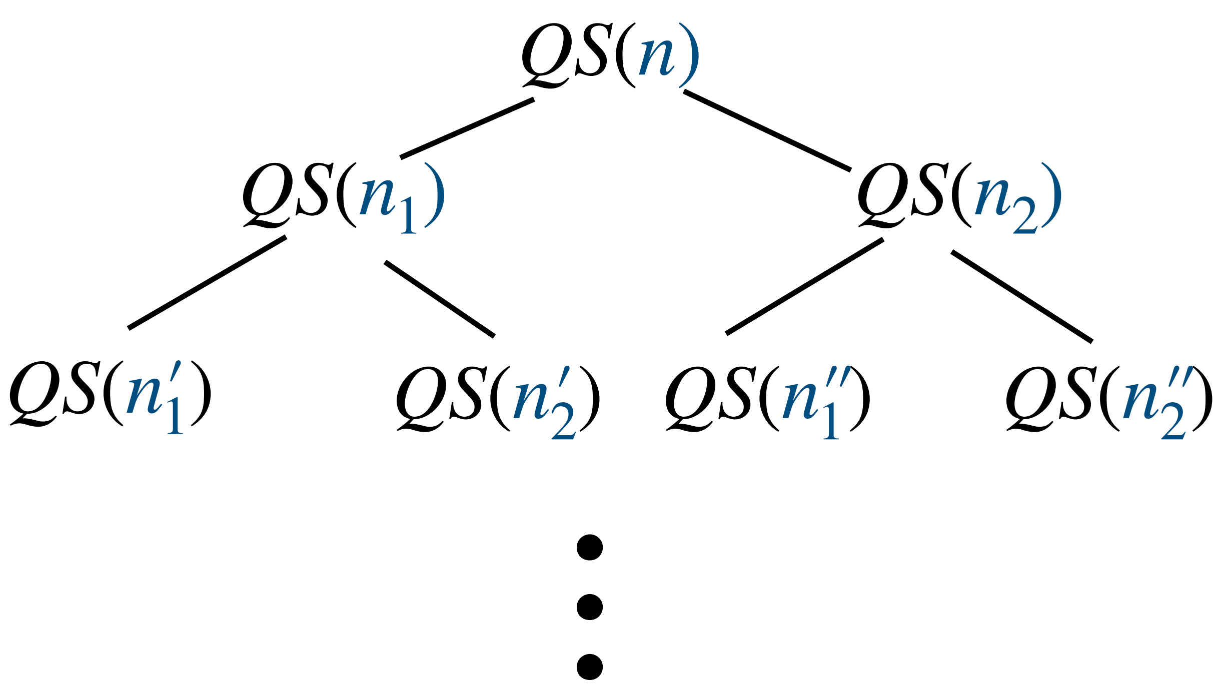
Lemma: Given two arbitrary elements $z_i, z_j \in Z_{ij}$, where $i < j$, the probability that they are compared is $2/(j - i + 1)$

Proof:

Probability of comparison

Lemma: Given two arbitrary elements $z_i, z_j \in Z_{ij}$, where $i < j$, the probability that they are compared is $2/(j - i + 1)$

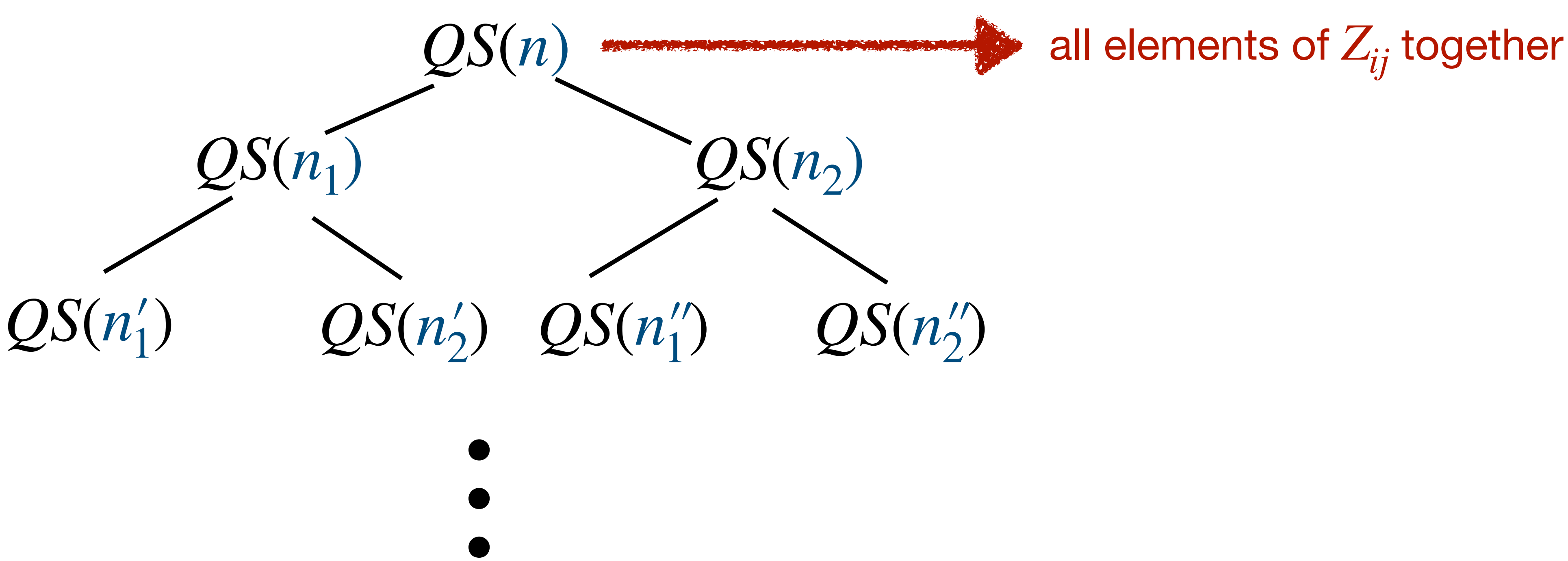
Proof:



Probability of comparison

Lemma: Given two arbitrary elements $z_i, z_j \in Z_{ij}$, where $i < j$, the probability that they are compared is $2/(j - i + 1)$

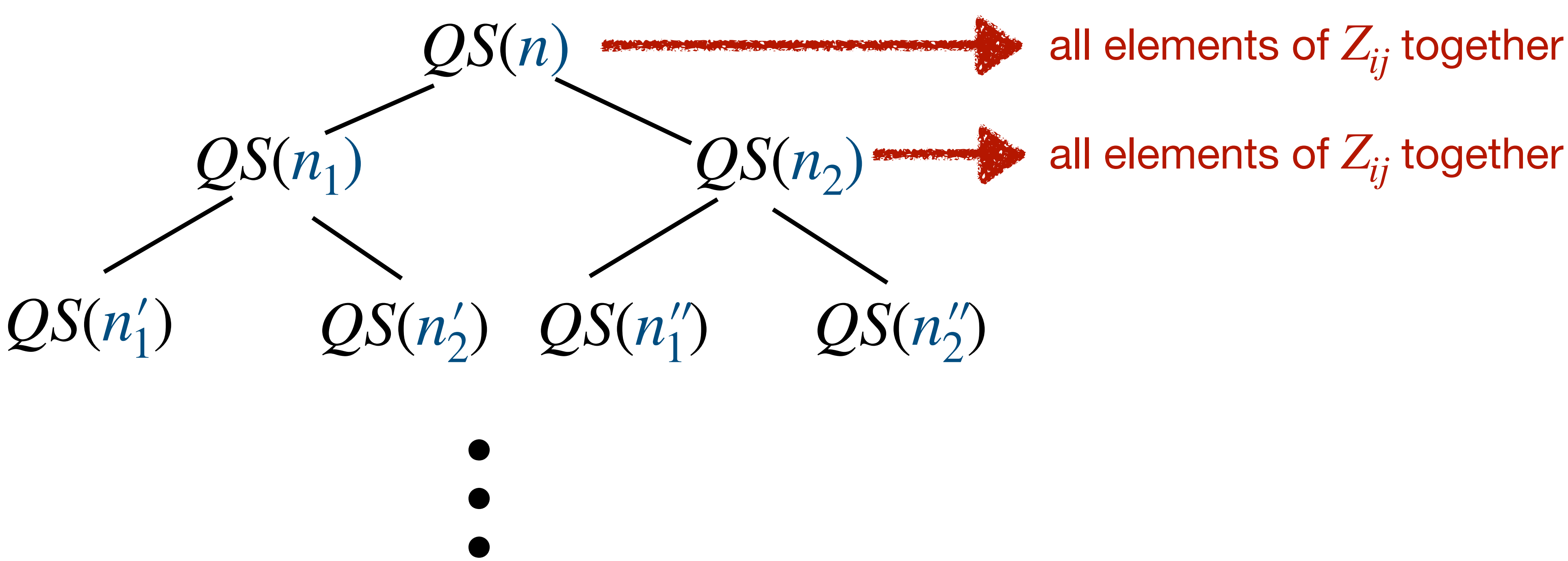
Proof:



Probability of comparison

Lemma: Given two arbitrary elements $z_i, z_j \in Z_{ij}$, where $i < j$, the probability that they are compared is $2/(j - i + 1)$

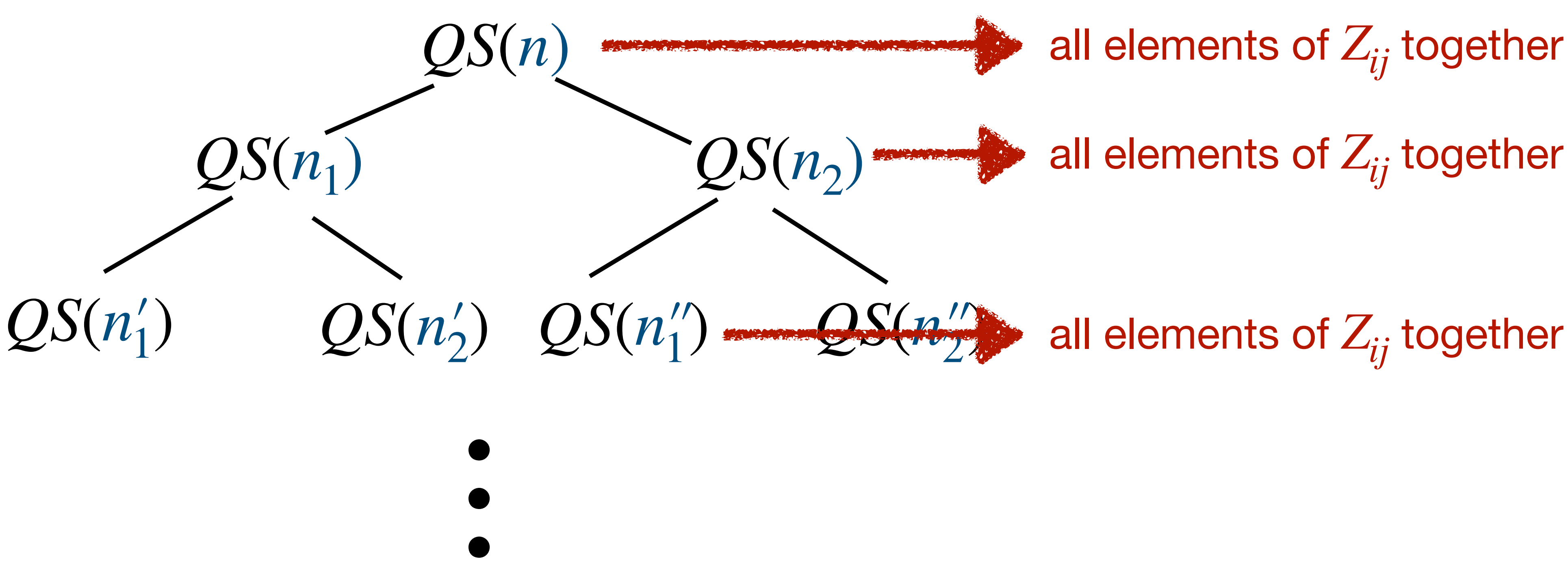
Proof:



Probability of comparison

Lemma: Given two arbitrary elements $z_i, z_j \in Z_{ij}$, where $i < j$, the probability that they are compared is $2/(j - i + 1)$

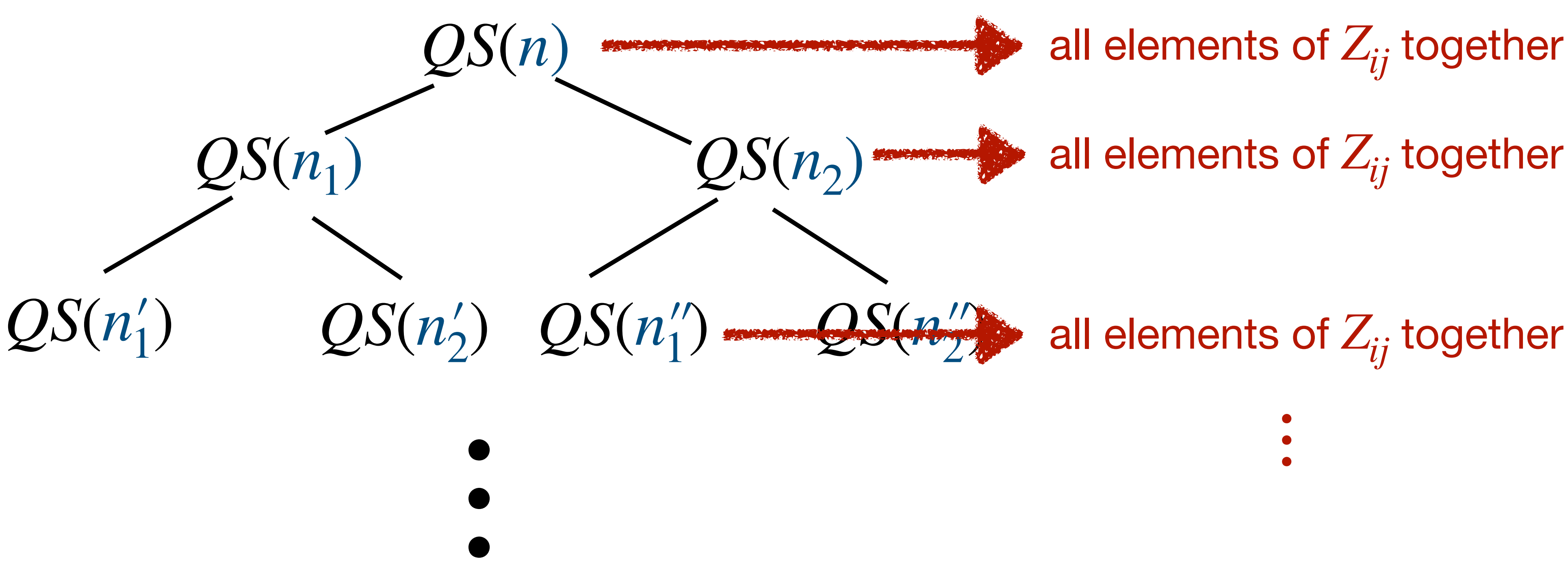
Proof:



Probability of comparison

Lemma: Given two arbitrary elements $z_i, z_j \in Z_{ij}$, where $i < j$, the probability that they are compared is $2/(j - i + 1)$

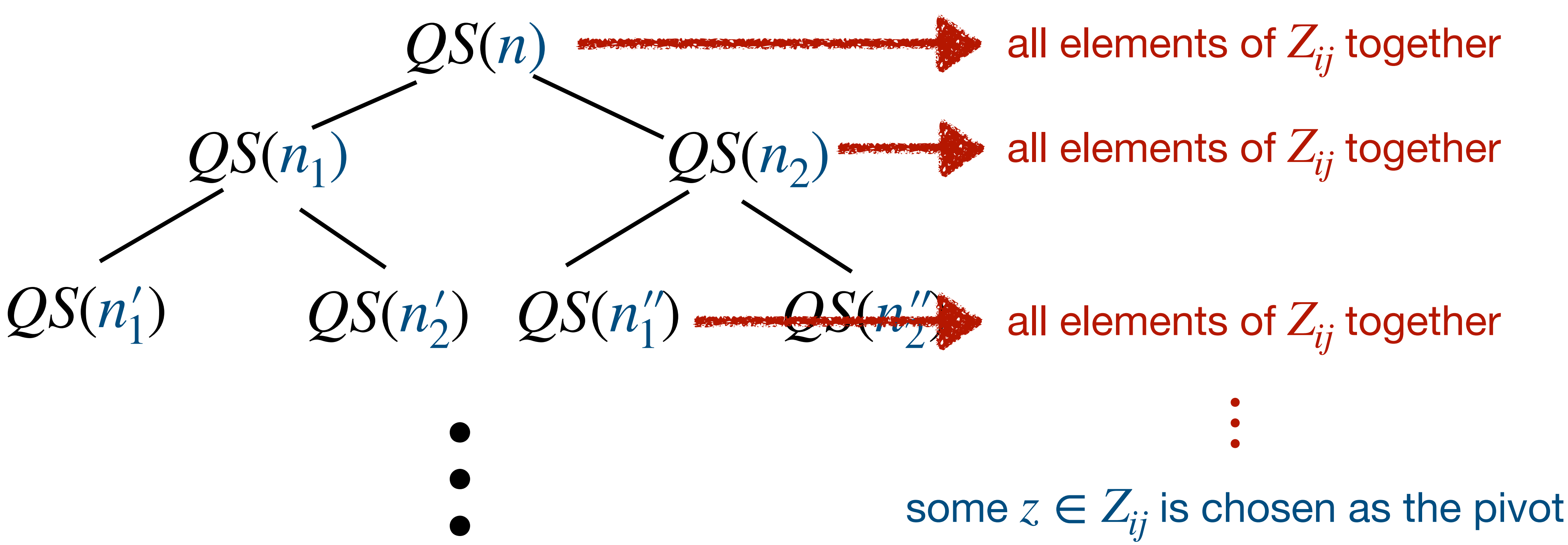
Proof:



Probability of comparison

Lemma: Given two arbitrary elements $z_i, z_j \in Z_{ij}$, where $i < j$, the probability that they are compared is $2/(j - i + 1)$

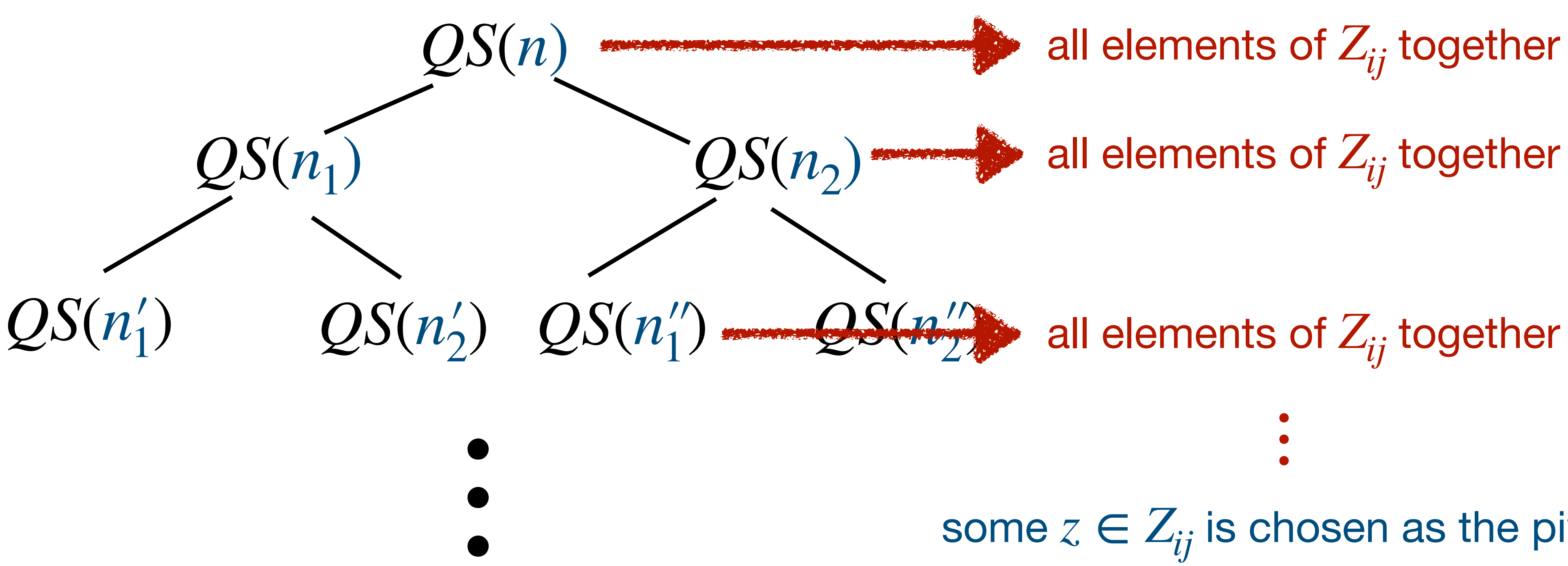
Proof:



Probability of comparison

Lemma: Given two arbitrary elements $z_i, z_j \in Z_{ij}$, where $i < j$, the probability that they are compared is $2/(j - i + 1)$

Proof:



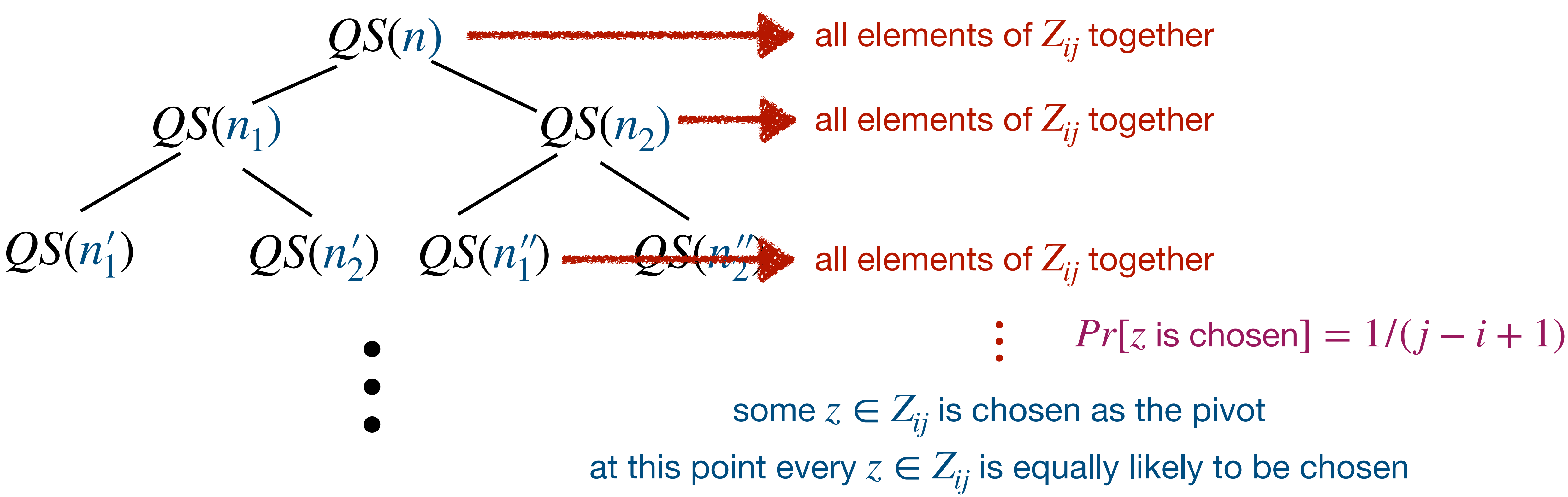
some $z \in Z_{ij}$ is chosen as the pivot

at this point every $z \in Z_{ij}$ is equally likely to be chosen

Probability of comparison

Lemma: Given two arbitrary elements $z_i, z_j \in Z_{ij}$, where $i < j$, the probability that they are compared is $2/(j - i + 1)$

Proof:



Probability of comparison

Lemma: Given two arbitrary elements $z_i, z_j \in Z_{ij}$, where $i < j$, the probability that they are compared is $2/(j - i + 1)$

Proof: $Pr[z \text{ is chosen}] = 1/(j - i + 1)$

Probability of comparison

Lemma: Given two arbitrary elements $z_i, z_j \in Z_{ij}$, where $i < j$, the probability that they are compared is $2/(j - i + 1)$

Proof: $Pr[z \text{ is chosen}] = 1/(j - i + 1)$

by the Useful Lemma, we have:

Probability of comparison

Lemma: Given two arbitrary elements $z_i, z_j \in Z_{ij}$, where $i < j$, the probability that they are compared is $2/(j - i + 1)$

Proof: $Pr[z \text{ is chosen}] = 1/(j - i + 1)$

by the Useful Lemma, we have:

$Pr[z_i$ **is compared with** $z_j] =$

Probability of comparison

Lemma: Given two arbitrary elements $z_i, z_j \in Z_{ij}$, where $i < j$, the probability that they are compared is $2/(j - i + 1)$

Proof: $Pr[z \text{ is chosen}] = 1/(j - i + 1)$

by the Useful Lemma, we have:

$Pr[z_i$ **is compared with** $z_j] = Pr[z_i$ **or** z_j **is the first pivot chosen from** $Z_{ij}]$

Probability of comparison

Lemma: Given two arbitrary elements $z_i, z_j \in Z_{ij}$, where $i < j$, the probability that they are compared is $2/(j - i + 1)$

Proof: $Pr[z \text{ is chosen}] = 1/(j - i + 1)$

by the Useful Lemma, we have:

$Pr[z_i$ **is compared with** $z_j] = Pr[z_i$ **or** z_j **is the first pivot chosen from** $Z_{ij}]$

$= Pr[z_i$ **is the first pivot chosen from** $Z_{ij}] +$

Probability of comparison

Lemma: Given two arbitrary elements $z_i, z_j \in Z_{ij}$, where $i < j$, the probability that they are compared is $2/(j - i + 1)$

Proof: $Pr[z \text{ is chosen}] = 1/(j - i + 1)$

by the Useful Lemma, we have:

$$Pr[z_i \textbf{ is compared with } z_j] = Pr[z_i \textbf{ or } z_j \textbf{ is the first pivot chosen from } Z_{ij}]$$
$$= Pr[z_i \textbf{ is the first pivot chosen from } Z_{ij}] +$$
$$Pr[z_j \textbf{ is the first pivot chosen from } Z_{ij}]$$

Probability of comparison

Lemma: Given two arbitrary elements $z_i, z_j \in Z_{ij}$, where $i < j$, the probability that they are compared is $2/(j - i + 1)$

Proof: $Pr[z \text{ is chosen}] = 1/(j - i + 1)$

by the Useful Lemma, we have:

$Pr[z_i$ **is compared with** $z_j] = Pr[z_i$ **or** z_j **is the first pivot chosen from** $Z_{ij}]$

the two events are independent

$= Pr[z_i$ **is the first pivot chosen from** $Z_{ij}] +$

$Pr[z_j$ **is the first pivot chosen from** $Z_{ij}]$

Probability of comparison

Lemma: Given two arbitrary elements $z_i, z_j \in Z_{ij}$, where $i < j$, the probability that they are compared is $2/(j - i + 1)$

Proof: $Pr[z \text{ is chosen}] = 1/(j - i + 1)$

by the Useful Lemma, we have:

$Pr[z_i$ **is compared with** $z_j] = Pr[z_i$ **or** z_j **is the first pivot chosen from** $Z_{ij}]$

the two events are independent

$= Pr[z_i$ **is the first pivot chosen from** $Z_{ij}] +$

$Pr[z_j$ **is the first pivot chosen from** $Z_{ij}]$

$$= \frac{2}{j - i + 1}$$

Average-case running time of Quicksort

Average-case running time of Quicksort

Indicator Random Variable: $X_{ij} = \mathbb{I}\{z_i \text{ is compared with } z_j\}$, for $1 \leq i < j \leq n$.

Average-case running time of Quicksort

Indicator Random Variable: $X_{ij} = \mathbb{I}\{z_i \text{ is compared with } z_j\}$, for $1 \leq i < j \leq n$.

By Useful Lemma, each pair is compared at most once, hence we have:

Average-case running time of `Quicksort`

Indicator Random Variable: $X_{ij} = \mathbb{I}\{z_i \text{ is compared with } z_j\}$, for $1 \leq i < j \leq n$.

By Useful Lemma, each pair is compared at most once, hence we have:

$$X = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij} \quad \text{and} \quad \mathbb{E}[X] = \mathbb{E}\left[\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}\right]$$

Average-case running time of Quicksort

$$\mathbb{E}[X] = \mathbb{E}\left[\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}\right]$$

Average-case running time of Quicksort

$$\mathbb{E}[X] = \mathbb{E}\left[\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} X_{ij}\right]$$

$$= \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} \mathbb{E}\left[X_{ij}\right]$$

Average-case running time of Quicksort

$$\mathbb{E}[X] = \mathbb{E}\left[\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}\right]$$

$$= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \mathbb{E}\left[X_{ij}\right]$$ *by linearity of expectation*

Average-case running time of Quicksort

$$\mathbb{E}[X] = \mathbb{E}\left[\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} X_{ij}\right]$$

$$= \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} \mathbb{E}\left[X_{ij}\right]$$ *by linearity of expectation*

$$= \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} Pr[z_i \text{ is compared with } z_j]$$

Average-case running time of Quicksort

$$\mathbb{E}[X] = \mathbb{E}\left[\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} X_{ij}\right]$$

$$= \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} \mathbb{E}\left[X_{ij}\right]$$ *by linearity of expectation*

$$\mathbb{E}[X_j] = Pr[X_j = 1] \cdot 1 + Pr[X_j = 0] \cdot 0 = Pr[X_j = 1]$$

$$= \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} Pr[z_i \text{ is compared with } z_j]$$

Average-case running time of Quicksort

$$\mathbb{E}[X] = \mathbb{E}\left[\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} X_{ij}\right]$$

$$= \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} \mathbb{E}\left[X_{ij}\right]$$ *by linearity of expectation*

$$\mathbb{E}[X_j] = Pr[X_j = 1] \cdot 1 + Pr[X_j = 0] \cdot 0 = Pr[X_j = 1]$$

$$= \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} Pr[z_i \text{ is compared with } z_j]$$

$$= \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} \frac{2}{j - i + 1}$$

Average-case running time of Quicksort

$$\mathbb{E}[X] = \mathbb{E}\left[\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} X_{ij}\right]$$

$$= \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} \mathbb{E}\left[X_{ij}\right]$$ *by linearity of expectation*

$\mathbb{E}[X_j] = Pr[X_j = 1] \cdot 1 + Pr[X_j = 0] \cdot 0 = Pr[X_j = 1]$

$$= \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} Pr[z_i \text{ is compared with } z_j]$$

$$= \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} \frac{2}{j-i+1}$$ *by probability lemma*

Average-case running time of Quicksort

$$\mathbb{E}[X] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1}$$

Average-case running time of Quicksort

$$\mathbb{E}[X] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1}$$

$$= \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{2}{k+1}$$

Average-case running time of Quicksort

$$\mathbb{E}[X] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1}$$

$$= \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{2}{k+1}$$ *by change of variables*

Average-case running time of Quicksort

$$\mathbb{E}[X] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1}$$

$$= \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{2}{k+1}$$ *by change of variables*

$$\leq \sum_{i=1}^{n-1} \sum_{k=1}^{n} \frac{2}{k}$$

Average-case running time of `Quicksort`

$$\mathbb{E}[X] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1}$$

$$= \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{2}{k+1}$$ *by change of variables*

$$\leq \sum_{i=1}^{n-1} \sum_{k=1}^{n} \frac{2}{k}$$

$$\leq 2nH_k$$

Average-case running time of Quicksort

$$\mathbb{E}[X] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1}$$

$$= \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{2}{k+1}$$ *by change of variables*

$$\leq \sum_{i=1}^{n-1} \sum_{k=1}^{n} \frac{2}{k}$$

$$\leq 2nH_k$$ $1 + 1/2 + \ldots + 1/k$

Average-case running time of Quicksort

$$\mathbb{E}[X] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1}$$

$$= \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{2}{k+1}$$ *by change of variables*

$$\leq \sum_{i=1}^{n-1} \sum_{k=1}^{n} \frac{2}{k}$$

$$\leq 2nH_k$$ $1 + 1/2 + \ldots + 1/k$

$$= O(n \log n)$$

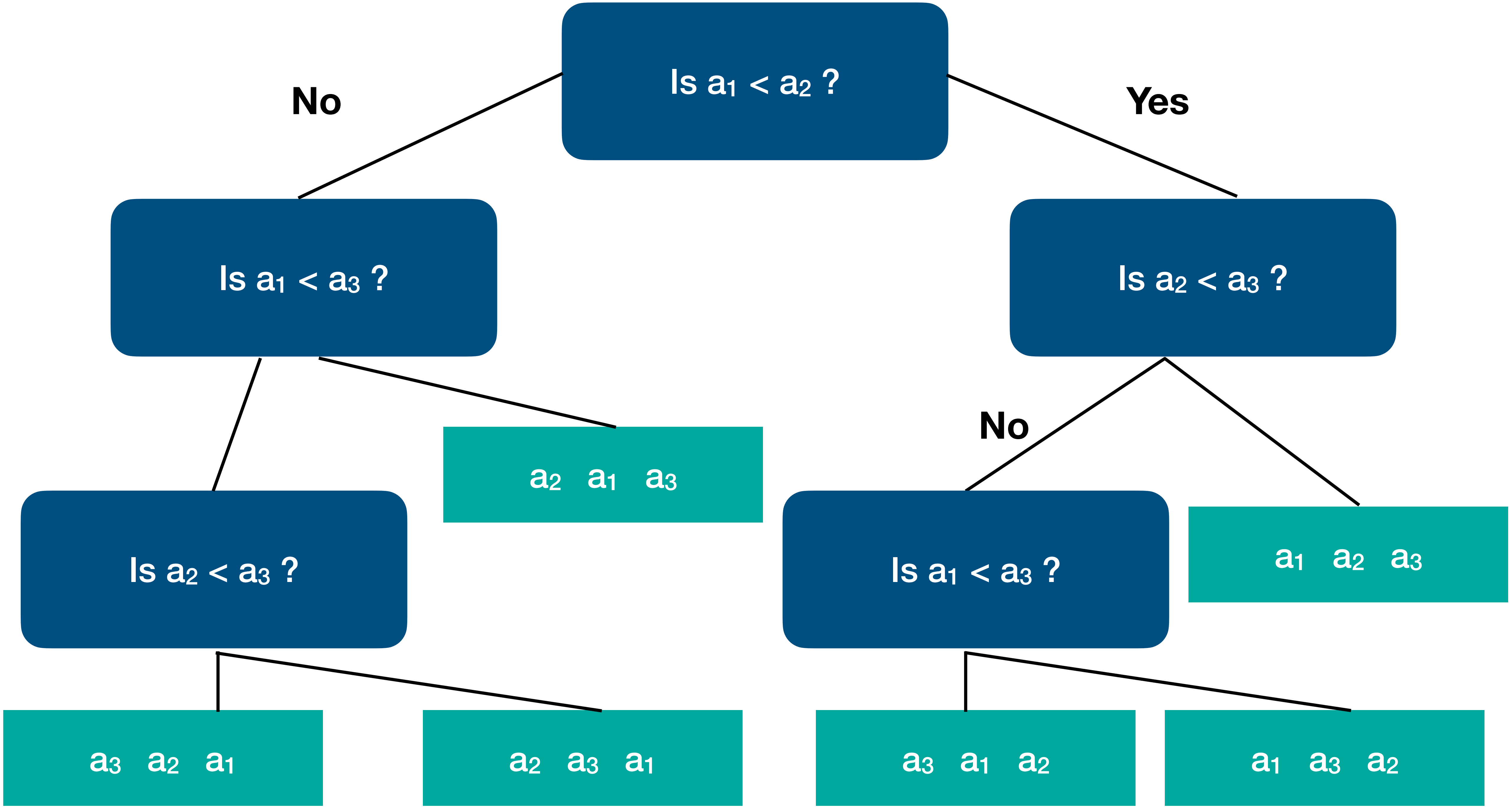
The Quicksort algorithm

```
Procedure Partition(A[i,...,j])

    Choose a pivot element x of A

    k = i

    For h = i to j do

        If A[h] < x

            Swap A[k] with A[h]
            k = k + 1

        Swap A[k] with A[h]

Return k
```

```
Algorithm Quicksort(A[i,...,j])

    y = Partition(A[i,...,j])
        Quicksort(A[i,...,y-1])
        Quicksort(A[y+1,...,j])
```

Q: Can you think of a version of the algorithm that will have worst-case running time $O(n \log n)$?

Lower bound for sorting



Lower bound for sorting

We need as many comparisons as the *depth* of the tree (length of the longest path from the root to the leaves).

The decision tree has $n!$ leaves

A leaf is a permutation of $\{a_1, a_2, \ldots, a_n\}$

Every possible permutation can appear as a leaf, since every possible permutation is a valid output.

Lower bound for sorting

Fact: Every binary tree of depth d has at most 2^d leaves.

Therefore the minimum number of comparisons is $\log_2(n!)$

We claim that $\log_2(n!) = \Omega(n \log n)$

$$
\begin{aligned}
\log_2(n!) &= \log_2(1 \cdot 2 \cdot , \ldots, \cdot n) \\
&= \log_2(1) + \log_2(2) + \ldots + \log_2(n) \\
&\geq \log_2(n/2) + \ldots + \log_2(n) \quad \textit{(half)} \\
&\geq \log_2(n/2) + \ldots + \log_2(n/2) = (n/2)\log_2(n/2)
\end{aligned}
$$

Worst-case lower bound for sorting

We need as many comparisons as the *depth* of the tree (length of the longest path from the root to the leaves).

The decision tree has $n!$ leaves

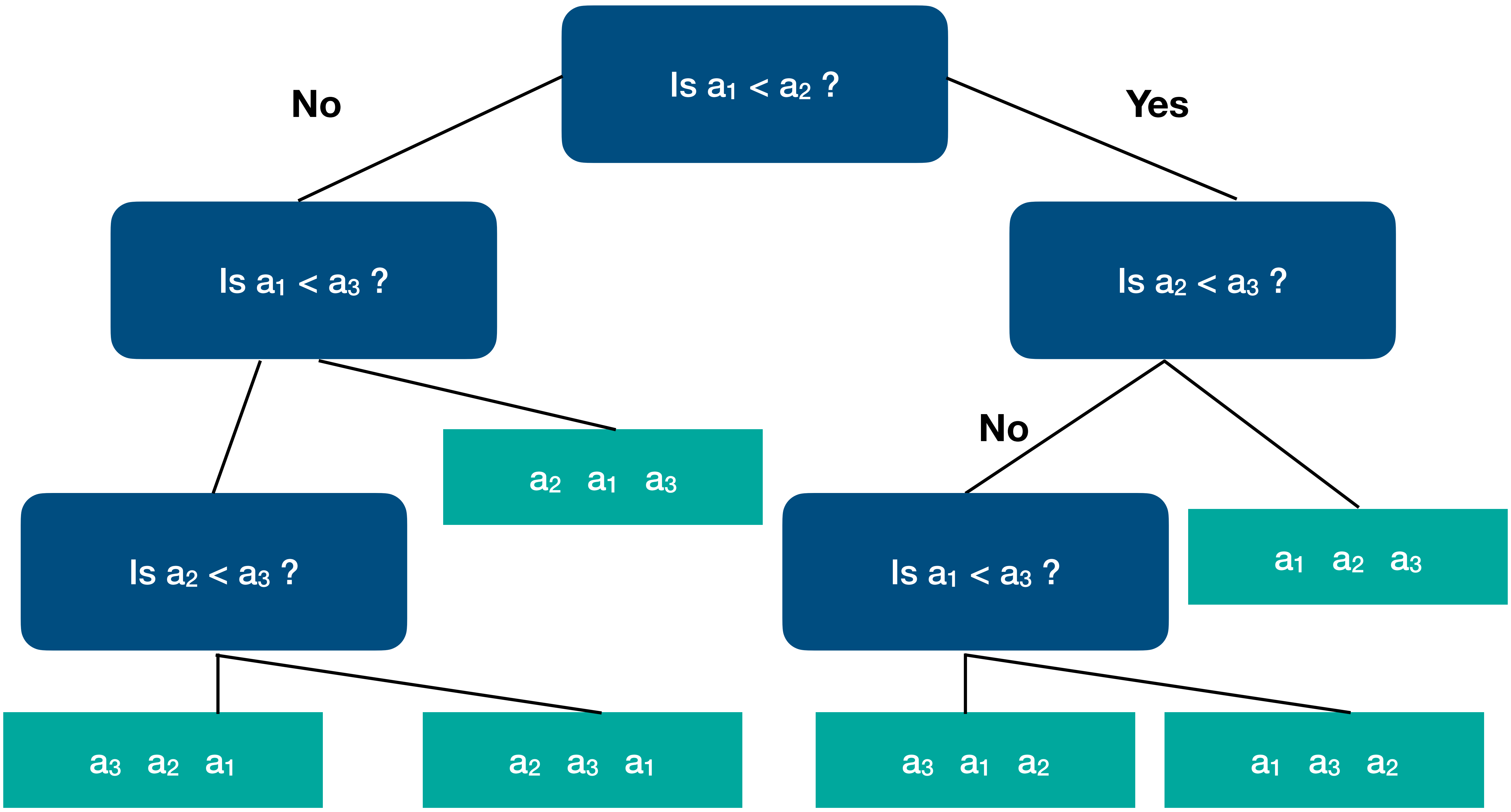
A leaf is a permutation of $\{a_1, a_2, \ldots , a_n\}$

Every possible permutation can appear as a leaf, since every possible permutation is a valid output.

Average-case lower bound for sorting

We need as many comparisons as the *average depth* of the tree (average length of the longest path from the root to a the leaves).

Average depth



Average-case lower bound for sorting

We need as many comparisons as the *average depth* of the tree (average length of the longest path from the root to a the leaves).

Average-case lower bound for sorting

We need as many comparisons as the *average depth* of the tree (average length of the longest path from the root to a the leaves).

Among all decision trees with a fixed number of leaves, which one has the smallest average depth?

Average-case lower bound for sorting

We need as many comparisons as the *average depth* of the tree (average length of the longest path from the root to a the leaves).

Among all decision trees with a fixed number of leaves, which one has the smallest average depth?

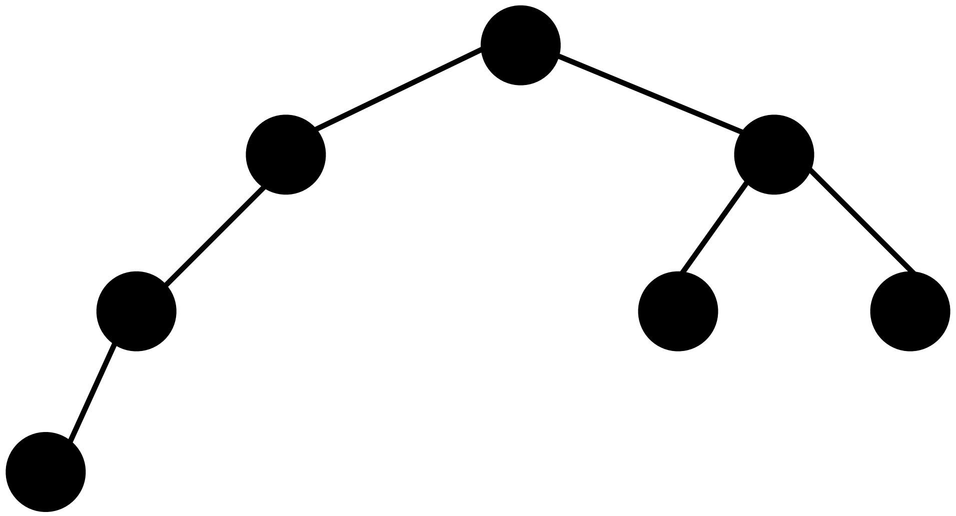
A completely balanced tree!

Average-case lower bound for sorting

We need as many comparisons as the *average depth* of the tree (average length of the longest path from the root to a the leaves).

Among all decision trees with a fixed number of leaves, which one has the smallest average depth?

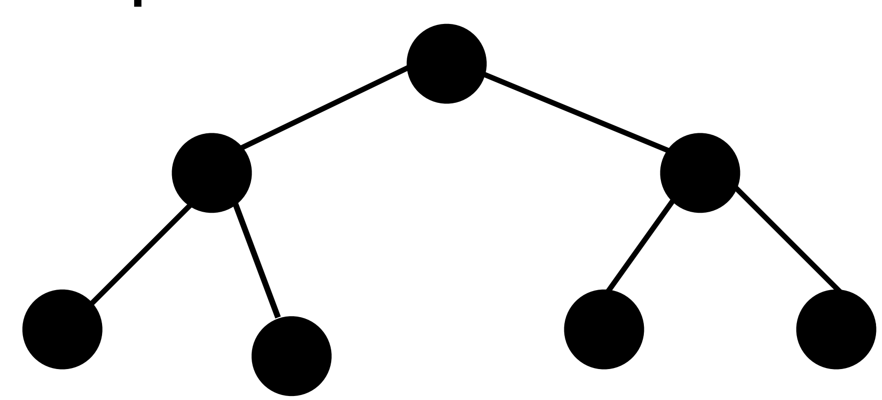
A completely balanced tree!



Average-case lower bound for sorting

We need as many comparisons as the *average depth* of the tree (average length of the longest path from the root to a the leaves).

Among all decision trees with a fixed number of leaves, which one has the smallest average depth?



A completely balanced tree!

Average-case lower bound for sorting

We need as many comparisons as the *average depth* of the tree (average length of the longest path from the root to a the leaves).

The depth of a balanced tree is $\Theta(\log_2 n!)$ and the analysis goes through as before.