Divide-and-Conquer: Quick Sort

Alexander S. Kulikov

Steklov Institute of Mathematics at St. Petersburg
Russian Academy of Sciences

Algorithmic Toolbox
Data Structures and Algorithms

Outline

Quick Sort

- comparison based algorithm
- running time: $O(n \log n)$ (on average)
- efficient in practice

Example: quick sort

6	4	8	2	9	3	9	4	7	6	1

Example: quick sort

6	4	8	2	9	3	9	4	7	6	1

partition with respect to $x = A[1]$

in particular, x is in its final position

1	4	2	3	4	6	6	9	7	8	9

≤ 6 $\qquad$ > 6

Example: quick sort

6	4	8	2	9	3	9	4	7	6	1

partition with respect to $x = A[1]$

in particular, x is in its final position

1	4	2	3	4	6	6	9	7	8	9

sort the two parts recursively

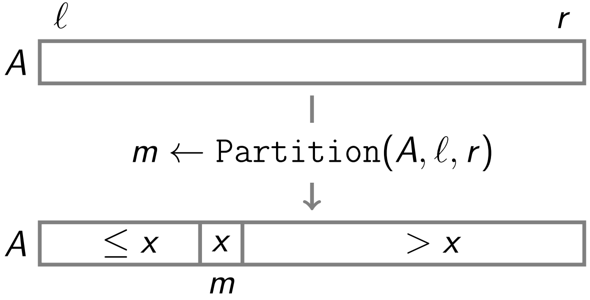
1	2	3	4	4	6	6	7	8	9	9

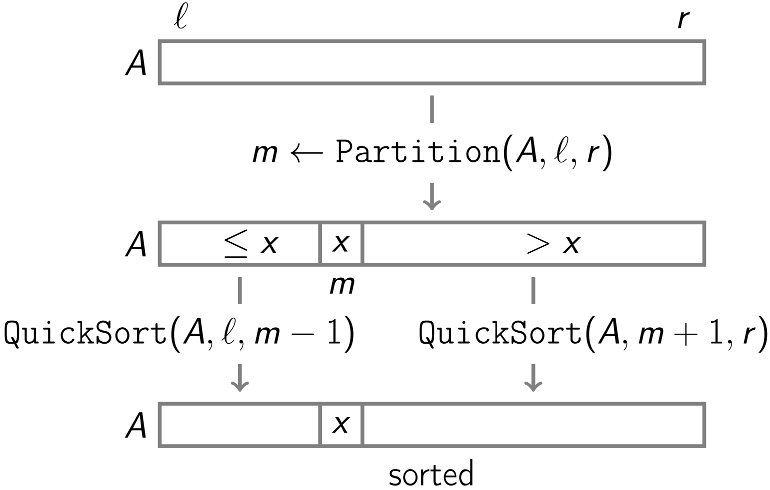
Outline

```
QuickSort(A, ℓ, r)

if ℓ ≥ r:
  return
m ← Partition(A, ℓ, r)
{A[m] is in the final position}
QuickSort(A, ℓ, m − 1)
QuickSort(A, m + 1, r)
```


ℓ
r
A


ℓ
r
A
$m \leftarrow \texttt{Partition}(A, \ell, r)$
A
$\leq x$
x
$> x$
m



ℓ
r
A
$m \leftarrow \texttt{Partition}(A, \ell, r)$
A
$\leq x$
x
$> x$
m
$\texttt{QuickSort}(A, \ell, m-1)$
$\texttt{QuickSort}(A, m+1, r)$
A
x
sorted

Partitioning: example

- the pivot is $x = A[\ell]$

Partitioning: example

- the pivot is $x = A[\ell]$
- move i from $\ell + 1$ to r maintaining the following invariant:

Partitioning: example

- the pivot is $x = A[\ell]$
- move i from $\ell + 1$ to r maintaining the following invariant:
  - $A[k] \leq x$ for all $\ell + 1 \leq k \leq j$

Partitioning: example

- the pivot is $x = A[\ell]$
- move i from $\ell + 1$ to r maintaining the following invariant:
  - $A[k] \leq x$ for all $\ell + 1 \leq k \leq j$
  - $A[k] > x$ for all $j + 1 \leq k \leq i$

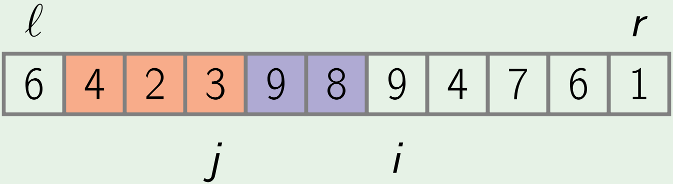
Partitioning: example

- the pivot is $x = A[\ell]$
- move i from $\ell + 1$ to r maintaining the following invariant:
  - $A[k] \leq x$ for all $\ell + 1 \leq k \leq j$
  - $A[k] > x$ for all $j + 1 \leq k \leq i$

ℓ										r
6	4	2	3	9	8	9	4	7	6	1
			j		i					

Partitioning: example

- the pivot is $x = A[\ell]$
- move i from $\ell + 1$ to r maintaining the following invariant:
  - $A[k] \le x$ for all $\ell + 1 \le k \le j$
  - $A[k] > x$ for all $j + 1 \le k \le i$



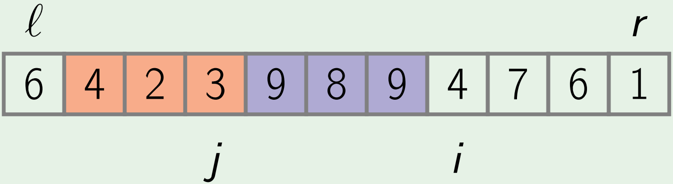
Partitioning: example

- the pivot is $x = A[\ell]$
- move i from $\ell + 1$ to r maintaining the following invariant:
  - $A[k] \leq x$ for all $\ell + 1 \leq k \leq j$
  - $A[k] > x$ for all $j + 1 \leq k \leq i$

ℓ										r
6	4	2	3	9	8	9	4	7	6	1
			j			i				

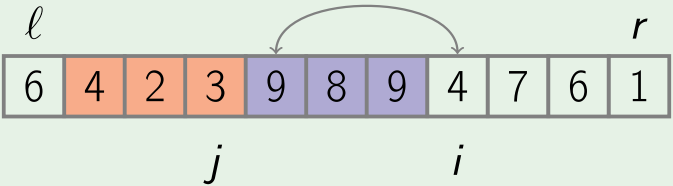
Partitioning: example

- the pivot is $x = A[\ell]$
- move i from $\ell + 1$ to r maintaining the following invariant:
  - $A[k] \leq x$ for all $\ell + 1 \leq k \leq j$
  - $A[k] > x$ for all $j + 1 \leq k \leq i$



Partitioning: example

- the pivot is $x = A[\ell]$
- move i from $\ell + 1$ to r maintaining the following invariant:
  - $A[k] \leq x$ for all $\ell + 1 \leq k \leq j$
  - $A[k] > x$ for all $j + 1 \leq k \leq i$



Partitioning: example

- the pivot is $x = A[\ell]$
- move i from $\ell + 1$ to r maintaining the following invariant:
  - $A[k] \leq x$ for all $\ell + 1 \leq k \leq j$
  - $A[k] > x$ for all $j + 1 \leq k \leq i$

ℓ										r
6	4	2	3	4	8	9	9	7	6	1
			j				i			

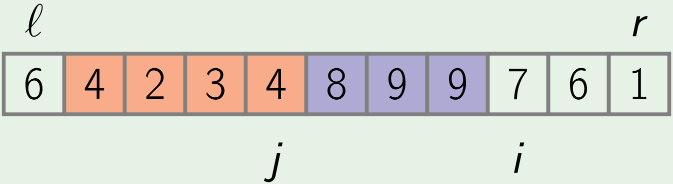
Partitioning: example

- the pivot is $x = A[\ell]$
- move i from $\ell + 1$ to r maintaining the following invariant:
  - $A[k] \leq x$ for all $\ell + 1 \leq k \leq j$
  - $A[k] > x$ for all $j + 1 \leq k \leq i$

ℓ										r
6	4	2	3	4	8	9	9	7	6	1
				j			i			

Partitioning: example

- the pivot is $x = A[\ell]$
- move i from $\ell + 1$ to r maintaining the following invariant:
  - $A[k] \leq x$ for all $\ell + 1 \leq k \leq j$
  - $A[k] > x$ for all $j + 1 \leq k \leq i$



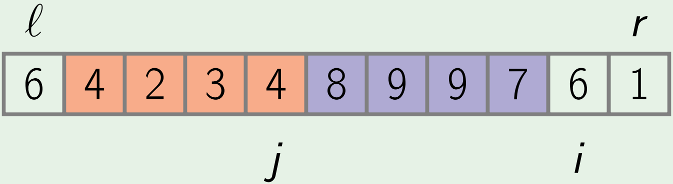
Partitioning: example

- the pivot is $x = A[\ell]$
- move i from $\ell + 1$ to r maintaining the following invariant:
  - $A[k] \leq x$ for all $\ell + 1 \leq k \leq j$
  - $A[k] > x$ for all $j + 1 \leq k \leq i$

ℓ										r
6	4	2	3	4	8	9	9	7	6	1
				j				i		

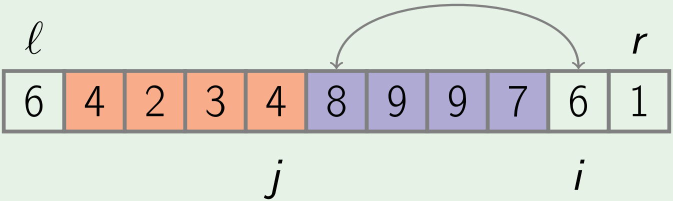
Partitioning: example

- the pivot is $x = A[\ell]$
- move i from $\ell + 1$ to r maintaining the following invariant:
  - $A[k] \leq x$ for all $\ell + 1 \leq k \leq j$
  - $A[k] > x$ for all $j + 1 \leq k \leq i$



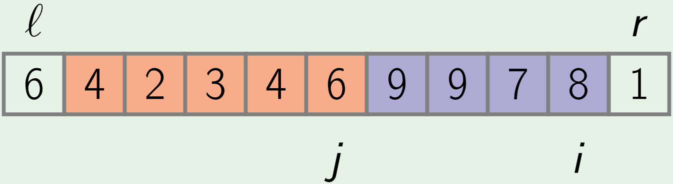
Partitioning: example

- the pivot is $x = A[\ell]$
- move i from $\ell + 1$ to r maintaining the following invariant:
  - $A[k] \leq x$ for all $\ell + 1 \leq k \leq j$
  - $A[k] > x$ for all $j + 1 \leq k \leq i$



Partitioning: example

- the pivot is $x = A[\ell]$
- move i from $\ell + 1$ to r maintaining the following invariant:
  - $A[k] \leq x$ for all $\ell + 1 \leq k \leq j$
  - $A[k] > x$ for all $j + 1 \leq k \leq i$



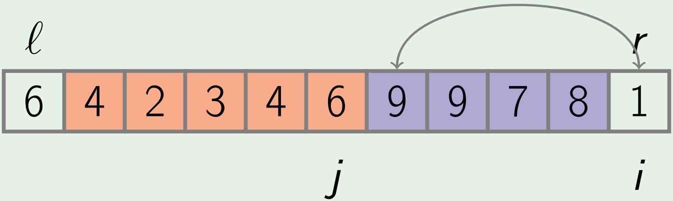
Partitioning: example

- the pivot is $x = A[\ell]$
- move i from $\ell + 1$ to r maintaining the following invariant:
  - $A[k] \leq x$ for all $\ell + 1 \leq k \leq j$
  - $A[k] > x$ for all $j + 1 \leq k \leq i$

ℓ										r
6	4	2	3	4	6	9	9	7	8	1
					j					i

Partitioning: example

- the pivot is $x = A[\ell]$
- move i from $\ell + 1$ to r maintaining the following invariant:
  - $A[k] \leq x$ for all $\ell + 1 \leq k \leq j$
  - $A[k] > x$ for all $j + 1 \leq k \leq i$



Partitioning: example

- the pivot is $x = A[\ell]$
- move i from $\ell + 1$ to r maintaining the following invariant:
  - $A[k] \leq x$ for all $\ell + 1 \leq k \leq j$
  - $A[k] > x$ for all $j + 1 \leq k \leq i$

ℓ										r
6	4	2	3	4	6	9	9	7	8	1
					j					i

Partitioning: example

- the pivot is $x = A[\ell]$
- move i from $\ell + 1$ to r maintaining the following invariant:
  - $A[k] \leq x$ for all $\ell + 1 \leq k \leq j$
  - $A[k] > x$ for all $j + 1 \leq k \leq i$

ℓ										r
6	4	2	3	4	6	9	9	7	8	1
						j				i

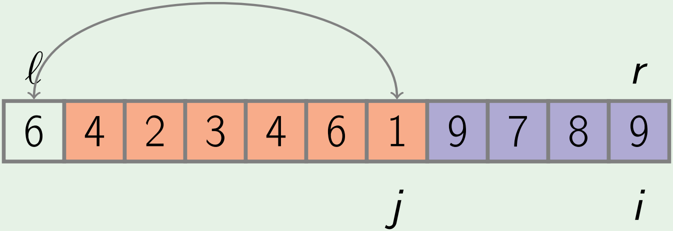
Partitioning: example

- the pivot is $x = A[\ell]$
- move i from $\ell + 1$ to r maintaining the following invariant:
  - $A[k] \leq x$ for all $\ell + 1 \leq k \leq j$
  - $A[k] > x$ for all $j + 1 \leq k \leq i$
- in the end, move $A[\ell]$ to its final place

ℓ										r
6	4	2	3	4	6	1	9	7	8	9
						j				i

Partitioning: example

- the pivot is $x = A[\ell]$
- move i from $\ell + 1$ to r maintaining the following invariant:
  - $A[k] \leq x$ for all $\ell + 1 \leq k \leq j$
  - $A[k] > x$ for all $j + 1 \leq k \leq i$
- in the end, move $A[\ell]$ to its final place



Partitioning: example

- the pivot is $x = A[\ell]$
- move i from $\ell + 1$ to r maintaining the following invariant:
  - $A[k] \leq x$ for all $\ell + 1 \leq k \leq j$
  - $A[k] > x$ for all $j + 1 \leq k \leq i$
- in the end, move $A[\ell]$ to its final place

ℓ										r
1	4	2	3	4	6	6	9	7	8	9
						j				i

$\texttt{Partition}(A, \ell, r)$

```
x ← A[ℓ]     {pivot}
j ← ℓ
for i from ℓ+1 to r:
   if A[i] ≤ x:
      j ← j+1
      swap A[j] and A[i]
   {A[ℓ+1...j] ≤ x, A[j+1...i] > x}
swap A[ℓ] and A[j]
return j
```

Outline

Unbalanced Partitions

- $T(n) = n + T(n-1)$:

  $$T(n) = n+(n-1)+(n-2)+\cdots = \Theta(n^2)$$

Unbalanced Partitions

- $T(n) = n + T(n-1)$:

  $$T(n) = n+(n-1)+(n-2)+\cdots = \Theta(n^2)$$

- $T(n) = n + T(n-5) + T(4)$:

  $$T(n) \geq n+(n-5)+(n-10)+\cdots = \Theta(n^2)$$

Balanced Partitions

- $T(n) = 2T(n/2) + n$:

$$T(n) = \Theta(n \log n)$$

Balanced Partitions

- $T(n) = 2T(n/2) + n$:

$$T(n) = \Theta(n \log n)$$

- $T(n) = T(n/10) + T(9n/10) + n$:

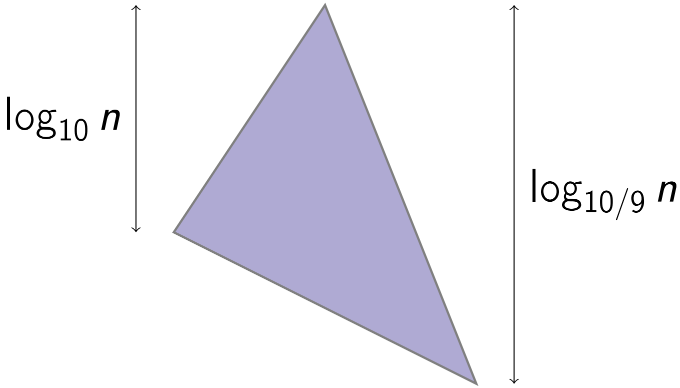
$$T(n) = \Theta(n \log n)$$

Balanced Partitions

$$T(n) = T(n/10) + T(9n/10) + O(n)$$

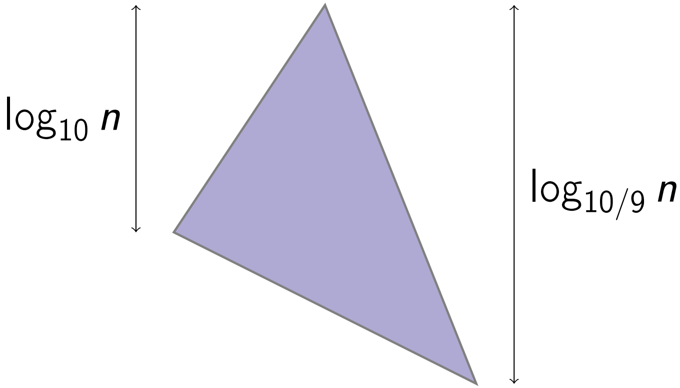
Balanced Partitions

$$T(n) = T(n/10) + T(9n/10) + O(n)$$



Balanced Partitions

$$T(n) = T(n/10) + T(9n/10) + O(n)$$



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

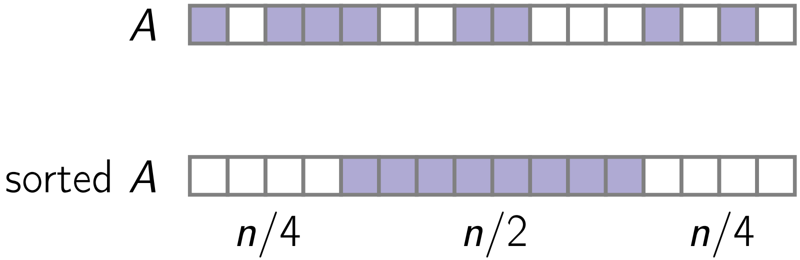
Random Pivot

```
RandomizedQuickSort(A, ℓ, r)

if ℓ ≥ r:
  return
k ← random number between ℓ and r
swap A[ℓ] and A[k]
m ← Partition(A, ℓ, r)
{A[m] is in the final position}
RandomizedQuickSort(A, ℓ, m − 1)
RandomizedQuickSort(A, m + 1, r)
```

Why Random?

half of the elements of A guarantees a balanced partition:



Theorem

Assume that all the elements of $A[1 \dots n]$ are pairwise different. Then the average running time of `RandomizedQuickSort`(A) is $O(n \log n)$ while the worst case running time is $O(n^2)$.

Theorem

Assume that all the elements of $A[1 \ldots n]$ are pairwise different. Then the average running time of `RandomizedQuickSort`(A) is $O(n \log n)$ while the worst case running time is $O(n^2)$.

Remark

Averaging is over random numbers used by the algorithm, but not over the inputs.

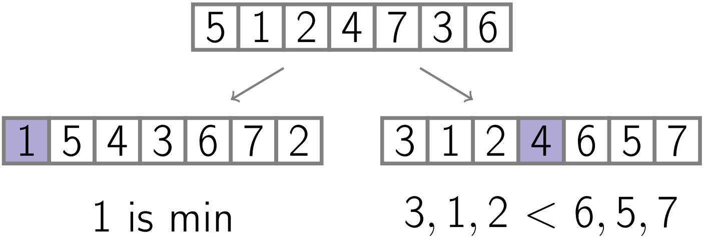
Outline

Proof Ideas: Comparisons

- the running time is proportional to the number of comparisons made

Proof Ideas: Comparisons

- the running time is proportional to the number of comparisons made
- balanced partition are better since they reduce the number of comparisons needed:



Proof Ideas: Probability

A	5	1	8	9	2	4	7	3	6
A'	1	2	3	4	5	6	7	8	9

Proof Ideas: Probability

A	5	1	8	9	2	4	7	3	6
A'	1	2	3	4	5	6	7	8	9

$$\text{Prob}\left(1 \text{ and } 9 \text{ are compared}\right) =$$

Proof Ideas: Probability

A	5	1	8	9	2	4	7	3	6
A'	1	2	3	4	5	6	7	8	9

$$\text{Prob}\left(\text{1 and 9 are compared}\right) = \frac{2}{9}$$

Proof Ideas: Probability

A	5	1	8	9	2	4	7	3	6
A'	1	2	3	4	5	6	7	8	9

$$\text{Prob}\left(1 \text{ and } 9 \text{ are compared}\right) = \frac{2}{9}$$

$$\text{Prob}\left(3 \text{ and } 4 \text{ are compared}\right) =$$

Proof Ideas: Probability

A	5	1	8	9	2	4	7	3	6
A'	1	2	3	4	5	6	7	8	9

$$\text{Prob}\left(1 \text{ and } 9 \text{ are compared}\right) = \frac{2}{9}$$

$$\text{Prob}\left(3 \text{ and } 4 \text{ are compared}\right) = 1$$

Proof

- let, for $i < j$,

$$\chi_{ij} = \begin{cases} 1 & A'[i] \text{ and } A'[j] \text{ are compared} \\ 0 & \text{otherwise} \end{cases}$$

Proof

- let, for $i < j$,

$$\chi_{ij} = \begin{cases} 1 & A'[i] \text{ and } A'[j] \text{ are compared} \\ 0 & \text{otherwise} \end{cases}$$

- for all $i < j$, $A'[i]$ and $A'[j]$ are either compared exactly once or not compared at all (as we compare with a pivot)

Proof

- let, for $i < j$,

$$\chi_{ij} = \begin{cases} 1 & A'[i] \text{ and } A'[j] \text{ are compared} \\ 0 & \text{otherwise} \end{cases}$$

- for all $i < j$, $A'[i]$ and $A'[j]$ are either compared exactly once or not compared at all (as we compare with a pivot)
- this, in particular, implies that the worst case running time is $O(n^2)$

Proof (continued)

- crucial observation: $\chi_{ij} = 1$ iff the first selected pivot in $A'[i \ldots j]$ is $A'[i]$ or $A'[j]$

Proof (continued)

- crucial observation: $\chi_{ij} = 1$ iff the first selected pivot in $A'[i \ldots j]$ is $A'[i]$ or $A'[j]$
- then $\text{Prob}(\chi_{ij}) = \frac{2}{j-i+1}$ and $\text{E}(\chi_{ij}) = \frac{2}{j-i+1}$

Proof (continued)

Then (the expected value of) the running time is

$$
\begin{aligned}
\mathrm{E} \sum_{i=1}^{n} \sum_{j=i+1}^{n} \chi_{ij} &= \sum_{i=1}^{n} \sum_{j=i+1}^{n} \mathrm{E}(\chi_{ij}) \\
&= \sum_{i<j} \frac{2}{j-i+1} \\
&\leq 2n \cdot \left(\frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n} \right) \\
&= \Theta(n \log n)
\end{aligned}
$$

Outline

Equal Elements

- what if all the elements of the given array are equal to each other?

Equal Elements

- what if all the elements of the given array are equal to each other?
- quick sort visualization

Equal Elements

- what if all the elements of the given array are equal to each other?
- quick sort visualization
- the array is always split into two parts of size 0 and $n - 1$

Equal Elements

- what if all the elements of the given array are equal to each other?
- quick sort visualization
- the array is always split into two parts of size 0 and $n-1$
- $T(n) = n + T(n-1) + T(0)$ and hence $T(n) = \Theta(n^2)$!

To handle equal elements, we replace the line

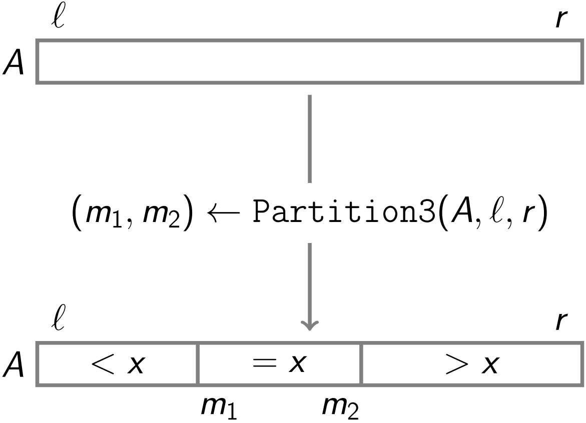
$$m \leftarrow \texttt{Partition}(A, \ell, r)$$

with the line

$$(m_1, m_2) \leftarrow \texttt{Partition3}(A, \ell, r)$$

such that

- for all $\ell \leq k \leq m_1 - 1$, $A[k] < x$
- for all $m_1 \leq k \leq m_2$, $A[k] = x$
- for all $m_2 + 1 \leq k \leq r$, $A[k] > x$


ℓ
r
A
$(m_1, m_2) \leftarrow \texttt{Partition3}(A, \ell, r)$
ℓ
r
A
$< x$
$= x$
$> x$
m_1
m_2

RandomizedQuickSort(A, ℓ, r)

```
if ℓ ≥ r:
  return
k ← random number between ℓ and r
swap A[ℓ] and A[k]
(m₁, m₂) ← Partition3(A, ℓ, r)
{A[m₁ ... m₂] is in final position}
RandomizedQuickSort(A, ℓ, m₁ − 1)
RandomizedQuickSort(A, m₂ + 1, r)
```

Outline

Tail Recursion Elimination

$\texttt{QuickSort}(A, \ell, r)$

```
while ℓ < r:
   m ← Partition(A, ℓ, r)
   QuickSort(A, ℓ, m − 1)
   ℓ ← m + 1
```

$\texttt{QuickSort}(A, \ell, r)$

```
while ℓ < r:
    m ← Partition(A, ℓ, r)
    if (m − ℓ) < (r − m):
        QuickSort(A, ℓ, m − 1)
        ℓ ← m + 1
    else:
        QuickSort(A, m + 1, r)
        r ← m − 1
```

QuickSort(A, ℓ, r)

```
while ℓ < r:
    m ← Partition(A, ℓ, r)
    if (m − ℓ) < (r − m):
        QuickSort(A, ℓ, m − 1)
        ℓ ← m + 1
    else:
        QuickSort(A, m + 1, r)
        r ← m − 1
```

Worst-case space requirement: $O(\log n)$

Intro Sort

- runs quick sort with a simple deterministic pivot selection heuristic (say, median of the first, middle, and last element)

Intro Sort

- runs quick sort with a simple deterministic pivot selection heuristic (say, median of the first, middle, and last element)
- if the recursion depth exceeds a certain threshold $c \log n$ the algorithm switches to heap sort

Intro Sort

- runs quick sort with a simple deterministic pivot selection heuristic (say, median of the first, middle, and last element)
- if the recursion depth exceeds a certain threshold $c \log n$ the algorithm switches to heap sort
- the running time is $O(n \log n)$ in the worst case

Conclusion

- Quick sort is a comparison based algorithm

Conclusion

- Quick sort is a comparison based algorithm
- Running time: $O(n \log n)$ on average, $O(n^2)$ in the worst case

Conclusion

- Quick sort is a comparison based algorithm
- Running time: $O(n \log n)$ on average, $O(n^2)$ in the worst case
- Efficient in practice