A simpler way is to choose a random element in \(A[p...r]\) as pivot, swap that with \(A[i]\), then partition as usual.

\[\text{RandPartition}(A, p, r)\]
1. \(i \leftarrow \text{Rand}(p, r)\)
2. \(A[i] \leftrightarrow A[r]\)
3. Return Partition \((A, p, r)\)

\[\text{RandQuicksort}(A, p, r)\]
1. If \(p < r\)
2. \(q \leftarrow \text{RandPartition}(A, p, r)\)
3. \(\text{RandQuicksort}(A, p, q-1)\)
4. \(\text{RandQuicksort}(A, q+1, r)\)

This is considered by some to be the algorithm of choice for sorting large inputs.
Ex.
MaxMin(A, p, r) finds the maximum and minimum elements in the subarray A[p...r].

\[ \min(m_1, m_2) \]

1.) if \( m_1 < m_2 \)
2.) return \( m_1 \)
3.) return \( m_2 \)

\( \max(M_1, M_2) \) is similar. Each does one comparison.

MaxMin(A, p, r) (Pre: \( p \leq r \))

1.) if \( p = r \)
2.) return \( \langle A[p], A[p] \rangle \)
3.) \( q \leftarrow \left\lfloor \frac{p + r}{2} \right\rfloor \)
4.) \( \langle m_1, M_1 \rangle \leftarrow \text{MaxMin}(A, q + 1, r) \)
5.) \( \langle m_2, M_2 \rangle \leftarrow \text{MaxMin}(A, p, q) \)
6.) return \( \langle \min(m_1, m_2), \max(M_1, M_2) \rangle \)

Let \( T(n) \) denote the number of comparisons performed by MaxMin(A, p, r) on arrays of length \( n \).
Then

\[ T(n) = \begin{cases} 
0 & \text{if } n = 1 \\
T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right) + 2 & \text{if } n \geq 2 
\end{cases} \]

**Exercise:**

Show that the exact solution is

\[ T(n) = 2n - 2 \]

This is no better than the obvious iterative algorithm.

**Exercise:**

Design a divide-and-conquer algorithm which finds maximum and minimum in exactly \( \lceil \frac{2n}{3} \rceil - 2 \) comparisons. (Hint: Section 9.1 describes an iterative algorithm to do this.)
**Example: The Selection Problem**

The *ith* order statistic of an array $A[1\ldots n]$ consisting of $n$ distinct elements is the $i^{th}$ smallest element. Equivalently, the $i^{th}$ order statistic is the unique element in $A$ which is greater than exactly $i-1$ other elements (where $1 \leq i \leq n$).

e.g. $i = 1$ gives the minimum
    $i = n$ gives the maximum

The $i^{th}$ order statistic is greater than or equal to exactly $i$ elements of $A$.

**Problem:** Given $A[1\ldots n]$, where all elements are distinct, determine the $i^{th}$ order statistic.

One approach would be to sort $A[1\ldots n]$, then return $A[i]$. In general this takes time $O(n \log n)$.

RandSelect is a randomized algorithm which finds the $i^{th}$ order statistic in linear time, on average.
Recall that \texttt{RandPartition(A, p, r)} splits the subarray \texttt{A[p..r]} into two subarrays \texttt{A[p..q]} and \texttt{A[q+1..r]}

\texttt{A[p..(q-1)] \leq A[q] \leq A[(q+1)..r]}

\texttt{RandSelect(A, p, r, i)} (Pre: \(1 \leq i \leq r-p+1\))

1. \texttt{if } p = r
2. \texttt{return } A[p]
3. \texttt{q} \leftarrow \texttt{RandPartition(A, p, r)}
4. \texttt{k} \leftarrow q - p + 1 // \texttt{k} is length of \texttt{A[p..q]}
5. \texttt{if } k = i
6. \texttt{return } A[q]
7. \texttt{else if } i < k
8. \texttt{return } \texttt{RandSelect(A, p, q-1, i)}
9. \texttt{else}
10. \texttt{return } \texttt{RandSelect(A, q+1, r, i-k)}

\texttt{RandSelect} is similar in some respects to \texttt{RandQuicksort} and \texttt{RandBinarySearch}. Like \texttt{Quicksort}, it randomly splits the subarray \texttt{A[p..r]} in order to exploit a good average-case runtime. Like \texttt{BinarySearch}, it recurs on only one subarray. Unlike \texttt{BinarySearch}, we seek not an index, but an array element.
Let \( t(n) \) denote the average number of (array) comparisons by RandSelect \((A, \ell, n, i)\).

Assume that each permutation of \( A[1..n] \) is equally likely; hence the return value of RandPartition is equally likely to be any of the numbers \( 1 \leq q \leq n \).

A priori \( t(n) \) depends on \( \ell \). We'll see that in fact it doesn't. Recall that RandPartition does \( (n-1) \) comparisons. Thus

\[
\sum_{q=1}^{n} \left( (n-1)P(i \leq q) t(q-1) + P(i > q) t(n-q) \right)
\]

\[
\frac{t(n)}{n}
\]

where

\[
P(i \leq q) = \frac{n-i}{n} \quad \text{and} \quad P(i > q) = \frac{i-1}{n}
\]

are the probabilities that \( q \) is in the range \( i < q \leq n \) and \( 1 < q < i \) respectively. Thus

\[
t(n) = (n-1) + \frac{1}{n} \sum_{q=1}^{n} \left( (n-1) t(q-1) + (i-1) t(n-q) \right)
\]

\[
= (n-1) + \frac{1}{n^2} \left[ \sum_{q=1}^{n-1} t(q) + (n-1) \sum_{q=1}^{n-1} t(q) \right]
\]
\[ t(n) = (n-1) + \frac{1}{n^2} (n-1 + i - 1) \sum_{q=1}^{n-1} t(q) \]

Therefore, \( t(n) \) does not depend on \( i \), as claimed earlier.

Observe that this recurrence is very similar to the one for the average run time of Quicksort.

**Exercise**
Show that \( t(n) = O(n) \).

Obviously, \( t(n) \geq n - 1 = \Omega(n) \). Prove that \( t(n) = O(n) \) by induction on \( n \).

**Induction Hypothesis**: \( \forall q \leq n-1 : t(q) \leq 2q \)

\[ t(n) \leq (n-1) + \frac{1}{n^2} (n-1 + i - 1) \sum_{q=1}^{n-1} 2q \]

\[ = (n-1) + \frac{1}{n^2} \cdot 2 \frac{n(n-1)}{2} \leq 2n \]

\[ \uparrow \text{ prove} \]
Exercise
Find the exact solution to this recurrence.

Answer:
\[ t(n) = (n-1) \left\{ 1 + \sum_{i=1}^{n-1} \frac{\lambda r^i (r-1)}{\lambda r^i n(n^2+n-1)} \right\} \]

where
\[ \lambda_n = \prod_{k=2}^{n} \left( \frac{k}{k^2+k+1} \right) \]