Thus
\[ \Omega(\ln(n)) = \ln(n+1) \leq H_n \leq 1 + \ln(n) = O(\ln(n)) \]

\[ \therefore H_n = \Theta(\ln(n)) = \Theta(\lg(n)). \]

Thus
\[ t(n) = \Theta(n \lg(n)) , \]

which is better than the worst case \( \Theta(n^2) \) run time.

Our assumption that all permutations of \( A[1\ldots n] \) are equally likely may not be well founded in practice. We may wish to sort a pre-sorted array, or one that is nearly sorted more often than not.

There are several ways to randomize quicksort to compensate for this.

One way is to simply apply a randomly chosen permutation to \( A[1\ldots n] \) before calling quicksort.
A simpler way is to choose a random element in $A[p...r]$ as pivot, swap that with $A[r]$, then partition as usual.

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

\[\text{Rand Quicksort}(A, p, r)\]
1. \( i \leftarrow p < r \)
2. \( t \leftarrow \text{Rand Partition}(A, p, r) \)
3. \( \text{Rand Quicksort}(A, p, t-1) \)
4. \( \text{Rand Quicksort}(A, t+1, r) \)

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

\[
\text{MaxMin}(A, p, r) \quad \text{(Pre : } p \leq r \text{)}
\]

1.) if \( p = r \)

2.) return \( (A[p], A[p]) \)

3.) \( q \leftarrow \left\lfloor \frac{p + r}{2} \right\rfloor \)

4.) \((m_1, M_1) \leftarrow \text{MaxMin}(A, q + 1, r)\)

5.) \((m_2, M_2) \leftarrow \text{MaxMin}(A, p, q)\)

6.) \text{return }\left( \min(m_1, m_2), \max(M_1, M_2) \right)\)

Let \( T(n) \) denote the number of comparisons performed by MaxMin(A, p, r) on arrays of length \( n \).
\[
T(n) = \begin{cases} 
0 & n = 1 \\
T\left(l \frac{n}{2}\right) + T\left(l \frac{n}{2}\right) + 2 & n \geq 2 
\end{cases}
\]

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 \( \lceil \frac{3n}{2} \rceil - 2 \) comparisons. (Hint: Section 9.1 describes an iterative algorithm to do this.)
Ex. The Selection Problem

The $i^{th}$ order statistic of an array $A[1...n]$ 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 $A[1...n]$ where all elements are distinct, determine the $i^{th}$ order statistic.

One approach would be to sort $A[1...n]$ 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 \textbf{RandPartition}(A, p, r) Splits The
Subarray A[p...r] into Two Subarrays Satisfying
\[ A[p..(q-1)] \leq A[q] \leq A[(q+1)\ldots r] \]

\textbf{RandSelect}(A, p, r, i) \hspace{1cm} (\text{Pre: } 1 \leq i \leq r-p+1)

1.) if \( p = r \)
2.) return \( A[p] \)
3.) \( q \leftarrow \text{RandPartition}(A, p, r) \)
4.) \( k \leftarrow q-p+1 \) \hspace{0.5cm} // \( k \) is length of \( A[p..q] \)
5.) if \( k = i \)
6.) return \( A[q] \)
7.) else if \( i < k \)
8.) return \textbf{RandSelect}(A, p, q-1, i)
9.) else
10.) return \textbf{RandSelect}(A, q+1, r, i-k)

\textbf{RandSelect} is similar in some respects to
both \textbf{QuickSort} and \textbf{BinarySearch}. Like \textbf{QuickSort}
\textbf{RandSelect} randomly splits the subarray \( A[p...r] \) in
order to exploit a good average case run-
time. Like \textbf{BinarySearch} it recurs on
only one subarray. Unlike \textbf{BinarySearch} we
seek not an index, but an \textbf{ARRAY ELEMENT}.\]
Let $t(n)$ denote the average number of array comparisons by $\text{RandSelect}(A, i, n, i)$.

Assume that each permutation of $A[1...n]$ is equally likely, hence the return value of $\text{RandPartition}$ is equally likely to be any of the numbers $1 \leq q \leq n$.

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

$$
t(n) = \frac{\sum_{q=1}^{n} \left( (n-1) + P(i < q) t(q-1) + P(i > q) t(n-q) \right)}{n}
$$

where

$$
P(i < q) = \frac{n-i}{n} \quad \text{and} \quad P(i > q) = \frac{i-1}{n}
$$

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

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

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

Therefore, \( t(n) \) does not depend on \( k \), 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 \( n \).

i.e. Show:

\( \forall n \geq 1 : t(n) \leq 2n \)

Induction Hypothesis: \( \forall 1 \leq n-1 : t(4) \leq 2n \)

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

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

Prove...
Exercise
Find the exact solution to this recurrence.

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

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