>>> A warm - up optional exercise
left to the students as individual reading.

Example of approximate computation of average case (Binary Search)

Carries on the details of computations in the textbook,
Section 1.6 .3 Average - Behavior Analysis, p. 57 - 59.

Results :

$T_{avg}(n) \approx Log2[n+1] - q \approx Log2[n] - q$

where q is the probability of sucessful search.

More precisely,

$T_{avg}(n) = Log2[n] - q + o(1)$

Also, for sufficiently large n (e.g., $n \geq 1000$) and any $0 \leq q \leq 1$:

$Log2[n] - q \leq T_{avg}(n) < Log2[n] + .1 - q$

The following two results will also be tested by a program posted on class website

For every n,
$Log2[n+1] \leq T^{fail}_{avg}(n) < Log2[n+1] + .1$

For every $n > 394$,
$Log2[n+1] - 1 \leq T^{succ}_{avg}(n) < Log2[n+1] - .9$

Binary search, ordered.

Find an item x in an ordered
array I based only of comparisons of x to elements ot I.

$I[0] \leq I[1] \leq I[2] \leq \ldots \leq I[i-1] \leq I[i] \leq \ldots \leq I[n-1]$

size (I) - number of elements to be searched (= n).

Assume size (I) $= n = 2^k - 1$

(for simplicity only;
the general case without the above assumption is analyzed at the end of this file,
with exact computations contained in files AverageCaseOptimalityBinSearch.nb
and AverageCaseOptimalityBinSearch.pdf posted on the class website)

$I[0] \le I[1] \le I[2] \le \ldots \le I[i-1] \le$
$I[i] \le \ldots \le I[n-1]$
^
midpoint $i = \left\lfloor \frac{n-1}{2} \right\rfloor$

(Exercise : Prove it !)

$n = 2^k - 1$

$n + 1 = 2^k$

k = Log2[n + 1];

We know already that k is the possibly largest number of comparisons
made by binary search, both in successful and unsuccessful case.

T (n) -
number of comparisons performed while searching of an entry in an n -
element array I.

Average - case running time for successful search

Let s_t be the number of elements of I that are
found after exactly t comparisons by the binary search.

$s_t = 2^{t-1}$

(Exercise. Prove it by induction on t !)
Let comp (i) be the number of comparisons needed to find I[i].

$T^{succ}_{avg}(n) =$

$$\sum_{i=0}^{n-1} \text{comp}(i) \times \Pr(i) = \sum_{i=0}^{n-1} \text{comp}(i) \times \frac{1}{n} = \frac{1}{n} \sum_{i=0}^{n-1} \text{comp}(i) = \frac{1}{n} \sum_{t=1}^{k} t \times s_t = [$$

because $s_t = 2^{t-1}$; note that $t \ge 1$ for one cannot find anything in 0 comparisons] =

$$\frac{1}{n} \sum_{t=1}^{k} t \times 2^{t-1}$$

(-n Log[2] + Log[1 + n] + n Log[1 + n]) / (n Log[2])

$\frac{1}{n}(n+1)\,\mathrm{Log2}[n+1]-1$

$\mathrm{Factor}\left[\frac{1}{n}(n+1)\,\mathrm{Log2}[n+1]-1\right]$

-((n Log[2] - Log[1 + n] - n Log[1 + n]) / (n Log[2]))

So, they are equal.

$\frac{1}{n}(n+1)\,\mathrm{Log2}[n+1]-1 =$

$= \frac{1}{n} n\,\mathrm{Log2}[n+1]-1+\frac{1}{n}\mathrm{Log2}[n+1] \approx$

$\left[\text{since } \frac{1}{n}\mathrm{Log2}[n+1] \approx 0 \text{ for large } n\right]$

$\approx \mathrm{Log2}[n+1]-1.$

More precisely,

$T^{succ}_{avg}(n) = \mathrm{Log2}[n+1]-1+o(1) =$

$= \mathrm{Log2}[n]-1+o(1).$

[For large n, Log2[n + 1] ≈ Log2[n], for instance,
for n > 1000, Log2[n + 1] < Log2[n × (1.001)] =
Log2[n] + Log2[1.001] = Log2[n] + 0.0014419741739062604`]

For unsuccessful search,

$T^{fail}_{avg}(n) = \mathrm{Log2}[n+1] =$

$= \mathrm{Log2}[n]+o(1).$

Hence, $T_{avg}(n) = q \times T^{succ}_{avg}(n) + (1-q) \times T^{fail}_{avg}(n) =$

$q \times \left(\frac{1}{n}(n+1)\,\mathrm{Log2}[n+1]-1\right) + (1-q) \times \mathrm{Log2}[n+1]$

((1 - q) Log[1 + n]) / Log[2] + q (-1 + ((1 + n) Log[1 + n]) / (n Log[2]))

Expand[%]

$-q + \frac{\mathrm{Log}[1+n]}{\mathrm{Log}[2]}$ + (q Log[1 + n]) / (n Log[2])

$\mathrm{Log2}[n+1] - q + \frac{1}{n} q\,\mathrm{Log2}[n+1]$

So,

$T_{avg}(n) \approx \text{Log2}[n+1] - q$.

More precisely,

$T_{avg}(n) = \text{Log2}[n+1] - q + o(1)$
$= \text{Log2}[n] - q + o(1)$.

[For large n, $\text{Log2}[n+1] \approx \text{Log2}[n]$, for instance,
for $n > 1000$, $\text{Log2}[n+1] < \text{Log2}[n \times (1.001)] =$
$\text{Log2}[n] + \text{Log2}[1.001] = \text{Log2}[n] + 0.0014419741739062604`$]

End of the warm - up exercise << <

Theorem 1.

Binary Search is average - case optimal in the class C of
search algorithms search an ordered array by comparisons of keys.

Input : array E of n elements;
key x.

Output : and index i s.t.
x = E[i]
if found (successful search);
otherwise (unsuccessful search)
a pair of idicies -∞ : 0 if x < E[0]
or
a pair of idices i : i + 1 if E[i] < x < E[i + 1]
or
a pair of idices n - 1 : ∞ if x > E[n - 1]

Note : We do not need assumption $n = 2^k - 1$ here.

Asserted even probability distribution :

All successful search outcomes are equally likely
and
all unsuccessful search outcomes are equally likely.

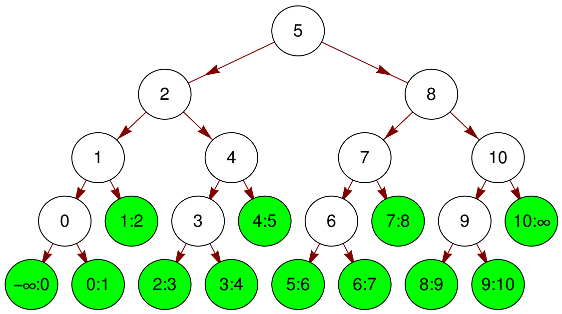
(A case of uneven probability distribution will be tackled in Chapter 10 Section 4.)

The result is proven for any algorithm that searches via decision tree;
in particular, for any algorithm that searches by comparisons of keys.

The computations below require the concept external
path length and internal path length in a binary tree.

Here is an example of a decision tree for Binry Search.

```
TreePlot[{5 → 2, 5 → 8, 2 → 1, 2 → 4, 8 → 7, 8 → 10, 0 → "-∞:0", 1 → 0,
  3 → "2:3", 4 → 3, 6 → "5:6", 7 → 6, 10 → 9, 10 → "10:∞", 0 → "0:1", 1 → "1:2",
  3 → "3:4", 4 → "4:5", 6 → "6:7", 7 → "7:8", 9 → "8:9", 9 → "9:10"},
 VertexLabeling → True, DirectedEdges → True, VertexRenderingFunction →
  ({White, EdgeForm[Black], Disk[#, .3], Black, Text[#2, #1]} &), AspectRatio → 0.55]
```



External nodes are green. These are non-decision nodes that represent the final outcomes of unsuccessful searches. All others are decision nodes or the final outcomes of successful searches.

The average number of comparisons performed by Binary Search with a decision tree T on n nodes is

$$T_{avg}(n) = q \times T_{avg}^{succ}(n) + (1 - q) \times T_{avg}^{fail}(n) = q \times \left(\frac{1}{n} \text{ipl}(T) + 1\right) + (1 - q) \times \left(\left(\text{ipl}(T) + 2n\right) / (n + 1)\right)$$

(in file : http : // csc.csudh.edu / suchenek / CSC401 / Mathematica / AverageCaseSearchLowerBound.nb).

Therefore, all we have to demonstrate that the decision tree of Binary Search has the minimum internal path length.

All levels of every decision tree for Binary Search has all levels that are full except, perhaps, for the last level.

Therefore, the internal path length for each decision tree on n nodes for Binary Search is equal to :

$$\sum_{i=1}^{n} \lfloor \text{Log2}[i] \rfloor.$$

Thus, the said decision tree has the minimum external path length.

End of proof.

Theorem 2.

The average number of comparisons of keys that Binary Search performs on the average while searching an n-element ordered array is equal to :

$$\left(1 + \frac{q}{n}\right) (\text{Log2}[n + 1] + \epsilon[n + 1]) - q \approx$$

$$\approx \text{Log2}[n + 1] + \epsilon[n + 1] - q,$$

where

$\beta[x\_] := 1 + x - 2^x$

$\theta[x\_] := \lceil x \rceil - x$

$\epsilon[x\_] := \beta[\theta[\text{Log2}[x]]]$

Proof. Let T be a deccision tree on n nodes for Binary Search.
As we shown in the proof of Theorem 1,

$$T_{avg}(n) = q \times \left(\frac{\text{ipl}(T)}{n} + 1\right) + (1 - q) \times \left((\text{ipl}(T) + 2n) / (n + 1)\right).$$

Since T has ther minimum internal path length, we conclude

$$T_{avg}(n) = q \times \left(\frac{1}{n}\text{ipl}_{min}(n) + 1\right) + (1 - q) \times \left((\text{ipl}_{min}(n) + 2n) / (n + 1)\right).$$

Applying computations from file file : http : // csc.csudh.edu / suchenek / CSC401 / Mathematica / AverageCaseSearchLowerBound.nb, we conclude that

$$T_{avg}(n) = \left(1 + \frac{q}{n}\right)(\text{Log2}[n + 1] + \epsilon[n + 1]) - q \approx$$

$$\approx \text{Log2}[n + 1] + \epsilon[n + 1] - q$$

End of proof.

Note.
Function ϵ oscillates between 0 and
$1 - \lg e + \lg \lg e \approx 0.08607133205593432$

```
Plot[{ϵ[n], 0.0860713, 1}, {n, 0.9, 2.2}, PlotTheme → "Classic",
 PlotStyle → {Thickness[Small]}, AspectRatio → 0.6,
 Ticks → {{1, 1.38629, 2, 22, 64, 177, 355, 512, 710, 1024}, {0, 0.0860713, 1}}]
```