Question: Try to proof that sorting algorithms based on comparison never faster than $O(n \log n)$.

sorting n distinct elements = $n!$ possible permutations.

A binary decision tree with height h has at most 2^k leaves

$$\therefore 2^h \geq n!$$
$$h \geq \log_2 (n!)$$

Applying Stirling's Approximation

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$
$$\log(n!) \approx \log_2 \left(\frac{n}{e}\right)^n = n\log n - n\log e$$
$$\therefore \log_2 (n!) = \Theta(n \log n)$$

The minimum height of the decision tree is $h = \Omega(n\log n)$

so the worst-case number of comparisons is $\Omega(n\log n)$

Question: Write down the recursive time function and analyze it by master theorem.

$$T(n) = T\left(\frac{n}{2}\right) + O(1)$$

$a=1$, $b=2$, $f(n) = O(1)$

compare with $n^{\log_b a} = n^0 = 1$

since $f(n) = \Theta(n^{\log_b a})$, so $T(n) = \Theta(\log n)$

Question: Rewrite above algorithm by while loop.

```
template <typename T>

int find(const T sorted_data[], int n, const T key) {

   int left = 0, right = n - 1;

   while (left <= right) {

      int mid = left + (right - left) / 2;

      if (sorted_data[mid] == key)

         return mid;

      else if (sorted_data[mid] > key)

         right = mid - 1;

      else

         left = mid + 1;

   }

   return n; // not found

}
```

Question: what's the time complexity of above algorithm (in worst case)?

The recursive algorithm used to solve f(x) = val on a continuous interval [a, b] uses binary search and the intermediate value theorem.
In each step, the range [a, b] is halved. The recursion continues until the absolute error between f(mid) and val is less than or equal to epi.

Thus, the maximum number of recursive calls required to narrow the interval to size ≤ epi is:

$$T(n) = O\left(\log_2 \frac{b-a}{\text{epi}}\right)$$

However, if the function f itself is expensive, and we assume the cost of each call to f(x) is O(f_time), the overall time complexity becomes:

$$O\left(f_{\text{time}} \cdot \log_2 \frac{b-a}{\text{epi}}\right)$$

Worst-case time complexity:

$$O\left(\log \frac{1}{\text{epi}}\right)$$

$$O\left(f_{\text{time}} \cdot \log \frac{1}{\text{epi}}\right) \quad \text{if } f(x) \text{ is expensive}$$

Question: What is the best k for searching key in n elements?

The time complexity of k-ary search is:

$$T(n) = T(n/k) + O(k) \Rightarrow O(k \cdot \log_k n)$$

To minimize this complexity, we can analyze the function $k \cdot \log_k n$,
Theoretically, the best value of k is when $k \approx e \approx 2.718$,
so the optimal integer value of k is:

$$k{=}3$$