1. The worst case time complexity of the standard quicksort algorithm is $\Omega(n^2)$, but by a suitable pivot selection one can achieve $O(n \log n)$ time. Explain how to achieve

(a) average time complexity $O(n \log n)$
(b) expected time complexity $O(n \log n)$
(c) worst case time complexity $O(n \log n)$.

Solution:
A bad pivot is one that is among the smallest or largest items in its partition. For example, we might define a pivot to be bad if it is among the first $\epsilon n$ or the last $\epsilon n$ items for some constant $0 < \epsilon < 1/2$. Quicksort achieves $O(n \log n)$ expected time if the probability of a good pivot is at least $p$ for some constant $p > 0$.

(a) The average complexity means averaging over all initial permutations of the input. Then any choice of pivot (that is not intentionally bad) is likely to be good. For example, we can choose the first item as the pivot.

(b) The expected complexity means the expectation over the random choices of a randomized quicksort. For example, choosing the pivot randomly achieves $O(n \log n)$ expected time for any input.

(c) To achieve worst case complexity $O(n \log n)$, we have to guarantee a good choice of a pivot. There is a linear time deterministic algorithm for computing the median of a set using comparisons. Median, of course, is the optimal choice of the pivot.

2. A full binary tree is a binary tree where every node is either a leaf or has two children. Show that every full binary tree with $n$ leaves has exactly $2n - 1$ nodes. Hint: Use induction.

Solution: Let $n = 1$. Then the tree only consists of the root thus contains $1 = 2 \cdot 1 - 1$ nodes, so the formula works for $n = 1$. Assume now that the formula holds for all $k < n$. By $n_L$ and $n_R$ denote the number of leaves in the left and right subtree of the root. From inductive assumption the left (right) subtree of the root contains $2n_L - 1$ ($2n_R - 1$) nodes, thus the total number of nodes in the tree is $(2n_L - 1) + (2n_R - 1) + 1 = 2(n_L + n_R) - 1 = 2n - 1$.

There also exists a simple direct proof. Consider a sequence of $n - 1$ trim operations performed on a tree with $n$ leaves. A trim operation removes an arbitrary pair of leaves that have a common parent. Each time the operation is performed, the number of leaves is reduced by one hence after $n - 1$ steps the tree only consists of a single root node. Each operation removed two nodes hence altogether $2(n - 1)$ nodes were removed. Therefore, by taking into account the remaining root, the tree must have initially contained $2(n - 1) + 1 = 2n - 1$ nodes.
3. Transform the following nondeterministic finite automaton into a deterministic finite automaton.

![NFA Diagram]

**Solution:** The states of the NFA are sets of the states of the DFA. The initial NFA state is \( \{A, B, D\} \) consisting of the DFA states reachable from the initial state \( A \) via \( \varepsilon \)-transitions. The NFA states reachable from the initial state and the transitions from them are given by the following table:

<table>
<thead>
<tr>
<th>State</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( {A, B, D} )</td>
<td>( {C} )</td>
<td>( {A, B, D} )</td>
</tr>
<tr>
<td>( {C} )</td>
<td>( {B, D} )</td>
<td>( \emptyset )</td>
</tr>
<tr>
<td>( {B, D} )</td>
<td>( {C} )</td>
<td>( {A, B, D} )</td>
</tr>
<tr>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
</tr>
</tbody>
</table>

The state diagram for the NFA is:

![NFA Diagram]

The state \( \emptyset \) could be removed if we do not require the transition function to be total. The states \( \{A, B, D\} \) and \( \{B, D\} \) could be merged since both are accept states and have the same transitions.
4. Write a program that computes the ten most frequent words in the english text file
http://pizzachili.dcc.uchile.cl/texts/nlang/english.50MB.gz
Note that the file is compressed with gzip and must be decompressed before processing.
You may consider the space character to be the only word separator, i.e., words can contain punctuation, newlines etc., but using a more sophisticated word parsing method is allowed too.
Be prepared to show the code and the output of your program.

**Solution:** Excluding any non-alphanumeric characters, the words and their frequencies are:

<table>
<thead>
<tr>
<th>Word</th>
<th>Count</th>
<th>Rel. freq.</th>
</tr>
</thead>
<tbody>
<tr>
<td>the</td>
<td>563792</td>
<td>5.9%</td>
</tr>
<tr>
<td>and</td>
<td>386061</td>
<td>4.0%</td>
</tr>
<tr>
<td>of</td>
<td>302218</td>
<td>3.1%</td>
</tr>
<tr>
<td>to</td>
<td>250761</td>
<td>2.6%</td>
</tr>
<tr>
<td>a</td>
<td>166341</td>
<td>1.7%</td>
</tr>
<tr>
<td>in</td>
<td>139458</td>
<td>1.5%</td>
</tr>
<tr>
<td>I</td>
<td>111195</td>
<td>1.2%</td>
</tr>
<tr>
<td>he</td>
<td>106906</td>
<td>1.1%</td>
</tr>
<tr>
<td>that</td>
<td>99089</td>
<td>1.0%</td>
</tr>
<tr>
<td>his</td>
<td>93871</td>
<td>0.98%</td>
</tr>
</tbody>
</table>

The C++ code to compute the frequencies is available at the course home page. The code uses std::unordered_map for storing counts for the distinct words.

An alternative solution is to use the following command line:

```
tr -cs '[:alnum:]' '
' < english.50MB | sed -e '/^$/d'
| sort | uniq -c | sort -nr | head -10
```

This uses sorting to group identical words instead of a data structure for counters.

5. The Fibonacci numbers are defined using the recurrence

\[
\begin{align*}
f_0 &= 0 \\
f_1 &= 1 \\
f_n &= f_{n-1} + f_{n-2}
\end{align*}
\]

Write two programs that compute \( f_n \) for a given \( n \) using

(a) a recursive function based directly on the recurrence
(b) dynamic programming.

What is the largest \( n \) for which you can compute \( f_n \) using program (a) in less than five seconds?

**Solution:** The C++ code to compute the Fibonacci numbers using both methods is available at the course home page. When compiled with `-O3`, the recursive implementation can go up to about \( n = 45 \) in five seconds (depending on the machine).