Problem 1. In our description of the Josephus problem we have participants standing in a circle and one sword. One participant has a sword. In one move, the participant with the sword kills the next participant in the counterclockwise direction, the cycle closes and the sword is passed to the next participant in the counterclockwise direction. Our goal is to determine who will survive if \( n \) participants start and we number/name them from 0 to \( n - 1 \). Assume that participants number 0 is the one who has the sword at the beginning.

(a) Show that if \( n \) is a power of 2, participant 0 is the survivor. Use mathematical induction.

**Answer.** Basis: \( n = 2^0 \). Participant 0 is the survivor. Inductive step, \( n = 2^{k+1} \). For \( i = 0, 1, \ldots, 2^k - 2 \) participant \( 2i \) kills participant \( 2i + 1 \) and passes the sword to participant \( 2i + 2 \); then participant \( 2 \times (2^k - 1) = 2^{k+1} - 2 \) kills \( 2^{k+1} - 1 \) and passes the sword back to participant 0. Thus the sword is back in the hands of 0 and the number of participants is \( 2^k \), so we can use the inductive hypothesis.

(b) Show that if \( n = p + r \) where \( p \) is a power of 2 and \( r < p \) then the survivor is participant \( 2r \). Hint: who gets the sword when the number of participants drops to \( p \)?

**Answer.** Actually, we can prove it by induction on \( r \). Basis: \( r = 0 \), proven in (a). Inductive step, \( r > 1 \). Participant 0 kills participant 1 and passes the sword to 2. We renumber the participants: \( i = 2, 3, \ldots, n - 1 \) gets new number \( i - 2 \), 0 gets new number \( n - 2 \). Now we have the situation from the inductive hypothesis: \( n' = p + (r - 1) \) participants, so participant \( 2(r - 1) \) survives, and it is not \( n - 2 \) (otherwise \( 2(r - 1) = n - 2 \equiv 2r = n = p + r \equiv r = p \)). The survivor \( 2r - 2 \) is re-numbered participant \( 2r \).
Problem 2. Recursive mergesort can be more efficient than iterative mergesort but copying before sorting is a major drain of running time. We use notation \((A, i, j)\) for array sector \(A[i], A[i+1], \ldots, A[j-1]\).

(a) Write code of a function that merges sorted sectors \((A, a, b)\) and \((A, c, d)\) onto \((A, c, d + b - a)\), assuming \(a \leq b \leq c \leq d\). It should move each array entry from these sectors exactly once.

```c
void merge(A, a, b, c, d)
{
    int i, j, k;
    i = b-1;
    j = d-1;
    k = j+b-a;
    while (i >= a && j >= c)
    while (i >= a)
        A[k--] = A[i--];
    while (j >= c)
        A[k--] = A[j--];
}
```

(b) Write recursive code that sorts fragment \((A, a, b)\) so that the result is in fragment \((A, b-1, 2b - a - 1)\). It should move array entries only using the calls to the merging function.

```c
void mergesort(A, a, b)
{
    int m;
    if (b-a < 2)
        return;
    m = (a+b)/2;
    mergesort(A, m, b);
    mergesort(A, a, m);
    merge(A, m-1, m-1+m-a, b-1, b-1+b-m);
}
```

(c) Explain how large array \(A\) should be to sort \((A, 0, n)\) using this method.

**Answer.** Array fragment \((A, 0, n)\) is moved during sorting to \((A, n-1, 2 \cdot n - 1)\), so we need \(A[2n-1]\).
**Problem 3.** You have as the input a connected undirected graph.

(a) Can you always give a single direction to each edge so the graph remains strongly connected? Show a counterexample.

**Answer.** Two node graph with one edge is the smallest counterexample.

(b) Sketch an algorithm to identify edges that make the graph not-strongly connected if we give them a single direction.

**Answer.** We can execute depth first search in our graph by calling \(dfs\_rec(r)\) from some node \(r\). Each node \(u\) will be an argument to recursive function call \(dfs\_rec(u)\). During that call we inspect nodes adjacent to \(u\); when we inspect \(v\) we have three cases:

i. Node \(v\) was not visited, we make call \(dfs\_rec(v)\) and we direct edge \(\{u, v\}\) as \((u, v)\), this is so-called **tree edge**.

ii. Node \(v\) was visited and it is the parent of \(v\), i.e. call \(dfs\_rec(u)\) is made from \(dfs\_rec(v)\); we do not do anything, because edge \(\{u, v\}\) as \((u, v)\) is already directed as \((v, u)\).

iii. Node \(v\) was visited and it is not the parent of \(v\). We direct edge \(\{u, v\}\) as \((u, v)\), this is so-called **back edge**.

Now we have a directed graph in which each edge that was before undirected has a single direction. In this graph we can find strongly connected components. Clearly, to make this graph strongly connected it suffices to do the following: if it has edge \((u, v)\) and \(u, v\) are in different strongly connected components, we add directed edge \((v, u)\). So the question is if it was necessary.

Suppose not, i.e. we could give directions to other edges in such a way that we could use only one of the edges \((u, v)\) and \((v, u)\) and the graph would still be strongly connected.

Because each node is visited during \(dfs\_rec(r)\), for each node \(w\) we have a path of directed tree edges from \(r\) to \(w\). If we created edge \((u, v)\) and nodes \(u, v\) are in different strongly connected components, such tree path from \(r\) to \(v\) uses edge \((u, v)\). Let \(A\) be the set of nodes on the tree path from \(r\) to \(u\), and \(B\) be the set of nodes visited during \(dfs\_rec(v)\). If there is an edge between \(A\) and \(B\) different than \(\{u, v\}\) then it would be encountered during \(dfs\_rec(v)\) and directed from \(B\) to \(A\), we would get a cycle including edge \((u, v)\) and \((u, v)\) would be in the same strongly connected component. If there is only one such edge, then every path from \(r\) to \(B\) and every path from \(B\) to \(r\) must use \(\{u, v\}\) so this edge must indeed have two directions.
**Problem 4.** Order the following functions so when \( f_i \) is followed by \( f_j \) then \( f_i = O(f_j) \).
Indicate the cases when \( f_i = \Theta(f_j) \).

\[
\begin{align*}
    f_1(n) &= (\ln n)^n \\
    f_2(n) &= (\ln n)! \\
    f_3(n) &= (n^{\ln n}) \\
    f_4(n) &= 5^n \\
    f_5(n) &= n \\
    f_6(n) &= 2^{\ln n}
\end{align*}
\]

Note: To receive partial credit for a not entirely correct answer, you must show your work neatly, and your work should not simply consist of evaluating the functions at a large value of \( \tilde{n} \) in the hope that this value is large enough to reveal the correct answer.

**Answer.**

\[
\begin{align*}
    f_6(n) &= (e^{\ln^2})^{\ln n} = (e^{\ln n})^{\ln 2} = n^{\ln^2} < n^{0.7} \ll n = f_5(n) \\
    f_2(n) &= (\ln n)! = (\ln n/e)^{\ln n} = e^{(\ln \ln n - 1)^{\ln n}} = n^{(\ln \ln n - 1)}, \text{ thus } f_5(n) \ll f_2(n) \ll f_3(n) \\
    f_3(n) &= n^{\ln n} = (e^{\ln n})^{\ln n} = e^{(\ln^2 n) n} \ll e^{n \ln 5} = (e^{\ln 5})^n = f_4(n)
\end{align*}
\]

Thus the ordering is \( f_6(n) \ll f_5(n) \ll f_2(n) \ll f_3(n) \ll f_4(n) \ll f_1(n) \)