PROBLEM 1

[1] \( \lg \lg n < \ln n, \lg n < (\lg n)^2 < \sqrt{n} < n < n \lg n < n^{1+\epsilon} < n^2, n^2 + \lg n < n^3 < n - n^3 + 7n^5 < 2^n, 2^{n-1} < e^n < n! \)

[2]

\[ f(n) = O(g(n)) \iff \exists c_1, \forall n > n_1, f(n) \leq c_1 \cdot g(n) \]
\[ g(n) = O(h(n)) \iff \exists c_2, \forall n > n_2, g(n) \leq c_2 \cdot h(n) \]
\[ \forall n > \max(n_1, n_2), f(n) \leq c_1 \cdot g(n) \leq c_1 \cdot c_2 \cdot h(n) \]

Thus
\[ \exists c = c_1 \cdot c_2, \forall n > n_0 = \max(n_1, n_2), f(n) \leq c \cdot h(n) \]

Thus
\[ f(n) = O(h(n)) \]

PROBLEM 2

There are two ways to solve this problem.

[1] Using a heap.
1) Put the heads of \( k \) lists into a heap - \( O(k \log k) \).
2) Remove the top of the heap (minimum element) and add it to the output list.
3) Remove the head mentioned in step 2 from the corresponding list and add the new head of the list to heap - \( O(\log k) \) because of the heap properties.
4) Repeat steps 2 and 3 until we do not have any elements left in the lists - need to iterate over \( n \) elements.

Overall time is \( O(n \log k) \)

[2] Merge 2 lists at a time. Assume \( k_0 = k \)

1) Run merge algorithm from merge sort for \( \frac{k_i}{2} \) pairs of lists. Each takes \( O(\frac{n}{k_i}) \). So overall this step takes \( O(n) \).
Now we have new \( k_{i+1} = \frac{k_i}{2} \) lists.
2) Repeat steps 1 for \( i = 0, ..., \log k \).
PROBLEM 3

[1] Use an array to store the bits. $BitFlip(i)$ is just a look up of $A[i]$ and changing the bit. $NearestOne(i)$ is a linear scan of the array and is $O(n)$.

[2] Maintain an array of tuples $(x_i, j)$, where $x_i$ is the bit at index $i$ and $j$ is the index of the nearest one of $x_i$.

$BitFlip(i)$ – Go to $A[i]$, flip the bit $x_i$ in the tuple and fix the look up tables for all values. If $x_i$ was one before, we need to not only find its nearest one index but also update all those locations $j$ in the array which had its index in the $NearestOne(j)$ field. If $x_i$ was zero, then its $NearestOne(i) = i$ now, and also it may have become the nearest one of other indices in the array so a linear scan is needed to find them and update their table. This operation thus takes $O(n)$.

$NearestOne(i)$ – Just go to $A[i]$ and output the value maintained in its $NearestOne$ field in $O(1)$ time.

PROBLEM 4

Build a balanced binary search tree on the indices of 1’s.

$BitFlip(i)$ – If $i$ is present in the tree, delete it. If it is not present in the tree, insert it. Both the search, and insert/delete take $O(\log n)$ time.

$NearestOne(i)$ – If $i$ is present in the tree, return $i$. If it is not, insert it temporarily and find $x = successor(i)$ and $y = predecessor(i)$. Return min $(x, y)$ and delete $i$ from the tree. These operations take $O(\log n)$ time as well.

PROBLEM 5

[1] Index of maximum element in the array = $k \mod n$ (assuming first index is 1).


Initialise $start = 0$, $end = n - 1$, $mid = start + end/2$

1) If $start < mid < end$ : return $end$.
2) If $start > mid < end$ : $end = mid$, $mid = start + end/2$ (recurse on left half).
3) If $start < mid > end$ : $start = mid$, $mid = start + end/2$ (recurse on right half).
4) If $start > mid > end$ : return $start$.
5) Repeat 1-4 till maximum value is returned.