Solution 3

1. Question 7-6a (page 163) **Fuzzy sorting of intervals**

   Suppose we have n closed intervals given by the array I, that is $I = [a_1, b_1], [a_2, b_2], \ldots, [a_n, b_n]$. The algorithm for fuzzy sorting is as follows, `Fuzzysort` takes as input the interval array, beginning index, p and the end index r of the sub-array which is to be sorted, in other words we sort $I[p \ldots r]$. When `Fuzzysort` returns, the array $I[p \ldots r]$ will be sorted.

   `Fuzzysort` uses a procedure called `Fuzzypartition`, which again takes as input I, p, r, considers $I[r]$ as the pivot, and rearranges I as follows: `Fuzzypartition` returns a two-numbered value (q_l, q_r). The rearrangement will be such that $I[p \ldots q_l]$ and $I[q_r \ldots r]$ have to be sorted, and $I[q_{l+1} \ldots q_{r-1}]$ is in the right place.

   Note that $I[k] = [a_k, b_k]$

   `Fuzzysort` (I, p, r)

   (a) if $p < r$ then
   (b) $\quad (q_l, q_r) \leftarrow$ `Fuzzypartition` (I, p, r)
   (c) $\quad$ `Fuzzysort` (I, p, q_l)
   (d) $\quad$ `Fuzzysort` (I, q_r, r)

   `Fuzzypartition` (I, p, r)

   (a) $x \leftarrow a_r$. Remember that $I[r] = [a_r, b_r]$.
   (b) $i \leftarrow (p - 1)$
   (c) for $j \leftarrow p$ to $(r - 1)$
   (d) $\quad$ if $a_j \leq x$ then
   (e) $\quad\quad i \leftarrow (i + 1)$
   (f) $\quad\quad$ exchange $I[i] \leftrightarrow I[j]$
   (g) exchange $I[i + 1] \leftrightarrow I[r]$
   (h) $q \leftarrow p - 1$
   (i) for $k \leftarrow p$ to i

(j) if $b_k \geq x$, then

(k) $q \leftarrow (q+1)$

(l) exchange $I[k] \leftrightarrow I[q]$

(m) for $l \rightarrow p$ to q

(n) exchange $I[l] \leftrightarrow I[i-l+p]$

(o) return $(i-q+p-1, i+2)$

Description of the algorithm: In `Fuzzypartition`, steps (a) through (g) are similar to quick sort. At the end of step (g), we will have the following property:

$a_k \leq a_{i+1}$ for $p \leq k \leq i$
$a_k > a_{i+1}$, for $i+2 \leq k \leq r$

In steps (h) through (l), we find elements in $I[p \ldots i]$ such that they overlap with $I[i+1]$, that is whose b value is $\geq a_{i+1}$, and these elements will be in positions $p \ldots q$ in I.

In steps (m) through (n), we swap the elements from $p \ldots q$, so that they are close to the pivot, that is we swap $I[p]$ with $I[i]$, $I[p+1]$ with $I[i-1]$ and so on.

Therefore at the end of (n) steps, we will have the intervals such that $I[p \ldots i-q+p-1]$ have both their a value and b value less than a_{i+1}. For $I[i-q+p \ldots i+1]$ will have overlapping intervals (a_{i+1} is value in all the intervals), and $I[i+2 \ldots r]$ have their a and b value greater than a_{i+1}.

The above gives the proof of correctness of the algorithm as well.

In the worst case, it takes $O(n^2)$ as for quick sort. In best case, it performs better than $O(n\ logn)$, because after one iteration, the left and the right interval arrays after `Fuzzypartition` can be arbitrarily small. So in the best case, it takes $\Theta(n)$ time. Also we can see that more the intervals overlap, the algorithm tends to become better.

2. Question 7.2-4 **Insertion sort is faster than quick sort for an almost sorted array**

   Insertion sort works by taking an element x, comparing it with the previous element and so on, until it finds a correct position for x. Therefore

when we consider an almost sorted array, the number of comparisons it has to make for any element will be very small. (For a fully sorted list, insertion sort takes $\Theta(n)$ steps).

Quick sort works very bad for an almost sorted array, when we choose the pivot as one of the end elements. This is because when we choose, say the last element, as the pivot, only very few elements will be greater than the pivot, whereas most of the elements will be less than the pivot. This is a worst case input for quick sort. (For a completely sorted list and the pivot is an end element, quick sort takes $\Theta(n^2)$ steps).

3. Question 8.1-2 **Asymptotically tight bounds for $lg(n!)$ without using Stirling's approximation**

   Proving an upper bound, i.e, $lg\ (n\ !) = O\ (n\ lg\ n)$

   $$\begin{aligned} \Sigma_{k=1}^{n} lg\ k &\leq \Sigma_{k=1}^{n} lg\ n \\ &= \text{n lg n} \end{aligned}$$

   Proving a lower bound, i.e, $lg\ (n\ !) = \Omega\ (n\ lg\ n)$

   $$\begin{aligned} \Sigma_{k=1}^{n} lg\ k &= \Sigma_{k=1}^{n/2} lg\ k + \Sigma_{k=n/2+1}^{n} lg\ k \\ &\geq \Sigma_{k=1}^{n/2} lg\ 1 + \Sigma_{k=n/2+1}^{n} lg\ n/2 \\ &= 0 + n/2\ lg\ n/2 \end{aligned}$$

   Thus we get that $lg\ (n!)$ is $O(n\ lg\ n)$ and $\Omega(n\ lg\ n)$. Hence we get that $lg\ (n!) = \Theta(n\ lg\ n)$.

4. Question 9.3-8 (page 193) **Finding the median of two sorted arrays in $O(lg\ n)$**

   Given two sorted lists A, B, to find the median. The algorithm is `Median` (A, B) and is given below. Note that the algorithm works by recursively finding median in *two sorted arrays of same size.*

   (a) if no. of elements in $A = 2$ (the no. of elements in B will also be 2), or no. of elements in $A = 1$ (the no. of elements in B will also be 1), find the median of the 4 (or 2) elements in constant time.

   `Median` (A, B)

(b) a_{mid} be the $\lfloor \frac{n+1}{2} \rfloor$ element in A, and b_{mid} be the $\lceil \frac{n+1}{2} \rceil$ element in B.

(c) If $a_{mid} \geq b_{mid}$, then consider $A' = A[1 \ldots \lfloor \frac{n+1}{2} \rfloor]$, $B' = [\lceil \frac{n+1}{2} \rceil \ldots n]$.

(d) If $a_{mid} < b_{mid}$, then consider $A' = A[\lfloor \frac{n+1}{2} \rfloor \ldots n]$, $B' = B[1 \ldots \lceil \frac{n+1}{2} \rceil]$

(e) `Median` (A', B')

Description of the algorithm The algorithm works as follows. We will find the "lower median" (the lower median of n elements is the $\lfloor (n+1)/2 \rfloor$ th element. Consider three cases:

Case 1: The number of elements in A is odd, and so the number of elements in B is also odd. In this case there exists only one median. By comparing the two medians, we find, $(n-1)/2$ elements which are smaller than $(n+1)/2$ elements in each array. Hence these $(n-1)/2$ elements have at least $(n+1)$ elements $\geq$ them, so we can safely discard these $(n-1)/2$ elements. Similarly we find $(n-1)/2$ elements which have $(n+1)$ elements $\leq$ them, and these can also be discarded. Therefore now we have to find median in these two sorted arrays, each of size $(n+1)/2$

Case 2: The number of elements in A is even, and $a_{mid} \geq b_{mid}$. In this case, we know $n/2$ elements in A which are $\geq n/2$ elements in A and $n/2+1$ elements in B and so they can be discarded. Similarly we know $n/2$ elements in B which are $\leq n/2+1$ elements in A and $n/2$ elements in B and they can be discarded. So now, we have to find the median in the two sorted arrays each of size $n/2$.

Case 3: The number of elements in A is even, and $a_{mid} < b_{mid}$. In this case, we know $n/2-1$ elements in A which are $\leq n/2+1$ elements in A and $n/2$ elements in B and so they can be discarded. Similarly we know $n/2-1$ elements in B which are $\geq n/2$ elements in A and $n/2+1$ elements in B and they can be discarded. So now, we have to find the median in the two sorted arrays each of size $n/2+1$.

The proof of correctness follows from above. The running time is $O(lg\ n)$, this also follows from above.

5. **Max. No. of regions produced by n circles**

We will go over this in class. You can show by induction that the max. no. of new regions produced by circle i is $2(i-1)$.

Hence you get the max. no. of regions produced by n circles as $2 + \Sigma_{i=2}^{n} 2(i-1) = n(n-1) + 2$.