Problem:

Input: Sorted lists $L_1, L_2 \ldots L_k$ (K sorted list)

Output: Combined sorted lists.

Time: $O(n \log K)$

(a) Let the lists be $L_1, L_2, L_3 \ldots L_k$ which are already sorted in which K is a size of lists. $(1 \leq k \leq n)$

(1) Divide: Divide the K sorted lists $L_1, L_2, \ldots L_k$ with $1 \leq K \leq n$. Each list L_i has size n_i.

(2) Conquer: First, Compare the first elements in the lists and insert the smallest element into new list L.

(a) if $L_1[i] \leq L_2[j]$
then $L.add(L_1[i])$

(b) Now increment the i in $L_1[i]$ and compare recursively.

(c) If $L_2[j] \leq L_1[i]$
then $L.add(L_2[j])$

(3) Combine: The elements in the two lists are combined into a single list L_3 which are sorted. This process takes $n_1 + n_2$ comparisions.

(4) Similarly L_3 and L_4, L_5 and L_6 .... L_{k-1} and L_k are combined into another lists which have n_3+n_4 and n_5+n_6 and - - - $n_{k-1}+n_k$ comparisons.

(5) Continue this process until all the lists are completely traversed

(6) Each level needs $O(n)$ comparisons and the lists are generated until k in $\log k$ levels. So time is $\underline{O(n \log k)}$

Problem 2:

2(a) Given array $A[1 \cdots n]$

If $i<j$ and $A[i] > A[j]$ then $(A[i], A[j])$ is called an inversion of A.

Given array : $\{14, 12, 17, 11, 19\} \leftarrow A[i]$

1 2 3 4 5 $\leftarrow i$

The inversions are ①	(14, 12)	$\because$ $14 > 12$ & $1 < 2$
②	(14, 11)	$\because$ $14 > 11$ & $1 < 4$
③	(12, 11)	$\because$ $12 > 11$ & $2 < 4$
④	(17, 11)	$\because$ $17 > 11$ & $3 < 4$

2(b) The array which is in a descending order has the most inversions.

Array : $A[n, n-1, n-2, \cdots, 3, 2, 1]$

The number of inversions is

$$= (n-1) + (n-2) + \cdots + 2 + 1$$

$$= \frac{(n-1)(n-1+1)}{2}$$

$$= \boxed{\frac{n(n-1)}{2}}$$

2(c) Input : Array : $A[1 \cdots n]$
Output : number of inversions.
Time : $O(n \log n)$.

This divide and conquer algorithm is similar to merge sort. The difference is that we have to calculate the number of inversions while merging the data.

Algorithm :

① If the size of an array (n) is less than or equal to 1 ($n \leq 1$), return 0. else we have to do the divide and conquer algorithm.

② Divide : Divide the array into two subarrays $A[1 \cdots n] = A[1 \cdots n/2] + A[n/2+1 \cdots n]$

③ Conquer : Now, compute the number of recursions in the two sub arrays & sort the two sub arrays $A[1 \cdots n/2]$ & $A[n/2+1 \cdots n]$. Let the number of inversions in $A[1 \cdots n/2] = I_1$ and the number of inversions in $A[n/2+1 \cdots n] = I_2$.

④ Combine: Now, combine the two sorted lists into one sorted list and compute the number of inversions in the merged sorted set.
inversion pair = $(A[i], A[j])$ s.t $A[i] > A[j]$ such that $1 \le i \le n/2$ and $n/2+1 \le j \le n$.
The number of iterations in the merged sorted list is $= I_3$.

⑤ Eventually, return number of inversion pairs = $\boxed{I_1 + I_2 + I_3}$.

⑥ As we implemented the merge sort algorithm here, the running time of entire algorithm is $\boxed{O(n \log n)}$.

3(a) $T(n) = 2 \cdot T(n/2) + n^4$; $T(n/2) = 2 \cdot T(n/4) + (n/2)^4$

$= 2\left(2 \cdot T(n/4) + (n/2)^4\right) + n^4$

$= 4 \cdot T(n/4) + \frac{n^4}{2^3} + n^4$; $T(n/4) = 2 \cdot T(n/8) + (n/4)^4$

$= 4\left(2 \cdot T(n/8) + (n/4)^4\right) + \frac{n^4}{2^3} + n^4$

$= 8 \cdot T(n/8) + \frac{n^4}{4^3} + \frac{n^4}{2^3} + \frac{n^4}{1^3}$

$= 2^3 \cdot T\left(n/2^3\right) + \frac{n^4}{4^3} + \frac{n^4}{2^3} + \frac{n^4}{1^3}$

$= 2^K \cdot T\left(n/2^K\right) + \left(\frac{1}{2^3}\right)^{K-1} \cdot n^4 + \cdots + \left(\frac{1}{2^3}\right)^2 \cdot n^4 + \left(\frac{1}{2^3}\right)^1 \cdot n^4 + \left(\frac{1}{2^3}\right)^0 n^4$

$= \boxed{2^K \cdot T\left(n/2^K\right)} + n^4 \left[\left(\frac{1}{2^3}\right)^{K-1} + \cdots + \left(\frac{1}{2^3}\right)^2 + \frac{1}{2^3} + 1\right]$

↳ This is in a form of Geometric progression $\left(\frac{a}{1-r}\right)$

G.P $= \frac{1}{1-1/2^3} = \boxed{8/7}$

$\frac{n}{2^K} = 1 \Rightarrow \boxed{n = 2^K}$

~~T(1)~~

$= n \cdot T(1) + n^4(8/7)$

So, the time complexity is $\boxed{O(n^4)}$

3(b) $T(n) = 4 \cdot T(\frac{n}{2}) + n$; $T(n/2) = 4 \cdot T(n/4) + n/2$

$= 4(4T(n/4) + n/2) + n$

$= 16\,T(n/4) + 2n + n$; $T(n/4) = 4 \cdot T(n/8) + n/4$

$= 16(4 \cdot T(n/8) + n/4) + 2n + n$

$= 64 \cdot T(n/8) + 4n + 2n + n$

$= 4^{K+1} \cdot T(n/2^{K+1}) + [2^K + \cdots + 2^2 + 2^1 + 2^0]\,n$

$= 4^{K+1} \cdot T(n/2^{K+1}) + [2^K - 1]\,n$

$O(1) \Rightarrow n/2^{K+1} = 1$

$n = 2^{K+1}$

$\boxed{\frac{n}{2} = 2^K}$

$= 2^{2K+2} + n(n/2 - 1)$

$= 4(2^K)^2 + n^2$

$= 4n^2 + n^2$

$= \boxed{n^2}$

$O(n^2)$

3(c) $T(n) = 2 \cdot T(n/2) + n \log n$; $T(n/2) = T(n/4) + n/2 \log n/2$

$= 2 \cdot (2T(n/4) + n/2 \log n/2) + n \log n$

$= 4T(n/4) + n \log n/2 + n \log n$; $T(n/4) = 2T(n/8) + n/4 \log n/4$

$= 4(2T(n/8) + n/4 \log n/4) + n \log n/2 + n \log n$

$= 8T(n/8) + n \log n/4 + n \log n/2 + n \log n$

$= 2^{K+1} T(n/2^{K+1}) + \cdots + n \log n/2^K + \cdots + n \log n/2^1 + n \log n$

$= \frac{n}{2^{K+1}} = 1 \Rightarrow \boxed{n = 2^{K+1}} \Rightarrow \log n/2 = K$

$= n\,O(1) + n[K \log n - [K \log 2 + (K-1) \log 2 + \cdots + 2 + 1]]$

$= n + n \cdot \log n/2 \cdot \log n - [K + (K-1) + \cdots + 2 + 1]$

$= n + n (\log n)^2 - \frac{K(K-1)}{2}$

$= n + n (\log n)^2 - (\log n/2)^2$

$= \boxed{O(n \log^2 n)}$

3(d) $T(n) = T(2/3 \cdot n) + n$;

$= T((2/3)^2 n) + 2/3 n + n$

$= T((2/3)^3 n) + (2/3)^2 n + (2/3) n + n$

$= T[(2/3)^{K+1} n] + [(2/3)^K n + \cdots + (2/3)^2 n + (2/3) n + n]$

$= T[(2/3)^{K+1} n] + n[1 + (2/3)^1 + (2/3)^2 + \cdots + (2/3)^K]$

$= T[(2/3)^{K+1} n] + n((2/3)^K - 1)$

$\Rightarrow (2/3)^{K+1} n = 1 \Rightarrow (2/3)^K n = \frac{3}{2} \Rightarrow \boxed{(2/3)^K = 3/2n}$

$= O(1) + n\left(\frac{3}{2n} - 1\right)$

$= 3/2 - n$

$= \boxed{O(n)}$

Problem (4):

Input: Array $A[1 \cdots n]$

Output: pair of indices (i, j) with $i < j$ s.t. $A[j] - A[i]$ is max of all pairs.

Time: $O(n \log n)$

(1) Divide: Divide the array into two sub arrays $A[1 \cdots n/2]$ & $A[n/2+1 \cdots n]$.
A_1 A_2

(2) Conquer: (a) if (i in A_1 & j in A_1)
then solve the A_1 recursively

(b) if (i in A_2 & j in A_2)
then solve the A_2 recursively

(c) if (i in A_1 & j in A_2)
find an index i with $1 \le i \le n/2$
find an index j with $n/2+1 \le j \le n$
s.t. $A[j] - A[i]$ is maximum.
This pair can be obtained by taking smallest element in A_1 and largest element in A_2.

(3) So, finally by using the divide and conquer approach, we can solve this problem.

(a) and (b) $\rightarrow$ Time $= T(n/2)$ each.

(c) $\rightarrow$ Time $= O(n)$

So, $T(n) = 2 \cdot T(n/2) + O(n)$

Solving recurrence

$T(n) = 2 \cdot T(n/2) + n$

$= 2(2 \cdot T(n/4) + n) + n$

$= 4T(n/4) + 2n + n$

$= 4(2 \cdot T(n/8) + n) + 2n + n$

$= 8T(n/8) + 4n + 2n + n$

$= 2^{K+1} T(n/2^{K+1}) + (2^K n + \cdots + 2^1 n + 2^0 n)$

$\Rightarrow \frac{n}{2^{K+1}} = 1 \Rightarrow n = 2^{K+1} \Rightarrow n/2 = 2^K \Rightarrow \log n/2 =$

$= n T(1) + n(2^0 + 2^1 + 2^2 + \cdots + 2^K)$

$= n T(1) + n(2^K - 1)$

$= n + n 2^K$

$T(n) = O(n \log n)$