Ans 1)

a. It would have to be $n^2 - 10n + 2 \leq c \cdot n^2$, because its $f(x) = O(g(x))$.

   by the Big-Oh definition, $T(n)$ is $O(n^2)$ if $T(n) \leq c \cdot n^2$ for some $n \geq n_0$

   if $n^2 - 10n + 2 \leq c \cdot n^2$

   then $1 - \frac{10}{n} + \frac{2}{n*n} \leq c$

   Therefore, the Big-Oh condition holds for $n \geq n_0 = 1$ and $c \geq -7$ (= 1 - 10 + 2)

   => **Valid**

b. It would have to be $2^{n*n} \leq c2^{2n}$, because its $f(x) = O(g(x))$.

   Here $f(x) = 2^{n*n}$ and $g(x) = 2^{2n}$

   => $\log(2^{n*n}) \leq \log(c.2^{2n})$

   => $n^2\log 2 \leq \log c + 2n\log 2$

   => $n^2 \leq \log c + 2n$

   Therefore, the Big-Oh condition cannot hold (the left side of the latter inequality is growing exponentially, so that there is no such constant factor c)

   => **Not valid**

c. It would have to be $n\log_2(n) \leq c \cdot n\log_{10}(n)$, because its $f(x) = O(g(x))$.

   $\log_2(n)$ and $\log_{10}(n)$ are asymptotically equal.

   Therefore, the Big-Oh condition holds.

   => **valid**

d. It would have to be $n\log_2(n) \leq c \cdot n$, because its $f(x) = O(g(x))$.

   Here $f(x) = n\log_2(n)$ and $g(x) = n$

   => $n\log_2(n) \leq c.n$

   => $\log_2(n) \leq c$

   Therefore, the Big-Oh condition cannot hold (the left side of the latter inequality is growing logarithmically, so that there is no such constant factor c)

   => **Not valid**

e. It would have to be $n2^n \leq c2^{2n}$, because its $f(x) = O(g(x))$.

   Here $f(x) = n2^n$ and $g(x) = 2^{2n}$

   => $\log n + n\log 2 \leq \log c + 2n\log 2$

=> $\log n + n \leq \log c + 2n$

Therefore, the Big-Oh condition cannot hold (the left side of the latter inequality is growing logarithmically, so that there is no such constant factor c)

=> **Not valid**

Ans 2) **<u>Algo 1:</u>**

First for loop i = 1 to n:

=> $1 + 2 + 3 \dots + n = O(n)$

Second for loop j = n to 1:

Third for loop k = j to 1:

=> $n(n + (n-1) + (n-2) + (n-3) \dots + 1) + (n-1)((n-1) + (n-2) + (n-3) \dots + 1)$

=> $O(n^2)$

Overall Time complexity:

=> $O(n) + O(n^2)$

=> $\mathbf{O(n^2)}$

<u>Algo 2:</u>

______For i = n to 1 and i = i/2

=> $n + n/2 + n/4 + n/8 \dots\dots n/2^k$

=> $n/2^k = 1$ => $n = 2^k$

=> $\log_2 n = k$

Overall time complexity:

=> $\mathbf{O(\log_2 n)}$

Ans 3) Given an array A of integers,

<u>Algorithm</u>

majority(A):

1. if len(A) == 0: return Null
2. if len(A) == 1: return A[0]
3. Half = len(A) / 2
4. left = majority(A[start, half])
5. right = majority(A[half+1, end])
6. if left == right: return left
7. if A.count(left) > half: return left

8. if A.count(right) > half: return right
9. Else: return Null

Recurrence relation : $T(n) = 2T(n/2) + O(n)$

$a = 2$, $b = 2$, $f(n) = O(n^{\log 2} * \log_2 n)$

By master theorem, time complexity is:

$T(n) =$ **$O(n\mathrm{Log}_2 n)$**

Ans 4) We compare the given recurrence relation with $T(n) = aT(n/b) + \theta(n^k \log^p n)$.

Then, we have-

$a = 3, b = 2, k = 2, p = 0$

Now, $a = 3$ and $b^k = 2^2 = 4$.

Clearly, $a < b^k$. So, we follow case-03.

Since $p = 0$, so we have-

$\Rightarrow T(n) = \theta(n^k \log^p n)$

$\Rightarrow T(n) = \theta(n^2 \log^0 n)$

Thus, **$T(n) = \theta(n^2)$**

Ans 5) $T(n) = 2T(n/2) + n \log2(n)$, where $T(n) = O(1)$

Determine cost of each level-

- Cost of level-0 = $n(\log(n/2^0) = O(n\log(n))$
- Cost of level-1 = $\log(n/2^1) = O(\log(n/2))$
- Cost of level-2 = $\log(n/2^2) = O(\log(n/4))$
- Cost of level-logn = $\log(n/2^{\log n}) = O(\log(n/2^{\log n})$

$\Rightarrow n((\log n - \log 2^0) + (\log n - \log 2^1) + (\log n - \log 2^2) + ... + \log n - (\log 2^{\log n}))$

$\Rightarrow n((\log n - 0) + (\log n - 1) + (\log n - 2) + ... + (\log n - \log n))$

$\Rightarrow n(\log n + (\log n - 1) + (\log n - 2) + ... + 2 + 1 + 0)$

Because of Gauss's sum: $0 + 1 + 2 + 3 + ... + k = k(k+1) / 2$. That means that this sum simplifies down to

=> n (log n)(1 + log n) / 2
=> **$O(n \log^2 n)$.**

Ans 6) Karatsuba's algorithm has $O(n\log_2(3))$ complexity.

when *a increases*, the number of subproblems determines the asymptotic running time.

By master theorem, in worst time complexity of the algorithm will be,

=> $T(n) = \Theta(n^{\log_b a}) = \Theta(n^{\log_4 a})$

=> $\Theta(n^{\log_2 \sqrt{a}})$

To make T(n) smaller than Karatuba's algorithm, $n^{\log\sqrt{a}}$ must be smaller than $n^{\log 3}$

=> $n^{\log\sqrt{a}} < n^{\log 3}$

=> $\log\sqrt{a} < \log 3$

=> $\sqrt{a} < 3$

=> $a < 9$

The largest integer value of a for which Professor Caesar's algorithm would be asymptotically faster than $O(n\log_2(3))$ is **8**.

.