Question 1.

1.a) Repeated substitution method

Expand $T(n)$

$= 2\left[2T(n/4) + (n/2)^2 \log(n/2)\right] + n^2 \log n$

$= 4T(n/4) + 2n^2(\log n - 1)$

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

$\Rightarrow T(n) = n^2(\log n)^2 + O(n^2 \log n)$

Base case:

$T(1) = 1^2(\log 1)^2 = 1$

Inductive step:

we assum that the formula is True for $n/2$,

we want to prove it's True for n

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

$= 2\left[(n/2)^2(\log(n/2))^2 + O((n/2)^2 \log(n/2))\right] + n^2 \log n$

$= n^2(\log n - 1)^2 + O(n^2 \log n) + n^2 \log n$

$\therefore T(n) = n^2(\log n)^2 + O(n^2 \log n)$

1.b) Master theorem

the recurrence fits the master theorem when $a=2, b=2,$

$f(n) = n^2 \log n$

comparing $n^{\log_b a}$ with $f(n) \Rightarrow n^{\log_2 2} = n \Rightarrow n^2 \log n$ is asymptotically larger than n

$\therefore$ we can't apply the m.t directly

2.a) RSM

Expand $T(n)$

$T(n) = T(n-1) + \log n$

$= [T(n-2) + \log(n-1)] + \log n$

$= [T(n-3) + \log(n-2) + \log(n-1)] + \log n$

... and so on

$\therefore = T(1) + \sum_{i=1}^{n} \log i \Rightarrow O(n \log n)$

$\Rightarrow T(n) = C + O(n \log n)$

2.b) MT

the recurrence doesn't fit the master theorem

3.a) RSM

Expand $T(n)$

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

$= 2[2T(n/4) + 3(n/2)^2] + 3n^2$

$= 4T(n/4) + 3n^2 + 3n^2$

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

... and so on

$\therefore T(n) = O(n^2 \log n)$

4.a) RSM

Expand $T(n)$

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

$= 3[3T(n/4) + n/2] + n$

$= 9T(n/4) + 3/2\, n + n$

$= 3^k T(n/2^k)\ K\, n$

... and so on

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

4.b) MT

the reccurence fits MT when $a=3$, $b=2$, $f(n)=n$

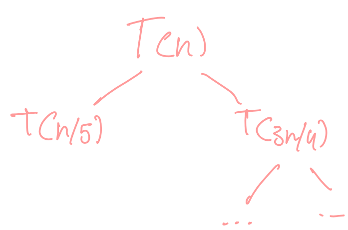
comparing $n^{\log_b a}$ with $f(n)$

$n^{\log_2 3} \simeq n^{1.58} \Rightarrow n$ is linear $\Rightarrow$ $f(n)$ is linear $\Rightarrow$ $f(n) < n^{1.58}$

we use case 1 of the master theorem

Question 2.

1.



cn cost

cn cost

* logarithmic depth ($\log n$)

branching factor = 2

in this case: $T(n) = O(n \log n)$

2.

$T(n)$ — n cost

$T(n/2)$ — n cost

...

* logarithmic depth
branching factor = 1

in this case: $T(n) = O(n \log n)$

Question 3.

a) input: a list of coefficients $[a_n, a_{n-1}, \ldots a_1, a_0]$, a value x_0
output: the value of the polynomial $p(x_0)$

set result to 0
for i in range of (n, -1, -1)
    set result to result * x_0 + a[i]
return result

b) the algorithm has time complexity of $O(n)$

c) no it is not possible

Question 4.

a) function findSecondLargest(A, start, end)

if end - start == 1

base case   return min(A[start], A[end]), max(A[start], A[end]) else

mid = (start + end)/2

leftmin, leftmax = findSecondLargest(A, start, mid)

rightmin, rightmax = findSecondLargest(A, mid+1, end)

if leftmax > rightmax

secondmax = max(leftmax, rightmin)

else

secondmax = max(rightmax, leftmin)

return min(leftmin, rightmin), secondmax

b) time complexity

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

$\therefore O(n\log n)$

Question 5.

a) the algorithm should recursively divide list into smaller sublist until each sublist contains only 1 element, then compute the sum, count, and average

b) function ComputeAverage(A, start, end)

if start == end return A[start]

else

mid = (start + end)/2

leftsum = computeAverage(A, start, mid)
rightsum = computeAverage(A, mid+1, end)
return (leftsum + rightsum)

c) $T(n) = 2T(n/2) + O(1)$

$\therefore O(n \log n)$