Date ____ 20 ____

Algorithms Assignment 1
21K-3153

(Q1) a

	9	19	14	29	13	4	23	10	37

step=2, k=19

	9	19	14	29	13	4	23	10	37
J--	19	9	14	29	13	4	23	10	37

step=3, k=14

	19	9	9	29	13	4	23	10	37
J--	19	14	9	29	13	4	23	10	37

step=4, k=29

	19	14	9	9	13	4	23	10	37
J--	19	M	14	9	13	4	23	10	37
J--	19	19	14	9	13	4	23	10	37
J--	29	19	14	9	13	4	23	10	37

~~step=5, k=13~~

step=5, k=13

	29	19	14	9	9	4	23	10	37
J--	29	19	14	13	9	4	23	10	37

step=6, k=4

	29	19	14	13	9	4	23	10	37

step 7, k=23

	29	19	14	13	9	4	4	10	37
J--	29	19	14	13	9	9	4	10	37
J--	29	19	14	13	13	9	4	10	37
J--	29	19	14	14	13	9	4	10	37
J--	29	19	19	14	13	9	4	10	37
J--	29	23	19	14	13	9	4	10	37

Date ____ 20__

step = 8, k = 10

	29	23	19	14	13	9	4	4	37
0--	29	23	19	14	13	9	9	4	37
5--	29	23	19	14	13	10	9	4	37

step = 9, k = 37

	29	23	19	14	13	10	9	4	4
5--	29	23	19	14	13	10	9	9	4
4--	29	23	19	14	13	10	10	9	4
3--	29	23	19	14	13	13	10	9	4
2--	29	23	19	14	14	13	10	9	4
1--	29	23	19	19	14	13	10	9	4
0--	29	23	23	19	14	13	10	9	4
5--	29	29	23	19	14	13	10	9	4
4--	37	29	23	19	14	13	10	9	4

Why in insertion sort $O(n^2)$:

Insertion sort fundamentally has two loops. The outer for loop and inner loop from swapping and comparing.
The outer for loop will always run, thus it is $O(n)$.
The inner loop will only run when the array is not sorted.

The inner loop does work proportional to number of swaps.
If an array is sorted in reverse e.g. 50 40 30 20 10, the element 50 will be compared against every element and so on.
Thus in the average case, inner loop will also work for $O(n)$

$O(n) \times O(n) = n^2 O(n^2)$ → for insertion sort average and worst case

Date ____ 20__

Loop invariance:

① condition = subarray A[0 .. i-1] is always sorted

Initialization:

Before first iteration (step=2), the array A[0 .. 1] will have only one element and it will be sorted.

Maintenance:

As the algorithm processes, insertion sort arranges the array in an asc/desc order by comparing and swapping. Thus A[0 .. i-1] will always be sorted since insertion sort always arranges array from beginning (0) to (i).

Termination:

Once insertion sort has finished, the array A[0 .. n] will always be sorted.

Date ____ 20 ____

Loop invariance:

- condition = subarray A[0 .. i-1] is always sorted

Initialization:

Before first iteration (step = 2), then array A[0 .. 1] will have only one element and it will be sorted.

Maintenance:

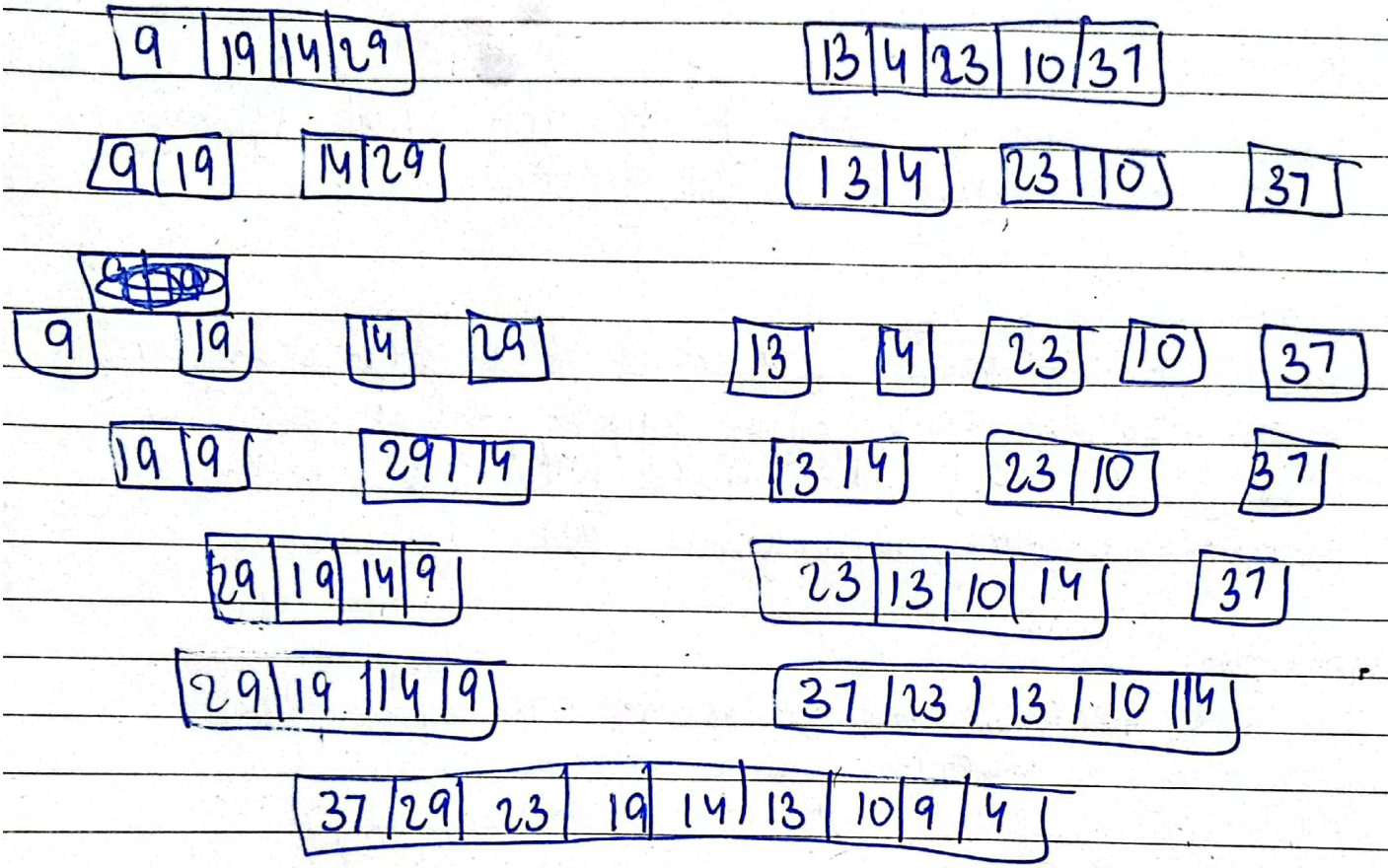
As the algorithm processes, insertion sort arrange the array in an asc/desc order by comparing and swapping. Thus A[0 .. i-1] will always be sorted since insertion sort always arranges array from beginning (0) to (i).

Termination:

Once insertion sort has finished, the array A[0 .. n] will always be sorted.

Date ____ 20__

Q2) 9 19 14 29 13 4 23 10 37



(b) divided into 3 parts

① merge (left, mid1) → T(n/3)
② merge (mid1, mid2) → T(n/3)
③ merge (mid2, right) → T(n/3)

merge sort:

merge () → 3(T(n/3)) = 3T(n/3) + O(n)
merge ()
merge ()

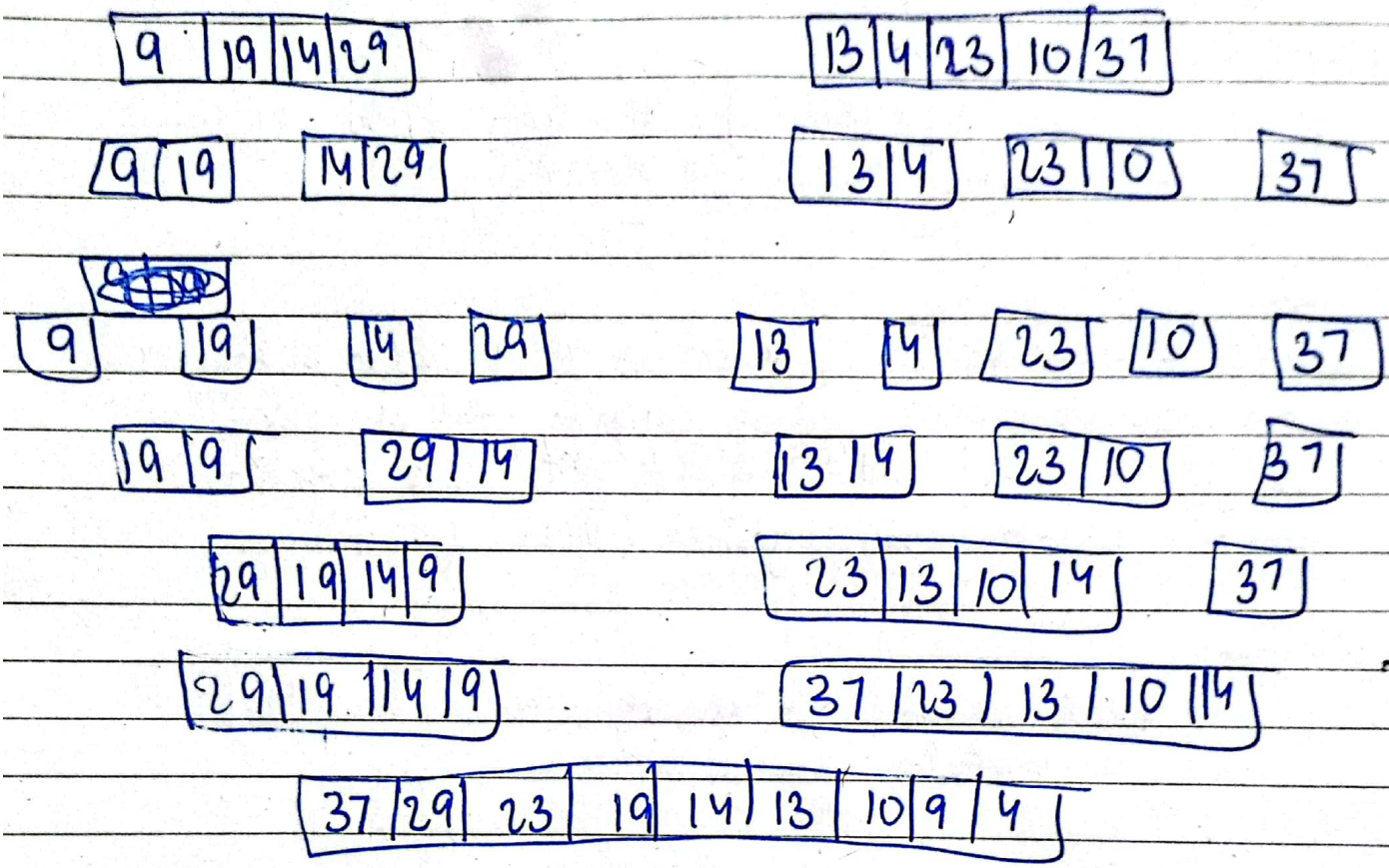
Master theorem:
3T(n/3) + cn

a = 3 b = 3 d = 1

$a = b^d \Rightarrow n^d \log n \Rightarrow n \log n$

Date 20

Q2) 9 19 14 29 13 4 23 10 37



(b) divided into 3 parts

① merge (left, mid1) → T(n/3)
② merge (mid1, mid2) → T(n/3)
③ merge (mid2, right) → T(n/3)

merge sort:

merge () → 3(T(n/3)) = 3T(n/3) + O(n)
merge ()
merge ()

Master theorem:
3T(n/3) + cn

$a = 3 \quad b = 3 \quad d = 1$

$a = b^d \Rightarrow n^d \log n \Rightarrow n \log n$

$3 = 3^1$

Date ____ 20__

(Q3)

i j

9 19 14 29 13 4 23 10 37

pivot as the last element : 37

p=37

~~19 9 14 29 13 4 23 10 37~~

p=37 37 19 14 29 13 4 23 10 9

p=9 37 19 14 29 13 23 10 9 4

p=10 (left+ right 9) same

p=4 (right) same

p=23 (left+ right 10) 37 29 23 14 10 19 13 10 9 4

p=13 37 29 23 14 19 13 10 9 4

p=14 37 29 23 19 14 13 10 9 4

Invariance:

condition: after every ~~partition~~ partition call

① pivot in it's correct place ② elements less than pivot on the left side

③ elements greater on the pivot on the right side (assuming lowest to highest)

initialization: ~~pivot has~~ after the first partition call, elements greater than pivot will be to it's right and lesser will be to it's left. before partition call, there will be no elements between i and j, thus invariant will be held.

maintenance: as algo progresses, array [left] will always be < pivot and array [right] will be ~~great~~ > pivot. This can only happen if pivot is in it's correct place, then all three hold up

Date _____ 20___

(b) Termination:

the final array will be sorted. Pivot will always be in its correct position, all elements to its left will be smaller. All elements to its right will be larger as it is a sorted array

~~A[left] < pivot < A[right]~~

$A[\text{left}] < \text{pivot} < A[\text{right}]$ (pivot)

(Q4) (b) $O(n^2)$

1. Firstly, sort the array using any $O(n^2)$ sorting algorithm, e.g bubble sort.
2. create two subarrays, empty, of n size
3. place 1st element in array one, last element in array 2. maintain a separate sum variables for each array.
4. maintain a pointer at the second last element $(2n-2)$.
5. add the largest possible elements to the array with the lowest sum until its sum is greater than the other array and until the array size $< n$.
6. once an element has been added to the subarray, decrement pointer to largest element remaining in the original array.
7. do until both arrays reach n size

Date 20

e.g 2 7 12 15 21 27 33 42

(sorted using bubble sort)

arr1

2

sum = 2

2 33

sum = 35

2 33 27

sum = 62

2 33 27 15

sum = 77

arr 2

42

sum = 42

42 21

sum = 63

42 21 12 7

sum = 82

balanced as possible

$O(n^2)$ + $O(n)$ → then $O(n^2)$

↓ bubble sort

⑤ $O(n \log n)$:

① divide the array into halves, and keep going recursively

② make 2 subarrays

③ during the process of merging back, place highest and lowest elements in separate arrays

④ maintain their sum variable, and place highest possible element in array with the lowest sum until sum > sum of other array or array size = n.

⑤ place all elements into 2 subarrays

$O(n \log n)$ + $O(n)$ → then $O(n \log n)$

↓ merging

Date 20

e.g 2 7 12 15 21 27 33 42

(sorted using bubble sort)

arr1

2

sum: 2

2 33

sum = 35

2 33 27

sum = 62

2 33 27 15

sum = 77

arr 2

42

sum = 42

42 21

sum = 63

42 21 12 7

sum = 82

as balanced as possible

$O(n^2)$ + $O(n)$ → thus $O(n^2)$

↓ bubble sort

⑤ $O(n \log n)$:

① divide the array into halves, and keep going recursively

② make 2 subarrays

③ during the process of merging back, place highest and lowest elements in separate arrays

④ maintain their sum variable, and place highest possible element in array with the lowest sum until sum > sum of other array or array size = n.

⑤ place all elements into 2 subarrays

$O(n \log n)$ + $O(n)$ → thus $O(n \log n)$

↓ merging

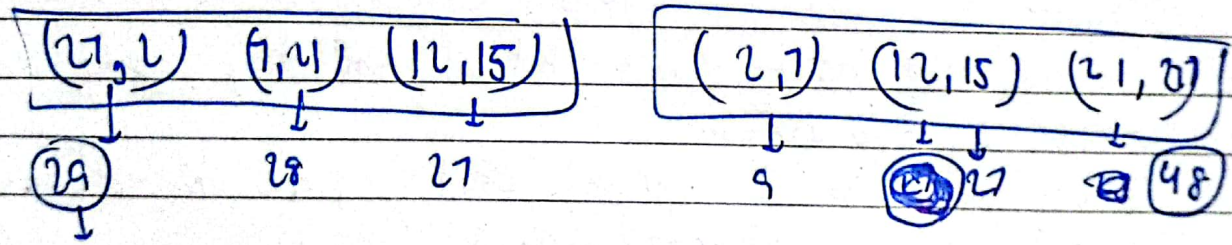
Date ____ 20__

(Q5) ① Use an nlogn sorting algorithm, like merge sort to sort the arrays

② Use two pointers, one at the beginning, one at the end, to make pairs. decrements last pointer, increment first pointer. This ensures that sum are a minimum as possible.

③ Once a pair have been made, the pair with the sum of highest element + lowest element will always be the lowest maximum.

2 7 12 15 21 27



lowest maximum

sorted using merge sort

$O(n \log n) + O(n)$

↓

thus $O(n \log n)$

Date _____ 20___

(Q6) a) $n^2+8n+15 = O(n^2)$. c ? n_0 = ?

$n^2+8n^2+15n^2 <= cn^2$

$24n^2 <= c(n^2)$

$c = 24$ $n_0 >= 1$

b) $5n^2\log n + 2n^2 = O(n^2\log n)$. c = ? n_0 ?

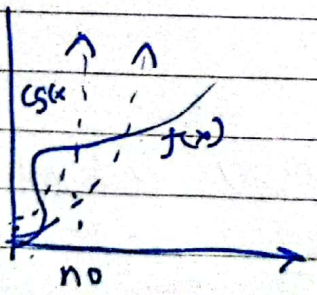
$5n^2\log n + 2n\log n <= c(n^2\log n)$

$7n^2\log n <= cn^2\log n$

$7 = c$ $n_0 >= 1$

(Q7) a speed :- operation relative to input (n)

Asymptotic bounds :- O, Ω, Θ

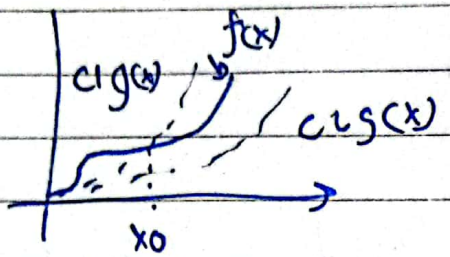


Big(O):
f(x) belongs to O(g(x)) if there is a c, for which $f(x) <= cg(x)$ beyond a certain point x_0 and for all $x >= x_0$

On the other hand : (Big Ω)
if $f(x) >= cg(x)$ for all $x >= x_0$,
then f(x) belongs to $O(\Omega)$.

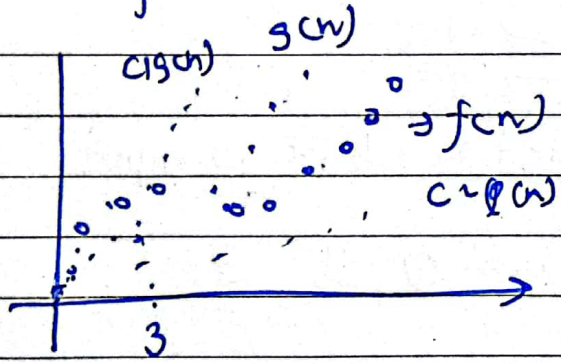
Big Θ:

if g(x) bounds f(x) both ~~lower~~ greater and lesser then f(x) does not towards sense $c_1 g(x)$ & $c_2 g(x)$.
Then it belongs to Θ(g(x))



Real vs Integer functions:

Algorithms are integer function.
n for integer variables



Big(Θ) (integer):

f(n) belongs to O(g(n)) if $c>0$ & $n_0>0$ such that
$f(n) <= cg(n)$
for $n >= n_0$

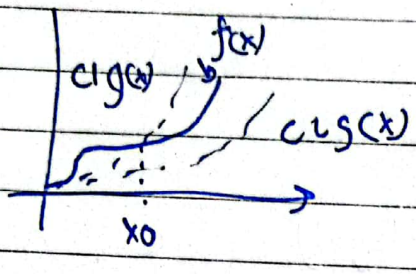
f(n) belongs to Ω(g(n)) if $c>0$ & $n_0>0$ such $f(n) >= g(n)$
for $n >= n_0$

f(n) belongs to Θ(g(n)) if $c_1, c_2 > 0$ & $n_0 > 0$
such that $f(n) <= c_1 g(n)$ & $f(n) >= c_2 g(n)$
for $n >= n_0$

Date_____20___

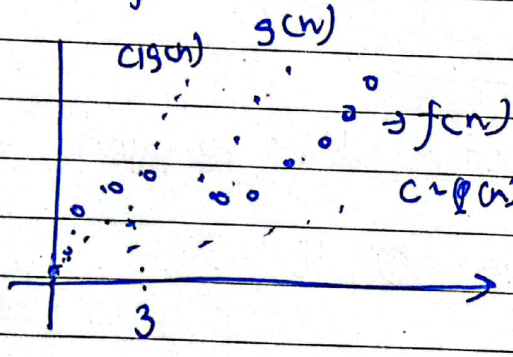
Big Θ:

if g(x) bounds f(x) both greater and lesser then f(x) does not bounds same $c_1 g(x)$ & $c_2 g(x)$.
Then it belongs to $\Theta(g(x))$



Real vs Integer functions:

Algorithms are integer function.
n for integer variables



Big(Θ) (integer):

f(n) belongs to $O(g(n))$ if $c>0$ & $n_0>0$ such that
$f(n) <= cg(n)$
for $n >= n_0$

$f(n)$ belongs to $\Omega(g(n))$ if $c>0$ & $n_0>0$ such $f(n) >= cg(n)$
for $n >= n_0$

$f(n)$ belongs to $\Theta(g(n))$ if $c_1, c_2 > 0$ & $n_0 > 0$
such that $f(n) <= c_1 g(n)$ & $f(n) >= c_2 g(n)$
for $n >= n_0$

Date _____ 20___

Q8 Master theorem:

(a) $T(n) = \sqrt{2}\, T(n/2) + \log n$

$\log n$ is now polynomial thus, function is non polynomial so master theorem cannot be applied

(b) $T(n) = 64\, T(n/8) - n^2 \log n$

~~$a=64 \quad b=8 \quad d = n^2 \log n$~~

$n^{\log_b a} \log^k n$

~~$n^{\log_8 64} \log^{64} n$~~

~~$n^{2\log_8 8} \log^8 n$~~

$n^{\log_8 64} \log^1 n$

$n^{2\log_8 8}$

$n^2 \log^1 n \rightarrow$ case 4

$$T(n) \in \Theta(n^2 \log^2 n)$$

(c) $T(n) = 16\, T(n/4) + n!$

$n!$ is not a polynomial expression because it is not raised to any power, thus $T(n)$ is not a polynomial expression and Master theorem cannot be applied

Date ____ 20 ____

(Q8) Master theorem:

(a) $T(n) = \sqrt{2}\, T(n/2) + \log n$

$\log n$ is now polynomial thus, function is non polynomial so master theorem cannot be applied

(b) $T(n) = 64\, T(n/8) - n^2 \log n$

~~$a=64 \quad b=8 \quad d = n^2 \log n$~~

$n^{\log_b a} \log^k n$

$n^{\log_8 64} \log^2 n$

$n^{2 \log_8 8}$

$n^2 \log^1 n \rightarrow$ case 4

$$T(n) \in \Theta(n^2 \log^2 n)$$

(c) $T(n) = 16\, T(n/4) + n!$

$n!$ is not a polynomial expression because it is not raised to any power, thus $T(n)$ is not a polynomial expression and Master theorem cannot be applied

$2 \cdot 2^{k-1}$

Date _____ 20___

(Q9):

(a) $T(n) = n\log n + 2T(\frac{n}{2})$

$n\log n + 2(\frac{n}{2}\log\frac{n}{2} + 2T(\frac{n}{4}))$

$n\log n + n\log\frac{n}{2} + 4T(n/4)$

$n\log\frac{n^2}{2} + 4T(\frac{n}{4})$

$4(\frac{n}{4}\log\frac{n}{4} + 2T(\frac{n}{16}))$

$n\log\frac{n^2}{2} + n\log\frac{n}{4} + 8T(\frac{n}{16})$

$n\log\frac{n^3}{8} + 8T(\frac{n}{16})$

$n(3\log n - 3) + 8T(\frac{n}{16})$

$n(2\log n - 2) + 4T(\frac{n}{8})$

$2k = a$

$n(k\log n - c) + 2^k T(\frac{n}{2^k})$

$2^k = n$

$k = \log_2 n$

$n(\log^2 n - c) + n(1)$

$n\log^2 n - c + n$

$T(n) \in n\log^2 n$

Date ____ 20__

(b) $T(n) = 8T\left(\frac{n}{2}\right) + n^2$

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

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

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

$3n^2 + 64T\left(\frac{n}{4}\right) \to k$

2^k

$3n^2 + 64\left[8T\left(\frac{n}{8}\right) + \left(\frac{n}{4}\right)^2 + 8T\left(\frac{n}{8}\right)\right]$

$3n^2 + 4n^2 + 512\,T\left(\frac{n}{8}\right)$

$7n^2 + 512\,T\left(\frac{n}{8}\right)$

2^{k-1} $\to$ $2^{k-1}n^2 + 8^k T\left(\frac{n}{k}\right)$

$k = n$

$2^{n-1}n^2 + 8^k T\left(\frac{n}{k}\right)$ $\quad k=n$

$8^n T(1)$

$\left(2^{n-1}\right) n^2 + 8^n$

$\left(\frac{2^n}{2}\right) n^2 + 8^n$

$\frac{1}{2}n^3 + 8^n$

$n^3 + 8^n \to T(n) \in O(n^3)$

Date ____ 20 ____

(10) (a) $n(f(n)) + O(f(n)) = \Theta(f(n))$

$a > b$ $\quad a \leq b$ $\quad = b$

$f(n) = n^2 + n^3$

$n^3 + n^2 = n^2$

worst case

thus false

(b) $f(n) + O(f(n)) = \Theta(f(n))$ take $f(n) = n^2$

better than n^2

$n^2 + n = n^2$

thus true

(c) $f(n) = n^2$

if $f(n) <= O(n^2)$ and $f(n) >= O(n)$

$g(n)$ $\quad h(n)$

$O(n^2)$ $\quad n^2 + n >= n^2$ ✓

thus true