T(0) =

$r^0 b + a\frac{1-r^0}{1-r}$ which is b, so the formula is true when $n = 0$. Now assum

$$T(n-1) = r^{n-1}b + a\frac{1-r^{n-1}}{1-r}.$$

Then we have

$$\begin{aligned}
T(n) &= rT(n-1) + a \\
&= r\left(r^{n-1}b + a\frac{1-r^{n-1}}{1-r}\right) + a \\
&= r^n b + \frac{ar - ar^n}{1-r} + a \\
&= r^n b + \frac{ar - ar^n + a - ar}{1-r} \\
&= r^n b + a\frac{1-r^n}{1-r}.
\end{aligned}$$

Question 4: (continued)

c. Use Decision tree and binary tree basic ideas to prove the following theorem:

"Every comparison based sorting algorithm has, for each n, running on input of size n, a worst case in which its running time is Ω(nlog n)".

How does comparison based sorting achieves Ω(nlog n) compared to O(n^2) running time of inversion bound sorts like insertion sort and bubble sort? Explain your answer.

- A decision tree has $n!$ leaves (n: input size)
- A binary tree has max 2^h leaves (h: tree height)
- We know that a decision tree is $\subseteq$ binary tree

$\therefore$ No. of leaves in decision tree is $\leq$ that of binary tree

$$\therefore n! \leq 2^h \quad \text{take Log:}$$

$$\log(n!) \leq \log 2^h$$

From Stirling's approximation: $n! > \left(\frac{n}{e}\right)^n$

$$\therefore \log\left(\frac{n}{e}\right)^n \leq h$$

$$\therefore h \geq n \log n - n \log e \quad \therefore \boxed{h \text{ is } \Omega(n.\log n)}$$

* Since "h" is the height, with represents the max. depth of any leave, which also represents the no. of comparisons in a decision tree, then no. of comparisons is $\Omega(n \log n)$.

- Inversion sorting makes no. of comparisons $\geq$ no. of inversions in the array. No. of inversions in array with size(n) is $\binom{n}{2} = \frac{n(n-1)}{2}$, which is $O(n^2)$. That's why inversion sorting has running time of $O(n^2)$.

?

→ (Divide & conquer)

40 -0.5

Continue of 4a:

Array before sorting:

0	1	2	3
4_a	4_b	4_c	4_d

Array after sorting:

4_c	4_d	4_b	4_a

Obviously, elements were not kept in their places ~~althoug~~ although they are equal. That's why quick sort is not stable.

11.5

Question 4: 13 points (4 + 4 + 5)

a. Show how Quicksort is not stable by using in-place random partitioning algorithm and the following 4 numbers {4a, 4b, 4c, 4d} (show all steps).

1- Pick Pivot : 4b

2- Swap it with last element:

A[]				
	4_a	4_d	4_c	4_b
	i		j	

3- Set pointers i, j as shown above.

4- A[i] & A[j] are swapped because ~~they are both ≤~~ A[i] ≥ pivot and A[j] ≤ pivot, thus they are both stuck.

After swapping, j is decremented and i is incremented. Array is as below:

A[]				
	4_c	4_d	4_a	4_b
		i j		

5- 4_d is swapped with itself and i is incremented, j is decremented, but now i > j. So, we stop and swap back the pivot with A[i]:

4_c	4_d	4_a	4_b
j		i	

→

4_c	4_d	4_b	4_a

please see back of paper

b. (i) Is mathematics decidable? Explain the Halting problem in your own words (no need to prove).

Mathematics in NOT decidable because given a Program P, we cannot tell if this program will halt, run finitely and stop with a result, or will be stuck in an infinite loop. The previous was the description of the halting problem, and it is the reason why math is undecidable. We have an algorithm to describe the halting problem, but none for solving it.

(ii) Is Mathematics Consistent? Explain your answer with an example.

Mathematics is not consistent.

Example ?

-1

9

Question 3: 11 points (4 + 3 + 4)

a. Suppose we perform a sequence of stack operations on a stack whose size never exceeds k. After every k operations, we make a copy of the entire stack for backup purposes. Show that the cost of n stack operations, including copying the stack, is O(n) by assigning suitable amortized costs to the various stack operations

- Let actual cost be as follows: $c_{pop} = 1$, $c_{push} = 1$, $c_{copy} = k$, since each cell is copied at cost of 1.
- Let amortize cost be as follows: $\hat{c}_{pop} = 3$, $\hat{c}_{push} = 3$, $\hat{c}_{copy} = 0$ $= n$
- After "n" operations (pop/push) the total cost will be $\left(\sum_{i=1}^{n} c_i + c_{copy} * \frac{n}{k}\right)$ because every $\frac{n}{k}$ times a copy will be made
- After "n" operation, the amortized cost is $3n = \sum_{i=1}^{n} \hat{c}_i$
- Amortized cost - Actual cost $= \sum_{i=1}^{n} \hat{c}_i - \sum_{i=1}^{n} c_i + \frac{n}{k} c_{copy}$

$= 3n - \cancel{2} n - \frac{n}{k} \cancel{k} = 3n - n - n = n$?

∴ Cost of "n" operations is $n \rightarrow$ is $O(n)$

∴ Cost per operation $= O(n/n) \rightarrow O(1)$

-1

b. Explain why Amortized analysis is better than Average Case analysis using probabilistic method.

Because probabilistic approach needs a complicated analysis and good estimation of the expected inputs and involves a lot of math. Amortized analysis covers all cases with a lot simpler effort. Its results may be not as solid as probabilities, but considering the simplicity, it is very good.

Question 3: 11 points (4 + 3 + 4)

c. Use RadixSort, with two bucket arrays and radix = 11, to sort the following array: [63, 1, 48, 53, 24, 10, 12, 30, 100, 115, 17]. Show all steps of the sorting procedure. Then explain why the running time is O(*n*).

Remainder

R[ ]

	100 12 1	24		48	115		17	30 63	53	10
0	1	2	3	4	5	6	7	8	9	10

The table above is filled as follows:
R[A[i] mod 11] = A[i] where A[i] is the i^{th} element of the input array and R[ ] is the element of the remainder bucket array.

Quotient

Q[ ]

10 1	17 12	30 24		53 48	63				100	115
0	1	2	3	4	5	6	7	8	9	10

Each ~~quo~~ element of the quotient array above "Q[i]" is filled using the formula: Q[A[i]/11] = A[i]
But here A[i] is collected from the ~~buckets in~~ remainder bucket array from left to right, and ~~down~~ bottom to top.

Finally, we obtain the sorted array by collecting the numbers from the quotient bucket: left to right & bottom ~~down~~ to top:

A_{sorted} = [1, 10, 12, 17, 24, 30, 48, 53, 63, 100, 115]

Running time analysis?

-1

Question 2: (continued)

e. Use Induction to show that

$$\textit{If } T(n) = rT(n-1) + a,\ T(0) = b, \textit{ and } r \neq 1 \textit{ then}$$

$$T(n) = r^n b + a\frac{1-r^n}{1-r} \quad \text{— (1)}$$

For all nonnegative integer a.

(1) Base case: Plug in n=0 into equation (1) above:

$T(0) = r^0 b + a\frac{1-r^0}{1-r} = b + a\left(\frac{1-1}{1-r}\right) = b \rightarrow$ Base case proven.

(2) Induction case: assume equation (1) is true for "n" and try to prove it for n+1. That is, assume

$r \cdot T(n-1) + a = r^{n} \cdot b + a\frac{1-r^{n}}{1-r}$?

and try to prove:

$r \cdot T(n) + a = r^{n+1} \cdot b + a\frac{1-r^{n+1}}{1-r}$

R.H.S. $= r^{n+1} \cdot b + a\frac{1-r^{n+1}}{1-r}$

?

Needs to be derived this using original equation with mathematical manipulations.

2.5

9.5

Question 2: 12 points (4 + 3 + 5)

For each of the following recurrences, derive an expression for the running time using iterative, substitution or Master Theorem.

a. Consider the following recurrence algorithm **[Use Master Theorem – See LAST Page]**

Code		Operations performed
`Procedure (Array A, int n) {`		
`If (n == 0) return True;`	2	Comparison + return
`for i = 1...n {`	$1+n+2n$	initialize counter + "n" comparisons + "n" increments + "n" assignment of increment
`A[i] = A[i] + 1;`	$4n$	
`}`		
`if (n>1) Procedure(A, n/2);`	$2+T(n/2)$	Comparison + Call + running time of call
`} // end procedure`		

i. (2 points) Write a recurrence equation for T(n)

$$T(n) = \begin{cases} 2 & n=0 \\ T(n/2)+7n+3 & \end{cases}$$

ii. (2 points) Solve recurrence equation using Master's method i.e. give an expression for the runtime T(n).

From the equation above:

$a=1,\ b=2,\ c=7,\ k=1,\ d=2 \Rightarrow a=1 < b^k = 2^1 = 2$

$\therefore T(n)$ is $\Theta(n)$

b. Use **Iterative method**

$$\begin{cases} T(n) = 3T(n-1)+1 \\ T(1) = 0 \end{cases}$$

$$\begin{aligned} T(n) &= 3T(n-1)+1 \\ &= 3(3T(n-2)+1)+1 \\ &= 3(3(3T(n-3)+1)+1)+1 \\ &= 3^3 \cdot T(n-3) + 3^2 + 3^1 + 3^0 \end{aligned}$$

Observing the pattern:

$$\begin{aligned} \therefore T(n) &= 3^k \cdot T(n-(k)) + 3^{k-1} + 3^{k-2} + 3^{k-3} + \dots + 3^1 + 3^0 \\ &= 3^k\, T(n-(k)) + \sum_{i=0}^{k-1} 3^i \\ &= 3^k\, T(n-(k)) + \frac{3^k-1}{3-1} \end{aligned}$$

Plug in $k = n-1$:

$$\therefore T(n) = 3^{n-1}\, T(1) + \frac{3^{n-1}-1}{3-1}$$

Since $T(1)=0$:

$$\therefore T(n) = \frac{3^{n-1}-1}{2}$$

$\boxed{\therefore T(n) \text{ is } O(3^n)}$

Question 1: (continued)

c. **Give a Big O estimate for f(x) = (x^3 + 2) log(x^2 + 1) + 4x^3**

1- since $\lim_{x\to\infty} \frac{x^3+2}{x^3} = \lim_{x\to\infty} \frac{1+2/x^3}{1} = 1 \therefore f_1(x) = x^3+2$ is $O(x^3) \rightarrow (1)$

2- Since x^2+1 eventually equals x^2

$\therefore \log(x^2+1) \approx \log x^2 = 2\log x$

since $2\log x \leq c \log x \rightarrow c \geq 2 \therefore f_2(x) = \log(x^2+1)$ is $O(\log x) \rightarrow (2)$

3- Since $\lim_{x\to\infty} \frac{4x^3}{x^3} = 4 \longrightarrow \therefore f_3(x) = 4x^3$ is $O(x^3) \rightarrow (3)$

Since $f(x) = f_1(x) \cdot f_2(x) + f_3(x)$, from (1), (2), (3):

~~$\therefore f(x)$ is $O(x^3$~~

$\therefore f(x)$ is $\max(O(x^3 \cdot \log x), O(x^3))$

$\therefore f(x)$ is $O(x^3 \cdot \log x)$

8

Question 1: 10 points (3 + 3 + 4)

a.

Show that $n^2 + 2n$ is $o(2^n)$

$$\lim_{n\to\infty} \frac{n^2+2n}{2^n} \overset{\lim}{=} \frac{2n+2}{c_1 \cdot 2^n}$$ "by applying L'Hôpital's rule"

$$= \lim_{n\to\infty} \frac{2}{c_2 \cdot 2^{2n}} = \lim_{n\to\infty} \frac{0}{c_3\, 2^{3n}} = 0$$

$\therefore f(n) = n^2 + 2n$ is $o(2^n)$

b. Determine whether f is O, o, Big omega or small omega of g where

f(n) = n ^ lg m and g(n) = m ^ lg n; Show your reasoning / work.

$F(n) = n^{\lg m}$, $g(n) = m^{\lg n}$

$$\lim_{n\to\infty} \frac{n^{\lg m}}{m^{\lg n}} = \lim_{n\to\infty} \frac{\lg m \cdot n^{(\lg m)-1}}{m^{\lg n} \cdot c_1 \cdot \frac{1}{n}}$$ $\boxed{\text{let } \log m = k \to 2^k = m}$

$$= \lim_{n\to\infty} \frac{k n^{k-1}}{2^{k \lg n} \cdot c_1/n}$$? taking L'Hôpital's rule k times:

$$\cancel{= \lim_{n\to\infty} \frac{k!}{C_2^{\lg n}}} = \lim_{n\to\infty} \frac{k!}{2^{c_2 \lg n} \cdot c_3/n^k}$$

$f(n) = n^k$, $g(n) = 2^{k \cdot \lg n}$

2 Since $\lg_a n < n$, $a \geq 2$ (Logarithms laws)

$\therefore g(n) \doteq 2^{k \lg n} < 2^{kn}$ $\therefore g(n)$ is $O(2^{kn})$ let $2^{kn} = h(n)$

-Take $\lim_{n\to\infty} \frac{n^k}{2^{kn}}$: apply L'Hôpital's rule k-times:

$$= \lim_{n\to\infty} \frac{k!}{c \cdot 2^{k^2 n}} = 0$$ $\therefore f(n)$ is $o(h(n))$?

$\therefore f(n)$ is $O(g(n))$