DSA Assignment 5

Connor Cash

March 2025

1.

a) Repeated Substitution

$T(n) = 3T(\frac{n}{4}) + 4n$

First Substitution
$T(n/4) = 4(\frac{n}{4}) + 3T(\frac{n}{16})$
$T(n) = 3(4(\frac{n}{4}) + 3T(\frac{n}{16})) + 4n$
$= 3n + 9T(\frac{n}{16}) + 4n$
$T(n) = 7n + 9T(\frac{n}{16})$
Second Substitution
$T(\frac{n}{16}) = 4(\frac{n}{16}) + 3T(\frac{n}{64})$
$T(n) = 7n + 9(4(\frac{n}{16}) + 3T(\frac{n}{64}))$
$= 7n + \frac{9}{4}n + 27T(\frac{n}{64})$
$T(n) = \frac{37}{4}n + 27T(\frac{n}{64})$
Third Substitution
$T(\frac{n}{64}) = 3T(\frac{n}{256}) + 4(\frac{n}{64})$
$= 3T(\frac{n}{256}) + (\frac{n}{16})$
$T(n) = \frac{37}{4}n + 27(3T(\frac{n}{256}) + (\frac{n}{16}))$
$= \frac{37}{4}n + 81T(\frac{n}{256}) + \frac{27}{16}n$
$T(n) = \frac{175}{16}n + 81T(\frac{n}{256})$
Then the pattern for the kth substitution is
$T(n) = 3^{k+1} * T(\frac{n}{4^{k+1}}) + C_k * n$
Where C_k is defined by the recurrence relation $C_0 = 4$ and $C_k = C_{k-1} + 3^k * \frac{4}{4^k}$
Then C_k is the sum of all previous terms plus $3^{k-1} * \frac{4}{4^{k-1}}$, so $C_k = (4 + 3^1 * \frac{4}{4^1} + 3^2 * \frac{4}{4^2} + ... + 3^{k-1} * \frac{4}{4^{k-1}})$
Then, $C_k = 4\sum_{i=0}^{k-1} \frac{3^i}{4^i}$
So, $T(n) = 3^{k+1} * T(\frac{n}{4^{k+1}}) + n * 4\sum_{i=0}^{k-1} \frac{3^i}{4^i}$

b) Proof

Claim: $T(n) = O(n)$
Want To Show: $\exists\ c,\ n_0 > 0$ s.t. $0 \leq T(n) \leq cg(n) \forall n \geq n_0$

Proof: I will prove the claim using induction.
Inductive Hypothesis: $\exists\ c > 0,\ n_0 > 0$ s.t. $\forall n \geq n_0\ T(n) \leq cn$
Inductive Step: Based on the inductive hypothesis $T(n) \leq cn$ is true for all numbers greater than or equal to n_0 and less than or equal to k.
So the recurrence is also true for $T(\frac{n}{4})$ since $n \geq 4n_0$ which is equivalent to $\frac{n}{4} \geq n_0$.
Then, $T(\frac{n}{4}) = c * \frac{n}{4}$
Which can be substituted into $T(n)$.
So $T(n) \leq 3(c * \frac{n}{4}) + 4n$
$T(n) \leq 3c + \frac{3n}{4} + 4n$
$T(n) \leq 12c + 3n + 16n$
$T(n) \leq 12c + 19n$
Let $h = 12c$ where h is a constant $T(n) \leq h + 19n$ Since h is a constant and does not grow for changes in n $19n$ dominates for large n.
Let $L = 19$ where L is a constant. Then $T(n) \leq L * n$
Thus by the definition of Big "O" notation $T(n) = O(n)$. Base Step: Let $n_0 \geq 4$, since this causes $\frac{n}{4} \geq 0$
Since the recurrence stops when $\frac{n}{4} = 1$ the recurrences $T(4), T(5), T(6), T(7)$ are constant operations.
The recursion is only happening once when any of these calls are made, but the 4n shows that $T(n) \geq cn$.
Thus by the definition of Big "O" notation the base case holds.
Therefore $T(n) = O(n)$.

c) master theorem

$a = 3, b = 4, d = 1$
$a < b^d$, so $T(n) = \Theta(n)$

2.

a) $T(n) = 3T(\frac{n}{5}) + n^2$

$a = 3\ b = 5\ d = 2\ b^d = 5^2 = 25$
$a < b^d$, so $T(n) = \Theta(n^2)$

b) $T(n) = 4T(\frac{n}{3}) + 7n$

$a = 4\ b = 3\ d = 1\ b^d = 3^1 = 3$
$a > b^d$, so $T(n) = \Theta(n^{(log_3 4)})$

c) $T(n) = 5T(\frac{n}{4}) + 10$

$a = 5\ b = 4\ d = 1\ b^d = 4^1 = 4$
$a > b^d$, so $T(n) = \Theta(n^{(log_4 5)})$

d) $T(n) = 9T(\frac{n}{3}) + n^4$

$a = 9$ $b = 3$ $d = 4$ $b^d = 3^4 = 81$
$a < b^d$, so $T(n) = \Theta(n^4)$

e) $T(n) = 6T(\frac{n}{8}) + n^3$

$a = 6$ $b = 8$ $d = 3$ $b^d = 8^3 = 512$
$a < b^d$, so $T(n) = \Theta(n^3)$

3)

CAP, COL, USD, SUN, JPY, VEE, ROW, JOB, COX, LOL, RAT, WOW, DOD, CAR, FIG, PIG, VIS, LOW, LOX, VEA, CAD, DOG, TSL

There will be three iterations and each iteration place all characters in the list into one of 26 buckets representing the order of the character in the alphabet. A = 0, Z = 25.

Iteration 1

A	0	VEA		
B	1	JOB		
C	2			
D	3	USD	DOD	CAD
E	4	VEE		
F	5			
G	6	FIG	PIG	DOG
H	7			
I	8			
J	9			
K	10			
L	11	COL	LOL	TSL
M	12			
N	13	SUN		
O	14			
P	15	CAP		
Q	16			
R	17	CAR		
S	18	VIS		
T	19	RAT		
U	20			
V	21			
W	22	ROW	WOW	LOW
X	23	COX	LOX	
Y	24	JPY		
Z	25			

After Iteration 1 VEA, JOB, USD, DOD, CAD, VEE, FIG, PIG, DOG, COL, LOL, TSL, SUN, CAP, CAR, VIS, RAT, ROW, WOW, LOW, COX, LOX, JPY

Iteration 2

A	0	CAD	CAP	CAR	RAT						
B	1										
C	2										
D	3										
E	4	VEA	VEE								
F	5										
G	6										
H	7										
I	8	FIG	PIG	VIS							
J	9										
K	10										
L	11										
M	12										
N	13										
O	14	JOB	DOD	DOG	COL	LOL	ROW	WOW	LOW	COX	LOX
P	15	JPY									
Q	16										
R	17										
S	18	USD	TSL								
T	19										
U	20	SUN									
V	21										
W	22										
X	23										
Y	24										
Z	25										

CAD, CAP, CAR, RAT, VEA, VEE, FIG, PIG, VIS, JOB, DOD, COL, LOL, ROW, WOW, LOW, COX, LOX, JPY, USD, TSL, SUN

Iteration 3

A	0					
B	1					
C	2	CAD	CAP	CAR	COL	COX
D	3	DOD	DOG			
E	4					
F	5	FIG				
G	6					
H	7					
I	8					
J	9	JOB	JPY			
K	10					
L	11	LOL	LOW	LOX		
M	12					
N	13					
O	14					
P	15	PIG				
Q	16					
R	17	RAT	ROW			
S	18	SUN				
T	19	TSL				
U	20	USD				
V	21	VEA	VEE	VIS		
W	22	WOW				
X	23					
Y	24					
Z	25					

Then the sorted list is [CAD, CAP, CAR, COL, COX, DOD, DOG, FIG, JOB, JPY, LOL, LOW, LOX, PIG, RAT, ROW, SUN, TSL, USD, VEA, VEE, VIS, WOW]

4)

The empty hash map with M=13 slots. Additionally, rehashing will be done at 0.75 load factor so the first rehashing will be done when 10 slots are full.

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	

Insert 25
1. Original Position
h1(25)=0
2. 0 Collisions
3. So the hash map becomes

0	25
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	

Insert 14
1.Original Position
h1(14)=4
2. 0 Collisions 3. So the hash map becomes

0	25
1	
2	
3	
4	14
5	
6	
7	
8	
9	
10	
11	
12	

Insert 9
1. Original Position
h1(9)=7
2. 0 Collisions
3. So the hash map becomes

0	25
1	
2	
3	
4	14
5	
6	
7	9
8	
9	
10	
11	
12	

Insert 7
1. Original Position
h1(7)=12
2. 0 collisions
3. So the hash map becomes

0	25
1	
2	
3	
4	14
5	
6	
7	9
8	
9	
10	
11	
12	7

Insert 5
1. Original Position
h1(5)=4, collision occurs so use double hash function $h_i(5) = (h1(5)+reverse(5)*i)\%13$
$h_1(5) = (h1(5) + reverse(5))\%13 = (4 + 5)\%13 = 9$
Key 9 is empty so no more hashing needs to be done.
2. 1 total collision.
3. 5 is inserted at key 9 in the hash map

0	25
1	
2	
3	
4	14
5	
6	
7	9
8	
9	5
10	
11	
12	7

Insert 3
1. Original Position
h1(3)=10, key 10 is empty so 3 is inserted
2. 0 collisions
3. The hash map becomes

0	25
1	
2	
3	
4	14
5	
6	
7	9
8	
9	5
10	3
11	
12	7

Insert 0
1. Original Position
h1(0)=0, key 0 is not empty so collision occurs
Rehashing, $h_1(0) = h1(0) + reverse(0) * 1 = 0 + 0 = 0$
Another collision occurs but at the same key, so there is no probe movement done by double hashing making 0 impossible to insert with the current hash map. This would lead to an infinite loop.
2. There are an infinite amount of collisions because the double hash function always returns 0 which is already full. 3. Since it is impossible to insert 0 at the hash map's current state and with the current hash function the hash hash map remains the same,

0	25
1	
2	
3	
4	14
5	
6	
7	9
8	
9	5
10	3
11	
12	7

Insert 21
1. Original Position
h1(21)=2, slot 2 is empty so 21 is inserted 2. 0 collisions
3.The hash map becomes,

0	25
1	
2	21
3	
4	14
5	
6	
7	9
8	
9	5
10	3
11	
12	7

Insert 6

1. Original Position
h1(6)=8, slot 8 is empty so 6 is inserted
2. 0 Collisions
3. The hash map becomes,

0	25
1	
2	21
3	
4	14
5	
6	
7	9
8	6
9	5
10	3
11	
12	7

Insert 33

1. Original Position
h1(33)=3, slot 3 is empty so 33 is inserted 2. 0 Collisions
3. The hash map becomes,

0	25
1	
2	21
3	33
4	14
5	
6	
7	9
8	6
9	5
10	3
11	
12	7

Insert 25
1. Original Positions
h1(25)=0, slot 0 is full so double hashing is done $h_1(25) = (h1(25) + 1 * reverse(25))\%13 = (0 + 1 * 52)\%13 = 52\%13 = 0$
$h_2 = h1(25) + 2 * reverse(25) = (2 * 52)\%13 = 0$
Since h1(25)=0 and the double hashing just multiplies a number that is divisible by 13 by an integer, the hash function will always by divisible by 13 and return 0, so there is an infinite amount of collisions. 2.Infinite amount of collisions. 3.Since it is impossible to insert 25, the hash map remains the same.

0	25
1	
2	21
3	33
4	14
5	
6	
7	9
8	6
9	5
10	3
11	
12	7

Insert 42
1. Original positions
h1(42)=10, slot 10 is full so double hashing occurs
$h_1(42) = (h1(42) + reverse(42) * 1)\%13 = (10 + 24)\%13 = 8$, slot 8 is full
$h_2(42) = (h1(42) + reverse(42) * 2)\%13 = (10 + 24 * 2)\%13 = 58\%13 = 6$, slot 6 is not full so 42 is inserted
2. There were 2 collisions.
3. The hash table becomes, 3.Since it is impossible to insert 25, the hash map remains the same.

0	25
1	
2	21
3	33
4	14
5	
6	42
7	9
8	6
9	5
10	3
11	
12	7

Since the load factor has now been exceeded $\frac{10}{13} \geq 0.75$, I am going to resize and rehash all elements in the previous hash table with M=29 slots

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	
16	
17	
18	
19	
20	
21	
22	
23	
24	
25	
26	
27	
28	

Rehashing
Insert 25
h1(25)=14, 0 collisions
Insert 14 h1(14)=11, 0 collisions
Insert 9 h1(9)=17, 0 collisions
Insert 7 h1(7)=9, 0 collisions
Insert 5 h1(5)=1, 0 collisions
Insert 3 h1(3)=23, 0 collisions
Insert 21 h1(21)=19, 0 collisions
Insert 6 h1(6)=5, 0 collisions
Insert 33, h1(33)=11, slot 11 is already full so double hashing is done
$h_1(33) = (h1(33) + reverse(33) * 1)\%29 = (11 + 33)\%29 = 15$, slot 15 is empty so 33 is inserted, 1 collisions
Insert 42, 0 collisions
h1(42)=25
Rehashing Done
So the table becomes,

0	
1	5
2	
3	
4	
5	6
6	
7	
8	
9	7
10	
11	14
12	
13	
14	25
15	33
16	
17	9
18	
19	21
20	
21	
22	
23	3
24	
25	42
26	
27	
28	

Insert 24
h1(24)=8, 0 collisions
Insert 107
h1(107)=25, slot 25 is already full so double hashing occurs
$h_1(107) = (h1(107) + reverse(107) * 1)\%29 = (25 + 701)\%29 = 1$, slot 1 is full so hashing continues
$h_2(107) = (h1(107) + reverse(107) * 2)\%29 = (25 + 701 * 2)\%29 = 6$, slot 6 is empty so 107 is inserted, 2 collisions
Then the final table is

0	
1	5
2	
3	
4	
5	6
6	107
7	
8	24
9	7
10	
11	14
12	
13	
14	25
15	33
16	
17	9
18	
19	
20	
21	19
22	
23	3
24	
25	42
26	
27	
28	

7. Algorithm Analysis

a) Radix Sort

Let n = s.length and m = max string length in s. The radix sort method calls the calcMax method which has time complexity $O(n)$ since calcMax loops through all Strings in s. Then the radixSort method contains a nested loop where the inner loop calls the buckets method. The buckets method has time complexity O(n) since the first loop in the method runs n times and the other loop is nested but runs O(27*n)=O(n). The n comes from the worst case scenario when all strings are in the same. For example, buckets[0].length = s.length represents the worst case. Then the total time complexity of the buckets method is O(2n)=O(n). Since the loop in the radixSort method is nested the total time complexity is O(m*n+n)=O(m*n). The space complexity is O(n) since the only space that is being allocated that grows as the input size grows is allocation and assignment of String[] temp = new String[s.length]. Note that int j is allocated in a nested loop but is always allocated 27 times, so the space complexity is still O(n).

b) Word Pattern

Time Complexity
Let n=pattern.length() and m=s.length() and p=max length word prior to delimiter or end of string in s ex: "apple—banana—grape—apple" p=6. The uniq method has time complexity O(26)+O(n)=O(26+n)+O(n). The word pattern method has a singular call to the uniq method and 2 loops. In the first loop the outer loop runs n times, but there are nested loops in an if statement. The if statement runs every time a delimeter is encounterd and the total number of delimeters is one less than n, so the if statement executes n-1 times. For every time the if statement executes the substring method is called which increments over a word in s. In the worst case this substring is longest word in s, which would be O(p) time complexity. Then the containsKey and containsValue method are both called which have worst case time complexity O(n) since each character in the pattern represents a word to be put into the hash map. With a perfect hash function containsKey is O(1) but since it is unknown if the hash function is perfect with the keys being provided I am going to have O(n) be the bound for both method calls. Then the worst case time complexity of the loop is O(m(p+2n)). This upper bound is assuming that if statement runs m times which is not the tightest upper bound but is a true upper bound since $m \geq n-1$ because if this were not true then not all letters in the pattern could map to a word. Then another substring call is made to ensure that the last word is included in the hash map, so more calls are made to containsValue. Then time complexity of these statements in lines 43-48 is O(p+n). The 2nd and final loop has time complexity O(n) since it runs n times. Ultimately the total time complexity would be O(n)+O(mp+2nm)+O(p+n)+O(n)=O(3n+p+mp+2nm)=O(mp+nm).
Space Complexity The only space that is being allocated that changes as the input increases is the HashMap. The maximum size of the HashMap occurs when the pattern has completely unique letters, so the maximum size would be n. Then the maximum amount of space that is allocated by the HashMap and the space complexity of the algorithm is O(n).