Assignment 5 - Hartz

1. Repeated Substitution

T(n) = 2T(n/4) + 2n

= 2(2T(n/4) + 2n)(n/4) + 2n = (((4n+Tn)n)/4) + 2n

= (((4n+(2T(n/4) + 2n)n)n)/4) + 2n = (((8n+4n^2+(n^2)T)n)/8) + 2n

= (((16n+8n^2 + 4n^3+(n^3)T)n)/16) + 2n

= (((32n + 16n^2 +8n^3 + 4n^4 +(n^4)T)n)/32) + 2n

(((j)n + (j-1)n^2 + (j-2)n^3… (j-x)n^(x-1) + n^(x-1)T)/j) + 2n, where j is the series of 2^2; j(1) = 4, j(2) = 8, j(3) = 16, and so forth.

2. Master Theorem

a. T(n) = 2T(n/4) + n^3
a is 2, b is 4, and d is 3 (d being the growth rate of the function at the end, n^3). b^d is 4^3, which is 64, and 2 < 64. Therefore, the first case of the master theorem applies: The time complexity is theta of n^d, or theta of n^3.

b. T(n) = 2T(n/2) + 6n^4^4

a is 2, b is 2, and d is 4. 2^4 is 16, which is greater than 2, so time comp is theta(n^4).

c. T(n) = 6T(n/7) + 23

a is 6, b is 7, d is 0. 7^0 = 1, which is less than 6. Therefore, the third case of the master theorem applies: The time complexity is theta(n raised to log base b(a)), or theta(n raised to log base 7(6)).

d. T(n) = 16T(n/4) + n^2

a is 16, b is 4, and d is 2. 4^2 is 16, which means a = b^d, leading to the second case. Therefore, time complexity is theta(n^(d) log n), or is theta(n^(2) log n)

e. T(n) = 7T(n/9) + n^3

a is 7, b is 9, and d is 3. 9^3 is 729, which is much greater than 7. Therefore, time complexity is theta(n^3).

3. Double Hashing

[43, 22, 10, 8, 7, 4, 0, 11, 3, 28, 43, 36]

I'm not sure if I'm understanding the double hashing correctly. My interpretation that I used here is as follows: first, go through h1, of course. After that, take the key, multiply by which probe you're on, and then go through Reverse, and modulo that against 11.

0: 43

1: 22

2: 11

3: 8

4: 4

5: 43

6: 3

7: 7

8: 28

9: 10

10:36

43 maps to 0. No probe sequence necessary, it goes to 0.

22 maps to 1. No probe sequence necessary, it goes to 1.

10 maps to 9. No probe sequence necessary, it goes to 9.

8 maps to 3. No probe sequence necessary, it goes to 3.

7 maps to 0. On probe one, 7 maps to 7.

4 maps to 3. On probe one, 4 maps to 4.

0 maps to 5. No probe sequence necessary, it goes to 5.

11 maps to 2. No probe sequence necessary, it goes to 2.

3 maps to 0. On probe one, 3 maps 3. On probe two, 3 maps to 6.

28 maps to 8. No probe sequence necessary, it goes to 8.

43 maps to 0. On probe one, 43 maps to 1. On probe two, 43 maps to 2. On probe three, 43 maps to 8. On probe four, 43 maps to 7. On probe 5, 43 maps to 6. On probe 6, 43 maps to 5.

36 maps to 10. No probe sequence necessary, it goes to 10.

4. Radix Sort

CAT, SBX, LOG, SUN, MUG, ROW, JOB, COX, LAP, RAT, PER, DAD, CAR, FIG, PIG, VIA, LOW,

LOX, TEA, ATE, ARE, DOG, TSL

I solved #5 first and then used that code to generate the answer to this, hence why the output is lowercase. The preceding number is the bucket the string was sorted into.

Pass one, on last letter:

1 via
1 tea
2 job
4 dad
5 ate
5 are
7 log
7 mug
7 fig
7 pig
7 dog
12 tsl
14 sun
16 lap
18 per
18 car
20 cat
20 rat
23 row
23 low
24 sbx
24 cox
24 lox

Pass two, on middle letter:

1 dad
1 lap
1 car
1 cat
1 rat
2 sbx
5 tea

5 per
9 via
9 fig
9 pig
15 job
15 log
15 dog
15 row
15 low
15 cox
15 lox
18 are
19 tsl
20 ate
21 mug
21 sun

Pass three, on first letter:

1 are
1 ate
3 car
3 cat
3 cox
4 dad
4 dog
6 fig
10 job
12 lap
12 log
12 low
12 lox
13 mug
16 per
16 pig
18 rat
18 row
19 sbx
19 sun
20 tea
20 tsl
22 via

7. Algo Analysis

4’s algorithm is the same as 5’s.

For #5, time complexity is O(n*d), where n is equal to the length of the input array and d is equal to the maximum amount of digits present in the input. This will be a constant, ultimately, so it's O(n), really. Space complexity for this method is O(n) as well, with n still being the size of the input array because I called .toCharArray within a for loop that runs n times, meaning a new char array is initialized n times.

For #6, the time complexity is O(n*k), where n is equal to the size of the first input and k is equal to the size of the second input. Radixsort is called on both, which has an O(j) time complexity with j being it's input, but more important, I have a nested for loop in my answer, which compares every element of n to every element of k. Space complexity for this method is O(n), as the loop variable for the nested k loop is initialized n times.