Amalia Karaman

Text Questions

1) Recurrence relation w/ repeated substitution

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

solve w/ master

Repeated sub:

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

Tn

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

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

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

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

$= 3^3 T(n/4^3) + 3^2 \times 4(n/4) + 8n$

$= 3^3 T(n/4^3) + 12n$

→ After k

$T(n) = 3^k T(n/4^k) + 4n \times k$

→ stop @ $n/4^k = 1$

$n = 4^k \Rightarrow k = \log_4 n$ ← plug

$T(n) = 3^{\log_4 n} \times T(1) + 4n \times \log_4 n$

$3^{\log_4 n} = n^{\log_4 3}$

Final: $T(n) = \Theta(n^{\log_4 3} + n \log n)$

↪ $T(n) = \Theta(n^{\log_4 3})$

Master method

$T(n) = aT(n/b) + f(n)$

$a = 3 \quad b = 4 \quad f(n) = 4n \rightarrow \Theta(n)$

$\log_4 3 \approx .792$

$f(n) = \Theta(n)$

? $n = \Omega(n^{.792+\varepsilon})$ for $\varepsilon > 0$

f(n) faster than $n^{\log_4 3}$ + 4n dominates recursively

Final: $T(n) = \Theta(n)$

2) Master Theorm

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

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

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

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

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

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

$a = 3$

$b = 5$

$f(n) = n^2$

$\log_b a = \log_5 3 \approx .682$

$f(n) = n^2$ vs $n^{\log_5 3}$

$\to n^2$ faster

$f(n) = \Omega(n^{\log_b a + \varepsilon})$ polynomial bigger

Regularity ✓

$a \times f(\frac{n}{b}) \le c \times f(n)$ for $c < 1$ + large n

yes ✓ $f(n) = n^2$ $f(\frac{n}{b}) = (\frac{n}{5})^2 = (\frac{n^2}{25})$ $3f(\frac{n}{5}) = 3n^2/25$ ✓ $< n^2$

Case 3 ✓ $T(n) = \Theta(n^2)$

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

$a = 4$

$b = 3$

$f(n) = 7n$

$\log_3 4 \approx 1.26$

$f(n) = \Theta(n)$ vs $n^{\log_3 4} = n^{1.26}$

Case 1

$T(n) = \Theta(n^{\log_3 4})$

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

$a = 5$

$b = 4$

$f(n) = 10 = \Theta(1)$

$\log_4 5 \approx 1.161$

$f(n) = \Theta(1)$ vs $n^{\log_4 5}$

$f(n)$ smaller than $n^{\log_4 5}$

case 1

$T(n) = \Theta(n^{\log_4 5})$

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

$a = 9$

$b = 3$

$f(n) = n^4$

$\log_3 9 = 2$

$f(n) = n^4$ vs n^2

n^4 faster growth

Regularity

$f(n) = n^4$

$f(\frac{n}{3}) = (\frac{n}{3})^4 = \frac{n^4}{81}$

$9 \times f(\frac{n}{3}) = 9n^4/81 = n^4/9 < n^4$ ✓

case 3

$T(n) = \Theta(n^4)$

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

$a = 6$

$b = 8$

$f(n) = n^3$

$\log_8 6 \approx .925$

$f(n) = n^3$ bigger than $n^{.925}$

~~Regularity~~

Regularity

$f(\frac{n}{8}) = (\frac{n}{8})^3 = n^3/512$

$6 \times f(\frac{n}{8}) = 6n^3/512 = n^3(\frac{6}{512}) < n^3$ ✓

Case 3

$T(n) = \Theta(n^3)$

3) Radix Sort

Input: CAP, COL, UVD, JUN, JPY, VEE, ROW, JOB, COX, LOL, RAT, WOW, DOD, CAR, FIG, PIG, VIS, LOW, LOX, VEA, CAD, DOG, TSL

3rd char:

CAD → D	DOG → G	LOL → L	ROW → W
CAP → P	DOD → D	LOW → W	JUN → N
CAR → R	FIG → G	LOX → X	TSL → L
COL → L	JOB → B	PIG → G	USP → D
COX → X	JPY → Y	RAT → T	VEE → E
			VEA → A
			VIS → S
			WOW → W

Sort:

B → JOB
D → CAD, DOD, USD
E → VEE
G → DOG, FIG, PIG
L → COL, LOL, TSL
N → JUN
P → CAP
R → CAR
S → VIS
T → RAT
W → LOW, ROW, WOW
X → COX, LOX
Y → JPY
A → VEA

Result: JOB, CAD, DOD, UVD, VEE, DOG, FIG, PIG, COL, LOL, TSL, JUN, CAP, CAR, VIS, RAT, LOW, ROW, WOW, COX, LOX, JPY, VEA

Sort by 2nd

JOB → O
CAD → A
DOD → O
USD → S
VEE → E
DOG → O
FIG → I
PIG → I
COL → O
LOL → O
TSL → S
SUN → U
CAP → A
CAR → A
VIS → I
RAT → A
LOW → O
ROW → O!
WOW → O
COX → O
LOX → O
JPY → P
VEA → E

Group/reorder:

A → CAD, CAP, CAR, RAT   P → JPY
E → VEE, VEA   S → USD, TSL
I → FIG, PIG, VIS   U → SUN
O → JOB, DOD, DOG, COL, LOL, LOW, ROW, WOW, COX, LOX

Result:

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

Sort by 1st

C → CAD, CAP, CAR, COL, COX

D → DOD, DOG

F → FIG

J → JOB, JPY

L → LOL, LOW, LOX

P → PIG

R → RAT, ROW

S → SUN

T → TSL

U → UJD

V → VEA, VEE, VIS

W → WOW

Final order!!!

→ CAD
CAP
CAR
COL
COX
DOD
DOG
FIG
JOB
JPY
LOL
LOW

1

LOX
PIG
RAT
ROW
SUN
TSL
UJD
VEA
VEE
VIV
WOW

2

#4: GIVEN

```
int h1(int key) {
    int x = (key + 19) * (key + 11);
    x = x / 15;
    x = x + key;
    x = x % M;
    return x;
}
```

HASH TABLE: M=13

Key	Calculation	Result (Index)
25	(25+19)*(25+11) = 44*36 = 1584 → 1584/15 = 105 + 25 = 130	130 % 13 = 0
14	(14+19)*(14+11) = 33*25 = 825 → 825/15 = 55 + 14 = 69	69 % 13 = 4
9	(9+19)*(9+11) = 28*20 = 560 → 560/15 = 37 + 9 = 46	46 % 13 = 7
7	(7+19)*(7+11) = 26*18 = 468 → 468/15 = 31 + 7 = 38	38 % 13 = 12
5	(5+19)*(5+11) = 24*16 = 384 → 384/15 = 25 + 5 = 30	30 % 13 = 4 (collision)
3	(3+19)*(3+11) = 22*14 = 308 → 308/15 = 20 + 3 = 23	23 % 13 = 10
0	(0+19)*(0+11) = 19*11 = 209 → 209/15 = 13 + 0 = 13	13 % 13 = 0 (collision)
21	(21+19)*(21+11) = 40*32 = 1280 → 1280/15 = 85 + 21 = 106	106 % 13 = 2
6	(6+19)*(6+11) = 25*17 = 425 → 425/15 = 28 + 6 = 34	34 % 13 = 8
33	(33+19)*(33+11) = 52*44 = 2288 → 2288/15 = 152 + 33 = 185	185 % 13 = 3

FIRST REHASH M=29

Key	Calculation	Result (Index)
25	130 % 29	13
14	69 % 29	11
9	46 % 29	17
7	38 % 29	9
5	30 % 29	1
3	23 % 29	23
0	13 % 29	13 (collision) → Reverse(0) = 0 → Step = 1 → Try 14
21	106 % 29	18
6	34 % 29	5
33	185 % 29	11 (collision) → Reverse(33) = 33 → Step = 4 → Try 15

FINAL REHASH: M=29

Key	Calculation	Result (Index)
25	(25+19)*(25+11) = 44*36 = 1584 → 1584/15 = 105 + 25 = 130	130 % 29 = 13
14	(14+19)*(14+11) = 33*25 = 825 → 825/15 = 55 + 14 = 69	69 % 29 = 11
9	(9+19)*(9+11) = 28*20 = 560 → 560/15 = 37 + 9 = 46	46 % 29 = 17
7	(7+19)*(7+11) = 26*18 = 468 → 468/15 = 31 + 7 = 38	38 % 29 = 9
5	(5+19)*(5+11) = 24*16 = 384 → 384/15 = 25 + 5 = 30	30 % 29 = 1
3	(3+19)*(3+11) = 22*14 = 308 → 308/15 = 20 + 3 = 23	23 % 29 = 23
0	(0+19)*(0+11) = 19*11 = 209 → 209/15 = 13 + 0 = 13	13 % 29 = 13 (collision) → Reverse(0)=0 → Step=1 → Try 14
21	(21+19)*(21+11) = 40*32 = 1280 → 1280/15 = 85 + 21 = 106	106 % 29 = 18
6	(6+19)*(6+11) = 25*17 = 425 → 425/15 = 28 + 6 = 34	34 % 29 = 5
33	(33+19)*(33+11) = 52*44 = 2288 → 2288/15 = 152 + 33 = 185	185 % 29 = 11 (collision) → Reverse(33)=33 → Step=4 → Try 15
25	130 % 29 = 13 (collision) → Reverse=52 → Step=23	Probes: 13 → 6 → Insert at 6
42	(42+19)*(42+11) = 61*53 = 3233 → 3233/15 = 215 + 42 = 257	257 % 29 = 25
24	(24+19)*(24+11) = 43*35 = 1505 → 1505/15 = 100 + 24 = 124	124 % 29 = 8
107	(107+19)*(107+11) = 126*118 = 14868 → 14868/15 = 991 + 107 = 1098	1098 % 29 = 25 (collision) → Reverse=701 → Step=5 → Probes: 25 → 1 → 6 → 11 → Try 16

#7: TIME AND SPACE COMPLEXITY OF 4-6

#4:

Time Complexity:
Average Case:
For each key, computing the first hash h1() and the secondary hash Reverse() is O(1). In the average case, insertions using double hashing take O(1) time if the load factor is low (few collisions). However, as the table fills, probing increases.

Worst Case:
If many collisions occur, double hashing degrades to O(n) per insertion (where *n* is the number of elements), since it may probe many slots before finding an empty one. Additionally, resizing the hash table is O(n) because every existing element must be rehashed and reinserted into the new table.
Since rehashing happens only occasionally (amortized), the overall time complexity for inserting *m* keys is O(m) on average, $O(m^2)$ in worst case (e.g., constant collisions or poorly distributed keys)

Space Complexity:
The hash table uses O(m) space, where *m* is the number of keys stored. Additional space is used during resizing (essentially a temporary second table), but it's also O(m). The space used by the Reverse() function is O(1) per key.

#5:
Time Complexity:
Radix sort runs in O(k × n), where *n* is the number of strings and *k* is the max string length. It performs *k* passes using counting sort, which takes linear time per pass. Each character comparison is constant time, so total work is proportional to the number of characters processed.
Space Complexity:
Uses O(n + r) space where *n* is for output and *r* (256 ASCII buckets) is constant. Extra memory is needed to hold buckets and sorted results, but remains linear in size.

#6:
Time Complexity:
Runs in O(n) where *n* is the number of words in the string. Each character in the pattern and word in the string is checked once. HashMap operations (insert and lookup) are constant time.
Space Complexity:
Requires O(n) space to store two HashMaps for tracking and the array of split words. Memory scales with the number of unique pattern characters and words.