<u>CS386D Database Systems</u>
<u>HW3 Solutions</u>

<u>Part A (20 pts)</u>

14.7.1

a) **speed**

Value	**Uncompressed**	**Compressed**
1.42	001000000000	1010
1.86	000000000010	11101010
2.00	000000001000	11101000
2.10	010000000000	01
2.20	000000110000	11011000
2.66	100000000000	00
2.80	000100000101	101111010101
3.20	000011000000	11010000

b) **ram**

Value	**Uncompressed**	**Compressed**
512	011010000000	010001
1024	100101101001	0010100100011010
2048	000000010110	1101110100

c) **hd**

Value	**Uncompressed**	**Compressed**
80	001000000000	1010
160	000000000011	1110101000
200	000000100000	110110

250	110110011000	00000100101000
300	000000000100	11101001
320	000001000000	110101

14.7.3 a.

If uncompressed bitmap is assumed, size of index = m * 1000000/8 bytes = 125000 m bytes
If compressed bitmap is assumed, size of index = m * 2 log(m) * 1000000/m bits = 250,000 log(m) bytes

Part B (30 pts)

Q1.

a. The entire alphabet can be added.

b. The number of characters has no impact on the bloom filter because bloom filter only depends on filter size m, number of hash functions k, and number of items added to the filter n. So the probability of false positives is expected to be the same. However, over multiple experiments the actual number of tries before a false positive occurs may vary.

Q2.

a. Probability of false positive: $(1 - (1 - 1/m)^{kn})^k$ = 0.02% [n = 1, m = 50, k = 3]
b. Probability of false positive: $(1 - (1 - 1/m)^{kn})^k$ = 34% [n = 20, m = 50, k = 3]

Q3. 12 keys.

$n = m/k * ln(2) = 50/3 * ln(2) \approx 12$

Part C (70 pts)

5.1.1

π_{speed}(PC) **with set semantics**	π_{speed}(PC) **with bag semantics**
2.66 2.10 1.42 2.80 3.20 2.20 2.00 1.86 3.06	2.66 2.10 1.42 2.80 3.20 3.20 2.20 2.20 2.00 2.80 1.86 2.80 3.06
Avg $\approx$ 2.37	Avg $\approx$ 2.48

5.1.2

π_{hd}(PC) **with set semantics**	π_{hd}(PC) **with bag semantics**
250 80 320 200 300 160	250 250 80 250 250 320 200 250 250 300 160 160 80
Avg $\approx$ 218.33	Avg $\approx$ 215.38

16.2.2.

b)

Relation R:

c1	**c2**
1	2

Relation S:

c1	**c2**
1	4

π_{c1}(R-S) = {3} while π_{c1}R - π_{c1}S = ϕ

Similarly, with projections with bag semantics, multiple copies of these tuples would lead to a Null set for π_{c1}R - π_{c1}S and non-empty set for π_{c1}(R-S).

c)

Relation R:

c1	c2
1	2
1	3

δ(π_{c1}R) = 1 while $\pi_{c1}\delta$(R) = {1,1}

Note: This is only one of many examples of relations that can be given to prove the statements.

2.

The outputs should correspond to the following queries:
i. SELECT * FROM R WHERE joinKey1 = 101
ii. SELECT * FROM R WHERE joinKey1 != 101
iii. SELECT * FROM R WHERE joinKey1 is Null
iv. SELECT Name, joinKey1 FROM R
v. SELECT joinKey1 FROM R
vi. SELECT * FROM R JOIN S ON joinKey1 = joinKey2
vii. SELECT * FROM R JOIN S ON joinKey1 != joinKey2
viii. SELECT * FROM R full outer JOIN S ON joinKey1 = joinKey2
ix. SELECT * FROM R left outer JOIN S ON joinKey1 = joinKey2
x. SELECT * FROM R WHERE EXISTS(SELECT 1 FROM S WHERE R.joinKey1 = S.joinKey2)
xi. SELECT * FROM R WHERE NOT EXISTS(SELECT 1 FROM S WHERE R.joinKey1 = S.joinKey2)

3.

vi.

```
SELECT * FROM R,S WHERE joinKey1 = joinKey2;
SELECT * FROM R inner JOIN S ON joinKey1 = joinkey2;
SELECT * FROM R cross JOIN S WHERE joinkey1 = joinkey2;
```

vii.

```
SELECT * FROM R, S WHERE joinKey1 != joinKey2;
SELECT * FROM R inner JOIN S ON joinKey1 != joinkey2;
SELECT * FROM R cross JOIN S WHERE joinkey1 != joinkey2;
```

4.

viii.

```
SELECT * FROM R left JOIN S ON joinKey1 = joinKey2
UNION
SELECT * FROM S right JOIN S ON joinKey1 = joinKey2;
```

ix.

```
SELECT *  FROM R INNER JOIN S ON  joinKey1 = joinKey2
UNION
SELECT R.thePrimaryKey, R.Name, R.joinkey1, NULL, NULL, NULL FROM R
WHERE NOT EXISTS (SELECT * FROM S WHERE joinKey1 = joinKey2);
```

x.

```
SELECT  R.thePrimaryKey, R.Name, R.joinKey1
FROM R INNER JOIN S ON joinKey1 = joinKey2;
```

xi.

```
SELECT R.thePrimaryKey, R.Name, R.joinKey1
FROM R LEFT JOIN S ON joinKey1= joinKey2 WHERE joinKey2 IS NUll;
```

Note: There are many other equivalent queries possible.