Question1

(a) # of tuples: 80000 * (1 / 5) * ((600 - 100) / 10000) = 800
Tuples / leaf: 80000 / 500 = 160
Cost of index search = 2 + 800 / 160 (+ 1) = 7 (+ 1)

(b) # of pages: 800 * (5 * 8) / 6000 = 6

Cost: ceil(6 / 23) * 4000 = 4000

(c) # of tuples: 800 * 600000 * (1 / 5000) = 96000
# of pages: (96000) * (8 * 6) / 6000 = 768 pages

768 / 100 = 8 groups
The whole process: r/w/r
We do not need to have any cost on read, cost(w) + cost(r) = 768 + 768 = 1536 pages

(d) Total cost: 7 (+1) + 4000 + 1536 = 5543 (+1) pages

Question2:

1. (R join S) join T

First cost (R join S): 200 + (200 / 200) * 1200 = 1400
For cost to join T: (8000 / 200) * 4000 = 160000

Total cost: 160000 + 1400 = 161400 pages

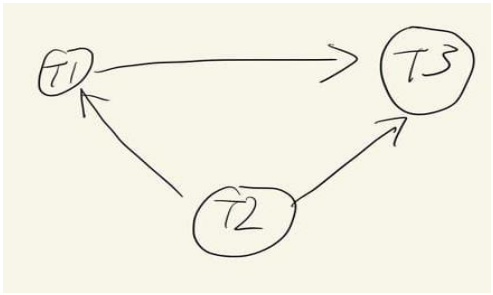
2. (R join T) join S

First cost (R join T): 200 + (200 / 200) * 4000 = 4200
For cost to join S: (2000 / 200) * 1200 = 12000

Total cost: 12000 + 4200 = 16200 pages

Question3:

(a) List all conflicts:
r1(x) w3(x)
r2(z) w3(z)
w2(w) r1(w)
w2(z) r3(z)
w2(z) w3(z)

(b) 

From the graph above, it is serializable. Since we can go from T2 , T1 , T3 with no cycle.

(c) Yes, it is possible for 2PL.

r2(z) w2(w) w2(z) r1(x) r1(y) r1(w) w1(y) r3(z) w3(x) w3(z)

Simple go from T2 T1 T3, once one finished and do not need any lock, we give the lock to another transaction.

Question4

Query1:

(a)

```
SELECT DISTINCT
   a.name
FROM
   artists a
   , songs s
   , spotify sp
WHERE
   a.id = s.artistid
   and s.id = sp.songid
   and sp.streams >= 10000000
ORDER BY
   a. name ASC;
```

(b)

```
Unique (cost=564.30..565.11 rows=163 width=11)
-> Sort (cost=564.30..564.71 rows=163 width=11)
Sort Key: a.name
-> Hash Join (cost=43.54..558.31 rows=163 width=11)
Hash Cond: (s.artistid = a.id)
-> Hash Join (cost=25.23..539.57 rows=163 width=8)
Hash Cond: (sp.songid = s.id)
```

-> Seq Scan on spotify sp (cost=0.00..513.91 rows=163 width=8)
Filter: (streams >= 10000000)
-> Hash (cost=16.77..16.77 rows=677 width=16)
-> Seq Scan on songs s (cost=0.00..16.77 rows=677 width=16)
-> Hash (cost=10.36..10.36 rows=636 width=19)
-> Seq Scan on artists a (cost=0.00..10.36 rows=636 width=19)

(c)
create index query6 on spotify(streams)
(d)

Unique (cost=236.40..237.22 rows=163 width=11)
-> Sort (cost=236.40..236.81 rows=163 width=11)
Sort Key: a.name
-> Hash Join (cost=49.09..230.42 rows=163 width=11)
Hash Cond: (s.artistid = a.id)
-> Hash Join (cost=30.78..211.68 rows=163 width=8)
Hash Cond: (sp.songid = s.id)
-> Bitmap Heap Scan on spotify sp (cost=5.55..186.02 rows=163
width=8)
Recheck Cond: (streams >= 10000000)
-> Bitmap Index Scan on query6 (cost=0.00..5.51 rows=163 width=0)
Index Cond: (streams >= 10000000)
-> Hash (cost=16.77..16.77 rows=677 width=16)
-> Seq Scan on songs s (cost=0.00..16.77 rows=677 width=16)
-> Hash (cost=10.36..10.36 rows=636 width=19)
-> Seq Scan on artists a (cost=0.00..10.36 rows=636 width=19)

Query2:
(a) select
   s.name as songname
   , a.name as artistname
   , count(distinct p.id) as numplayed
from
   songs s
   join bilboard b on b.songid = s.id
   join artists a on a.id = s.artistid
   left join playedonradio p on p.songid = s.id
where

```
   s.danceability >= 0.9
   and b.rank <= 10
group by
   s.id
   , s.name
   , a.name
order by
   numplayed desc
   , songname asc
   , artistname asc
;
```

(b)

```
Sort (cost=1291.57..1294.15 rows=1033 width=45)
Sort Key: (count(DISTINCT p.id)) DESC, s.name, a.name
-> GroupAggregate (cost=1162.07..1239.85 rows=1033 width=45)
Group Key: s.id, a.name
-> Incremental Sort (cost=1162.07..1221.77 rows=1033 width=41)
Sort Key: s.id, a.name
Presorted Key: s.id
-> Merge Join (cost=1161.90..1181.71 rows=1033 width=41)
Merge Cond: (s.id = b.songid)
-> Sort (cost=995.89..997.27 rows=551 width=41)
Sort Key: s.id
-> Hash Join (cost=36.87..970.80 rows=551 width=41)
Hash Cond: (s.artistid = a.id)
-> Hash Right Join (cost=18.56..951.03 rows=551 width=38)
Hash Cond: (p.songid = s.id)
-> Seq Scan on playedonradio p (cost=0.00..809.30 rows=46630
width=12)
-> Hash (cost=18.46..18.46 rows=8 width=34)
-> Seq Scan on songs s (cost=0.00..18.46 rows=8 width=34)
Filter: (danceability >= '0.9'::double precision)
-> Hash (cost=10.36..10.36 rows=636 width=19)
-> Seq Scan on artists a (cost=0.00..10.36 rows=636 width=19)
-> Sort (cost=166.01..169.27 rows=1305 width=8)
Sort Key: b.songid
-> Seq Scan on bilboard b (cost=0.00..98.47 rows=1305 width=8)
Filter: (rank <= 10)
```

(c)

```
create index query1 on songs(danceability);
create index query1rank on bilboard(rank);
```

(d)

```
Sort (cost=1260.83..1263.42 rows=1033 width=45)
Sort Key: (count(DISTINCT p.id)) DESC, s.name, a.name
-> GroupAggregate (cost=1131.34..1209.12 rows=1033 width=45)
Group Key: s.id, a.name
-> Incremental Sort (cost=1131.34..1191.04 rows=1033 width=41)
Sort Key: s.id, a.name
Presorted Key: s.id
-> Merge Join (cost=1131.16..1150.97 rows=1033 width=41)
Merge Cond: (s.id = b.songid)
-> Sort (cost=991.92..993.30 rows=551 width=41)
Sort Key: s.id
-> Hash Join (cost=32.90..966.83 rows=551 width=41)
Hash Cond: (s.artistid = a.id)
-> Hash Right Join (cost=14.59..947.07 rows=551 width=38)
Hash Cond: (p.songid = s.id)
-> Seq Scan on playedonradio p (cost=0.00..809.30 rows=46630
width=12)
-> Hash (cost=14.49..14.49 rows=8 width=34)
-> Bitmap Heap Scan on songs s (cost=4.34..14.49 rows=8 width=34)
Recheck Cond: (danceability >= '0.9'::double precision)
-> Bitmap Index Scan on query1 (cost=0.00..4.33 rows=8 width=0)
Index Cond: (danceability >= '0.9'::double precision)
-> Hash (cost=10.36..10.36 rows=636 width=19)
-> Seq Scan on artists a (cost=0.00..10.36 rows=636 width=19)
-> Sort (cost=139.24..142.50 rows=1305 width=8)
Sort Key: b.songid
-> Bitmap Heap Scan on bilboard b (cost=18.40..71.71 rows=1305
width=8)
Recheck Cond: (rank <= 10)
-> Bitmap Index Scan on query1rank (cost=0.00..18.07 rows=1305
width=0)
Index Cond: (rank <= 10)
```