1.
query: select * from section;

chosen plan: "Seq Scan on section (cost=0.00..2.00 rows=100 width=28) (actual time=0.014..0.027 rows=100 loops=1)"
"Planning time: 0.063 ms"
"Execution time: 0.058 ms"

reason: simple select statement uses seq scanning in file.

2.
query: create index nam_idx on student(name);
explain analyze select * from student where name='Schrefl';

chosen plan: "Bitmap Heap Scan on student (cost=4.31..14.16 rows=4 width=24) (actual time=0.037..0.040 rows=4 loops=1)"
" Recheck Cond: ((name)::text = 'Schrefl'::text)"
" Heap Blocks: exact=3"
" -> Bitmap Index Scan on nam_idx (cost=0.00..4.31 rows=4 width=0) (actual time=0.028..0.029 rows=4 loops=1)"
"   Index Cond: ((name)::text = 'Schrefl'::text)"
"Planning time: 1.328 ms"
"Execution time: 0.079 ms"

reason: searching for some value after creating an index does an Bitmap heap scan in the created index.

3. query: select * from student where name='rumat' and id='0000';

chosen plan: "Index Scan using name_idx on student (cost=0.28..8.30 rows=1 width=24) (actual time=0.022..0.022 rows=0 loops=1)"
" Index Cond: ((name)::text = 'rumat'::text)"
" Filter: ((id)::text = '0000'::text)"
"Planning time: 0.779 ms"
"Execution time: 0.045 ms"

reason: searching for two values simultaneously does an index scan and then filters for the other attribute.

4.
query: select * from takes, student
where (takes.id, takes.course_id) = ('83314', '960') and student.id = '0000';

chosen plan: "Nested Loop (cost=0.56..16.61 rows=1 width=48) (actual time=0.069..0.069 rows=0 loops=1)"
" -> Index Scan using takes_pkey on takes (cost=0.29..8.31 rows=1 width=24) (actual time=0.043..0.047 rows=2 loops=1)"
"   Index Cond: (((id)::text = '83314'::text) AND ((course_id)::text = '960'::text))"

" -> Index Scan using student_pkey on student (cost=0.28..8.29 rows=1 width=24) (actual time=0.009..0.009 rows=0 loops=2)"
"   Index Cond: ((id)::text = '0000'::text)"
"Planning time: 1.030 ms"
"Execution time: 0.131 ms"

reason: searching for a tuple in a relation does multiple index scans inside a nested loop.

5.
query:
select * from (select * from takes order by id) as temp_s , (select * from student order by id) as temp_t
where temp_t.id = temp_s.id;

chosen plan:
"Merge Join (cost=0.56..3011.55 rows=30000 width=48) (actual time=0.031..46.859 rows=30000 loops=1)"
" Merge Cond: ((student.id)::text = (takes.id)::text)"
" -> Index Scan using student_pkey on student (cost=0.28..130.27 rows=2000 width=24) (actual time=0.013..1.680 rows=2000 loops=1)"
" -> Materialize (cost=0.29..2481.28 rows=30000 width=24) (actual time=0.011..30.048 rows=30000 loops=1)"
"   -> Index Scan using takes_pkey on takes (cost=0.29..2106.28 rows=30000 width=24) (actual time=0.009..24.540 rows=30000 loops=1)"
"Planning time: 0.458 ms"
"Execution time: 48.936 ms"

reason: using order by clause with join query uses merge join (as given in the hint)

6.
after adding limit 10 to the previous query, the plan:

"Limit (cost=0.56..1.57 rows=10 width=48) (actual time=0.032..0.052 rows=10 loops=1)"
" -> Merge Join (cost=0.56..3011.55 rows=30000 width=48) (actual time=0.032..0.049 rows=10 loops=1)"
"   Merge Cond: ((student.id)::text = (takes.id)::text)"
"   -> Index Scan using student_pkey on student (cost=0.28..130.27 rows=2000 width=24) (actual time=0.017..0.017 rows=1 loops=1)"
"   -> Materialize (cost=0.29..2481.28 rows=30000 width=24) (actual time=0.010..0.023 rows=10 loops=1)"
"   -> Index Scan using takes_pkey on takes (cost=0.29..2106.28 rows=30000 width=24) (actual time=0.007..0.018 rows=10 loops=1)"
"Planning time: 3.247 ms"
"Execution time: 0.100 ms"

since the algorithm did not change,

a different query where it changes:

select * from prereq natural join takes order by course_id;

chosen plan before adding limit 10:
"Merge Join (cost=2759.39..3191.53 rows=28481 width=28) (actual time=20.703..26.118 rows=16860 loops=1)"
" Merge Cond: ((prereq.course_id)::text = (takes.course_id)::text)"
" -> Sort (cost=5.32..5.57 rows=100 width=8) (actual time=0.066..0.075 rows=100 loops=1)"
"   Sort Key: prereq.course_id"
"   Sort Method: quicksort Memory: 29kB"
"   -> Seq Scan on prereq (cost=0.00..2.00 rows=100 width=8) (actual time=0.004..0.009 rows=100 loops=1)"
" -> Sort (cost=2750.90..2825.90 rows=30000 width=24) (actual time=20.496..22.001 rows=34008 loops=1)"
"   Sort Key: takes.course_id"
"   Sort Method: quicksort Memory: 3112kB"
"   -> Seq Scan on takes (cost=0.00..520.00 rows=30000 width=24) (actual time=0.004..4.811 rows=30000 loops=1)"
"Planning time: 4.190 ms"
"Execution time: 26.789 ms"

chosen plan after adding limit 10:

"Limit (cost=0.43..9.17 rows=10 width=28) (actual time=3.165..3.199 rows=10 loops=1)"
" -> Nested Loop (cost=0.43..24904.39 rows=28481 width=28) (actual time=3.164..3.197 rows=10 loops=1)"
"   -> Index Only Scan using prereq_pkey on prereq (cost=0.14..13.64 rows=100 width=8) (actual time=1.953..1.954 rows=2 loops=1)"
"   Heap Fetches: 2"
"   -> Index Scan using takes_pkey on takes (cost=0.29..245.38 rows=353 width=24) (actual time=0.603..0.619 rows=5 loops=2)"
"   Index Cond: ((course_id)::text = (prereq.course_id)::text)"
"Planning time: 2.702 ms"
"Execution time: 3.222 ms"

hence after adding limit 10, the algorithm changes from merge join to index scan inside nested loop.

Reason: takes is a small relation, hence the nested is preferred for lesser time than merge join.

7.
time taken for creating index is 142 ms
time taken for dropping index is 2 ms

8.
8.1
begin;
Query returned successfully with no result in 15 msec

8.2
"Delete on course (cost=0.00..4.50 rows=1 width=6) (actual time=0.061..0.061 rows=0 loops=1)"
" -> Seq Scan on course (cost=0.00..4.50 rows=1 width=6) (actual time=0.012..0.035 rows=1 loops=1)"
"   Filter: ((course_id)::text = '400'::text)"
"   Rows Removed by Filter: 199"
"Planning time: 1.406 ms"
"Trigger for constraint section_course_id_fkey on course: time=1.397 calls=1"
"Trigger for constraint prereq_course_id_fkey on course: time=1.587 calls=1"
"Trigger for constraint prereq_prereq_id_fkey on course: time=0.380 calls=1"
"Trigger for constraint teaches_course_id_fkey on section: time=2.255 calls=2"
"Trigger for constraint takes_course_id_fkey on section: time=9.364 calls=2"
"Execution time: 15.075 ms"

8.3
rollback;
Query returned successfully with no result in 11 msec.

8.4
create index ind1 on section(course_id);
Query returned successfully with no result in 22 msec.

create index ind2 on prereq(course_id);
Query returned successfully with no result in 23 msec.

create index ind3 on prereq(prereq_id);
Query returned successfully with no result in 23 msec.

create index ind4 on teaches(course_id, sec_id, semester, year);
Query returned successfully with no result in 22 msec.

create index ind5 on takes(course_id, sec_id, semester, year);
Query returned successfully with no result in 142 msec.

8.5
delete index ind1;
Query returned successfully with no result in 2 msec.

Delete index ind2;
Query returned successfully with no result in 0 msec.

delete index ind3;
Query returned successfully with no result in 1 msec.

delete index ind4;
Query returned successfully with no result in 12 msec.

delete index ind5;
Query returned successfully with no result in 8 msec.

8.6
while executing drop, the time is lesser because we postgresql does not read the whole data, it can only drop the allocated data directly.