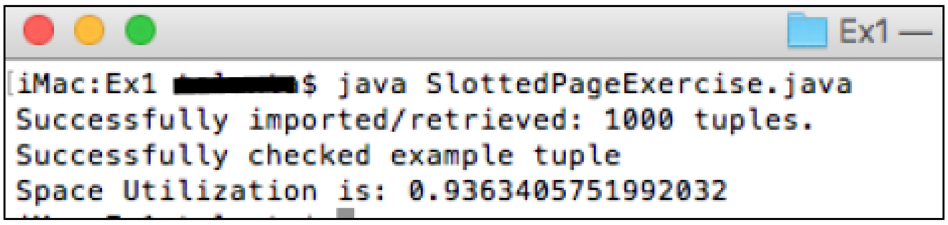
Question 1:



```
iMac:Ex1 [redacted]$ java SlottedPageExercise.java
Successfully imported/retrieved: 1000 tuples.
Successfully checked example tuple
Space Utilization is: 0.9363405751992032
```

Question 2:

a)

PRS:

t1	t2	t3	t4	t5	t6	t7	t8	t9	t10	t11	t12
〈 (T1,A)	(T1,B)	(T1,E)	(T2,F)	(T1,C)	(T1,D)	(T2,D)	(T2,E)	(T2,C)	(T1,G)	(T1,D)	(T1,A)

t13	t14	t15	t16	t17
(T1,F)	(T1,F)	(T1,C)	(T2,A)	(T2,B) 〉

Contains these SRS:

1. (AB) -> LRS = 2
2. (EF) -> LRS = 2
3. (CDDE) -> LRS = 3
4. (C) -> LRS = 1
5. (G) -> LRS = 1
6. (D) -> LRS = 1
7. (A) -> LRS = 1
8. (FF) -> LRS = 1
9. (C) -> LRS = 1
10. (AB) -> LRS = 2

Sequentiality:
S(1) = 0.6
S(2) = 0.9
S(3) = 1.0

Locality:

T1: W(t1,3) = {A}	w(t1,3) = 1
T2: W(t1,3) = {}	w(t1,3) = 0
T1: W(t2,3) = {AB}	w(t2,3) = 2
T2: W(t2,3) = {}	w(t2,3) = 0
T1: W(t3,3) = {ABE}	w(t3,3) = 3
T2: W(t3,3) = {}	w(t3,3) = 0
T1: W(t4,3) = {ABE}	w(t4,3) = 3
T2: W(t4,3) = {F}	w(t4,3) = 1
T1: W(t5,3) = {BEC}	w(t5,3) = 3
T2: W(t5,3) = {F}	w(t5,3) = 1

T1: W(t6,3) = {ECD}   w(t6,3) = 3
T2: W(t6,3) = {F}   w(t6,3) = 1
T1: W(t7,3) = {ECD}   w(t7,3) = 3
T2: W(t7,3) = {FD}   w(t7,3) = 2
T1: W(t8,3) = {ECD}   w(t8,3) = 3
T2: W(t8,3) = {FDE}   w(t8,3) = 3
T1: W(t9,3) = {ECD}   w(t9,3) = 3
T2: W(t9,3) = {DEC}   w(t9,3) = 3
T1: W(t10,3) = {CDG}   w(t10,3) = 3
T2: W(t10,3) = {DEC}   w(10t,3) = 3
T1: W(t11,3) = {DG}   w(t11,3) = 2
T2: W(t11,3) = {DEC}   w(t11,3) = 3
T1: W(t12,3) = {GDA}   w(t12,3) = 3
T2: W(t12,3) = {DEC}   w(t12,3) = 3
T1: W(t13,3) = {DAF}   w(t13,3) = 3
T2: W(t13,3) = {DEC}   w(t13,3) = 3
T1: W(t14,3) = {AF}   w(t14,3) = 2
T2: W(t14,3) = {DEC}   w(t14,3) = 3
T1: W(t15,3) = {FC}   w(t15,3) = 2
T2: W(t15,3) = {DEC}   w(t15,3) = 3
T1: W(t16,3) = {FC}   w(t16,3) = 2
T2: W(t16,3) = {ECA}   w(t16,3) = 3
T1: W(t17,3) = {FC}   w(t17,3) = 2
T2: W(t17,3) = {CAB}   w(t17,3) = 3
Sum: 78
window size = 3
n = 34
AL(3)= (78/3)/34 = 13/17 = 0.765

b)

Sum: 562+445+408+373+144+115+102+78+31+20 = 2278

i) How small can the buffer be, if we want no more than 8% of cache-misses?

2278 * 0.08 = 182.24

Stack depth 6 is the smallest option that complies with not having more than 8% cache-misses.

ii) How many cache hits (in %) will there be, given a buffer size of 4?

(562+445+408+373)/2278 = 0.785

There will be 78.5 % hits with a buffer size of 4.

iii) How many cache misses are avoided, if the buffer size is increased from 4 to 6?

144+115 = 259

259 misses are avoided.

Question 3:
a)

D, B, A, C, B, B, C, A, D, D, C, B, C, A, C, B, A

Clock:	LRU-k (k=2)	GClock
- empty - D miss D1 - B miss D1B1 - A miss D1B1A1 - C miss B0A0C1 - B hit B1A0C1 - B hit B1A0C1 - C hit B1A0C1 - A hit B1A1C1 - D miss A0C0D1 - D hit A0C0D1 - C hit A0C1D1 - B miss C1D1B1 - C hit C1D1B1 - A miss	- empty - D miss D(inf) - B miss D(inf)B(inf) - A miss D(inf)B(inf)A(inf) - C miss B(inf)A(inf)C(inf) - B hit A(inf)C(inf)B(3) - B hit A(inf)C(inf)B(1) - C hit A(inf)C(3)B(2) - A hit A(5)C(4)B(3) - D miss A(6)C(5)B(4) - D miss C(6)B(5)D(1) - C hit B(6)C(4)D(2) - B hit B(6)C(5)D(3) - C hit B(7)D(4)C(2) - A miss	- empty - D miss D(2) - B miss D(2)B(2) - A miss D(2)B(2)A(2) -C miss B(0)A(0)C(2) -B hit B(1)A(0)C(2) -B hit B(2)A(0)C(2) -C hit B(2)A(0)C(3) - A hit B(2)A(1)C(3) -D miss B(0)C(2)D(2) -D hit B(0)C(2)D(3) -C hit B(0)C(3)D(3) - B hit B(1)C(3)D(3) -C hit B(1)C(4)D(3) -A miss

D0B0A1
- C miss
B0A1C1
- B hit
B1A1C1
- A hit
B1A1C1
 	
-> 8 misses
	A(6)D(5)C(3)
	- C hit
	A(7)D(6)C(2)
	- B miss
	D(7)B(4)C(3)
	- A miss
	B(5)C(4)A(3)
	
	-> 9 misses
		C(3)D(2)A(2)
		-C hit
		C(4)D(2)A(2)
		-B miss
		C(1)A(0)B(2)
		-A hit
		C(1)A(1)B(2)
		
		-> 7 misses

b) optimal replacement strategy:

-
empty
- D miss
D
- B miss
DB
- A miss
DAB
- C miss
ABC
- B hit
ABC
- B hit
ABC
- C hit
ABC
- A hit
ABC
- D miss
ABC
- D miss
ABC
- C hit
ABC
- B hit
ABC
- C hit
ABC
- A hit
ABC
- C hit
ABC
- B hit
ABC
- A hit

ABC

-> 6 misses

c)

1. FIFO has less misses than LRU

PRS: ABACB	
FIFO	LRU
-	-
empty	empty
-A miss	-A miss
A	A
-B miss	-B miss
AB	AB
-A hit	-A hit
AB	BA
-C miss	-C miss
BC	AC
-B hit	-B miss
BC	CB
-> 3 misses	-> 4 misses

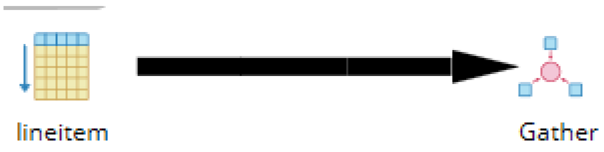
2. FIFO has less hits than LRU

PRS: ABACA	
FIFO	LRU
-	-
empty	empty
-A miss	-A miss
A	A
-B miss	-B miss
AB	AB
-A hit	-A hit
AB	BA

-C miss BC -A miss CA -> 1 hit	-C miss AC -A hit AC -> 2 hits

Question 4:

a) **SELECT * FROM lineitem WHERE l_suppkey < 10**



```
"[
  {
    "Plan": {
      "Node Type": "Gather",
      "Parallel Aware": false,
      "Startup Cost": 1000,
      "Total Cost": 145325.81,
      "Plan Rows": 5677,
      "Plan Width": 117,
      "Workers Planned": 2,
      "Single Copy": false,
      "Plans": [
        {
          "Node Type": "Seq Scan",
          "Parent Relationship": "Outer",
          "Parallel Aware": true,
          "Relation Name": "lineitem",
          "Alias": "lineitem",
          "Startup Cost": 0,
          "Total Cost": 143758.11,
          "Plan Rows": 2365,
          "Plan Width": 117,
          "Filter": "(l_suppkey < 10)"
        }
      ]
    }
  }
]"
```

Description:

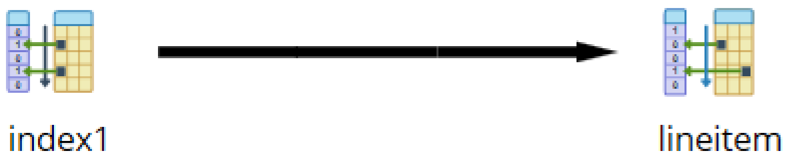
According to the query plan is realized in two steps. First, a Sequential scan has happened on table lineitem. The cost of sequential scan is 143758.11. If there would Index existed on the column used in WHERE clause, planner would might use Index scan only resulting in less Cost. Also, we can see that Filter is applied in the first node, which also adds cost for CPU computation for the WHERE clause. In the second the Gathering process is done on selected tuples earlier. It adds only small amount of Cost.

b) CREATE INDEX index1 ON lineitem(l_suppkey);

```
CREATE INDEX

Query returned successfully in 2 min 49 secs.
```

Query plan:



"[
  {
    "Plan": {
      "Node Type": "Bitmap Heap Scan",
      "Parallel Aware": false,
      "Relation Name": "lineitem",
      "Alias": "lineitem",
      "Startup Cost": 108.43,
      "Total Cost": 18645.28,
      "Plan Rows": 5677,
      "Plan Width": 117,
      "Recheck Cond": "(l_suppkey < 10)",
      "Plans": [
        {
          "Node Type": "Bitmap Index Scan",
          "Parent Relationship": "Outer",
          "Parallel Aware": false,
          "Index Name": "index1",
          "Startup Cost": 0,
          "Total Cost": 107.01,
          "Plan Rows": 5677,
          "Plan Width": 0,
          "Index Cond": "(l_suppkey < 10)"
        }
      ]
    }
  }
]"

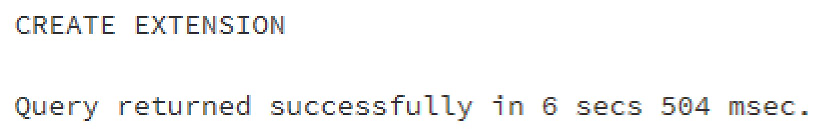
Description:
After creating an Index the Total cost has decreased drastically as expected. That happened because in the first step the Planner used Bitmap Index Scan. Here the Filter condition was used as Index condition. In the second step it simply applied Bitmap Heap Scan and rechecked the condition. Overall the usage of Index has decreased the Total Cost for almost 8 times.

c) **EXPLAIN (ANALYZE, BUFFERS) SELECT * FROM orders WHERE o_totalprice < 100**

c.1.
```
"Gather  (cost=1000.00..34922.50 rows=150 width=107) (actual time=28273.722..31215.309 rows=0 loops=1)"
"  Workers Planned: 2"
"  Workers Launched: 2"
"  Buffers: shared hit=2080 read=24015"
"  ->  Parallel Seq Scan on orders  (cost=0.00..33907.50 rows=62 width=107) (actual time=24361.952..24361.953 rows=0 loops=3)"
"        Filter: (o_totalprice < '100'::numeric)"
"        Rows Removed by Filter: 500000"
"        Buffers: shared hit=2080 read=24015"
"Planning Time: 1.238 ms"
"Execution Time: 31215.336 ms"
```
c.2.
```
"Gather  (cost=1000.00..34922.50 rows=150 width=107) (actual time=7560.713..9810.292 rows=0 loops=1)"
"  Workers Planned: 2"
"  Workers Launched: 2"
"  Buffers: shared hit=2176 read=23919"
"  ->  Parallel Seq Scan on orders  (cost=0.00..33907.50 rows=62 width=107) (actual time=2518.905..2518.905 rows=0 loops=3)"
"        Filter: (o_totalprice < '100'::numeric)"
"        Rows Removed by Filter: 500000"
"        Buffers: shared hit=2176 read=23919"
"Planning Time: 0.107 ms"
"Execution Time: 9810.314 ms"
```
c.3.
```
"Gather  (cost=1000.00..34922.50 rows=150 width=107) (actual time=8331.435..11428.874 rows=0 loops=1)"
"  Workers Planned: 2"
"  Workers Launched: 2"
"  Buffers: shared hit=2240 read=23855"
"  ->  Parallel Seq Scan on orders  (cost=0.00..33907.50 rows=62 width=107) (actual time=2471.418..2471.418 rows=0 loops=3)"
"        Filter: (o_totalprice < '100'::numeric)"
"        Rows Removed by Filter: 500000"
"        Buffers: shared hit=2240 read=23855"
"Planning Time: 0.067 ms"
"Execution Time: 11428.891 ms"
```

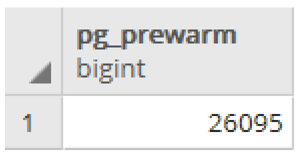
Description:
Above given the results of 3 Planner analysis for the same query run 3 times. Overall all plans are samely consists of two steps. In the first step a Parallel Sequential Scan is performed on Orders table and Filter is applied. In the second step final Gathering of tuples happened.
Here we can see that in consequent runs of the same query the Buffers shared hit had increased, which also resulted in decrease of the actual run time of the query. It means that more pages already fetched from the Disk are retained in the Buffer.

d) **CREATE EXTENSION pg_prewarm;**
SELECT pg_prewarm('orders');
EXPLAIN (ANALYZE, BUFFERS) SELECT * FROM orders WHERE o_totalprice < 100



```
CREATE EXTENSION

Query returned successfully in 6 secs 504 msec.
```



	pg_prewarm bigint
1	26095

```
"Gather  (cost=1000.00..34922.50 rows=150 width=107) (actual time=8803.948..10644.799 rows=0 loops=1)"
"  Workers Planned: 2"
"  Workers Launched: 2"
"  Buffers: shared hit=15509 read=10586"
"  ->  Parallel Seq Scan on orders  (cost=0.00..33907.50 rows=62 width=107) (actual time=1902.709..1902.709 rows=0 loops=3)"
"        Filter: (o_totalprice < '100'::numeric)"
"        Rows Removed by Filter: 500000"
"        Buffers: shared hit=15509 read=10586"
"Planning Time: 0.101 ms"
"Execution Time: 10644.819 ms"
```

Description:
In Table prewarming 26095 pages were processed. As a result, we can see in a Query plan that the most of the pages were fetched from the Buffer. But, here we still have reads from the Disk. Despite running the pg_prewarm, depending on the Buffer management strategies it could happen that it did not retain all pages in the Buffer. Therefore we still see some reads from the Disk.