Names: Bob Skowron, Jason Walker
Keys: rskowron, jwalker
SVN: jwalker: https://svn.seas.wustl.edu/repositories/jwalker/cse427s_fl17/

1. a. Define the transition matrix M:

	a	**b**	**c**
a	.333	.333	.333
b	.5	0	.5
c	0	.5	.5

Transpose M and multiply by v = $[.333.333.333]^T$

Iterating this multiplication gives:

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
a	0.278	0.231	0.235	0.230	0.231	0.231	0.231	0.231	0.231	0.231
b	0.278	0.315	0.304	0.309	0.307	0.308	0.308	0.308	0.308	0.308
c	0.444	0.454	0.461	0.461	0.462	0.461	0.462	0.462	0.462	0.462

b. Here we introduce $\beta = .8$. The matrix M and vector v from above still apply.
Iterating, we get:

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
a	0.289	0.259	0.261	0.259	0.259	0.259	0.259	0.259	0.259	0.259
b	0.289	0.313	0.307	0.309	0.309	0.309	0.309	0.309	0.309	0.309
c	0.422	0.428	0.432	0.432	0.432	0.432	0.432	0.432	0.432	0.432

c. a.

Source	**Degree**	**Destinations**
A	3	B,C,D
B	2	A,D
C	1	E
D	2	B,C

b.

Source	**Degree**	**Destinations**
A	3	A,B,C
B	2	A,B
C	2	B,C

Using 4-byte integers for coordinates of an element and an 8-byte double for the value, then we need 16 bytes per nonzero entry. If we list the nonzero entries by column, however, we can thus represent a column by one integer for the out-degree, and one integer per nonzero entry in that column, giving the row number where that entry is located. This is only 4 bytes per nonzero entry and 4 bytes for the column.

Names: Bob Skowron, Jason Walker
Keys: rskowron, jwalker
SVN: jwalker: https://svn.seas.wustl.edu/repositories/jwalker/cse427s_fl17/

2. a. M = RDD of (columnId, (rowId, value)); v = RDD of (id, value)
mmult = join v to M
flatMap to get (rowId, M_value * v_value)
reduceByKey (add up values)

b. Stages are operations that can run on the same data partitioning in parallel across executors/nodes. Tasks within a stage are operations executed by one executor/node that are pipelined together.
Stage1: join, flatMap Stage2: reduceByKey

c. RDDs:
M ={(1, (1, 16)), (2, (1, 2)), (3, (1, 3)), (4, (1, 13))
(1, (2, 5)), (2, (2, 11)), (3, (2, 10)), (4, (2, 8))
(1, (3, 9)), (2, (3, 7)), (3, (3, 6)), (4, (3, 12))
(1, (4, 4)), (2, (4, 14)), (3, (4, 15)), (4, (4, 1))}
v = {(1, 1), (2, 2), (3, 3), (4, 4)}

d. Results:
join: {((1, (1, 16)), (1, 1)), ((2, (1, 2)), (2, 2)), ((3, (1, 3)), (3, 3)), ((4, (1, 13)), (4, 4)) ((1, (2, 5)) , (1, 1)), ((2, (2, 11)) , (2, 2)), ((3, (2, 10)) , (3, 3)), ((4, (2, 8)) , (4, 4)) ((1, (3, 9)) , (1, 1)), ((2, (3, 7)) , (2, 2)), ((3, (3, 6)) , (3, 3)), ((4, (3, 12)) , (4, 4)) ((1, (4, 4)) , (1, 1)), ((2, (4, 14)) , (2, 2)), ((3, (4, 15)) , (3, 3)), ((4, (4, 1)) , (4, 4))}
flatMap: {(1,16*1),(1,2*2),(1,3*3),(1,13*4)
(2,5*1),(2,11*2),(2,10*3),(2,8*4)
(3,9*1),(3,7*2),(3,6*3),(3,12*4)
(4,4*1),(4,14*2),(4,15*3),(4,1*4)}
reduceByKey:{(1,81)
(2,89)
(3,89)
(4,81)}

e. Input = (m * n) + n
join = (m*2n)
flatMap = (m*n)
reduceByKey = (n)
Total = 2(m*n)+ (m*2n)+2n

f. A potential memory bottleneck could occur during the re-partition between the flatMap and the reduceByKey.

Names: Bob Skowron, Jason Walker
Keys: rskowron, jwalker
SVN: jwalker: https://svn.seas.wustl.edu/repositories/jwalker/cse427s_fl17/

3. a. See SVN

   b. *spark-submit CountJPGs.py /loudacre/weblogs*
   Number of JPGs: 64,978
   The Driver program and processing are done locally. The result (print statement) is output to the command line.

   c. *spark-submit –master yarn-client CountJPGs.py /loudacre/weblogs*
   The Driver program is run locally. The processing is done on the cluster. The result (print statement) is output to the command line.

   d. 1 stage and 311 tasks were executed

   e. *spark-submit –master yarn-cluster CountJPGs.py /loudacre/weblogs*
   The Driver program and processing are completed on the cluster. The result (print statement) is output to a log file on the cluster.

Names: Bob Skowron, Jason Walker
Keys: rskowron, jwalker
SVN: jwalker: https://svn.seas.wustl.edu/repositories/jwalker/cse427s_fl17/

4. a. To generate a sample of the data you can use the unix commands head or tail to create a file with a specified subset of data. Using PIG, you could load the data and then write out a sample file with LIMIT and STORE. Both of these would only take the first or last n records. If you wanted to generate a more representative sample, you could use PIG with SAMPLE or unix with shuf.

   It is much faster to test PIG scripts with a local subset since PIG will generate a MapReduce job based on the script. If you run the script on the cluster with the full set of data, this would be equivalent to running an entire MR job on the data which could take a long time.

   b. *hadoop fs -cat /dualcore/ad_data1/part* | head -100 >test_ad_data.txt*

   c. (diskcentral.example.com,68)
   (megawave.example.com,96)
   (megasource.example.com,100)
   (salestiger.example.com,141)

   d. (bassoonenthusiast.example.com,1246)
   (grillingtips.example.com,4800)
   (footwear.example.com,4898)
   (cofeenews.example.com,5106)

Names: Bob Skowron, Jason Walker
Keys: rskowron, jwalker
SVN: jwalker: https://svn.seas.wustl.edu/repositories/jwalker/cse427s_fl17/

5. a. Included later

   b. (TABLET,3193033)
   (DUALCORE,2888747)
   (DEAL,2717098)

Names: Bob Skowron, Jason Walker
Keys: rskowron, jwalker
SVN: jwalker: https://svn.seas.wustl.edu/repositories/jwalker/cse427s_fl17/

PIG script for Problem 4, parts (c) and (d)

```
-- TODO (A): Replace 'FIXME' to load the test_ad_data.txt file.
--data = LOAD 'test_ad_data.txt' AS (campaign_id:chararray,
--             date:chararray, time:chararray,
--             keyword:chararray, display_site:chararray,
--             placement:chararray, was_clicked:int, cpc:int);

-- load with file patterns
data = LOAD '/dualcore/ad_data[0-9]/part*' AS (campaign_id:chararray,
             date:chararray, time:chararray,
             keyword:chararray, display_site:chararray,
             placement:chararray, was_clicked:int, cpc:int);


-- TODO (B): Include only records where was_clicked has a value of 1
wasClickedOne = FILTER data BY was_clicked == 1;


-- TODO (C): Group the data by the appropriate field
siteGroup = GROUP wasClickedOne BY display_site;


/* TODO (D): Create a new relation which includes only the
 *           display site and the total cost of all clicks
 *           on that site
 */
siteTotalCost = FOREACH siteGroup GENERATE group as display_site, SUM(wasClickedOne.cpc) AS total_cost;


-- TODO (E): Sort that new relation by cost (ascending)
orderedTotalCost = ORDER siteTotalCost BY total_cost ASC;

-- TODO (F): Display just the first three records to the screen
top4Records = LIMIT orderedTotalCost 4;
DUMP top4Records;
```

Names: Bob Skowron, Jason Walker
Keys: rskowron, jwalker
SVN: jwalker: https://svn.seas.wustl.edu/repositories/jwalker/cse427s_fl17/

PIG script for Problem 5, part (a)

```
-- load data
data = LOAD '/dualcore/ad_data[0-9]/part*' AS (campaign_id:chararray,
             date:chararray, time:chararray,
             keyword:chararray, display_site:chararray,
             placement:chararray, was_clicked:int, cpc:int);

-- group data by keyword
keywordGroup = GROUP data BY keyword;

-- calculate total cost
keywordTotalCost = FOREACH keywordGroup GENERATE group as keyword, SUM(data.cpc) AS total_cost;

-- sort descending
orderedTotalCost = ORDER keywordTotalCost BY total_cost DESC;

-- display top 3 records
top3Records = LIMIT orderedTotalCost 3;
DUMP top3Records;
```