Question 1

(a)

1. Spark is based on memory, trying to do tasks and keep more data on memory, while MapReduce has to read from and write to disk.
2. Spark has rich data source such Local files, S3, HDFS, HBase, but MapReduce is built on HDFS.
3. Spark is multi-threaded framework, while MapReduce executor is based on multi-processes.
4. Spark supports SparkSQL, GraphX, SparkStreaming and MLlib module, it is more suitable and efficient to do iterative computation, real-time streaming tasks, graph tasks, machine learning task, compared with MapReduce.

(b)

In distributed system, there will be latency in data consistency across different nodes/partitions. If we guaranteed consistency, every node/partition has to wait for data update before reading data, which will be conflict with high availability. On the other hand, if we guaranteed availability, if the data changes in one node, and accessing data in other node, the data we read will not be latest, which is conflicting with consistency.

Question 2

(a)

```
class Question2()
    method mapper(_,line):
        words = line.split(" ")
        for word in words:
            yield word,1

    method combiner(self, word, counts):
         yield word,sum(counts)

    method reducer_init():
        self.top_k_list = []

    method reducer(word, counts):
        total_counts = sum(counts)
        if len(self.top_k_list) <= k:
            self.top_k_list.append((word,total_counts))
        else:
            # assume we can find the word in top_k_list with minimal occurrence by function min_occurrence()
            if total_counts > min_occurrence(self.top_k_list):

                # replace the word in top-k_list with minimal occurrence with current word and its occurrence
                replace(self.top_k_list, (word, total_counts))

     method reducer_final():
       // we sort by count first, then sort by term
       top_k_list = self.top_k_list.sort(lambda x => x[1],x[0])
       for i in top_k_list:
           word = i[0]
           counts = i[1]
           emit (word, counts)

     job_step { mapper ,combiner ,reducer_init,reducer,reducer_final}

     conf = {
        'mapreduce.map.output.key.field.separator': ' ',
        'mapreduce.partition.keypartitioner.options': '-k1,1',
        'partitioner': 'org.apache.hadoop.mapred.lib.KeyFieldBasedPartitioner',
        'mapreduce.job.output.key.comparator.class': 'org.apache.hadoop.mapreduce.lib.partition.KeyFieldBasedComparator',
        'mapreduce.partition.keycomparator.options': '-k1,1'
     }
```

Question 3

1.

```
val temp: RDD[(String, Iterable[Int])] = docs.flatMap(x => {x._2.split(" ").map(word
=> (word,x._1))}).groupByKey()

// sort by term, docID sorted increasingly
val InvList: RDD[(String, List[Int])] = temp.sortByKey().map(x =>(x._1,
x._2.toList.sortBy(x => x)))
```

2.

```
// init src node attribute to be -1
val initialGraph = graph.mapVertices((id, _) =>
  if (id == s) -1  else 0)

// node with -1 means reachable from src
val node_reachable = initialGraph.pregel(initialMsg = 0)(
  (id, dist, newDist) => {
      if (newDist == -1)  // curr node is reachable, update current node attribute
        newDist
      else
        dist  // received value is not -1, no update
  },
  // sendMsg
  triplet =>{
    if (triplet.srcAttr == -1) {
        Iterator((triplet.dstId,-1)) // send -1 to neighbour, if curr node's attribute
is -1
    } else
      Iterator.empty
  }
  ,
  // mergeMsg
  (x, y) => {
    if (x == -1 || y == -1) -1
    else 0 // any value is ok, we only care -1
  }
)

// count how many -1 in all vertices, -1 for src node
val result = node_reachable.vertices.collect().reduce((x,y) => (x._2+y._2)) ...
val final = result - 1
```

Question 4

(i)

2-shingles set for document A is

Set_A = {**the_sky**, **sky_is,** is_blue, blue_the, **the_sun**, sun_is, **is_bright**}

2-shingles set for document B is

Set_B = {**the_sun,** sun_in, in_the, **the_sky**, **sky_is**,**is_bright**}

Jaccard similarity = |Set_A ∩Set_B |/ |Set_A ∪ Set_B| = 4/9

(ii)

Matrix

	Doc A	Doc B
the_sky	1	1
sky_is	1	1
is_blue	1	0
blue_the	1	0
the_sun	1	1
sun_is	1	0
is_bright	1	1
sun_in	0	1
in_the	0	1

h1(n) = 5n - 1 mod 9

h2(n) = 2n + 1 mod 9

	Doc A	Doc B
	Infinite	Infinite
	Infinite	Infinite
h1(0)=8	8	8
h2(0)=1	1	1
h1(1)=4	4	4
h2(1)=3	1	1
h1(2)=0	0	4
h2(2)=5	1	1
h1(3)=5	0	4

h2(3)=7	1	1
h1(4)=1	0	1
h2(4)=0	0	0
h1(5)=6	0	1
h2(5)=2	0	0
h1(6)=2	0	1
h1(6)=4	0	0
h1(7)=7	0	1
h1(7)=6	0	0
h1(8)=3	0	1
h1(8)=8	0	0

Result signature:
docA doc B
0 1
0 0

(iii)
1 - (1 – t^r)^b = 1 – (1-0.6^2)^5

Question 5

1.

Initial bit array (bloom filter) of size 7

[0,0,0,0,0,0,0]

S = {“hi”,”big”,”data”}

For key “hi”:

h1(“hi”) = (7+8) mod 7 = 1

h2(“hi”) = 2 mod 7 = 2

Update bit 1 and bit 2 to be 1 in bit array.

bit array : [0,1,1,0,0,0,0]

For key “big”:

h1(“big”) = (1+8+6) mod 7 = 1

h2(“big”) = 3 mod 7 = 3

Update bit 1 and bit 3 to be 1 in bit array.

bit array : [0,1,1,1,0,0,0]

For key “data”:

h1(“data”) = (3+0+19+0) mod 7 = 1

h2(“data”) = 4 mod 7 = 4

Update bit 1 and bit 6 to be 1 in bit array.

bit array : [0,1,1,1,1,0,0]

2.

bit array from previous question: [0,1,1,1,1,0,0]

For key “comp”

h1(“comp”) = (2+14+12+15) mod 7 = 1

h2(“comp”) = 4 mod 7 = 4

The first bit and 4^{th} bit in bit array are 1, “comp” is contained in S.

3. False positive example: “c”

Question 6

(a)

It is a collaborative filtering-based recommendation. . The possible logic behind will be many similar users which has same taste as user 1 also like movie B, based on user-based CF, the Movie B is recommended.

If it is a content-based system, it would be more likely to recommend Movie C to the user, since the Movie C is more similar to user 1's interest than Movie B (e.g For Movie C, both 2000s and D2 are satisfied).

(b)

Items = [A,C,D,E], B no included

u1 = (2, 0, 4, 0), u2 = (1, 0, 4,0), u3 = (0, 0, 3, 2)

cos(u1, u2) = $\frac{u1 \cdot u2}{|u1|\,|u2|}$ = $\frac{2*1+0+4*4+0}{\sqrt{4+16}*\sqrt{1+16}}$ = 0.98

cos(u1, u3) = $\frac{u1 \cdot u3}{|u1|\,|u3|}$ = $\frac{0+0+4*3+0}{\sqrt{4+16}*\sqrt{9}}$ = 0.74

Sim(u1,u2) = 0.98
Sim(u1,u3) = 0.74

Assume N = 2, both user 2 and user 3 are the two most similar users to user 1.

The rating of use1 to movie B

$r_{u1,mb} = \frac{\sum_{y \in N} Sim(u1,y) * r_{y,mb}}{\sum_{y \in N} Sim(u1,y)} = \frac{0.98*3+0.74*5}{0.98+0.78} = 3.86 \approx 4$

The predicted rating of user 1 to movie B is 4.