Design Discussion (20 points total)
PreProcessor

```
Map(Key k,Value v)
{
pageName= get page name from value
filecontents= get filecontents from value
LinkedPages=xmlParser.parse(filecontents)

emit(pagename,graph)

For (String each linked page: LinkedPages)
{
emit (linkedpagename,emptygraph)
}


Reduce(key pagename,value graph)
{
every page , (dangling or non dangling) inceremt the global counter

if (graph)
{
emit (pagename,graph)
}
if (emptygraph)
{
emit (pagename,emptygraph)
}
```

The preprocess discussion:
All page names with a tilde is removed and wikiparser is called to find all its connected nodes ,outlinks to all other nodes.
If the node has no outlinks ,it is treated as a dangling , an empty graph structure is emitted for that node.
The reducer phase is used to sum up the total number of pages.

Page Rank
Mapper

```
Map(key pagename ,value node)
{

if (firstTime)
```

```
{
pagerank=1/totalPages;
}
else
{
pagerank=node.pagerank+delta/totalPages
}
node.pagerank=pagerank
emit (pageName,graph)
 if (node.adList is Empty)
{
emit (“DanglingNode”,node)
}else
{
for each outlinkNodes in node.adList
{
cnode=new Node();
cnode.pageRank=node.pagerank/node.adList.size()
emit (outlinkNodes,cnode)
}
}
}
```

Combiner(key pageName, Iterable<Node> nodes)

```
If (key==”Dangling”)
{
for each node in nodes
delta=delta+node.pagerank

node.pageRank=delta

emit (“Dangling”,node)
return

}Node n
 for(Node node:values)
                    {
//if it is not a dangling node maintain the graph
                        if (!node.isDangling)
                        {
                                n.pageLinks=node.pageLinks;
                        }
   //In Combiner all the contributiong pageRanks are added up

                        PageRankSoFar=PageRankSoFar+node.pageRank;
                    }
n.pageRank=PageRankSoFar;
```

```
emit(key, n);
```

Reducer

```
Reduce(key PageName, Iterable<Node> nodes)
If (key==”Dangling”)
{
for each node in nodes
delta=delta+node.pagerank
update global delta counter
}
for(Node node:values)
                {
//if it is not a dangling node maintain the graph
                        if (!node.isDangling)
                        {
                                n.pageLinks=node.pageLinks;
                        }
  //In Reducerreducer ,all the contributiong pageRanks are added up

                        PageRankSoFar=PageRankSoFar+node.pageRank;
                }

                //page rank is calculated with the formula
                double pageRank=(a/totalPages)+((1-a)*(PageRankSoFar));
                n.pageRank=pageRank;
        emit(key, n);
```

PageRank Discussion
In the Map phase ,
The contribution each node would make to its connected children is calculated .
For the first time , This contribution is calculated as 1/totalnumber of pages .
In the reduce phase all the the dangling nodes come together in one reduce call and all their page rank are accumulated and passed on to the next map phase and added to the page rank calculate in the i-1 iteration thus truly calculating page rank in the i+1 iteration.

TopK

Mapper

```
//the Mapper is responsible for for finding the local topK records
//algorithm taken from module .
```

```
priorityQueue q
setup ()
{
initialize the q
}

map(key,vaue)
q.add()
if (q.size>100)
{
q.remove()
}


cleanup
{
//emit the local 100 records
}
```

Reducer TopK

```
//The Reducer emits the topK global records
Reduce(Nullwritatle,topk)
q.add()
if (q.size>100)
{
q.remove()
}
```

Report the amount of data transferred from Mappers to Reducers, and from Reducers to HDFS, in each iteration of the PageRank computation. Does it change over time? (5 points)

NO, it does not change with each iteration .
It works with the same set of pages recalculating their PR over and over till the result is found after 10 iterations .

```
Map input records=15518
Map output records=100
Map output bytes=1819
Map output materialized bytes=2025
Input split bytes=118
Combine input records=0
Combine output records=0
Reduce input groups=1
Reduce shuffle bytes=2025
Reduce input records=100
Reduce output records=100
```

Run your program in Elastic MapReduce (EMR) on the four provided bz2 files, which comprise the full English Wikipedia data set from 2006, using the following two configurations:
Report for both configurations (i) pre-processing time, (ii) time to run ten iterations of PageRank, and (iii) time to find the top-100 pages. There should be 2*3=6 time values. (6 points)

Unsuceessful at aws run

Critically evaluate the runtime results by comparing them against what you had expected to see and discuss your findings. Make sure you address the following question: Which of the computation phases showed a good speedup? If a phase seems to show fairly poor speedup, briefly discuss possible reasons—make sure you provide concrete evidence, e.g., numbers from the log file or analytical arguments based on the algorithm's properties. (4 points)

Although I was unsuccessful in running in aws but I will answer the
PreProcess/PageRank
Preprocess and PageRank should show fairly good speedup as we can expect good distribution of work .
TopK
As compared to top k , increasing the number of reducer nodes would not result in good speedup as the reducer needs to wait for all the mapper to complete its local top k nodes finding .

Report the top-100 Wikipedia pages with the highest PageRanks, along with their rank values and sorted from highest to lowest, for both the simple and full datasets. Do they seem reasonable based on your intuition about important information on Wikipedia? (5 points)
As evident from the local input ,

The top 100 wikipedia pages are related to country related and year related pages as would be expected from most Wikipedia articles.