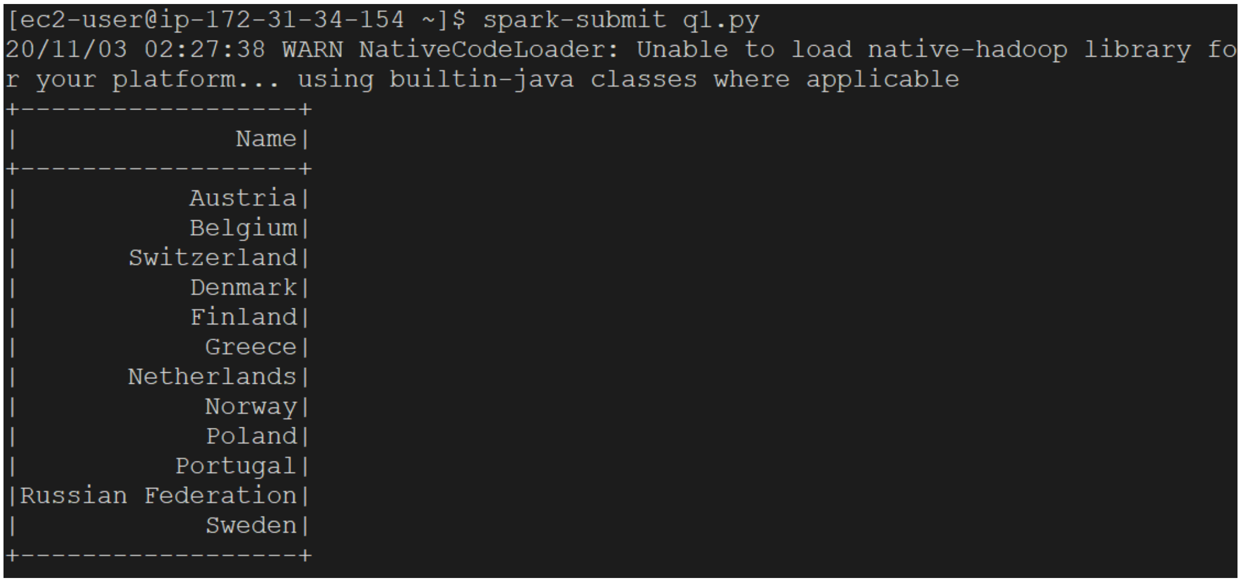
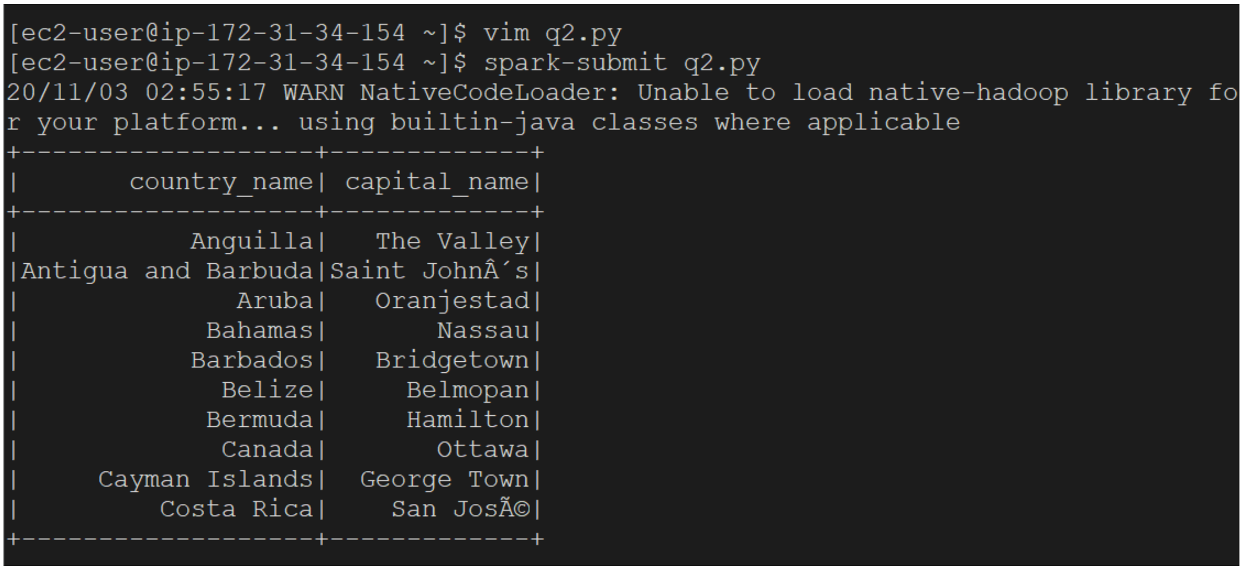
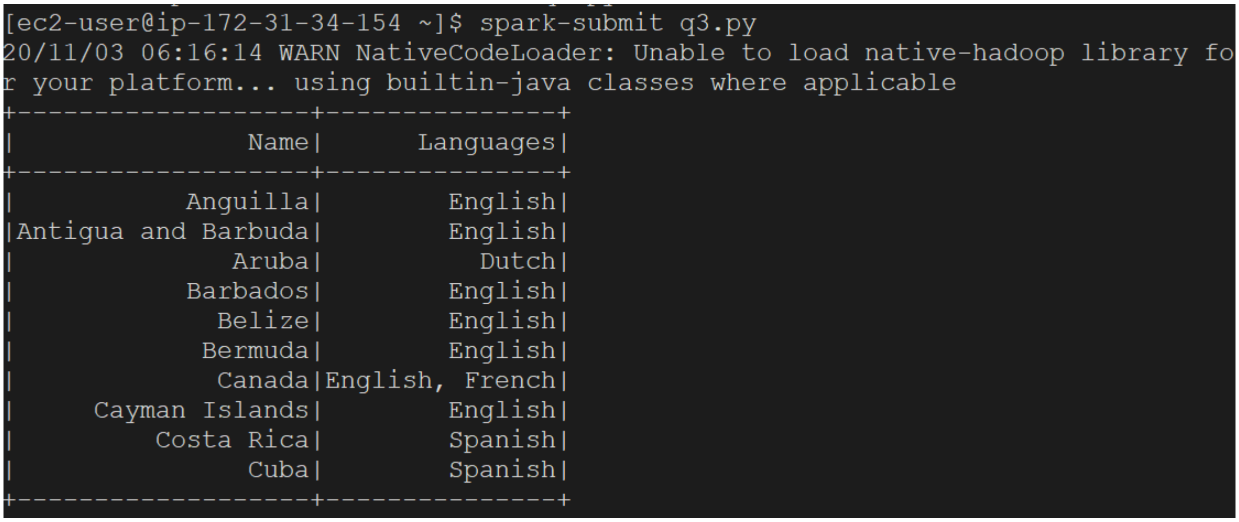


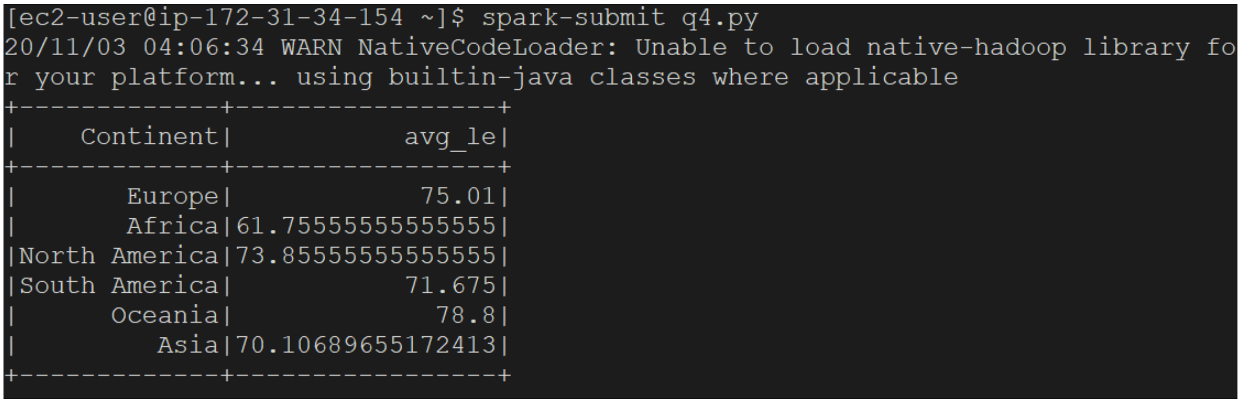
```
[ec2-user@ip-172-31-34-154 ~]$ spark-submit q1.py
20/11/03 02:27:38 WARN NativeCodeLoader: Unable to load native-hadoop library fo
r your platform... using builtin-java classes where applicable
+------------------+
|              Name|
+------------------+
|           Austria|
|           Belgium|
|       Switzerland|
|           Denmark|
|           Finland|
|            Greece|
|       Netherlands|
|            Norway|
|            Poland|
|          Portugal|
|Russian Federation|
|            Sweden|
+------------------+
```



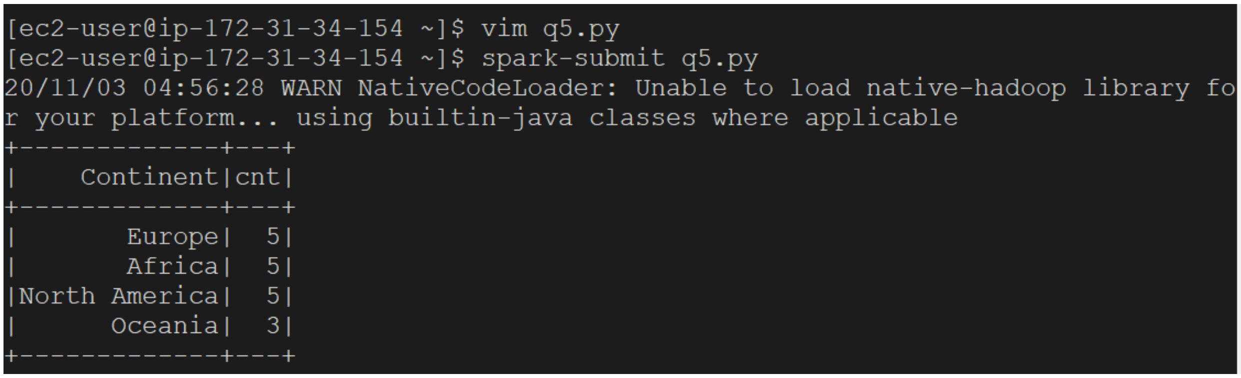
```
[ec2-user@ip-172-31-34-154 ~]$ vim q2.py
[ec2-user@ip-172-31-34-154 ~]$ spark-submit q2.py
20/11/03 02:55:17 WARN NativeCodeLoader: Unable to load native-hadoop library fo
r your platform... using builtin-java classes where applicable
+-------------------+-------------+
|       country_name| capital_name|
+-------------------+-------------+
|           Anguilla|   The Valley|
|Antigua and Barbuda|Saint JohnÂ´s|
|              Aruba|   Oranjestad|
|            Bahamas|       Nassau|
|           Barbados|   Bridgetown|
|             Belize|     Belmopan|
|            Bermuda|     Hamilton|
|             Canada|       Ottawa|
|     Cayman Islands|  George Town|
|         Costa Rica|     San JosÃ©|
+-------------------+-------------+
```



```
[ec2-user@ip-172-31-34-154 ~]$ spark-submit q3.py
20/11/03 06:16:14 WARN NativeCodeLoader: Unable to load native-hadoop library fo
r your platform... using builtin-java classes where applicable
+-------------------+---------------+
|               Name|      Languages|
+-------------------+---------------+
|           Anguilla|        English|
|Antigua and Barbuda|        English|
|              Aruba|          Dutch|
|           Barbados|        English|
|             Belize|        English|
|            Bermuda|        English|
|             Canada|English, French|
|     Cayman Islands|        English|
|         Costa Rica|        Spanish|
|               Cuba|        Spanish|
+-------------------+---------------+
```

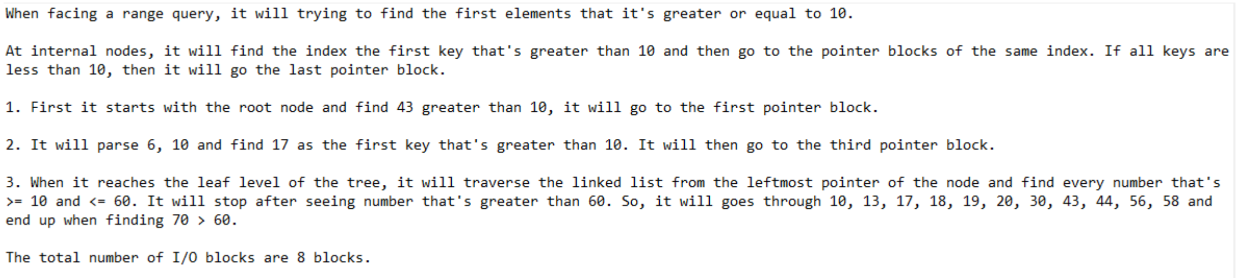


```
[ec2-user@ip-172-31-34-154 ~]$ spark-submit q4.py
20/11/03 04:06:34 WARN NativeCodeLoader: Unable to load native-hadoop library fo
r your platform... using builtin-java classes where applicable
+-------------+-----------------+
|    Continent|           avg_le|
+-------------+-----------------+
|       Europe|            75.01|
|       Africa|61.75555555555555|
|North America|73.85555555555555|
|South America|           71.675|
|      Oceania|             78.8|
|         Asia|70.10689655172413|
+-------------+-----------------+
```



```
[ec2-user@ip-172-31-34-154 ~]$ vim q5.py
[ec2-user@ip-172-31-34-154 ~]$ spark-submit q5.py
20/11/03 04:56:28 WARN NativeCodeLoader: Unable to load native-hadoop library fo
r your platform... using builtin-java classes where applicable
+-------------+---+
|    Continent|cnt|
+-------------+---+
|       Europe|  5|
|       Africa|  5|
|North America|  5|
|      Oceania|  3|
+-------------+---+
```

2.1



When facing a range query, it will trying to find the first elements that it's greater or equal to 10.

At internal nodes, it will find the index the first key that's greater than 10 and then go to the pointer blocks of the same index. If all keys are less than 10, then it will go the last pointer block.

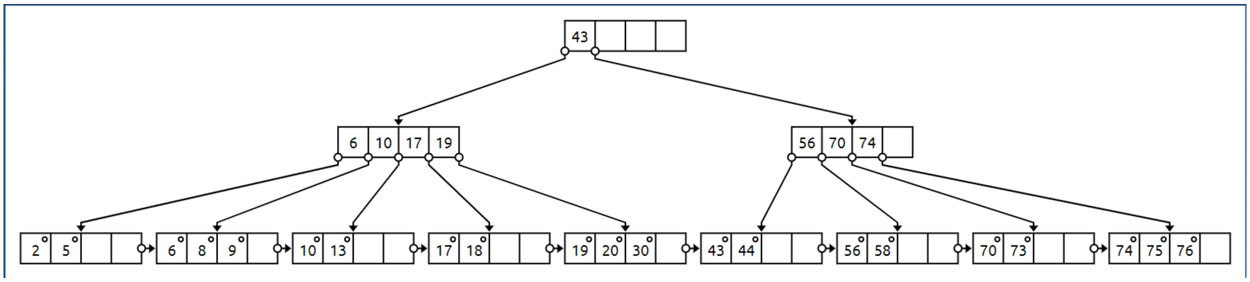
1. First it starts with the root node and find 43 greater than 10, it will go to the first pointer block.

2. It will parse 6, 10 and find 17 as the first key that's greater than 10. It will then go to the third pointer block.

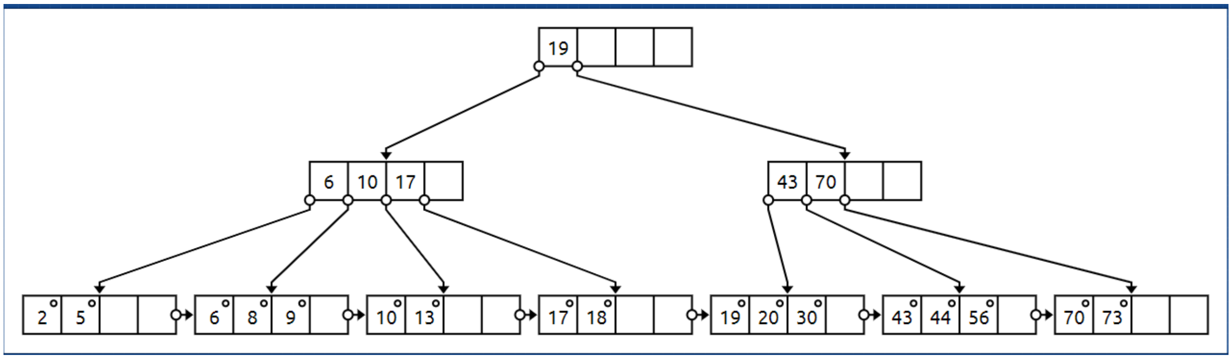
3. When it reaches the leaf level of the tree, it will traverse the linked list from the leftmost pointer of the node and find every number that's >= 10 and <= 60. It will stop after seeing number that's greater than 60. So, it will goes through 10, 13, 17, 18, 19, 20, 30, 43, 44, 56, 58 and end up when finding 70 > 60.

The total number of I/O blocks are 8 blocks.

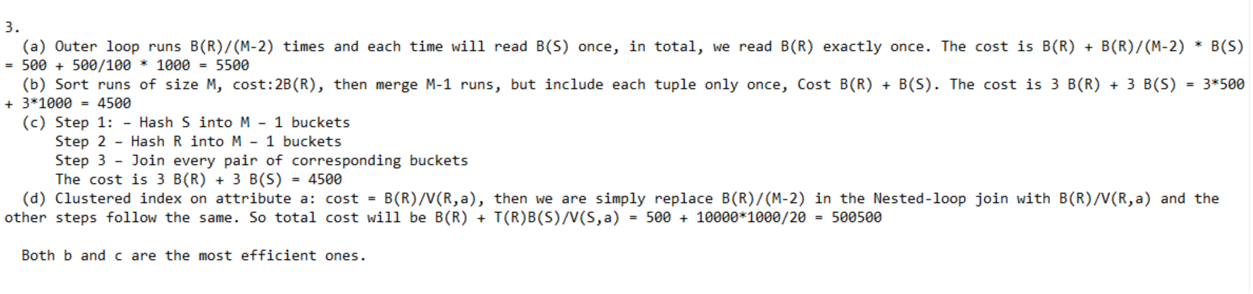
2.2



2.3



3



```
3.
  (a) Outer loop runs B(R)/(M-2) times and each time will read B(S) once, in total, we read B(R) exactly once. The cost is B(R) + B(R)/(M-2) * B(S)
= 500 + 500/100 * 1000 = 5500
  (b) Sort runs of size M, cost:2B(R), then merge M-1 runs, but include each tuple only once, Cost B(R) + B(S). The cost is 3 B(R) + 3 B(S) = 3*500
+ 3*1000 = 4500
  (c) Step 1: – Hash S into M – 1 buckets
      Step 2 – Hash R into M – 1 buckets
      Step 3 – Join every pair of corresponding buckets
      The cost is 3 B(R) + 3 B(S) = 4500
  (d) Clustered index on attribute a: cost = B(R)/V(R,a), then we are simply replace B(R)/(M-2) in the Nested-loop join with B(R)/V(R,a) and the
other steps follow the same. So total cost will be B(R) + T(R)B(S)/V(S,a) = 500 + 10000*1000/20 = 500500

  Both b and c are the most efficient ones.
```