Ex. No. 2	Word Count Using Apache Spark
Date of Exercise	28/07/2025

Aim:

To implement a **Word Count** program using Apache Spark and analyze how distributed processing works with transformations and actions.

Procedure:

1. **Create Input File :**

   Apache Spark is fast

   Spark is open source

   Spark runs in memory

2. **Launch PySpark:**

```
pyspark
```

3. **Read the input file:**

```
lines = sc.textFile("sample.txt")
```

4. **Tokenize lines into words:**

```
words = lines.flatMap(lambda line: line.split(" "))
```

5. **Map each word to a key-value pair:**

```
wordPairs = words.map(lambda word: (word.lower(), 1))
```

6. **Aggregate the counts by word:**

```
wordCounts = wordPairs.reduceByKey(lambda a, b: a + b)
```

7. **Display the result:**

```
for word, count in wordCounts.collect():
print(f"{word}: {count}")
```

Code:

```
!pip install findspark pyspark

import findspark

findspark.init()

from pyspark import SparkContext, SparkConf

from pyspark import SparkContext, SparkConf


# Initialize SparkContext

conf =
SparkConf().setMaster(&quot;local[*]&quot;).setAppName(&quot;WordCount&quot;)

sc = SparkContext(conf=conf)


# Load text file (update to your path)

text_rdd = sc.textFile(&quot;path/to/your/textfile.txt&quot;)


# Tokenize lines into words

words = text_rdd.flatMap(lambda line: line.split(&quot; &quot;))


# Map each word to (word, 1)

pairs = words.map(lambda word: (word, 1))


# Reduce by key to count occurrences

counts = pairs.reduceByKey(lambda a, b: a + b)


# Sort by count descending

sorted_counts = counts.sortBy(lambda x: x[1], ascending=False)
```

```
# Collect and show results
for word, count in sorted_counts.collect():
print(f&quot;{word}: {count}&quot;)

# Stop SparkContext
sc.stop()
```

Output:

```
apache: 1
spark: 3
is: 2
fast: 1
open: 1
source: 1
runs: 1
in: 1
memory: 1
```

Result:

The Word Count application was successfully implemented using Apache Spark, demonstrating distributed processing using RDD transformations and actions.