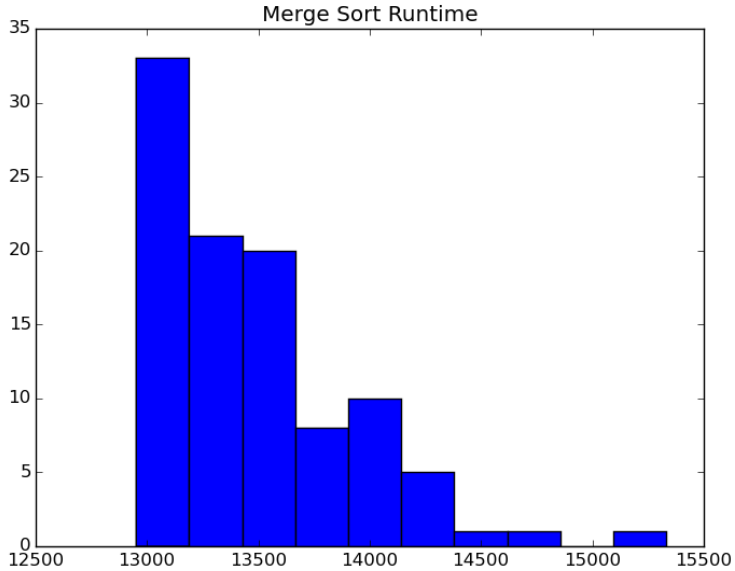
CSIC3070U Assignment 1 LO, Cheuk Him Calvin #100514352

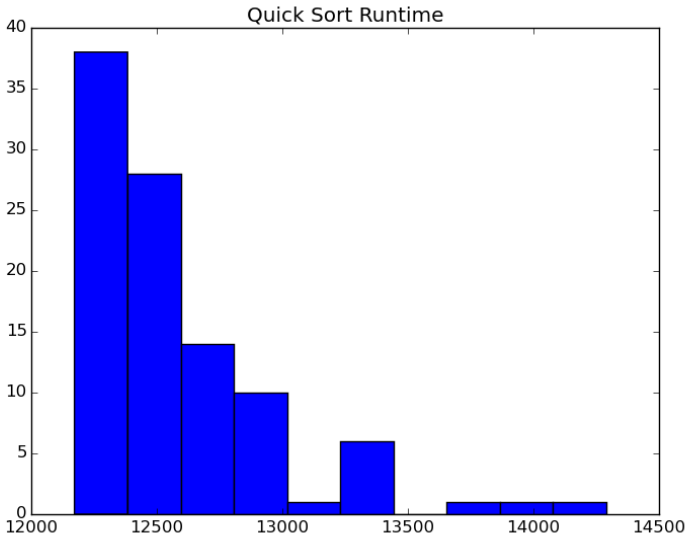
(1)
(1.1) Return true if the list contains any zero else return false
(1.2) T(n) = O(1) + O(1) + T(n/3)
(1.3) O(n/3 + 2)
(1.4)

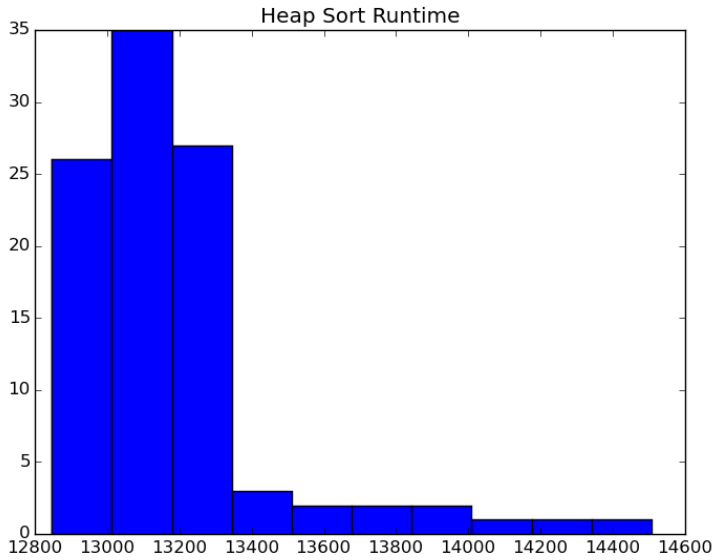
```
function f(x: List[Integer]) {
  var n: Integer = Length(x) {
    for int i = 0; i < n ; i++) {
       if x[i] == 0    {
       return true
    }
}
return false
}
```

(1.5)
They are different. The complexity of iterative is O(n) while the complexity of recursive is O(n/3 + 2). Recursive is more efficient.

(2)
(2.1)
Assume each char is 6 bits and each string is 75 bytes. The object head is 12 bytes. So, the total memory usage is 12 + (75*1 million) = 75 mb
(2.2) See Attachment.
(2.3)







(3)

(3.1)

Assume each char is 6 bits and each string is 75 bytes. The object head is 12 bytes. So, the total memory usage is 12 + (75*100 million) = 7500000012 Byte = 7.5 GB.

(3.2) See Attachment.

(3.3)

```
function sort(File file)
ArrayList<File> files
int chunk_size = 1024*1024*1024
Reader reader = new Reader(file)

// create a String arrayList
ArrayList<String> StringArrayList = new ArrayList<String>();

// Fill the arrayList
String line = ""
while (line not empty)
        int current_chunk_size = 0
        // Fill in the ArrayList from file
        while ((line = reader.readLine() not NULL) &&( current_chunk_size < chunk_size))
                StringArrayList.add(line)
                currentchunksize + line.length()

        // Sorting
        quickSort(StringArrayList)

        // Store in temp file
        File tmp = new tempfile
        PrintwWriter writer = new PrintWriter(tmp)
        for (String s in StringArrayList)
                PrintWrriter.write(s)

        files.add(tmap)

        clear StringArrayList


// use N-way merge to merge every chunk file
ArrayList<String> sorted = merge(files)

// Write sorted to file
PrintWriter writer = new PrintWriter("sorted.txt")
for (String s in sorted)
        writer.write(s)


// merge
function: ArrayList<String> merge (List<File> files)
ArrayList<String> sorted = new ArrayList<String>

Comparator<String> cmp: compare string_1 and string 2
Comparator<CusFileBuffer> cmp: compare the first string of each file

PriorityQueue<CusFileBuffer> pq

Add the files as file buffer to the PriorityQueue
```

```
while (pq.size() greater than zero)
        poll the first buffer
        pop the string from the buffer
        add the string the sorted ArrayList

close all buffer reader
return sorted

// Customize buffer class
class cusFileBuffer
BufferedReader reaer
File ofile
String line
boolean empty

Constructor(File file):
        set the ofile variable
        set the reader
        read()
boolean empty:
        return empty
void read:
        readL() from the file
void close:
        close the reader
String peek:
        return line
String pop:
        String temp = peek()
        read()
        return temp
```

(3.4) O(nlogn + nk^2) while is chunk_size. nlogn = complexity of quick sort for n chunk. nk^2 = complexity of n way merge

(3.5)

see Attachment.