Cover Page

Name: David Chen Salas

Section: 2023 Fall Term (1) Algorithms I CSCI 700 231[25504] (Queens College)

Project#: 8

Project Name: Radix sort for strings

Due Date: 12/1/2023 Friday before midnight

Algorithm Steps:

Step 0: inFile, outFile1, deBugFile open via args []

hashTable[2][tableSize] establish and initialize as given in the above.

Step 1: firstReading (inFile, deBugFile)

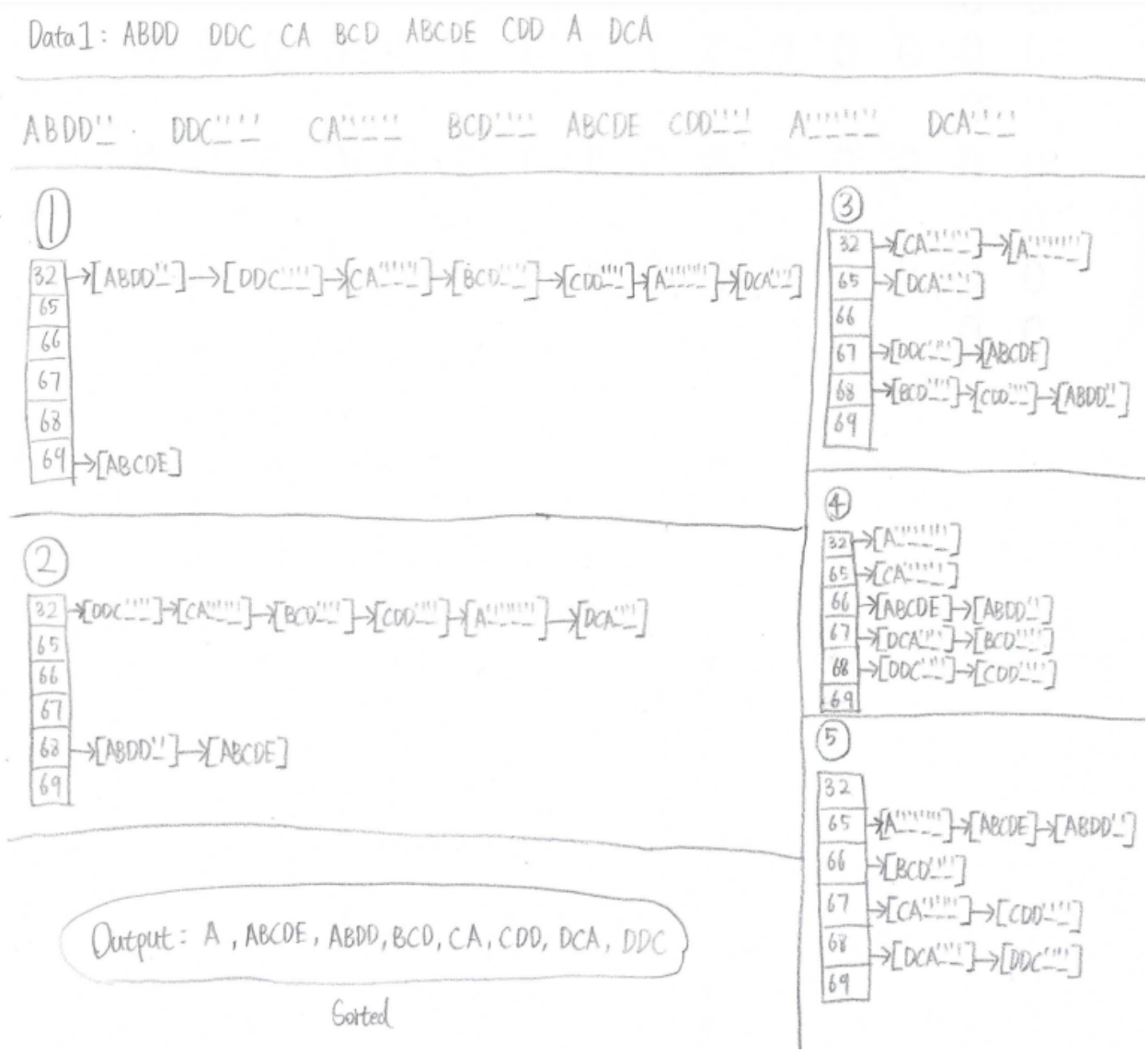
Step 2: close inFile

reopen inFile.

Step 3: RSort (inFile, outFile1, deBugFile)

Step 4: close all files

Illustration



Source Code

```
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Scanner;

public class ChenSalasD_Project8_Main {

   static Scanner inFile;
   static FileWriter outFile1, deBugFile;

   static class listNode{
      String data;
      listNode next;

      listNode(){
         data = "dummy";
         next = null;
      }

      listNode(String data, listNode next){
         this.data = data;
         this.next = next;
      }
   }

   static class LLQ{
      listNode head;
      listNode tail;

      LLQ(){
         head = new listNode();
         tail = head;
      }
   }
   public static void insertQ(LLQ llq, listNode newNode) {
      llq.tail.next = newNode;
      llq.tail = newNode;
   }

   public static listNode deleteQ(LLQ llq){
      if(llq.head.next != null){
         listNode tmp = llq.head.next;
         if(tmp == llq.tail){
            llq.tail = llq.head;
         }
         llq.head.next = tmp.next;
         tmp.next = null;
         return tmp;
      }
      return null;
   }

   public static boolean isEmpty(LLQ llq){
      return llq.head == llq.tail;
   }

   static int tableSize = 256;
   static LLQ[][] hashTable;
   static String data;
   static int currentTable, nextTable;
```

```
    static int longestStringLength;
    static int currentIndex;

    public static void main(String[] args) throws IOException {
        inFile = new Scanner(new FileReader(args[0]));
        outFile1 = new FileWriter(args[1]);
        deBugFile = new FileWriter(args[2]);

        hashTable = new LLQ[2][tableSize];
        for(int i = 0; i < 2; i++){
            for(int j = 0; j < tableSize; j++){
                hashTable[i][j] = new LLQ();
            }
        }

        firstReading(inFile, deBugFile);
        inFile.close();

        inFile = new Scanner(new FileReader(args[0]));
        RSort(inFile, outFile1, deBugFile);

        inFile.close();
        outFile1.close();
        deBugFile.close();
    }

    public static void firstReading(Scanner inFile, FileWriter deBugFile) throws IOException {
        deBugFile.write("Performing first Reading to find the longest data string in the input file");
        longestStringLength = 0;
        while(inFile.hasNext()) {
            data = inFile.next();
            if(longestStringLength < data.length()){
                longestStringLength = data.length();
            }
        }
    }

    public static void RSort(Scanner inFile, FileWriter outFile1, FileWriter deBugFile) throws IOException {
        deBugFile.write("Entering RSort, Performing Radix Sort.\n");
        populateFirstTable(inFile, outFile1, deBugFile);

        nextTable = currentTable;
        while(currentIndex >= 0){
            currentTable = (++currentTable) % 2;
            deBugFile.write("In RSort, after swap tables, currentIndex = " + currentIndex + "; currentTable = " + currentTable + ",
nextTable = " + nextTable + "\n");
            int tableIndex = 0;
            listNode newNode;
            String tempData;
            int hashIndex;
            while(tableIndex < tableSize){
                while(!isEmpty(hashTable[nextTable][tableIndex])) {
                    newNode = deleteQ(hashTable[nextTable][tableIndex]);
                    if (newNode != null) {
                        tempData = newNode.data;
                        hashIndex = tempData.charAt(currentIndex);
                        insertQ(hashTable[currentTable][hashIndex], newNode);
                    }
                }
                tableIndex++;
            }
            printTable(hashTable[currentTable], deBugFile);
            currentIndex--;
```

```
            nextTable = currentTable;
        }
        printSortedData(nextTable, outFile1);
        deBugFile.write("Leaving RSort, Performing Radix Sort.\n");
    }


    public static void populateFirstTable(Scanner inFile, FileWriter outFile1, FileWriter deBugFile) throws IOException {
        deBugFile.write("Entering populateFirstTable().\n");
        currentIndex = longestStringLength - 1;
        currentTable = 0;

        String paddedData;
        listNode newNode;
        int hashIndex;
        while(inFile.hasNext()){
            data = inFile.next();
            paddedData = padString(data);
            newNode = new listNode(paddedData, null);
            hashIndex = paddedData.charAt(currentIndex);
            deBugFile.write("In RSort, paddedData is " + paddedData + "; currentIndex = " + currentIndex + "; hashIndex = " +
hashIndex + "; currentTable = " + currentTable + "\n");
            insertQ(hashTable[currentTable][hashIndex], newNode);
        }
        deBugFile.write("Finish insert all paddedData into the hashTable[0], the hashTable shown below");
        printTable(hashTable[0], deBugFile);
        printTable(hashTable[0], outFile1);
        deBugFile.write("Leaving populateFirstTable().\n");
    }

    public static String padString(String data) {
        StringBuilder str = new StringBuilder(data);
        while(str.length() < longestStringLength){
            str.append(" ");
        }
        return str.toString();
    }

    public static void printTable(LLQ[] llq, FileWriter outFile) throws IOException {
        listNode tmp;
        for(int i = 0; i < tableSize; i++){
            if(!isEmpty(llq[i])){
                tmp = llq[i].head;
                outFile.write("Table[" + currentTable + "][" + i + "]: ");
                while(tmp.next != null){
                    outFile.write("(" + tmp.data + ", " + tmp.next.data + ")");
                    outFile.write(" -> ");
                    tmp = tmp.next;
                }
                outFile.write("(" + tmp.data + ", NULL) --> NULL\n");
            }
        }
        outFile.write("\n");
    }

    public static void printSortedData(int nextTable, FileWriter outFile) throws IOException {
        listNode tmp;
        StringBuilder str;
        for(int i = 0; i < tableSize; i++){
            if(!isEmpty(hashTable[nextTable][i])){
                tmp = hashTable[nextTable][i].head.next;
                while(tmp != null){
                    str = new StringBuilder("");
```

```
                for(int j = 0; j < tmp.data.length()-1; j++){
                    if(tmp.data.charAt(j) != ' ') {
                        str.append(tmp.data.charAt(j));
                    }
                }
                outFile.write(str + " ");
                tmp = tmp.next;
            }
        }
    }
  }

}
```

Program Output

Data1

outFile1.txt

```
Table[0][32]: (dummy, ABDD ) -> (ABDD , DDC  ) -> (DDC  , CA   ) -> (CA   , BCD  ) ->
(BCD  , CDD  ) -> (CDD  , A    ) -> (A    , DCA  ) -> (DCA  , NULL) --> NULL
Table[0][69]: (dummy, ABCDE) -> (ABCDE, NULL) --> NULL

A ABCD ABDD BCD CA CDD DCA DDC
```

deBugFile.txt

Performing first Reading to find the longest data string in the input fileEntering RSort, Performing Radix Sort.
Entering populateFirstTable().
In RSort, paddedData is ABDD ; currentIndex = 4; hashIndex = 32; currentTable = 0
In RSort, paddedData is DDC ; currentIndex = 4; hashIndex = 32; currentTable = 0
In RSort, paddedData is CA   ; currentIndex = 4; hashIndex = 32; currentTable = 0
In RSort, paddedData is BCD ; currentIndex = 4; hashIndex = 32; currentTable = 0
In RSort, paddedData is ABCDE; currentIndex = 4; hashIndex = 69; currentTable = 0
In RSort, paddedData is CDD ; currentIndex = 4; hashIndex = 32; currentTable = 0
In RSort, paddedData is A   ; currentIndex = 4; hashIndex = 32; currentTable = 0
In RSort, paddedData is DCA ; currentIndex = 4; hashIndex = 32; currentTable = 0
Finish insert all paddedData into the hashTable[0], the hashTable shown belowTable[0][32]: (dummy, ABDD) -> (ABDD , DDC) -> (DDC , CA   ) -> (CA   , BCD) -> (BCD , CDD) -> (CDD , A   ) -> (A   , DCA) -> (DCA , NULL) --> NULL
Table[0][69]: (dummy, ABCDE) -> (ABCDE, NULL) --> NULL

Leaving populateFirstTable().
In RSort, after swap tables, currentIndex = 4; currentTable = 1, nextTable = 0
Table[1][32]: (dummy, ABDD) -> (ABDD , DDC) -> (DDC , CA   ) -> (CA   , BCD) -> (BCD , CDD) -> (CDD , A   ) -> (A   , DCA) -> (DCA , NULL) --> NULL
Table[1][69]: (dummy, ABCDE) -> (ABCDE, NULL) --> NULL

In RSort, after swap tables, currentIndex = 3; currentTable = 0, nextTable = 1
Table[0][32]: (dummy, DDC) -> (DDC , CA   ) -> (CA   , BCD) -> (BCD , CDD) -> (CDD , A   ) -> (A   , DCA) -> (DCA , NULL) --> NULL
Table[0][68]: (dummy, ABDD) -> (ABDD , ABCDE) -> (ABCDE, NULL) --> NULL

In RSort, after swap tables, currentIndex = 2; currentTable = 1, nextTable = 0
Table[1][32]: (dummy, CA   ) -> (CA   , A   ) -> (A   , NULL) --> NULL
Table[1][65]: (dummy, DCA) -> (DCA , NULL) --> NULL
Table[1][67]: (dummy, DDC) -> (DDC , ABCDE) -> (ABCDE, NULL) --> NULL
Table[1][68]: (dummy, BCD) -> (BCD , CDD) -> (CDD , ABDD) -> (ABDD , NULL) --> NULL

In RSort, after swap tables, currentIndex = 1; currentTable = 0, nextTable = 1
Table[0][32]: (dummy, A   ) -> (A   , NULL) --> NULL
Table[0][65]: (dummy, CA   ) -> (CA   , NULL) --> NULL
Table[0][66]: (dummy, ABCDE) -> (ABCDE, ABDD) -> (ABDD , NULL) --> NULL
Table[0][67]: (dummy, DCA) -> (DCA , BCD) -> (BCD , NULL) --> NULL
Table[0][68]: (dummy, DDC) -> (DDC , CDD) -> (CDD , NULL) --> NULL

In RSort, after swap tables, currentIndex = 0; currentTable = 1, nextTable = 0
Table[1][65]: (dummy, A   ) -> (A   , ABCDE) -> (ABCDE, ABDD) -> (ABDD , NULL) --> NULL
Table[1][66]: (dummy, BCD) -> (BCD , NULL) --> NULL
Table[1][67]: (dummy, CA   ) -> (CA   , CDD) -> (CDD , NULL) --> NULL
Table[1][68]: (dummy, DCA) -> (DCA , DDC) -> (DDC , NULL) --> NULL

Leaving RSort, Performing Radix Sort.

Data2

outFile1.txt

```
Table[0][32]: (dummy, cCaAbb ) -> (cCaAbb , CcaabB ) -> (CcaabB , BbAa   ) -> (BbAa
, aA     ) -> (aA     , AAbb   ) -> (AAbb   , zxcccc ) -> (zxcccc , XyZzz  ) -> (XyZzz
, xyyk   ) -> (xyyk   , ZZZZ   ) -> (ZZZZ   , Hijk   ) -> (Hijk   , jIJkL  ) -> (jIJkL
, Acc    ) -> (Acc    , aCC    ) -> (aCC    , Bdd    ) -> (Bdd    , bggff  ) -> (bggff
, aAaA   ) -> (aAaA   , AccaCC ) -> (AccaCC , NULL) --> NULL
Table[0][71]: (dummy, AbCdEfG) -> (AbCdEfG, NULL) --> NULL

AAbb AbCdEf Acc AccaCC BbAa Bdd CcaabB Hijk XyZzz ZZZZ aA aAaA aCC bggff cCaAbb jIJkL
xyyk zxcccc
```

deBugFile.txt

Performing first Reading to find the longest data string in the input fileEntering RSort, Performing Radix Sort.
Entering populateFirstTable().
In RSort, paddedData is AbCdEfG; currentIndex = 6; hashIndex = 71; currentTable = 0
In RSort, paddedData is cCaAbb ; currentIndex = 6; hashIndex = 32; currentTable = 0
In RSort, paddedData is CcaabB ; currentIndex = 6; hashIndex = 32; currentTable = 0
In RSort, paddedData is BbAa   ; currentIndex = 6; hashIndex = 32; currentTable = 0
In RSort, paddedData is aA   ; currentIndex = 6; hashIndex = 32; currentTable = 0
In RSort, paddedData is AAbb   ; currentIndex = 6; hashIndex = 32; currentTable = 0
In RSort, paddedData is zxcccc ; currentIndex = 6; hashIndex = 32; currentTable = 0
In RSort, paddedData is XyZzz ; currentIndex = 6; hashIndex = 32; currentTable = 0
In RSort, paddedData is xyyk   ; currentIndex = 6; hashIndex = 32; currentTable = 0
In RSort, paddedData is ZZZZ   ; currentIndex = 6; hashIndex = 32; currentTable = 0
In RSort, paddedData is Hijk   ; currentIndex = 6; hashIndex = 32; currentTable = 0
In RSort, paddedData is jIJkL ; currentIndex = 6; hashIndex = 32; currentTable = 0
In RSort, paddedData is Acc   ; currentIndex = 6; hashIndex = 32; currentTable = 0
In RSort, paddedData is aCC   ; currentIndex = 6; hashIndex = 32; currentTable = 0
In RSort, paddedData is Bdd   ; currentIndex = 6; hashIndex = 32; currentTable = 0
In RSort, paddedData is bggff ; currentIndex = 6; hashIndex = 32; currentTable = 0
In RSort, paddedData is aAaA   ; currentIndex = 6; hashIndex = 32; currentTable = 0
In RSort, paddedData is AccaCC ; currentIndex = 6; hashIndex = 32; currentTable = 0
Finish insert all paddedData into the hashTable[0], the hashTable shown belowTable[0][32]: (dummy, cCaAbb) -> (cCaAbb , CcaabB) -> (CcaabB , BbAa   ) -> (BbAa   , aA   ) -> (aA   , AAbb   ) -> (AAbb   , zxcccc) -> (zxcccc , XyZzz) -> (XyZzz , xyyk   ) -> (xyyk   , ZZZZ   ) -> (ZZZZ   , Hijk   ) -> (Hijk   , jIJkL) -> (jIJkL , Acc   ) -> (Acc   , aCC   ) -> (aCC   , Bdd   ) -> (Bdd   , bggff) -> (bggff , aAaA   ) -> (aAaA   , AccaCC) -> (AccaCC , NULL) --> NULL
Table[0][71]: (dummy, AbCdEfG) -> (AbCdEfG, NULL) --> NULL

Leaving populateFirstTable().
In RSort, after swap tables, currentIndex = 6; currentTable = 1, nextTable = 0
Table[1][32]: (dummy, cCaAbb) -> (cCaAbb , CcaabB) -> (CcaabB , BbAa   ) -> (BbAa   , aA   ) -> (aA   , AAbb   ) -> (AAbb   , zxcccc) -> (zxcccc , XyZzz) -> (XyZzz , xyyk   ) -> (xyyk   , ZZZZ   ) -> (ZZZZ   , Hijk   ) -> (Hijk   , jIJkL) -> (jIJkL , Acc   ) -> (Acc   , aCC   ) -> (aCC   , Bdd   ) -> (Bdd   , bggff) -> (bggff , aAaA   ) -> (aAaA   , AccaCC) -> (AccaCC , NULL) --> NULL
Table[1][71]: (dummy, AbCdEfG) -> (AbCdEfG, NULL) --> NULL

In RSort, after swap tables, currentIndex = 5; currentTable = 0, nextTable = 1
Table[0][32]: (dummy, BbAa   ) -> (BbAa   , aA   ) -> (aA   , AAbb   ) -> (AAbb   , XyZzz) -> (XyZzz , xyyk   ) -> (xyyk   , ZZZZ   ) -> (ZZZZ   , Hijk   ) -> (Hijk   , jIJkL) -> (jIJkL , Acc   ) -> (Acc   , aCC   ) -> (aCC   , Bdd   ) -> (Bdd   , bggff) -> (bggff , aAaA   ) -> (aAaA   , NULL) --> NULL
Table[0][66]: (dummy, CcaabB) -> (CcaabB , NULL) --> NULL
Table[0][67]: (dummy, AccaCC) -> (AccaCC , NULL) --> NULL
Table[0][98]: (dummy, cCaAbb) -> (cCaAbb , NULL) --> NULL
Table[0][99]: (dummy, zxcccc) -> (zxcccc , NULL) --> NULL
Table[0][102]: (dummy, AbCdEfG) -> (AbCdEfG, NULL) --> NULL

In RSort, after swap tables, currentIndex = 4; currentTable = 1, nextTable = 0
Table[1][32]: (dummy, BbAa   ) -> (BbAa   , aA   ) -> (aA   , AAbb   ) -> (AAbb   , xyyk   ) -> (xyyk   , ZZZZ   ) -> (ZZZZ   , Hijk   ) -> (Hijk   , Acc   ) -> (Acc   , aCC   ) -> (aCC   , Bdd   ) -> (Bdd   , aAaA   ) -> (aAaA   , NULL) --> NULL

Table[1][67]: (dummy, AccaCC) -> (AccaCC , NULL) --> NULL
Table[1][69]: (dummy, AbCdEfG) -> (AbCdEfG, NULL) --> NULL
Table[1][76]: (dummy, jIJkL) -> (jIJkL , NULL) --> NULL
Table[1][98]: (dummy, CcaabB) -> (CcaabB , cCaAbb) -> (cCaAbb , NULL) --> NULL
Table[1][99]: (dummy, zxcccc) -> (zxcccc , NULL) --> NULL
Table[1][102]: (dummy, bggff) -> (bggff , NULL) --> NULL
Table[1][122]: (dummy, XyZzz) -> (XyZzz , NULL) --> NULL

In RSort, after swap tables, currentIndex = 3; currentTable = 0, nextTable = 1
Table[0][32]: (dummy, aA) -> (aA , Acc) -> (Acc , aCC) -> (aCC , Bdd) -> (Bdd , NULL) --> NULL
Table[0][65]: (dummy, aAaA) -> (aAaA , cCaAbb) -> (cCaAbb , NULL) --> NULL
Table[0][90]: (dummy, ZZZZ) -> (ZZZZ , NULL) --> NULL
Table[0][97]: (dummy, BbAa) -> (BbAa , AccaCC) -> (AccaCC , CcaabB) -> (CcaabB , NULL) --> NULL
Table[0][98]: (dummy, AAbb) -> (AAbb , NULL) --> NULL
Table[0][99]: (dummy, zxcccc) -> (zxcccc , NULL) --> NULL
Table[0][100]: (dummy, AbCdEfG) -> (AbCdEfG, NULL) --> NULL
Table[0][102]: (dummy, bggff) -> (bggff , NULL) --> NULL
Table[0][107]: (dummy, xyyk) -> (xyyk , Hijk) -> (Hijk , jIJkL) -> (jIJkL , NULL) --> NULL
Table[0][122]: (dummy, XyZzz) -> (XyZzz , NULL) --> NULL

In RSort, after swap tables, currentIndex = 2; currentTable = 1, nextTable = 0
Table[1][32]: (dummy, aA) -> (aA , NULL) --> NULL
Table[1][65]: (dummy, BbAa) -> (BbAa , NULL) --> NULL
Table[1][67]: (dummy, aCC) -> (aCC , AbCdEfG) -> (AbCdEfG, NULL) --> NULL
Table[1][74]: (dummy, jIJkL) -> (jIJkL , NULL) --> NULL
Table[1][90]: (dummy, ZZZZ) -> (ZZZZ , XyZzz) -> (XyZzz , NULL) --> NULL
Table[1][97]: (dummy, aAaA) -> (aAaA , cCaAbb) -> (cCaAbb , CcaabB) -> (CcaabB , NULL) --> NULL
Table[1][98]: (dummy, AAbb) -> (AAbb , NULL) --> NULL
Table[1][99]: (dummy, Acc) -> (Acc , AccaCC) -> (AccaCC , zxcccc) -> (zxcccc , NULL) --> NULL
Table[1][100]: (dummy, Bdd) -> (Bdd , NULL) --> NULL
Table[1][103]: (dummy, bggff) -> (bggff , NULL) --> NULL
Table[1][106]: (dummy, Hijk) -> (Hijk , NULL) --> NULL
Table[1][121]: (dummy, xyyk) -> (xyyk , NULL) --> NULL

In RSort, after swap tables, currentIndex = 1; currentTable = 0, nextTable = 1
Table[0][65]: (dummy, aA) -> (aA , aAaA) -> (aAaA , AAbb) -> (AAbb , NULL) --> NULL
Table[0][67]: (dummy, aCC) -> (aCC , cCaAbb) -> (cCaAbb , NULL) --> NULL
Table[0][73]: (dummy, jIJkL) -> (jIJkL , NULL) --> NULL
Table[0][90]: (dummy, ZZZZ) -> (ZZZZ , NULL) --> NULL
Table[0][98]: (dummy, BbAa) -> (BbAa , AbCdEfG) -> (AbCdEfG, NULL) --> NULL
Table[0][99]: (dummy, CcaabB) -> (CcaabB , Acc) -> (Acc , AccaCC) -> (AccaCC , NULL) --> NULL
Table[0][100]: (dummy, Bdd) -> (Bdd , NULL) --> NULL
Table[0][103]: (dummy, bggff) -> (bggff , NULL) --> NULL
Table[0][105]: (dummy, Hijk) -> (Hijk , NULL) --> NULL
Table[0][120]: (dummy, zxcccc) -> (zxcccc , NULL) --> NULL
Table[0][121]: (dummy, XyZzz) -> (XyZzz , xyyk) -> (xyyk , NULL) --> NULL

In RSort, after swap tables, currentIndex = 0; currentTable = 1, nextTable = 0
Table[1][65]: (dummy, AAbb) -> (AAbb , AbCdEfG) -> (AbCdEfG, Acc) -> (Acc , AccaCC) -> (AccaCC , NULL) --> NULL
Table[1][66]: (dummy, BbAa) -> (BbAa , Bdd) -> (Bdd , NULL) --> NULL
Table[1][67]: (dummy, CcaabB) -> (CcaabB , NULL) --> NULL
Table[1][72]: (dummy, Hijk) -> (Hijk , NULL) --> NULL
Table[1][88]: (dummy, XyZzz) -> (XyZzz , NULL) --> NULL
Table[1][90]: (dummy, ZZZZ) -> (ZZZZ , NULL) --> NULL
Table[1][97]: (dummy, aA) -> (aA , aAaA) -> (aAaA , aCC) -> (aCC , NULL) --> NULL
Table[1][98]: (dummy, bggff) -> (bggff , NULL) --> NULL
Table[1][99]: (dummy, cCaAbb) -> (cCaAbb , NULL) --> NULL
Table[1][106]: (dummy, jIJkL) -> (jIJkL , NULL) --> NULL
Table[1][120]: (dummy, xyyk) -> (xyyk , NULL) --> NULL
Table[1][122]: (dummy, zxcccc) -> (zxcccc , NULL) --> NULL

Leaving RSort, Performing Radix Sort.

Data3

outFile1.txt

```
Table[0][32]: (dummy, Cindy      ) -> (Cindy      , Alam       ) -> (Alam       ,
Siliang    ) -> (Siliang    , Andrade    ) -> (Andrade    , Hyungbin   ) -> (Hyungbin
, Robert     ) -> (Robert     , Hammad     ) -> (Hammad     , Jianhui    ) -> (Jianhui
, Nahian     ) -> (Nahian     , Yuhang     ) -> (Yuhang     , Kevin      ) -> (Kevin
, Matthew    ) -> (Matthew    , Naiem      ) -> (Naiem      , Frederick  ) ->
(Frederick  , Michael    ) -> (Michael    , Jiawei     ) -> (Jiawei     , Ye         )
-> (Ye         , Rupert     ) -> (Rupert     , Weiting    ) -> (Weiting    , Yulin
) -> (Yulin      , Russell    ) -> (Russell    , Justin     ) -> (Justin     , Juan
) -> (Juan       , Jason      ) -> (Jason      , Kenley     ) -> (Kenley     , Bret
) -> (Bret       , Wong       ) -> (Wong       , Calvin     ) -> (Calvin     , Matthew
) -> (Matthew    , Zaynab     ) -> (Zaynab     , Nectario   ) -> (Nectario   , Lei
) -> (Lei        , Steven     ) -> (Steven     , Martin     ) -> (Martin     ,
Jonathan   ) -> (Jonathan   , Christos   ) -> (Christos   , Carlos     ) -> (Carlos
, Bijaya     ) -> (Bijaya     , Asher      ) -> (Asher      , Wilber     ) -> (Wilber
, Archimed   ) -> (Archimed   , Aaron      ) -> (Aaron      , Abeesh     ) -> (Abeesh
, Umair      ) -> (Umair      , Joshua     ) -> (Joshua     , Lei        ) -> (Lei
, Murgray    ) -> (Murgray    , Jordon     ) -> (Jordon     , Rajendra   ) ->
(Rajendra   , Amreen     ) -> (Amreen     , Jamil      ) -> (Jamil      , Thomas     )
-> (Thomas     , Shelley    ) -> (Shelley    , Vincenzo   ) -> (Vincenzo   , Alexis
) -> (Alexis     , Jason      ) -> (Jason      , Shahan     ) -> (Shahan     , Angel
) -> (Angel      , Edwin      ) -> (Edwin      , Talha      ) -> (Talha      , Shadman
) -> (Shadman    , Gurnoor    ) -> (Gurnoor    , Krzysztof  ) -> (Krzysztof  , Eunhee
) -> (Eunhee     , Rajin      ) -> (Rajin      , Arellano   ) -> (Arellano   , Piyush
) -> (Piyush     , Guo        ) -> (Guo        , Ilma       ) -> (Ilma       , Kaur
) -> (Kaur       , Jurgen     ) -> (Jurgen     , Bashir     ) -> (Bashir     ,
Jhonatan   ) -> (Jhonatan   , Landerer   ) -> (Landerer   , Mohammed   ) -> (Mohammed
, Patel      ) -> (Patel      , Shaharin   ) -> (Shaharin   , Ziyu       ) -> (Ziyu
, Zhang      ) -> (Zhang      , Eliyahu    ) -> (Eliyahu    , Rajesh     ) -> (Rajesh
, Mandal     ) -> (Mandal     , Romanbir   ) -> (Romanbir   , Shaharin   ) ->
(Shaharin   , Juan       ) -> (Juan       , Stefan     ) -> (Stefan     , Petersen   )
-> (Petersen   , Tavoli     ) -> (Tavoli     , Ali        ) -> (Ali        , Varadi
) -> (Varadi     , John       ) -> (John       , Navi       ) -> (Navi       , Jorge
) -> (Jorge      , Ng         ) -> (Ng         , Juan       ) -> (Juan       , Marc
) -> (Marc       , Panzer     ) -> (Panzer     , Rayamajhee ) -> (Rayamajhee ,
Christian  ) -> (Christian  , Mcdonald   ) -> (Mcdonald   , Sean       ) -> (Sean
, Uliano     ) -> (Uliano     , Hassan     ) -> (Hassan     , Mohamed    ) -> (Mohamed
, MinasSaad  ) -> (MinasSaad  , Weipei     ) -> (Weipei     , Dashi      ) -> (Dashi
, Ryan       ) -> (Ryan       , Guerrero   ) -> (Guerrero   , Joshua     ) -> (Joshua
, Bashir     ) -> (Bashir     , Rajin      ) -> (Rajin      , Arellano   ) ->
(Arellano   , Tenzin     ) -> (Tenzin     , Ortiz      ) -> (Ortiz      , Muhtasim   )
-> (Muhtasim   , Juan       ) -> (Juan       , Roberto    ) -> (Roberto    , Chen
) -> (Chen       , Erik       ) -> (Erik       , Rambaran   ) -> (Rambaran   , Gildian
) -> (Gildian    , Hossain    ) -> (Hossain    , Kenneth    ) -> (Kenneth    ,
SharmaSam  ) -> (SharmaSam  , Paguay     ) -> (Paguay     , Sharad     ) -> (Sharad
, Rahman     ) -> (Rahman     , Genesis    ) -> (Genesis    , Roman      ) -> (Roman
, Kenny      ) -> (Kenny      , Wu         ) -> (Wu         , NULL) --> NULL
Table[0][114]: (dummy, Christopher) -> (Christopher, Christopher) -> (Christopher,
NULL) --> NULL

Aaron Abeesh Alam Alexis Ali Amreen Andrade Angel Archimed Arellano Arellano Asher
Bashir Bashir Bijaya Bret Calvin Carlos Chen Christian Christophe Christophe Christos
Cindy Dashi Edwin Eliyahu Erik Eunhee Frederick Genesis Gildian Guerrero Guo Gurnoor
Hammad Hassan Hossain Hyungbin Ilma Jamil Jason Jason Jhonatan Jianhui Jiawei John
Jonathan Jordon Jorge Joshua Joshua Juan Juan Juan Juan Jurgen Justin Kaur Kenley
Kenneth Kenny Kevin Krzysztof Landerer Lei Lei Mandal Marc Martin Matthew Matthew
Mcdonald Michael MinasSaad Mohamed Mohammed Muhtasim Murgray Nahian Naiem Navi
Nectario Ng Ortiz Paguay Panzer Patel Petersen Piyush Rahman Rajendra Rajesh Rajin
Rajin Rambaran Rayamajhee Robert Roberto Roman Romanbir Rupert Russell Ryan Sean
Shadman Shahan Shaharin Shaharin Sharad SharmaSam Shelley Siliang Stefan Steven Talha
Tavoli Tenzin Thomas Uliano Umair Varadi Vincenzo Weipei Weiting Wilber Wong Wu Ye
Yuhang Yulin Zaynab Zhang Ziyu
```

deBugFile.txt

Performing first Reading to find the longest data string in the input fileEntering RSort, Performing Radix Sort.
Entering populateFirstTable().
In RSort, paddedData is Cindy ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Alam ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Siliang ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Andrade ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Hyungbin ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Robert ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Hammad ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Jianhui ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Nahian ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Yuhang ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Kevin ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Matthew ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Naiem ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Frederick ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Christopher; currentIndex = 10; hashIndex = 114; currentTable = 0
In RSort, paddedData is Michael ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Jiawei ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Ye ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Rupert ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Weiting ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Yulin ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Russell ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Justin ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Juan ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Jason ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Kenley ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Bret ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Wong ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Calvin ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Matthew ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Zaynab ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Nectario ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Lei ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Steven ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Martin ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Jonathan ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Christos ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Carlos ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Bijaya ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Asher ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Wilber ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Archimed ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Aaron ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Abeesh ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Umair ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Joshua ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Lei ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Murgray ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Jordon ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Rajendra ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Amreen ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Jamil ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Thomas ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Shelley ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Vincenzo ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Alexis ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Jason ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Shahan ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Angel ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Edwin ; currentIndex = 10; hashIndex = 32; currentTable = 0

In RSort, paddedData is Talha   ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Shadman   ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Gurnoor   ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Krzysztof ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Eunhee   ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Rajin   ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Arellano   ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Piyush   ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Guo   ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Ilma   ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Kaur   ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Jurgen   ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Bashir   ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Jhonatan   ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Landerer   ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Mohammed   ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Patel   ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Shaharin   ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Ziyu   ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Zhang   ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Eliyahu   ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Rajesh   ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Mandal   ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Romanbir   ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Shaharin   ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Juan   ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Stefan   ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Petersen   ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Tavoli   ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Ali   ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Varadi   ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is John   ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Navi   ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Jorge   ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Ng   ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Juan   ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Marc   ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Panzer   ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Christopher; currentIndex = 10; hashIndex = 114; currentTable = 0
In RSort, paddedData is Rayamajhee ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Christian ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Mcdonald   ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Sean   ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Uliano   ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Hassan   ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Mohamed   ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is MinasSaad ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Weipei   ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Dashi   ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Ryan   ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Guerrero   ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Joshua   ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Bashir   ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Rajin   ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Arellano   ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Tenzin   ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Ortiz   ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Muhtasim   ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Juan   ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Roberto   ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Chen   ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Erik   ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Rambaran   ; currentIndex = 10; hashIndex = 32; currentTable = 0

In RSort, paddedData is Gildian ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Hossain ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Kenneth ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is SharmaSam ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Paguay ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Sharad ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Rahman ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Genesis ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Roman ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Kenny ; currentIndex = 10; hashIndex = 32; currentTable = 0
In RSort, paddedData is Wu ; currentIndex = 10; hashIndex = 32; currentTable = 0
Finish insert all paddedData into the hashTable[0], the hashTable shown belowTable[0][32]: (dummy, Cindy) -> (Cindy , Alam) -> (Alam , Siliang) -> (Siliang , Andrade) -> (Andrade , Hyungbin) -> (Hyungbin , Robert) -> (Robert , Hammad) -> (Hammad , Jianhui) -> (Jianhui , Nahian) -> (Nahian , Yuhang) -> (Yuhang , Kevin) -> (Kevin , Matthew) -> (Matthew , Naiem) -> (Naiem , Frederick) -> (Frederick , Michael) -> (Michael , Jiawei) -> (Jiawei , Ye) -> (Ye , Rupert) -> (Rupert , Weiting) -> (Weiting , Yulin) -> (Yulin , Russell) -> (Russell , Justin) -> (Justin , Juan) -> (Juan , Jason) -> (Jason , Kenley) -> (Kenley , Bret) -> (Bret , Wong) -> (Wong , Calvin) -> (Calvin , Matthew) -> (Matthew , Zaynab) -> (Zaynab , Nectario) -> (Nectario , Lei) -> (Lei , Steven) -> (Steven , Martin) -> (Martin , Jonathan) -> (Jonathan , Christos) -> (Christos , Carlos) -> (Carlos , Bijaya) -> (Bijaya , Asher) -> (Asher , Wilber) -> (Wilber , Archimed) -> (Archimed , Aaron) -> (Aaron , Abeesh) -> (Abeesh , Umair) -> (Umair , Joshua) -> (Joshua , Lei) -> (Lei , Murgray) -> (Murgray , Jordon) -> (Jordon , Rajendra) -> (Rajendra , Amreen) -> (Amreen , Jamil) -> (Jamil , Thomas) -> (Thomas , Shelley) -> (Shelley , Vincenzo) -> (Vincenzo , Alexis) -> (Alexis , Jason) -> (Jason , Shahan) -> (Shahan , Angel) -> (Angel , Edwin) -> (Edwin , Talha) -> (Talha , Shadman) -> (Shadman , Gurnoor) -> (Gurnoor , Krzysztof) -> (Krzysztof , Eunhee) -> (Eunhee , Rajin) -> (Rajin , Arellano) -> (Arellano , Piyush) -> (Piyush , Guo) -> (Guo , Ilma) -> (Ilma , Kaur) -> (Kaur , Jurgen) -> (Jurgen , Bashir) -> (Bashir , Jhonatan) -> (Jhonatan , Landerer) -> (Landerer , Mohammed) -> (Mohammed , Patel) -> (Patel , Shaharin) -> (Shaharin , Ziyu) -> (Ziyu , Zhang) -> (Zhang , Eliyahu) -> (Eliyahu , Rajesh) -> (Rajesh , Mandal) -> (Mandal , Romanbir) -> (Romanbir , Shaharin) -> (Shaharin , Juan) -> (Juan , Stefan) -> (Stefan , Petersen) -> (Petersen , Tavoli) -> (Tavoli , Ali) -> (Ali , Varadi) -> (Varadi , John) -> (John , Navi) -> (Navi , Jorge) -> (Jorge , Ng) -> (Ng , Juan) -> (Juan , Marc) -> (Marc , Panzer) -> (Panzer , Rayamajhee) -> (Rayamajhee , Christian) -> (Christian , Mcdonald) -> (Mcdonald , Sean) -> (Sean , Uliano) -> (Uliano , Hassan) -> (Hassan , Mohamed) -> (Mohamed , MinasSaad) -> (MinasSaad , Weipei) -> (Weipei , Dashi) -> (Dashi , Ryan) -> (Ryan , Guerrero) -> (Guerrero , Joshua) -> (Joshua , Bashir) -> (Bashir , Rajin) -> (Rajin , Arellano) -> (Arellano , Tenzin) -> (Tenzin , Ortiz) -> (Ortiz , Muhtasim) -> (Muhtasim , Juan) -> (Juan , Roberto) -> (Roberto , Chen) -> (Chen , Erik) -> (Erik , Rambaran) -> (Rambaran , Gildian) -> (Gildian , Hossain) -> (Hossain , Kenneth) -> (Kenneth , SharmaSam) -> (SharmaSam , Paguay) -> (Paguay , Sharad) -> (Sharad , Rahman) -> (Rahman , Genesis) -> (Genesis , Roman) -> (Roman , Kenny) -> (Kenny , Wu) -> (Wu , NULL) --> NULL
Table[0][114]: (dummy, Christopher) -> (Christopher, Christopher) -> (Christopher, NULL) --> NULL

Leaving populateFirstTable().
In RSort, after swap tables, currentIndex = 10; currentTable = 1, nextTable = 0
Table[1][32]: (dummy, Cindy) -> (Cindy , Alam) -> (Alam , Siliang) -> (Siliang , Andrade) -> (Andrade , Hyungbin) -> (Hyungbin , Robert) -> (Robert , Hammad) -> (Hammad , Jianhui) -> (Jianhui , Nahian) -> (Nahian , Yuhang) -> (Yuhang , Kevin) -> (Kevin , Matthew) -> (Matthew , Naiem) -> (Naiem , Frederick) -> (Frederick , Michael) -> (Michael , Jiawei) -> (Jiawei , Ye) -> (Ye , Rupert) -> (Rupert , Weiting) -> (Weiting , Yulin) -> (Yulin , Russell) -> (Russell , Justin) -> (Justin , Juan) -> (Juan , Jason) -> (Jason , Kenley) -> (Kenley , Bret) -> (Bret , Wong) -> (Wong , Calvin) -> (Calvin , Matthew) -> (Matthew , Zaynab) -> (Zaynab , Nectario) -> (Nectario , Lei) -> (Lei , Steven) -> (Steven , Martin) -> (Martin , Jonathan) -> (Jonathan , Christos) -> (Christos , Carlos) -> (Carlos , Bijaya) -> (Bijaya , Asher) -> (Asher , Wilber) -> (Wilber , Archimed) -> (Archimed , Aaron) -> (Aaron , Abeesh) -> (Abeesh , Umair) -> (Umair , Joshua) -> (Joshua , Lei) -> (Lei , Murgray) -> (Murgray , Jordon) -> (Jordon , Rajendra) -> (Rajendra , Amreen) -> (Amreen , Jamil) -> (Jamil , Thomas) -> (Thomas , Shelley) -> (Shelley , Vincenzo) -> (Vincenzo , Alexis) -> (Alexis , Jason) -> (Jason , Shahan) -> (Shahan , Angel) -> (Angel , Edwin) -> (Edwin , Talha) -> (Talha , Shadman) -> (Shadman , Gurnoor) -> (Gurnoor , Krzysztof) -> (Krzysztof , Eunhee) -> (Eunhee , Rajin) -> (Rajin , Arellano) -> (Arellano , Piyush) -> (Piyush , Guo) -> (Guo , Ilma) -> (Ilma , Kaur) -> (Kaur , Jurgen) -> (Jurgen , Bashir) -> (Bashir , Jhonatan) -> (Jhonatan , Landerer) -> (Landerer , Mohammed) -> (Mohammed , Patel) -> (Patel , Shaharin) -> (Shaharin , Ziyu) -> (Ziyu , Zhang) -> (Zhang , Eliyahu) -> (Eliyahu , Rajesh) -> (Rajesh , Mandal) -> (Mandal , Romanbir) -> (Romanbir , Shaharin) -> (Shaharin , Juan) -> (Juan , Stefan) -> (Stefan , Petersen) -> (Petersen , Tavoli) -> (Tavoli , Ali) -> (Ali , Varadi) -> (Varadi , John) ->

(John , Navi) -> (Navi , Jorge) -> (Jorge , Ng) -> (Ng , Juan) -> (Juan , Marc) -> (Marc , Panzer) -> (Panzer , Rayamajhee) -> (Rayamajhee , Christian) -> (Christian , Mcdonald) -> (Mcdonald , Sean) -> (Sean , Uliano) -> (Uliano , Hassan) -> (Hassan , Mohamed) -> (Mohamed , MinasSaad) -> (MinasSaad , Weipei) -> (Weipei , Dashi) -> (Dashi , Ryan) -> (Ryan , Guerrero) -> (Guerrero , Joshua) -> (Joshua , Bashir) -> (Bashir , Rajin) -> (Rajin , Arellano) -> (Arellano , Tenzin) -> (Tenzin , Ortiz) -> (Ortiz , Muhtasim) -> (Muhtasim , Juan) -> (Juan , Roberto) -> (Roberto , Chen) -> (Chen , Erik) -> (Erik , Rambaran) -> (Rambaran , Gildian) -> (Gildian , Hossain) -> (Hossain , Kenneth) -> (Kenneth , SharmaSam) -> (SharmaSam , Paguay) -> (Paguay , Sharad) -> (Sharad , Rahman) -> (Rahman , Genesis) -> (Genesis , Roman) -> (Roman , Kenny) -> (Kenny , Wu) -> (Wu , NULL) --> NULL
Table[1][114]: (dummy, Christopher) -> (Christopher, Christopher) -> (Christopher, NULL) --> NULL

In RSort, after swap tables, currentIndex = 9; currentTable = 0, nextTable = 1
Table[0][32]: (dummy, Cindy) -> (Cindy , Alam) -> (Alam , Siliang) -> (Siliang , Andrade) -> (Andrade , Hyungbin) -> (Hyungbin , Robert) -> (Robert , Hammad) -> (Hammad , Jianhui) -> (Jianhui , Nahian) -> (Nahian , Yuhang) -> (Yuhang , Kevin) -> (Kevin , Matthew) -> (Matthew , Naiem) -> (Naiem , Frederick) -> (Frederick , Michael) -> (Michael , Jiawei) -> (Jiawei , Ye) -> (Ye , Rupert) -> (Rupert , Weiting) -> (Weiting , Yulin) -> (Yulin , Russell) -> (Russell , Justin) -> (Justin , Juan) -> (Juan , Jason) -> (Jason , Kenley) -> (Kenley , Bret) -> (Bret , Wong) -> (Wong , Calvin) -> (Calvin , Matthew) -> (Matthew , Zaynab) -> (Zaynab , Nectario) -> (Nectario , Lei) -> (Lei , Steven) -> (Steven , Martin) -> (Martin , Jonathan) -> (Jonathan , Christos) -> (Christos , Carlos) -> (Carlos , Bijaya) -> (Bijaya , Asher) -> (Asher , Wilber) -> (Wilber , Archimed) -> (Archimed , Aaron) -> (Aaron , Abeesh) -> (Abeesh , Umair) -> (Umair , Joshua) -> (Joshua , Lei) -> (Lei , Murgray) -> (Murgray , Jordon) -> (Jordon , Rajendra) -> (Rajendra , Amreen) -> (Amreen , Jamil) -> (Jamil , Thomas) -> (Thomas , Shelley) -> (Shelley , Vincenzo) -> (Vincenzo , Alexis) -> (Alexis , Jason) -> (Jason , Shahan) -> (Shahan , Angel) -> (Angel , Edwin) -> (Edwin , Talha) -> (Talha , Shadman) -> (Shadman , Gurnoor) -> (Gurnoor , Krzysztof) -> (Krzysztof , Eunhee) -> (Eunhee , Rajin) -> (Rajin , Arellano) -> (Arellano , Piyush) -> (Piyush , Guo) -> (Guo , Ilma) -> (Ilma , Kaur) -> (Kaur , Jurgen) -> (Jurgen , Bashir) -> (Bashir , Jhonatan) -> (Jhonatan , Landerer) -> (Landerer , Mohammed) -> (Mohammed , Patel) -> (Patel , Shaharin) -> (Shaharin , Ziyu) -> (Ziyu , Zhang) -> (Zhang , Eliyahu) -> (Eliyahu , Rajesh) -> (Rajesh , Mandal) -> (Mandal , Romanbir) -> (Romanbir , Shaharin) -> (Shaharin , Juan) -> (Juan , Stefan) -> (Stefan , Petersen) -> (Petersen , Tavoli) -> (Tavoli , Ali) -> (Ali , Varadi) -> (Varadi , John) -> (John , Navi) -> (Navi , Jorge) -> (Jorge , Ng) -> (Ng , Juan) -> (Juan , Marc) -> (Marc , Panzer) -> (Panzer , Christian) -> (Christian , Mcdonald) -> (Mcdonald , Sean) -> (Sean , Uliano) -> (Uliano , Hassan) -> (Hassan , Mohamed) -> (Mohamed , MinasSaad) -> (MinasSaad , Weipei) -> (Weipei , Dashi) -> (Dashi , Ryan) -> (Ryan , Guerrero) -> (Guerrero , Joshua) -> (Joshua , Bashir) -> (Bashir , Rajin) -> (Rajin , Arellano) -> (Arellano , Tenzin) -> (Tenzin , Ortiz) -> (Ortiz , Muhtasim) -> (Muhtasim , Juan) -> (Juan , Roberto) -> (Roberto , Chen) -> (Chen , Erik) -> (Erik , Rambaran) -> (Rambaran , Gildian) -> (Gildian , Hossain) -> (Hossain , Kenneth) -> (Kenneth , SharmaSam) -> (SharmaSam , Paguay) -> (Paguay , Sharad) -> (Sharad , Rahman) -> (Rahman , Genesis) -> (Genesis , Roman) -> (Roman , Kenny) -> (Kenny , Wu) -> (Wu , NULL) --> NULL
Table[0][101]: (dummy, Rayamajhee) -> (Rayamajhee , Christopher) -> (Christopher, Christopher) -> (Christopher, NULL) --> NULL

In RSort, after swap tables, currentIndex = 8; currentTable = 1, nextTable = 0
Table[1][32]: (dummy, Cindy) -> (Cindy , Alam) -> (Alam , Siliang) -> (Siliang , Andrade) -> (Andrade , Hyungbin) -> (Hyungbin , Robert) -> (Robert , Hammad) -> (Hammad , Jianhui) -> (Jianhui , Nahian) -> (Nahian , Yuhang) -> (Yuhang , Kevin) -> (Kevin , Matthew) -> (Matthew , Naiem) -> (Naiem , Michael) -> (Michael , Jiawei) -> (Jiawei , Ye) -> (Ye , Rupert) -> (Rupert , Weiting) -> (Weiting , Yulin) -> (Yulin , Russell) -> (Russell , Justin) -> (Justin , Juan) -> (Juan , Jason) -> (Jason , Kenley) -> (Kenley , Bret) -> (Bret , Wong) -> (Wong , Calvin) -> (Calvin , Matthew) -> (Matthew , Zaynab) -> (Zaynab , Nectario) -> (Nectario , Lei) -> (Lei , Steven) -> (Steven , Martin) -> (Martin , Jonathan) -> (Jonathan , Christos) -> (Christos , Carlos) -> (Carlos , Bijaya) -> (Bijaya , Asher) -> (Asher , Wilber) -> (Wilber , Archimed) -> (Archimed , Aaron) -> (Aaron , Abeesh) -> (Abeesh , Umair) -> (Umair , Joshua) -> (Joshua , Lei) -> (Lei , Murgray) -> (Murgray , Jordon) -> (Jordon , Rajendra) -> (Rajendra , Amreen) -> (Amreen , Jamil) -> (Jamil , Thomas) -> (Thomas , Shelley) -> (Shelley , Vincenzo) -> (Vincenzo , Alexis) -> (Alexis , Jason) -> (Jason , Shahan) -> (Shahan , Angel) -> (Angel , Edwin) -> (Edwin , Talha) -> (Talha , Shadman) -> (Shadman , Gurnoor) -> (Gurnoor , Eunhee) -> (Eunhee , Rajin) -> (Rajin , Arellano) -> (Arellano , Piyush) -> (Piyush , Guo) -> (Guo , Ilma) -> (Ilma , Kaur) -> (Kaur , Jurgen) -> (Jurgen , Bashir) -> (Bashir , Jhonatan) -> (Jhonatan , Landerer) -> (Landerer , Mohammed) -> (Mohammed , Patel) -> (Patel , Shaharin) -> (Shaharin , Ziyu) -> (Ziyu , Zhang) -> (Zhang , Eliyahu) -> (Eliyahu , Rajesh) -> (Rajesh , Mandal) -> (Mandal , Romanbir) -> (Romanbir , Shaharin) -> (Shaharin , Juan) -> (Juan , Stefan) -> (Stefan , Petersen) -> (Petersen , Tavoli) -> (Tavoli , Ali) -> (Ali , Varadi) -> (Varadi , John) -> (John , Navi) -> (Navi , Jorge) ->

(Jorge , Ng) -> (Ng , Juan) -> (Juan , Marc) -> (Marc , Panzer) -> (Panzer , Mcdonald) -> (Mcdonald , Sean) -> (Sean , Uliano) -> (Uliano , Hassan) -> (Hassan , Mohamed) -> (Mohamed , Weipei) -> (Weipei , Dashi) -> (Dashi , Ryan) -> (Ryan , Guerrero) -> (Guerrero , Joshua) -> (Joshua , Bashir) -> (Bashir , Rajin) -> (Rajin , Arellano) -> (Arellano , Tenzin) -> (Tenzin , Ortiz) -> (Ortiz , Muhtasim) -> (Muhtasim , Juan) -> (Juan , Roberto) -> (Roberto , Chen) -> (Chen , Erik) -> (Erik , Rambaran) -> (Rambaran , Gildian) -> (Gildian , Hossain) -> (Hossain , Kenneth) -> (Kenneth , Paguay) -> (Paguay , Sharad) -> (Sharad , Rahman) -> (Rahman , Genesis) -> (Genesis , Roman) -> (Roman , Kenny) -> (Kenny , Wu) -> (Wu , NULL) --> NULL
Table[1][100]: (dummy, MinasSaad) -> (MinasSaad , NULL) --> NULL
Table[1][101]: (dummy, Rayamajhee) -> (Rayamajhee , NULL) --> NULL
Table[1][102]: (dummy, Krzysztof) -> (Krzysztof , NULL) --> NULL
Table[1][104]: (dummy, Christopher) -> (Christopher, Christopher) -> (Christopher, NULL) --> NULL
Table[1][107]: (dummy, Frederick) -> (Frederick , NULL) --> NULL
Table[1][109]: (dummy, SharmaSam) -> (SharmaSam , NULL) --> NULL
Table[1][110]: (dummy, Christian) -> (Christian , NULL) --> NULL

In RSort, after swap tables, currentIndex = 7; currentTable = 0, nextTable = 1
Table[0][32]: (dummy, Cindy) -> (Cindy , Alam) -> (Alam , Siliang) -> (Siliang , Andrade) -> (Andrade , Robert) -> (Robert , Hammad) -> (Hammad , Jianhui) -> (Jianhui , Nahian) -> (Nahian , Yuhang) -> (Yuhang , Kevin) -> (Kevin , Matthew) -> (Matthew , Naiem) -> (Naiem , Michael) -> (Michael , Jiawei) -> (Jiawei , Ye) -> (Ye , Rupert) -> (Rupert , Weiting) -> (Weiting , Yulin) -> (Yulin , Russell) -> (Russell , Justin) -> (Justin , Juan) -> (Juan , Jason) -> (Jason , Kenley) -> (Kenley , Bret) -> (Bret , Wong) -> (Wong , Calvin) -> (Calvin , Matthew) -> (Matthew , Zaynab) -> (Zaynab , Lei) -> (Lei , Steven) -> (Steven , Martin) -> (Martin , Carlos) -> (Carlos , Bijaya) -> (Bijaya , Asher) -> (Asher , Wilber) -> (Wilber , Aaron) -> (Aaron , Abeesh) -> (Abeesh , Umair) -> (Umair , Joshua) -> (Joshua , Lei) -> (Lei , Murgray) -> (Murgray , Jordon) -> (Jordon , Amreen) -> (Amreen , Jamil) -> (Jamil , Thomas) -> (Thomas , Shelley) -> (Shelley , Alexis) -> (Alexis , Jason) -> (Jason , Shahan) -> (Shahan , Angel) -> (Angel , Edwin) -> (Edwin , Talha) -> (Talha , Shadman) -> (Shadman , Gurnoor) -> (Gurnoor , Eunhee) -> (Eunhee , Rajin) -> (Rajin , Piyush) -> (Piyush , Guo) -> (Guo , Ilma) -> (Ilma , Kaur) -> (Kaur , Jurgen) -> (Jurgen , Bashir) -> (Bashir , Patel) -> (Patel , Ziyu) -> (Ziyu , Zhang) -> (Zhang , Eliyahu) -> (Eliyahu , Rajesh) -> (Rajesh , Mandal) -> (Mandal , Juan) -> (Juan , Stefan) -> (Stefan , Tavoli) -> (Tavoli , Ali) -> (Ali , Varadi) -> (Varadi , John) -> (John , Navi) -> (Navi , Jorge) -> (Jorge , Ng) -> (Ng , Juan) -> (Juan , Marc) -> (Marc , Panzer) -> (Panzer , Sean) -> (Sean , Uliano) -> (Uliano , Hassan) -> (Hassan , Mohamed) -> (Mohamed , Weipei) -> (Weipei , Dashi) -> (Dashi , Ryan) -> (Ryan , Joshua) -> (Joshua , Bashir) -> (Bashir , Rajin) -> (Rajin , Tenzin) -> (Tenzin , Ortiz) -> (Ortiz , Juan) -> (Juan , Roberto) -> (Roberto , Chen) -> (Chen , Erik) -> (Erik , Gildian) -> (Gildian , Hossain) -> (Hossain , Kenneth) -> (Kenneth , Paguay) -> (Paguay , Sharad) -> (Sharad , Rahman) -> (Rahman , Genesis) -> (Genesis , Roman) -> (Roman , Kenny) -> (Kenny , Wu) -> (Wu , NULL) --> NULL
Table[0][97]: (dummy, Rajendra) -> (Rajendra , MinasSaad) -> (MinasSaad , SharmaSam) -> (SharmaSam , Christian) -> (Christian , NULL) --> NULL
Table[0][99]: (dummy, Frederick) -> (Frederick , NULL) --> NULL
Table[0][100]: (dummy, Archimed) -> (Archimed , Mohammed) -> (Mohammed , Mcdonald) -> (Mcdonald , NULL) --> NULL
Table[0][104]: (dummy, Rayamajhee) -> (Rayamajhee , NULL) --> NULL
Table[0][109]: (dummy, Muhtasim) -> (Muhtasim , NULL) --> NULL
Table[0][110]: (dummy, Hyungbin) -> (Hyungbin , Jonathan) -> (Jonathan , Jhonatan) -> (Jhonatan , Shaharin) -> (Shaharin , Shaharin) -> (Shaharin , Petersen) -> (Petersen , Rambaran) -> (Rambaran , NULL) --> NULL
Table[0][111]: (dummy, Nectario) -> (Nectario , Vincenzo) -> (Vincenzo , Arellano) -> (Arellano , Guerrero) -> (Guerrero , Arellano) -> (Arellano , Krzysztof) -> (Krzysztof , NULL) --> NULL
Table[0][112]: (dummy, Christopher) -> (Christopher, Christopher) -> (Christopher, NULL) --> NULL
Table[0][114]: (dummy, Landerer) -> (Landerer , Romanbir) -> (Romanbir , NULL) --> NULL
Table[0][115]: (dummy, Christos) -> (Christos , NULL) --> NULL

In RSort, after swap tables, currentIndex = 6; currentTable = 1, nextTable = 0
Table[1][32]: (dummy, Cindy) -> (Cindy , Alam) -> (Alam , Robert) -> (Robert , Hammad) -> (Hammad , Nahian) -> (Nahian , Yuhang) -> (Yuhang , Kevin) -> (Kevin , Naiem) -> (Naiem , Jiawei) -> (Jiawei , Ye) -> (Ye , Rupert) -> (Rupert , Yulin) -> (Yulin , Justin) -> (Justin , Juan) -> (Juan , Jason) -> (Jason , Kenley) -> (Kenley , Bret) -> (Bret , Wong) -> (Wong , Calvin) -> (Calvin , Zaynab) -> (Zaynab , Lei) -> (Lei , Steven) -> (Steven , Martin) -> (Martin , Carlos) -> (Carlos , Bijaya) -> (Bijaya , Asher) -> (Asher , Wilber) -> (Wilber , Aaron) -> (Aaron , Abeesh) -> (Abeesh , Umair) -> (Umair , Joshua) -> (Joshua , Lei) -> (Lei , Jordon) -> (Jordon , Amreen) -> (Amreen , Jamil) -> (Jamil , Thomas) -> (Thomas , Alexis) -> (Alexis , Jason) -> (Jason , Shahan) -> (Shahan