# POLITECNICO DI TORINO


# OSES LAB #5–Heartbeat moniton

Department of CONTROL AND COMPUTER ENGINEERING (DAUIN)
Master's degree in Mechatronic Engineering
2023/2024
Operating systems for embedded systems– Prof. Violante
Laboratory 05
Wednesday, 18 January 2024

Student:
s320296 DI MARTINO GIULIO

## Exercise

A heartbeat sensor has to be implemented on Arduino Uno, as follows:

- The PPG sensor is simulated on Simulide using a waveform generator with these

  settings:
  - Id:WaveGen-16
  - Voltage:5V
  - Frequency:1Hz.
  - Trianglewaveshape.

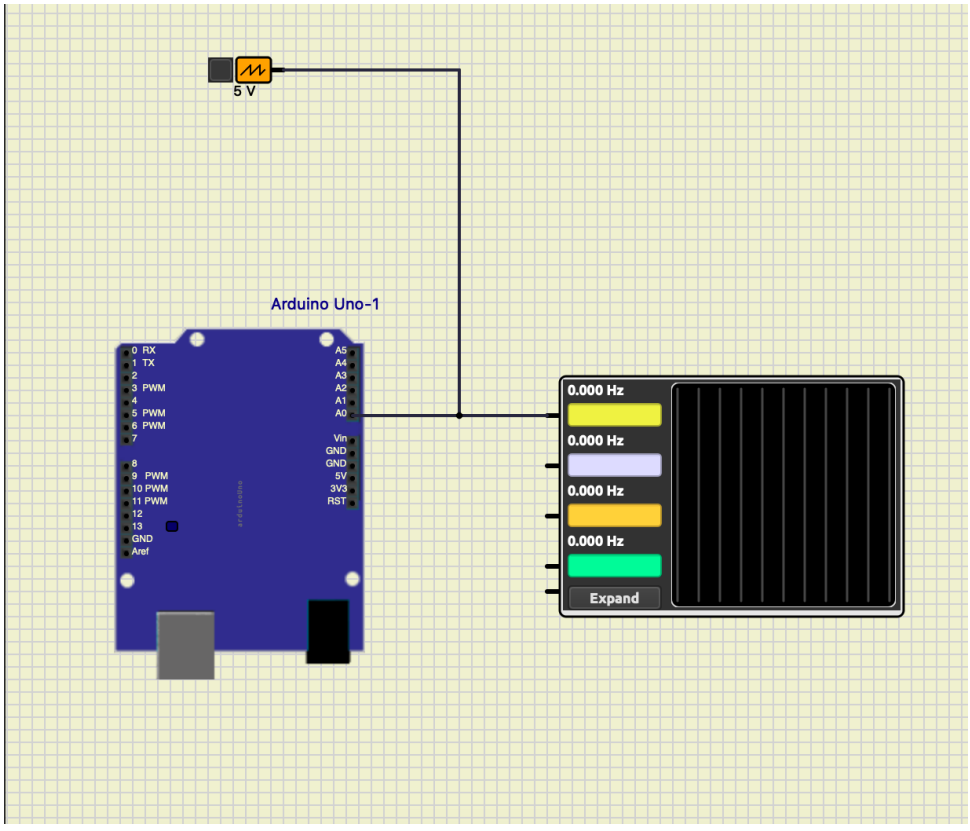
- The signal coming from waveform generator has to be sampled with a frequency of 50 Hz. From the signal is asked to the student to:

  - Convertthesignalinaproperformat. ○ Collectthedatainabuffer.
  - Implementacircularbuffer.

- Once filled the buffer, every 50 samples a task is activated. This task must: ○ Findthepeaksinthebuffer.
  - Calculatethepeak-to-peakdistance(RR).
  - CalculatethepunctualHeartRateasHR=60/RR.

  - PrinttheHeartRatevalueusingtheserialport.

- The Simulide scheme to build-up is the follow:



## C-Code

```
##include "tpl_os.h"
#include "Arduino.h"
#include "tpl_com.h"

DeclareAlarm(a20msec);
DeclareAlarm(a1000msec);

#define K 137

int circularBuffer[K];
static int itemCount = 0;
static int queue = 0;

void setup()
{
  pinMode(A0,INPUT);
  Serial.begin(115200); //115200 bps, 8N1

}

// la task s deve essere attivata ogni 50 Hz = 20 ms
TASK(TaskS) {
  GetResource(Sem);

  int sensorValue = analogRead(A0);
  Serial.print("itemCount: ");
  Serial.print(itemCount);
  Serial.print(" A0: ");
  Serial.print(sensorValue);
  Serial.print("\t");

  Serial.print("head:");
  Serial.println(itemCount%K);
  Serial.print("\t\t");

  circularBuffer[itemCount%K] = sensorValue;

  itemCount++;

  if(itemCount%K==queue%K)
    queue++;
  Serial.print("queue: ");
  Serial.print(queue%K);
  Serial.print("\n");
  ReleaseResource(Sem);
  }


  TASK(TaskD) {

    static int i;
    static int RR=0;
    static int HR;
    static int Max=0;
    static int Min=0;
    static int T;
    static int index;

    GetResource(Sem);

    Max=circularBuffer[0];
    Min=circularBuffer[0];

    if(itemCount > K)
      T=K;
    else
      T=itemCount;

    for (i=0;i<T;i++) {

      index=(queue+i)%K;
```

```
      Serial.print("data:\t ");
      Serial.print(index);
      Serial.print("\n");

    if (circularBuffer[index]>Max)

      Max=circularBuffer[index];

    if (circularBuffer[index]<=Min)

      Min=circularBuffer[index];
    }


    RR=Max-Min;
    Serial.print("\nRR:");
    Serial.print(RR);
    HR=60*100/RR;
    Serial.print("\nHR%:");
    Serial.print(HR);
    Serial.print("\n");
    ReleaseResource(Sem);
    }
```

I define the circular buffer dimension K=134 because 1 Hz correspond to 60 bpm, the e lower heart rate value is 45 bpm that correspond to 0.75 Hz that correspond to 1333.3 ms. To have a sample I need 20 ms so to the circular buffer must have the dimension of 133.3 that is big enough to contain enough to contain at least 2 consecutive beats. I chose 134 because is an integer number. After the simulation I notice that the perfect value for K = 137 because there is a discrepancy due to Arduino

```
TASK(TaskS) {
  GetResource(Sem);
  Serial.print("itemCount: ");
  Serial.print(itemCount);
  Serial.print("\t");
  Serial.println(itemCount%K);
  int sensorValue = analogRead(A0);
  int item=itemCount%K;
  Serial.print("head:");

  Serial.print("\t");

  circularBuffer[itemCount%K] = sensorValue;

  itemCount++;

  if(itemCount%K==queue%K)
    queue++;
  Serial.print("queue: ");
  Serial.print(queue%K);
  Serial.print("\n");
  ReleaseResource(Sem);
  }
```

This task is the task that sample the data and send this data to the circular buffer I use two variable `itemCount` and `queue` for implement the circular buffer correctly:

- Circular Buffer Management: The modulo operation ensures a circular behavior for the buffer. The head index, represented by **itemCount % K**, is printed to the serial monitor for tracking purposes. The system also increments **itemCount** to manage the circular buffer's insertion index.
- Queue Management: The code introduces a mechanism to manage a "queue," denoted by the variable **queue**. If the current index in the circular buffer (**itemCount % K**) is equal to

the queue index (**queue % K**), the queue index is incremented. This mechanism likely serves to handle the circular buffer as a queue.

```
TASK(TaskD) {

    static int i;
    static int RR=0;
    static int HR;
    static int Max=0;
    static int Min=0;
    static int T;
    static int index;

    GetResource(Sem);

    Max=circularBuffer[0];
    Min=circularBuffer[0];

    if(itemCount > K)
      T=K;
    else
      T=itemCount;

    for (i=0;i<T;i++) {

      index=(queue+i)%K;

      Serial.print("data: ");
      Serial.print(index);

    if (circularBuffer[index]>Max)

      Max=circularBuffer[index];

    if (circularBuffer[index]<=Min)

      Min=circularBuffer[index];
    }


    RR=Max-Min;
    Serial.print("\nRR:");
    Serial.print(RR);
    HR=60*100/RR; // percent value
    Serial.print("\nHR%:");
    Serial.print(HR);
    Serial.print("\n");
    ReleaseResource(Sem);
    }
```

This task is useful to check each item in the circular buffer and find the max and the min after that I find the peak-to-peak distance RR and the punctual Heart Rate

```
if(itemCount > K)
      T=K;
    else
      T=itemCount;
```

With this part I check if the circular buffer has been filled once becouse if it's the first run is useless to check each value of the buffer

OIL_FILE:

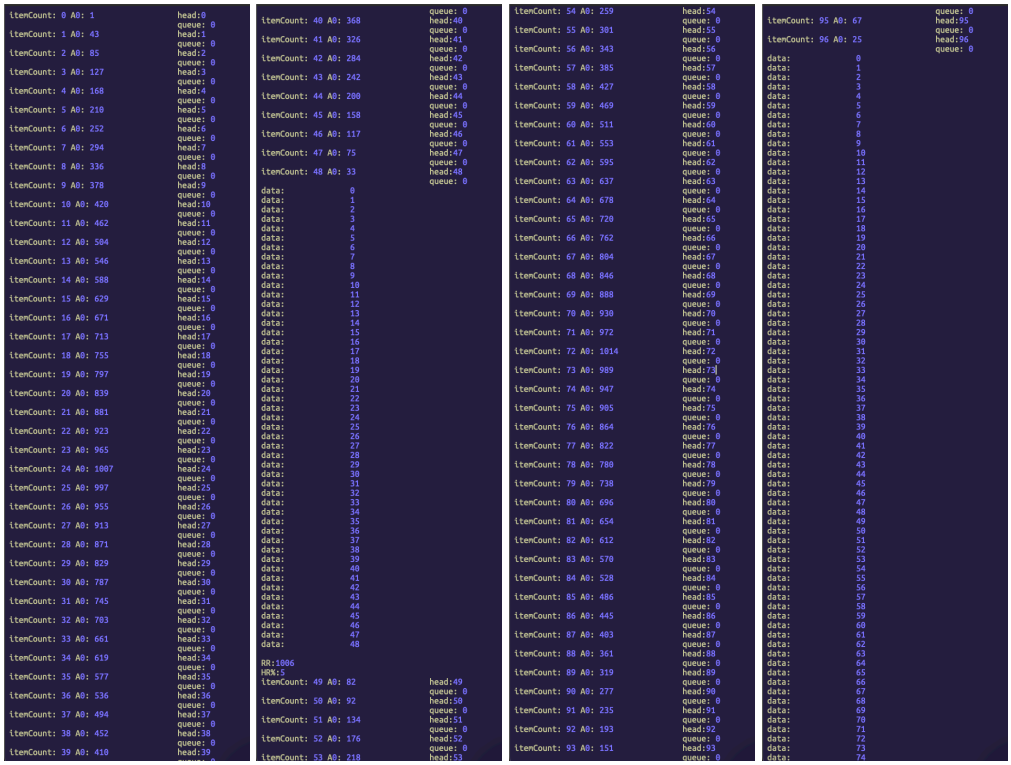
```
OIL_VERSION = "2.5" : "test" ;

CPU test {
  OS config {
    STATUS = STANDARD;
```

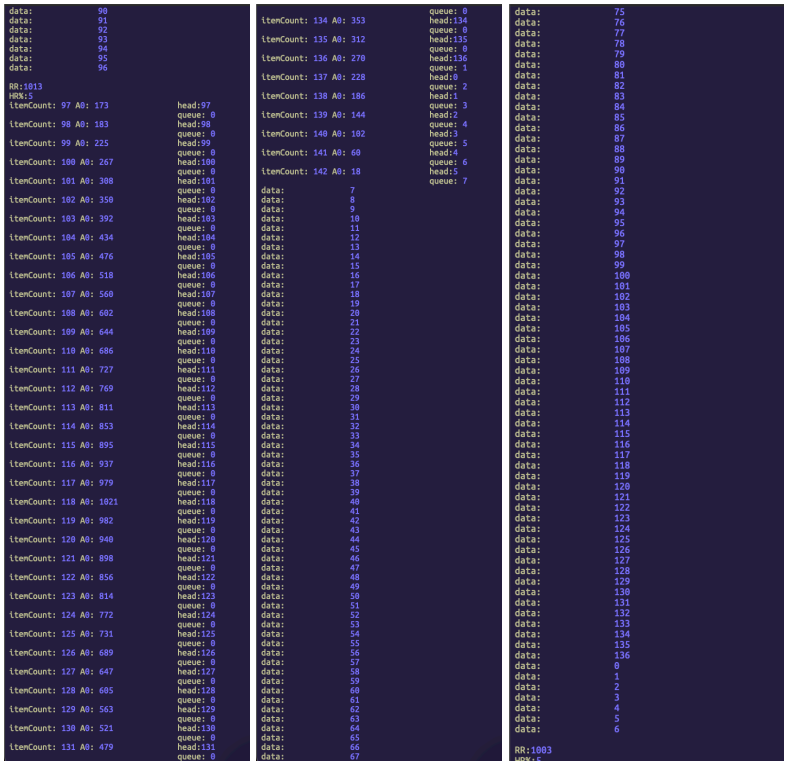
```
    BUILD = TRUE {
      TRAMPOLINE_BASE_PATH = "../../../..";
      APP_NAME = "lab51";
      APP_SRC = "lab5.1.cpp";
      CPPCOMPILER = "avr-g++";
      COMPILER = "avr-gcc";
      LINKER = "avr-gcc";
      ASSEMBLER = "avr-gcc";
      COPIER = "avr-objcopy";
      SYSTEM = PYTHON;
      LIBRARY = serial;
    };
    SYSTEM_CALL = TRUE;
  };

  APPMODE stdAppmode {};

  RESOURCE Sem{
  RESOURCEPROPERTY = STANDARD;
   };

  ALARM a20ms {
    COUNTER= SystemCounter;
    ACTION = ACTIVATETASK {TASK = TaskS; };
    AUTOSTART = TRUE {ALARMTIME = 20;CYCLETIME = 20;APPMODE = stdAppmode; };
  };

  ALARM a1000ms {
    COUNTER= SystemCounter;
    ACTION = ACTIVATETASK {TASK = TaskD; };
    AUTOSTART = TRUE {ALARMTIME = 976;CYCLETIME = 976;APPMODE = stdAppmode; };
  };




  TASK TaskS {
    PRIORITY = 2;
    AUTOSTART = TRUE { APPMODE = stdAppmode;};
    ACTIVATION = 1;
    SCHEDULE = FULL;
    RESOURCE= Sem;
  };


  TASK TaskD {
    PRIORITY = 1;
    AUTOSTART = FALSE;
    ACTIVATION = 1;
    SCHEDULE = FULL;
    RESOURCE= Sem;
  };


};
```

I use two task with periodicity of 20 ms and 1000 ms the task D  run each 1000 ms becouse I need that the Task S sample 50 data.

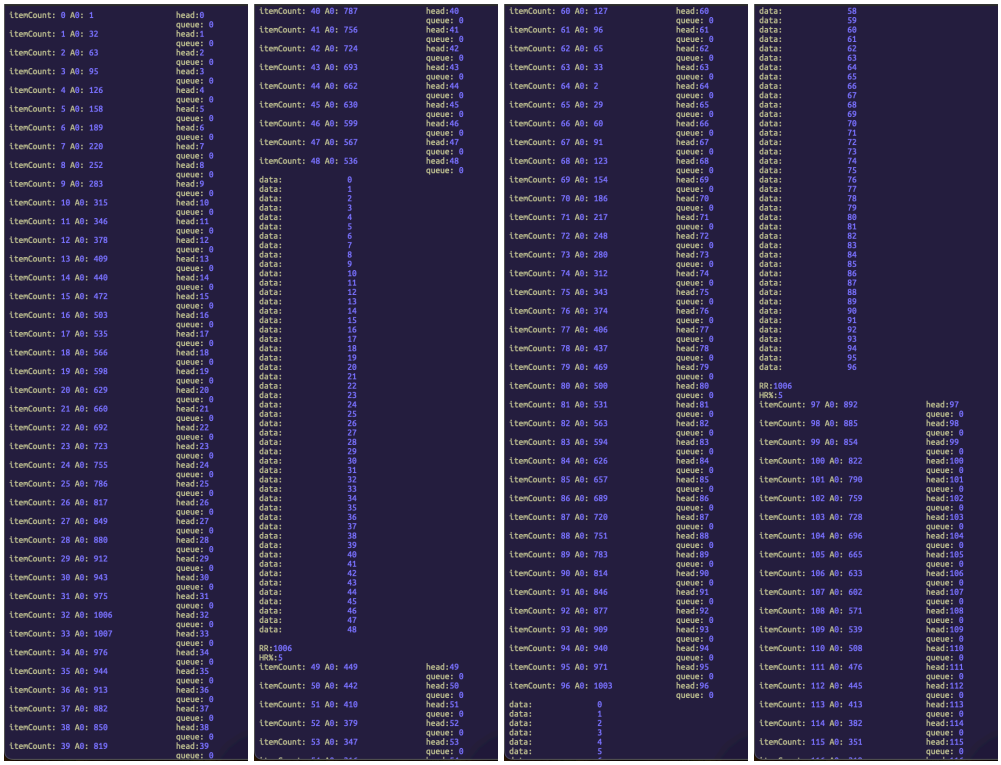
SIMULATION

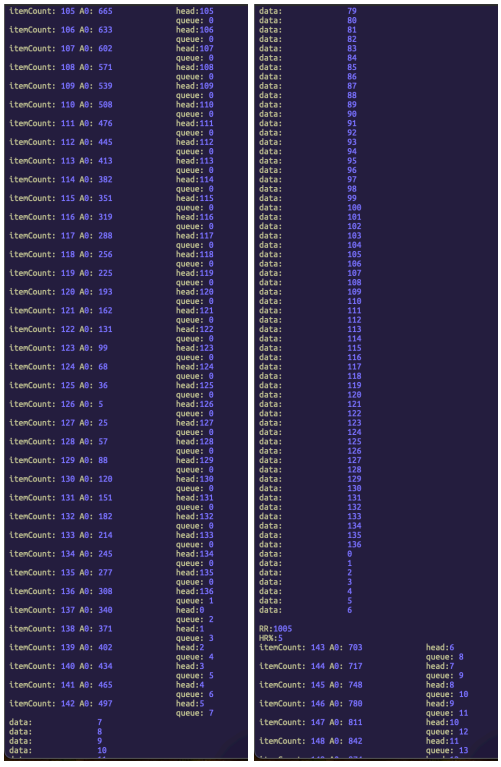
-1HZ



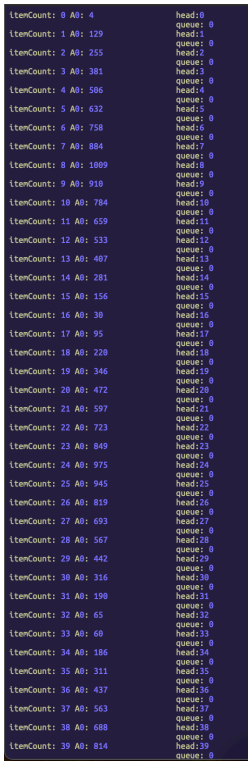


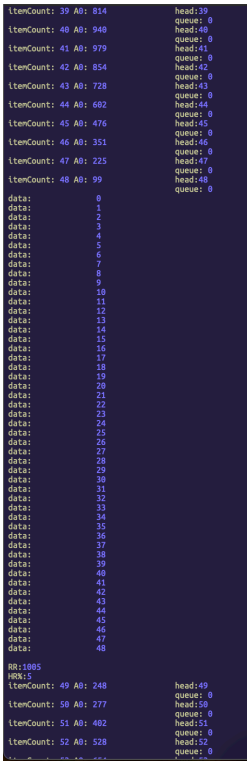
- 0.75 Hz

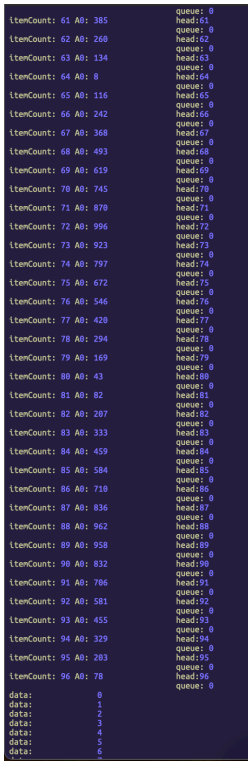


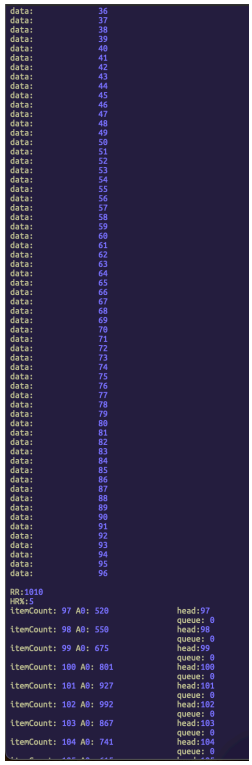


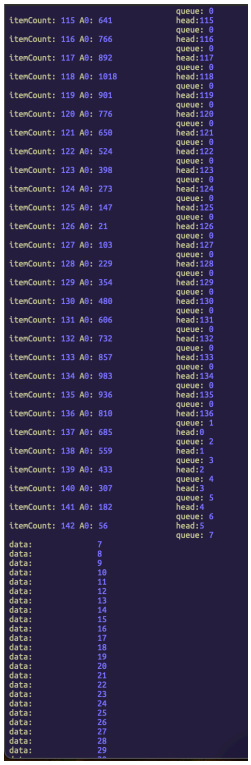
- 3Hz

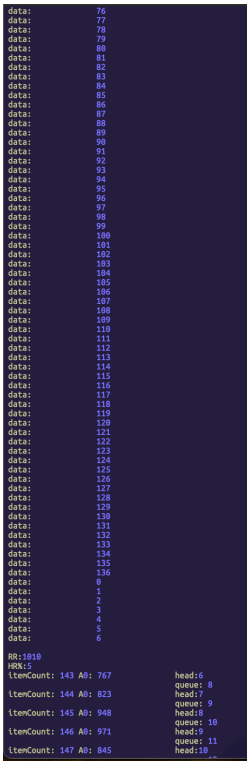

itemCount: 0 A0: 4 head:0 queue: 0
itemCount: 1 A0: 129 head:1 queue: 0
itemCount: 2 A0: 255 head:2 queue: 0
itemCount: 3 A0: 381 head:3 queue: 0
itemCount: 4 A0: 506 head:4 queue: 0
itemCount: 5 A0: 632 head:5 queue: 0
itemCount: 6 A0: 758 head:6 queue: 0
itemCount: 7 A0: 884 head:7 queue: 0
itemCount: 8 A0: 1009 head:8 queue: 0
itemCount: 9 A0: 910 head:9 queue: 0
itemCount: 10 A0: 784 head:10 queue: 0
itemCount: 11 A0: 659 head:11 queue: 0
itemCount: 12 A0: 533 head:12 queue: 0
itemCount: 13 A0: 407 head:13 queue: 0
itemCount: 14 A0: 281 head:14 queue: 0
itemCount: 15 A0: 156 head:15 queue: 0
itemCount: 16 A0: 30 head:16 queue: 0
itemCount: 17 A0: 95 head:17 queue: 0
itemCount: 18 A0: 220 head:18 queue: 0
itemCount: 19 A0: 346 head:19 queue: 0
itemCount: 20 A0: 472 head:20 queue: 0
itemCount: 21 A0: 597 head:21 queue: 0
itemCount: 22 A0: 723 head:22 queue: 0
itemCount: 23 A0: 849 head:23 queue: 0
itemCount: 24 A0: 975 head:24 queue: 0
itemCount: 25 A0: 945 head:25 queue: 0
itemCount: 26 A0: 819 head:26 queue: 0
itemCount: 27 A0: 693 head:27 queue: 0
itemCount: 28 A0: 567 head:28 queue: 0
itemCount: 29 A0: 442 head:29 queue: 0
itemCount: 30 A0: 316 head:30 queue: 0
itemCount: 31 A0: 190 head:31 queue: 0
itemCount: 32 A0: 65 head:32 queue: 0
itemCount: 33 A0: 60 head:33 queue: 0
itemCount: 34 A0: 186 head:34 queue: 0
itemCount: 35 A0: 311 head:35 queue: 0
itemCount: 36 A0: 437 head:36 queue: 0
itemCount: 37 A0: 563 head:37 queue: 0
itemCount: 38 A0: 688 head:38 queue: 0
itemCount: 39 A0: 814 head:39 queue: 0


itemCount: 39 A0: 814 head:39 queue: 0
itemCount: 40 A0: 940 head:40 queue: 0
itemCount: 41 A0: 979 head:41 queue: 0
itemCount: 42 A0: 854 head:42 queue: 0
itemCount: 43 A0: 728 head:43 queue: 0
itemCount: 44 A0: 602 head:44 queue: 0
itemCount: 45 A0: 476 head:45 queue: 0
itemCount: 46 A0: 351 head:46 queue: 0
itemCount: 47 A0: 225 head:47 queue: 0
itemCount: 48 A0: 99 head:48 queue: 0
data: 0
data: 1
data: 2
data: 3
data: 4
data: 5
data: 6
data: 7
data: 8
data: 9
data: 10
data: 11
data: 12
data: 13
data: 14
data: 15
data: 16
data: 17
data: 18
data: 19
data: 20
data: 21
data: 22
data: 23
data: 24
data: 25
data: 26
data: 27
data: 28
data: 29
data: 30
data: 31
data: 32
data: 33
data: 34
data: 35
data: 36
data: 37
data: 38
data: 39
data: 40
data: 41
data: 42
data: 43
data: 44
data: 45
data: 46
data: 47
data: 48
RR:1005
HRN:5
itemCount: 49 A0: 248 head:49 queue: 0
itemCount: 50 A0: 277 head:50 queue: 0
itemCount: 51 A0: 402 head:51 queue: 0
itemCount: 52 A0: 528 head:52 queue: 0


itemCount: 61 A0: 385 queue: 0 head:61 queue: 0
itemCount: 62 A0: 260 head:62 queue: 0
itemCount: 63 A0: 134 head:63 queue: 0
itemCount: 64 A0: 8 head:64 queue: 0
itemCount: 65 A0: 116 head:65 queue: 0
itemCount: 66 A0: 242 head:66 queue: 0
itemCount: 67 A0: 368 head:67 queue: 0
itemCount: 68 A0: 493 head:68 queue: 0
itemCount: 69 A0: 619 head:69 queue: 0
itemCount: 70 A0: 745 head:70 queue: 0
itemCount: 71 A0: 870 head:71 queue: 0
itemCount: 72 A0: 996 head:72 queue: 0
itemCount: 73 A0: 923 head:73 queue: 0
itemCount: 74 A0: 797 head:74 queue: 0
itemCount: 75 A0: 672 head:75 queue: 0
itemCount: 76 A0: 546 head:76 queue: 0
itemCount: 77 A0: 420 head:77 queue: 0
itemCount: 78 A0: 294 head:78 queue: 0
itemCount: 79 A0: 169 head:79 queue: 0
itemCount: 80 A0: 43 head:80 queue: 0
itemCount: 81 A0: 82 head:81 queue: 0
itemCount: 82 A0: 207 head:82 queue: 0
itemCount: 83 A0: 333 head:83 queue: 0
itemCount: 84 A0: 459 head:84 queue: 0
itemCount: 85 A0: 584 head:85 queue: 0
itemCount: 86 A0: 710 head:86 queue: 0
itemCount: 87 A0: 836 head:87 queue: 0
itemCount: 88 A0: 962 head:88 queue: 0
itemCount: 89 A0: 958 head:89 queue: 0
itemCount: 90 A0: 832 head:90 queue: 0
itemCount: 91 A0: 706 head:91 queue: 0
itemCount: 92 A0: 581 head:92 queue: 0
itemCount: 93 A0: 455 head:93 queue: 0
itemCount: 94 A0: 329 head:94 queue: 0
itemCount: 95 A0: 203 head:95 queue: 0
itemCount: 96 A0: 78 head:96 queue: 0
data: 0
data: 1
data: 2
data: 3
data: 4
data: 5
data: 6


data: 36
data: 37
data: 38
data: 39
data: 40
data: 41
data: 42
data: 43
data: 44
data: 45
data: 46
data: 47
data: 48
data: 49
data: 50
data: 51
data: 52
data: 53
data: 54
data: 55
data: 56
data: 57
data: 58
data: 59
data: 60
data: 61
data: 62
data: 63
data: 64
data: 65
data: 66
data: 67
data: 68
data: 69
data: 70
data: 71
data: 72
data: 73
data: 74
data: 75
data: 76
data: 77
data: 78
data: 79
data: 80
data: 81
data: 82
data: 83
data: 84
data: 85
data: 86
data: 87
data: 88
data: 89
data: 90
data: 91
data: 92
data: 93
data: 94
data: 95
data: 96
RR:1010
HRN:5
itemCount: 97 A0: 520 head:97 queue: 0
itemCount: 98 A0: 550 head:98 queue: 0
itemCount: 99 A0: 675 head:99 queue: 0
itemCount: 100 A0: 801 head:100 queue: 0
itemCount: 101 A0: 927 head:101 queue: 0
itemCount: 102 A0: 992 head:102 queue: 0
itemCount: 103 A0: 867 head:103 queue: 0
itemCount: 104 A0: 741 head:104 queue: 0


queue: 0
itemCount: 115 A0: 641 head:115 queue: 0
itemCount: 116 A0: 766 head:116 queue: 0
itemCount: 117 A0: 892 head:117 queue: 0
itemCount: 118 A0: 1018 head:118 queue: 0
itemCount: 119 A0: 901 head:119 queue: 0
itemCount: 120 A0: 776 head:120 queue: 0
itemCount: 121 A0: 650 head:121 queue: 0
itemCount: 122 A0: 524 head:122 queue: 0
itemCount: 123 A0: 398 head:123 queue: 0
itemCount: 124 A0: 273 head:124 queue: 0
itemCount: 125 A0: 147 head:125 queue: 0
itemCount: 126 A0: 21 head:126 queue: 0
itemCount: 127 A0: 103 head:127 queue: 0
itemCount: 128 A0: 229 head:128 queue: 0
itemCount: 129 A0: 354 head:129 queue: 0
itemCount: 130 A0: 480 head:130 queue: 0
itemCount: 131 A0: 606 head:131 queue: 0
itemCount: 132 A0: 732 head:132 queue: 0
itemCount: 133 A0: 857 head:133 queue: 0
itemCount: 134 A0: 983 head:134 queue: 0
itemCount: 135 A0: 936 head:135 queue: 0
itemCount: 136 A0: 810 head:136 queue: 1
itemCount: 137 A0: 685 head:0 queue: 2
itemCount: 138 A0: 559 head:1 queue: 3
itemCount: 139 A0: 433 head:2 queue: 4
itemCount: 140 A0: 307 head:3 queue: 5
itemCount: 141 A0: 182 head:4 queue: 6
itemCount: 142 A0: 56 head:5 queue: 7
data: 7
data: 8
data: 9
data: 10
data: 11
data: 12
data: 13
data: 14
data: 15
data: 16
data: 17
data: 18
data: 19
data: 20
data: 21
data: 22
data: 23
data: 24
data: 25
data: 26
data: 27
data: 28
data: 29


data: 76
data: 77
data: 78
data: 79
data: 80
data: 81
data: 82
data: 83
data: 84
data: 85
data: 86
data: 87
data: 88
data: 89
data: 90
data: 91
data: 92
data: 93
data: 94
data: 95
data: 96
data: 97
data: 98
data: 99
data: 100
data: 101
data: 102
data: 103
data: 104
data: 105
data: 106
data: 107
data: 108
data: 109
data: 110
data: 111
data: 112
data: 113
data: 114
data: 115
data: 116
data: 117
data: 118
data: 119
data: 120
data: 121
data: 122
data: 123
data: 124
data: 125
data: 126
data: 127
data: 128
data: 129
data: 130
data: 131
data: 132
data: 133
data: 134
data: 135
data: 136
data: 0
data: 1
data: 2
data: 3
data: 4
data: 5
data: 6
RR:1010
HRN:5
itemCount: 143 A0: 767 head:6 queue: 8
itemCount: 144 A0: 823 head:7 queue: 9
itemCount: 145 A0: 948 head:8 queue: 10
itemCount: 146 A0: 971 head:9 queue: 11
itemCount: 147 A0: 845 head:10