Victor-2567375 Marco-2554643 Jannis-2555689

Aufgabe 1)

1.2)

```
const int buzzer_pin=37;
const int trig_pin=35;
const int echo_pin=33;
const int button_pin=3;
const unsigned long t_debounce=100;

unsigned long sonar_timeout=90*1000;

unsigned long sample_time_ms=100;
unsigned long measure_time_ms=30000;
unsigned long t=0;

const int array_len=measure_time_ms/sample_time_ms;
unsigned long array_measures[30000/100];
unsigned int i=0;

unsigned int state=0;

int button_pressed=0;

unsigned long triggerAndFetchSonar();
unsigned long deltat();
void cleanArray();
float median(long unsigned int*  a,int size);
void merge_sort(long unsigned int  a[],long unsigned int  b[],int size);
void partial_merge_sort(long unsigned int a[],
    long unsigned int b[],int begin, int end);
void merge(long unsigned int  a[],long unsigned int b[],
    int begin, int  mid,int end);
void copy(long unsigned int  a[],long unsigned int  b[],int begin, int end);


void setup() {
  // put your setup code here, to run once:
  pinMode(button_pin, INPUT);
  pinMode(trig_pin, OUTPUT);
  pinMode(echo_pin, INPUT);
  Serial.begin(9600);
 cleanArray();
  t=millis();
}

void loop() {
```

```
// put your main code here, to run repeatedly:
//Serial.println(state);
switch (state) {
    case 0:
      // do something
      button_pressed=digitalRead(button_pin);
      if(button_pressed==HIGH){
        state=1;
        t=millis();
      }
      break;
    case 1:
      // do something
      button_pressed=digitalRead(button_pin);
      if(button_pressed==LOW && deltat()>=t_debounce){
        state=2;
        t=millis();
        i=0;
      }

      break;
    case 2:
      if(deltat()>=(i+1)*sample_time_ms){
        state=3;
      }
      break;
    case 3:
     array_measures[i++]=triggerAndFetchSonar();

      if(i==array_len){
        state=5;
      }else if(digitalRead(button_pin)==HIGH){
        state=4;
        t=millis();
      }else{
        state=2;
      }
      break;
    case 4:
      button_pressed=digitalRead(buzzer_pin);
      if(button_pressed==LOW && deltat()>=t_debounce){
        state=5;
      }
      break;
    case 5:
      Serial.print("Median: ");
      Serial.println(median(array_measures,i));
      cleanArray();
      state=0;
```

```
        //while(true);
        break;

      default:
        // do something
        state=0;
        Serial.println("RESETING FSM");
  }
}


unsigned long triggerAndFetchSonar () {
  digitalWrite ( trig_pin ,LOW ) ;
  delayMicroseconds (2) ;
  digitalWrite ( trig_pin ,HIGH ) ;
  delayMicroseconds (10) ;
  digitalWrite ( trig_pin ,LOW ) ;
  return pulseIn ( echo_pin ,HIGH , sonar_timeout ) ;
}

unsigned long deltat(){
  return(millis()-t);
}

void cleanArray(){
  for (int j=0;j<array_len;j++){
    array_measures[j]=-1;
  }
}

float median(long unsigned int*  a,int size){
  long unsigned int b[size];
 merge_sort(a,b,size);
  if(size%2!=0){
    return a[size/2];
  }else{
    return (a[size/2]+a[size/2-1])/2.0;
  }
}

void merge_sort(long unsigned int  a[],long unsigned int  b[],int size)
{
 partial_merge_sort(a, b, 0,size);
}
void partial_merge_sort(long unsigned int a[],
    long unsigned int b[],int begin, int end)
{
 if (end  -begin < 2)
 return;
```

```
 // Split and sort
 int mid =begin + (end -begin) / 2;
 partial_merge_sort(a, b,begin, mid);
 partial_merge_sort(a, b, mid,end);

 // Merge and copy
 merge(a, b,begin, mid,end);
 copy(a, b,begin, end);
}

void merge(long unsigned int a[],long unsigned int b[],int begin,
    int mid,int end) {
 int i_begin =begin;
 int i_mid = mid;
 for (int j =begin; j <end; j++){
   if (i_begin < mid && (i_mid >=end || a[i_begin] <= a[i_mid]))
    b[j] = a[i_begin++];
   else
    b[j] = a[i_mid++];
   }
}

void copy(long unsigned int a[],long unsigned int b[],
    int begin, int end) {
 for (int k =begin; k <end; k++)
 a[k] = b[k];
}
```

1.3)

```
/*
Der Durchschnitt des Arrays und der Median können unterschiedlich
sein, da sie unterschiedliche Herangehensweisen zur Bestimmung
von Mittelwerten datellen. Der Median beschreibt den Wert, der nach
den Messungen und sortierten Auflistung an der mittleren Stelle
steht (wobei es praktisch wäre eine ungerade Anzahl an Messungen zu
haben, da dann eine Messung immer an mittlerer Stelle steht).
Beim Durchschnittlichen Wert wird nicht das Ergebnis einer Messung
genommen, sondern das Ergebniss von allen Messungen betrachtet und
durch die Anzahl an Messungen geteilt.

*/

const int buzzer_pin=37;
const int trig_pin=35;
const int echo_pin=33;
const int button_pin=3;
const unsigned long t_debounce=100;

unsigned long sonar_timeout=90*1000;

unsigned long sample_time_ms=100;
unsigned long measure_time_ms=30000;
unsigned long t=0;

const intarray_len=measure_time_ms/sample_time_ms;
unsigned longarray_measures[30000/100];
unsigned int i=0;

unsigned int state=0;

int button_pressed=0;

unsigned longtriggerAndFetchSonar();
unsigned long deltat();
void cleanArray();
float media(long unsigned int* a,int size);
float median(long unsigned int*  a,int size);
void merge_sort(long unsigned int  a[],long unsigned int  b[],int size);
voidpartial_merge_sort(long unsigned int a[],
    long unsigned int b[],int begin, int end);
void merge(long unsigned int  a[],long unsigned int b[],
    int begin, int  mid,int end);
void copy(long unsigned int  a[],long unsigned int  b[],int begin, int end);


void setup() {
```

```
  // put your setup code here, to run once:
  pinMode(button_pin, INPUT);
  pinMode(trig_pin, OUTPUT);
  pinMode(echo_pin, INPUT);
  Serial.begin(9600);
  cleanArray();
  t=millis();
}

void loop() {
  // put your main code here, to run repeatedly:
  //Serial.println(state);
  switch (state) {
      case 0:
        // do something
        button_pressed=digitalRead(button_pin);
        if(button_pressed==HIGH){
          state=1;
          t=millis();
        }
        break;
      case 1:
        // do something
        button_pressed=digitalRead(button_pin);
        if(button_pressed==LOW && deltat()>=t_debounce){
          state=2;
          t=millis();
          i=0;
        }

        break;
      case 2:
        if(deltat()>=(i+1)*sample_time_ms){
          state=3;
        }
        break;
      case 3:
       array_measures[i++]=triggerAndFetchSonar();

        if(i==array_len){
          state=5;
        }else if(digitalRead(button_pin)==HIGH){
          state=4;
          t=millis();
        }else{
          state=2;
        }
        break;
      case 4:
```

```
        button_pressed=digitalRead(buzzer_pin);
        if(button_pressed==LOW && deltat()>=t_debounce){
          state=5;
        }
        break;
      case 5:
        Serial.print("Median: ");
        Serial.println(median(array_measures,i));
        Serial.print("Media: ");
        Serial.println(media(array_measures,i));
        cleanArray();
        state=0;
        //while(true);
        break;

      default:
        // do something
        state=0;
        Serial.println("RESETING FSM");
  }
}


unsigned long triggerAndFetchSonar () {
  digitalWrite ( trig_pin ,LOW ) ;
  delayMicroseconds (2) ;
  digitalWrite ( trig_pin ,HIGH ) ;
  delayMicroseconds (10) ;
  digitalWrite ( trig_pin ,LOW ) ;
  return pulseIn ( echo_pin ,HIGH , sonar_timeout ) ;
}

unsigned long deltat(){
  return(millis()-t);
}

void cleanArray(){
  for (int j=0;j<array_len;j++){
    array_measures[j]=-1;
  }
}

float media(long unsigned int* a,int size){
  float sum=0;
  for(int i=0;i<size;i++){
    sum+=a[i];
  }
  return sum/i;
}
```

```
float median(long unsigned int*  a,int size){
  long unsigned int b[size];
 merge_sort(a,b,size);
  if(size%2!=0){
    return a[size/2];
  }else{
    return (a[size/2]+a[size/2-1])/2.0;
  }
}

void merge_sort(long unsigned int  a[],long unsigned int  b[],int size)
{
 partial_merge_sort(a, b, 0,size);
}
void partial_merge_sort(long unsigned int a[],
    long unsigned int b[],int begin, int end)
{
 if (end  -begin < 2)
 return;

 // Split and sort
 int mid  =begin + (end  -begin) / 2;
 partial_merge_sort(a, b,begin, mid);
 partial_merge_sort(a, b, mid,end);

 // Merge and copy
 merge(a,  b,begin,  mid,end);
 copy(a,  b,begin, end);
}

void merge(long unsigned int  a[],long unsigned int b[],
    int begin, int  mid,int end) {
 int i_begin  =begin;
 int i_mid = mid;
 for (int j  =begin;  j  <end; j++){
   if (i_begin < mid && (i_mid >=end || a[i_begin] <= a[i_mid]))
    b[j] = a[i_begin++];
   else
    b[j] = a[i_mid++];
   }
}

void copy(long unsigned int  a[],long unsigned int  b[],int begin, int end){
 for (int k  =begin;  k  <end; k++)
 a[k] = b[k];
}
```

```
1.4)

const int buzzer_pin=37;
const int trig_pin=35;
const int echo_pin=33;

unsigned longtimeout_min=150,timeout_max=1000;
unsigned long freq_min=1, freq_max=4000;
unsigned long sonar_timeout=1000;

float x=0;

unsigned longtriggerAndFetchSonar();


void setup() {
  // put your setup code here, to run once:
  pinMode(buzzer_pin, OUTPUT);
  pinMode(trig_pin, OUTPUT);
  pinMode(echo_pin, INPUT);
  Serial.begin(9600);

}

void loop() {
  // put your main code here, to run repeatedly:
  unsigned longt=triggerAndFetchSonar();
  if(t<timeout_min){
    t=timeout_min;
  }else if(t>timeout_max){
    t=timeout_max;
  }
  int f=map(t,timeout_min,timeout_max,freq_min,freq_max);
 x=(4.0*x+f)/5.0;

  tone(buzzer_pin,x);
  Serial.println(t);
  Serial.println(f);
  Serial.println(x);
  Serial.println("----");
  delay(100);
}

unsigned long triggerAndFetchSonar () {
  digitalWrite ( trig_pin ,LOW ) ;
  delayMicroseconds (2) ;
  digitalWrite ( trig_pin ,HIGH ) ;
  delayMicroseconds (10) ;
  digitalWrite ( trig_pin ,LOW ) ;
```

```
  return pulseIn ( echo_pin ,HIGH , sonar_timeout ) ;
}
```

Aufgabe 2)

```
void generate_test_data(int test_array[],int size){
 for (int i = 0 ; i <size; i++){
     test_array[i] =random(0,10001);
  }
}

unsigned long test_insertion_sort(int test_array[],int size){
  unsigned long t =millis();
  insertion_sort(test_array,size);
  unsigned long r =millis()-t;
  return r;
}

unsigned long test_merge_sort(int test_array[],int size){
  int buf[size];
  unsigned long t =millis();
  merge_sort(test_array, buf,size);
  unsigned long r =millis()-t;
  return r;
}

void merge_sort(int a[],int b[],int size)
{
 partial_merge_sort(a, b, 0,size);
}
void partial_merge_sort(int a[],int b[],int begin, int end)
{
 if (end -begin < 2)
 return;

 // Split and sort
 int mid =begin + (end -begin) / 2;
 partial_merge_sort(a, b,begin, mid);
 partial_merge_sort(a, b, mid,end);

 // Merge and copy
 merge(a, b,begin, mid,end);
 copy(a, b,begin, end);
}

void merge(int a[],int b[],int begin, int mid,int end) {
 int i_begin =begin;
 int i_mid = mid;
 for (int j =begin; j <end; j++){
   if (i_begin < mid && (i_mid >=end || a[i_begin] <= a[i_mid]))
    b[j] = a[i_begin++];
   else
```

```
    b[j] = a[i_mid++];
   }
}

void copy(int  a[],int  b[],int begin, int end) {
 for (int k  =begin;  k  <end; k++)
 a[k] = b[k];
}

void insertion_sort(int a[],int size){
   for(int i  =  0;  i  <size; i++){
     int j = i;
     if(j != 0){
       while(a[j] < a[j-1]){
        int tmp = a[j];
        a[j]=a[j-1];
        a[j-1]=tmp;
        j = j-1;
        if(j == 0){
         break;
        }
       }
       }
   }
}
void setup(){
  Serial.begin(9600);
  Serial.println("Laufzeitanalyse");
  Serial.println("===============");

  for (int i = 100 ; i < 1000; i += 100){
     int test_data_1[i];
    generate_test_data(test_data_1, i);
     unsigned long time_is = test_insertion_sort(test_data_1, i);

     int test_data_2[i];
    generate_test_data(test_data_2, i);
     unsigned long time_ms = test_merge_sort(test_data_2, i);

     Serial.print("Anzahl Elemente: ");
     Serial.println(i);
     Serial.println("Geschwindigkeit:");
       Serial.print("InsertionsSort: ");
     Serial.print(time_is);
     Serial.println(" ms");
     Serial.print("MergeSort: ");
     Serial.print(time_ms);
     Serial.println(" ms");
   }
```

```
}

void loop(){

}
```

2.5

a)

Insertionsort ist schneller da es direkt über das Array läuft ohne sortieren zu müssen und Mergesort trotzdem die Merge-Operationen ausführen muss.

b)

Merge-Sort ist schneller da die Laufzeit bei Mergesort nur bedingt von dem Sortiertheits-Grad abhängt während Insertionsort bei jedem neuen unsortierten Element über alle vorherigen Elemente iterieren muss.