Arduino Forum

Development => Other Software Development => Topic started by: robtillaart on Nov 21, 2014, 09:38 am

Title: **some sorting algorithms compared**
Post by: **robtillaart** on **Nov 21, 2014, 09:38 am**

While working on some code I needed a ~~basic~~ C++ sorting algorithm for a uint8_t array.
So I implemented five different ones to see the perfomance difference.
Code: [Select]

```
//
//    FILE: sort.ino
//  AUTHOR: Rob Tillaart
// VERSION: 0.1.00
// PURPOSE: compare sorting methods
//    DATE: 2014-11-20
//     URL:
//
// Released to the public domain
//

uint8_t array[250];

uint32_t start;
uint32_t stop;

void setup()
{
  Serial.begin(115200);
  Serial.println("Start Sort\n");


  randomSeed(1);
  for (uint8_t i=0; i< 250; i++) array[i] = random(256);
  start = micros();
  bubbleSort(array, 250);
  stop = micros();
  Serial.print("bubbleSort:\t");
  Serial.println(stop-start);
  //  for (uint8_t i=0; i< 250; i++)
  //  {
  //    Serial.print(array[i]);
  //    Serial.print(" ");
  //    if (i%10==0) Serial.println();
  //  }
  Serial.println();


  randomSeed(1);
  for (uint8_t i=0; i< 250; i++) array[i] = random(256);
  start = micros();
  shellSort(array, 250);
  stop = micros();
  Serial.print("shellSort:\t");
  Serial.println(stop-start);
  //  for (uint8_t i=0; i< 250; i++)
  //  {
  //    Serial.print(array[i]);
  //    Serial.print(" ");
  //    if (i%10==0) Serial.println();
  //  }
  Serial.println();


  randomSeed(1);
  for (uint8_t i=0; i< 250; i++) array[i] = random(256);
  start = micros();
  insertSort(array, 250);
  stop = micros();
  Serial.print("insertSort:\t");
  Serial.println(stop-start);
  //  for (uint8_t i=0; i< 250; i++)
  //  {
  //    Serial.print(array[i]);
  //    Serial.print(" ");
  //    if (i%10==0) Serial.println();
  //  }
  Serial.println();


  randomSeed(1);
  for (uint8_t i=0; i< 250; i++) array[i] = random(256);
  start = micros();
  combSort11(array, 250);
  stop = micros();
  Serial.print("combSort11:\t");
  Serial.println(stop-start);
  //  for (uint8_t i=0; i< 250; i++)
  //  {
  //    Serial.print(array[i]);
  //    Serial.print(" ");
  //    if (i%10==0) Serial.println();
  //  }
  Serial.println();


  randomSeed(1);
  for (uint8_t i=0; i< 250; i++) array[i] = random(256);
  start = micros();
  quickSort(array, 250);
  stop = micros();
  Serial.print("quickSort:\t");
  Serial.println(stop-start);
  //  for (uint8_t i=0; i< 250; i++)
  //  {
  //    Serial.print(array[i]);
  //    Serial.print(" ");
  //    if (i%10==0) Serial.println();
  //  }
  Serial.println();


}

void loop()
{
}

void bubbleSort(uint8_t * ar, uint8_t n)
{
  // bubble sort with flag
  for (uint8_t i=0; i< n-1; i++)
  {
    bool flag = true;
    for (uint8_t j=1; j< n-i; j++)
    {
      if (ar[j-1] > ar[j])
      {
        uint8_t t = ar[j-1];
        ar[j-1] = ar[j];
        ar[j] = t;
        flag = false;
      }
    }
    if (flag) break;
  }
}


void shellSort( uint8_t *ar, uint8_t n)
{
  uint8_t i, temp, flag = 1;
  uint8_t d = n;
  while( flag || (d > 1))      // boolean flag (true when not equal to 0)
  {
    flag = 0;           // reset flag to 0 to check for future swaps
    d = (d+1) / 2;
    for (i = 0; i < (n - d); i++)
    {
      if (ar[i + d] < ar[i])
      {
        temp = ar[i + d];      // swap positions i+d and i
        ar[i + d] = ar[i];
        ar[i] = temp;
        flag = 1;                  // tells swap has occurred
      }
    }
  }
}


void combSort11(uint8_t *ar, uint8_t n)
{
  uint8_t i, j, gap, swapped = 1;
  uint8_t temp;

  gap = n;
  while (gap > 1 || swapped == 1)
  {
    gap = gap * 10 / 13;
    if (gap == 9 || gap == 10) gap = 11;
    if (gap < 1) gap = 1;
    swapped = 0;
    for (i = 0, j = gap; j < n; i++, j++)
    {
      if (ar[i] > ar[j])
      {
        temp = ar[i];
        ar[i] = ar[j];
        ar[j] = temp;
        swapped = 1;
      }
    }
  }
}

void quickSort(uint8_t *ar, uint8_t n)
{
  if (n < 2)
    return;
  uint8_t p = ar[n / 2];
  uint8_t *l = ar;
  uint8_t *r = ar + n - 1;
  while (l <= r) {
    if (*l < p) {
      l++;
    }
    else if (*r > p) {
      r--;
    }
    else {
      int t = *l;
      *l = *r;
      *r = t;
      l++;
      r--;
    }
  }
  quickSort(ar, r - ar + 1);
  quickSort(l, ar + n - l);
}

void insertSort(uint8_t * ar, uint8_t n)
{
  uint8_t t, z, temp;
  for (t = 1; t < n; t++)
  {
    z = t;
    while( (z > 0) && (ar[z] < ar[z - 1] ))
    {
      temp = ar[z];
      ar[z] = ar[z - 1];
      ar[z - 1] = temp;
      z--;
    }
  }
}
```

output

```
Start Sort
bubbleSort:   49992
```

```
shellSort:    36376
insertSort:   21928
combSort11:   4728
quickSort:    3260
```

Quicksort is a clear winner, but combSort11 is a good second place as it doesn't use recursion it probably uses less RAM. I did not measure that yet as my goal was primary performance.

Title: **Re: some sorting algorithms compared**
Post by: **robtillaart** on **Nov 21, 2014, 11:59 am**

Quick RAM investigation - UNO, IDE 1.5.4

Reverse sorted array of 250 values (to enforce worst case behavior quickSort)

```
QUICKSORT

memory before:  1483
lowest memory:  1387
--------------------------
Max RAM used:    96 bytes


COMBSORT11
memory before:  1483
lowest memory:  1462
--------------------------
Max RAM used:    21 bytes
```

So quicksort does not use much memory in absolute sense, but relative to CombSort11 it is about 4.6 times as much.

note: redo timing test with array size above 255.

Title: **Re: some sorting algorithms compared**
Post by: **robtillaart** on **Nov 21, 2014, 01:12 pm**

---

redid the test with array of 1500 elements (+ bit optimized combsort11 + added bubblesort)

Code: [Select]

```
//
//    FILE: sort.ino
//  AUTHOR: Rob Tillaart
// VERSION: 0.1.01
// PURPOSE: compare some sorting methods
//    DATE: 2014-11-20
//     URL:
//
// Released to the public domain
//

// #include <FreeRam.h>

uint8_t array[1500];

uint32_t start;
```

```
uint32_t stop;

uint32_t fr = 2222;


void testSort( void(*sf)(uint8_t *, uint16_t), uint16_t sz, char *name)
{
  randomSeed(1);
  for (uint16_t i=0; i<sz; i++) array[i] = random(256);

  start = micros();
  sf(array, sz);
  stop = micros();

  Serial.print(name);
  Serial.print("\t");
  Serial.println(stop-start);

  //  for (uint16_t i=0; i< sz; i++)
  //  {
  //    Serial.print(array[i]);
  //    Serial.print(" ");
  //    if (i%10==0) Serial.println();
  //  }

  Serial.println();
}


void setup()
{
  Serial.begin(115200);
  Serial.println("Start Sort\n");

  testSort(bubbleSort, 1500, "bubbleSort");
  testSort(bubbleFlagSort, 1500, "bubbleFlagSort");
  testSort(shellSort, 1500, "shellSort");
  testSort(insertSort, 1500, "insertSort");
  testSort(combSort11, 1500, "combSort11");
  testSort(quickSort, 1500, "quickSort");

  //  Serial.println("QSORT RAM:\t");
  //  Serial.println(fr);
  //  Serial.println("RAM:\t");
  //  Serial.println(freeRam());
}

void loop()
{
}


void bubbleFlagSort(uint8_t * ar, uint16_t n)
{
  // bubble sort with flag
  for (uint16_t i=0; i< n-1; i++)
  {
    bool flag = true;
    for (uint16_t j=1; j< n-i; j++)
    {
      if (ar[j-1] > ar[j])
      {
        uint8_t t = ar[j-1];
        ar[j-1] = ar[j];
        ar[j] = t;
        flag = false;
      }
    }
    if (flag) break;
  }
}


void bubbleSort(uint8_t * ar, uint16_t n)
{
  // bubble sort with flag
  for (uint16_t i=0; i< n-1; i++)
  {
    for (uint16_t j=1; j< n-i; j++)
    {
      if (ar[j-1] > ar[j])
      {
        uint8_t t = ar[j-1];
        ar[j-1] = ar[j];
        ar[j] = t;
      }
    }
  }
}


void shellSort( uint8_t *ar, uint16_t n)
{
  uint16_t i, temp;
  uint8_t flag = 1;
  uint16_t d = n;
  while( flag || (d > 1))      // boolean flag (true when not equal to 0)
  {
    flag = 0;           // reset flag to 0 to check for future swaps
    d = (d+1) / 2;
    for (i = 0; i < (n - d); i++)
    {
      if (ar[i + d] < ar[i])
      {
        temp = ar[i + d];      // swap positions i+d and i
        ar[i + d] = ar[i];
        ar[i] = temp;
        flag = 1;                  // tells swap has occurred
      }
    }
  }
}


void combSort11(uint8_t *ar, uint16_t n)
{
  uint16_t i, j;
  uint16_t gap;
  uint8_t swapped = 1;
  uint8_t temp;

  gap = n;
  while (gap > 1 || swapped == 1)
  {
    if (gap > 1)
    {
      gap = gap * 10/13;
      if (gap == 9 || gap == 10) gap = 11;
    }
    swapped = 0;
    for (i = 0, j = gap; j < n; i++, j++)
    {
      if (ar[i] > ar[j])
      {
        temp = ar[i];
        ar[i] = ar[j];
        ar[j] = temp;
        swapped = 1;
      }
    }
  }
}


void quickSort(uint8_t *ar, uint16_t n)
{
  if (n < 2)
  {
    // uint32_t s = freeRam();
    // if (fr > s) fr = s;
    return;
  }
  uint8_t p = ar[n / 2];
  uint8_t *l = ar;
  uint8_t *r = ar + n - 1;
  while (l <= r) {
    if (*l < p) {
      l++;
    }
    else if (*r > p) {
      r--;
    }
    else {
      int t = *l;
      *l = *r;
      *r = t;
      l++;
      r--;
    }
  }
  quickSort(ar, r - ar + 1);
  quickSort(l, ar + n - l);
}


void insertSort(uint8_t * ar, uint16_t n)
{
  uint16_t t, z, temp;
  for (t = 1; t < n; t++)
  {
    z = t;
    while( (z > 0) && (ar[z] < ar[z - 1] ))
    {
      temp = ar[z];
      ar[z] = ar[z - 1];
      ar[z - 1] = temp;
      z--;
    }
  }
}
```

output

```
Start Sort
bubbleSort:     1167696
bubbleFlagSort: 1203072
shellSort:      1228124
insertSort:     636660
combSort11:     36504
quickSort:      22800
```

Difference between quickSort and combSort11 grows in absolute sense but relative it is still only 1.6x slower. The others are way behind 30-60 times slower

RAM QUICKSORT

memory before: 225
lowest memory: 21

Max RAM used: 204 bytes   (that's 10% of UNO's RAM)

(OK enough played today ;)

Title: **Re: some sorting algorithms compared**
Post by: **pYro_65** on **Nov 22, 2014, 07:02 am**

Using mega + older Arduino IDE compiler:
Quote

```
bubbleSort   1167760
bubbleFlagSort   1203144
shellSort   1228200
insertSort   636692
combSort11   36512
quickSort   22876
```

Using a mega on the new compiler:
Quote

```
bubbleSort   1202792
bubbleFlagSort   1238100
```

```
shellSort   1435748
insertSort   672680
combSort11   42624
quickSort   22492
```

Strange huh? All got slower except quickSort.

Title: **Re: some sorting algorithms compared**
Post by: **robtillaart** on **Nov 22, 2014, 11:00 am**

Good and important observation, ~15% slower

how about the code size generated by the two compilers?

Title: **Re: some sorting algorithms compared**
Post by: **hans_oberlander** on **Jun 18, 2015, 09:21 pm**

Interesting experiment! I've read the whole thread and I'm trying to replicate your experiment (for 16 bit numbers), but having trouble getting the code to run.
I tried looking at your code but there's no comments anywhere and the way testSort interprets its arguments goes way over my head. Could anyone help me out? I'm getting the error:

error: cannot convert 'int (*)[4]' to 'int*' for argument '1' to 'void quickSort(int*, int)'

Which I guess means that the 'int' that quickSort works on cannot point to the array that I fed it, because it's not compatible? Don't really understand this.

Minimum code to reproduce this is here:
Code: [Select]

```
int myArray[] = {4,8,2,11};
int sortedArray[4];

void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:
    sortedArray = quickSort(&myArray, 4);

}


void quickSort(int *ar, int n)
//the local 'ar' variable, through the dereference operator * now
//points to the array that you fed it with the dereference operator &
{
  if (n < 2) //array too small
    return;
  int p = ar[n / 2];
  int *l = ar;
  int *r = ar + n - 1;
  while (l <= r) {
    if (*l < p) {
      l++;
    }
    else if (*r > p) {
      r--;
    }
    else {
      int t = *l;
      *l = *r;
      *r = t;
      l++;
      r--;
    }
  }
  quickSort(ar, r - ar + 1);
  quickSort(l, ar + n - l);
}
```

Title: **Re: some sorting algorithms compared**
Post by: **HazardsMind** on **Jun 18, 2015, 09:55 pm**

Quicksort doesn't return anything and no need to reference the array.

Title: **Re: some sorting algorithms compared**
Post by: **robtillaart** on **Jun 18, 2015, 11:44 pm**

error: cannot convert 'int (*)[4]' to 'int*' for argument '1' to 'void quickSort(int*, int)'

The name of the array is already the address so you do not need the & before myArray.

try this
Code: [Select]

```
int myArray[] = {4, 8, 2, 11};
int sortedArray[4];

void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);

}

void loop() {
  // put your main code here, to run repeatedly:
  quickSort(myArray, 4);

  for (int i = 0; i < 4; i++)
    Serial.println(myArray[i]);
  delay(2000);
}


void quickSort(int *ar, int n)
//the local 'ar' variable, through the dereference operator * now
//points to the array that you fed it with the dereference operator &
{
  if (n < 2) //array too small
    return;
  int p = ar[n / 2];
  int *l = ar;
  int *r = ar + n - 1;
  while (l <= r) {
    if (*l < p) {
      l++;
    }
    else if (*r > p) {
      r--;
    }
    else {
      int t = *l;
      *l = *r;
      *r = t;
      l++;
      r--;
    }
  }
  quickSort(ar, r - ar + 1);
  quickSort(l, ar + n - l);
}
```