

Re: "Sorting" assignment

Source: <http://coding.derkeiler.com/Archive/General/comp.programming/2008-02/msg00141.html>

- *From:* spinoza1111 <spinoza1111@xxxxxxxx>
 - *Date:* Wed, 6 Feb 2008 02:04:37 -0800 (PST)
-

On Feb 6, 5:15 pm, "Clive D. W. Feather" <cl...@on-the-train.demon.co.uk> wrote:

In article

<99cf437b-e8d3-4d3c-92cc-909fb9d85...@xx>, spinoza1111<spinoza1...@x> writes

Bubble sorting is how not to sort.

Here is a C example of a simple program that sorts a list of random numbers using the method of exchanging between two bins (partition exchange) that I mentioned in my first post.

An algorithm which most people know as "quicksort", of course.

And many others prefer to call partition exchange because "quicksort" sounds too commercial.

As I said, this algorithm grabs an arbitrary member and then compares it to every other member, throwing each compared member into bin A or bin B depending on whether it is greater than, or less than (or equal to) the arbitrary member (called the pivot).

Actually, it doesn't. It grabs the member at the left end of the array to be sorted. If the array is already sorted, this means that you end up with an N^2 algorithm instead of an $N \log N$ one. About the only worse choice you could have made is to use the one at the right end, since the way you've coded it means that you'd also swap every single member of the array with itself.

Re: "Sorting" assignment

It is more usual to pick the middle of the array as the pivot, or even to pick an element at random.

News flash. The array is unsorted. This means that choosing the left end IS picking an element at random.

It is usual to point out that quicksort has this pathological behaviour for certain inputs.

Thanks for saving me the trouble. It do.

But when A and B are one cell, this means that $A \leq \text{pivot} < B$ for all members because the two recursive calls you see in the code cover all the values.

This makes no sense to me. If A or B are one cell, your code doesn't attempt to sort them (since there's nothing to do).

It makes no sense to you because you don't understand it. Yet. You will. I am in the text giving a little peroration to show that that the recursion gets down to one.

But that means that the entire array is sorted in rather less time than a bubble sort,

Actually, if the array is sorted before you start, your code will be rather slower than a bubble sort. If it's in reverse order, it will be much slower.

True. This can however be checked.

which maximizes the number of times an entry must travel to get to its final position.

Re: "Sorting" assignment

In some circumstances. There are others – as we'll see – where bubble sort is actually a sensible thing to use.

For any arbitrary entry, the probability of its having to move reduces each time the quicksort is recursively called.

If the array is initially random, the probability that an entry has to move is almost exactly 0.5 *at every stage*. This is because each entry has a 50% chance of being greater than the pivot each time.

This is what intelligent workers might do on a farm when sorting apples.

You are a patronising git at times.

And so are you. Wiglaf was a Geat, wasn't he?

I remain by my contention that algorithms aren't invented by mathematicians.

They wouldn't use bubble sort. I think a programmer invented that time sink.

Here is the code. Examine it in a monospace font such as Courier New.

```
// ***** Quicksort (partition exchange) example *****
```

```
#include <stdio.h>  
#include <stdlib.h>
```

Re: "Sorting" assignment

```
const int SIZE = 27;
```

```
int quicksortPartition  
(int intArray[],  
 int intLeft,  
 int intRight,  
 int intPivotIndex)
```

I find this style difficult to read. The type of a variable is often the least important thing about it. I know you love Hungarian notation, but many of us find it merely hides the essentials in a morass of unimportant detail. For example, "intArray" tells me *NOTHING* about the parameter that I can't already tell from its declaration (no matter what it is called, it's an int and it's an array). I feel that it would be far better to name that parameter on the basis of what it *does*, such as "values" or even (though I wouldn't do it myself) "to_be_sorted".

"The type of a variable is often the least important thing about it"?
I should say that the type of a variable is the MOST important thing about it.

Given your advocacy of object-oriented programming in the past, I am extremely surprised that you force the data to be integers. It would be far more sensible to abstract the type away. That is, at the top of the program have "typedef int datatype" and declare the first parameter as "datatype values []".

That would be the version I am working on for my own further use. This uses C Sharp. I don't think typedef is OO programming.

Furthermore, the correct type to use for an index into an array is size_t rather than int. Of course, since you blithely subtract 1 from the index, you'll need to trap that special case. But you won't mind doing that – after all, you were complaining the other day that (I think) Brian Kernigan didn't do that but relied on his data being zero-terminated.

What is the justification for making the index of the pivot a parameter to this function? If you were choosing the pivot via separate code it

Re: "Sorting" assignment

might be justified, but you don't – you pivot on the left hand value. It would make the code far clearer if the choice of pivot was done within the partition function. It would also make it easier for the student to experiment with different choices, or for you to do mid–array or random selection.

```
{  
    int intPivotValue = intArray[intPivotIndex];
```

If this is meant to be educational code, then a comment of the form:

```
/* Move the pivot value out of the way at the right hand end */
```

would have been helpful. Comments further along would also be helpful; for example, in the main loop you could point out how `intStoreIndex` and `intIndex1` partition the array into bin A, bin B, and unchecked data.

```
intArray[intPivotIndex] = intArray[intRight];  
intArray[intRight] = intPivotValue;
```

I don't criticise your use of arrays and indices rather than pointers, though many people would. In practice it's a matter of what you feel comfortable with.

```
int intStoreIndex = intLeft;
```

You can't do that! Declaring variables after the first line of executable code is an invention of the evil corporation–controlled standards bodies and is not part of True C before the military–industrial complex perverted it.

Git. Wiglaf was a Geat, wasn't he?

To be precise, *I* added it to the 1999 C Standard. That is, I did all the donkey work of writing the wording to allow it to be included, including working out some nasty boundary cases (such as the interaction with variable length arrays).

Nice going (seriously).

Re: "Sorting" assignment

```
int intIndex1;  
int intExchange;  
for (intIndex1 = intLeft;  
    intIndex1 < intRight;  
    intIndex1++)
```

If you're going to surrender and use C99 features, then why not:

```
for (int intIndex1 = intLeft; intIndex1 < intRight; intIndex1++)
```

I thought about doing this but was uncertain about the scope of intIndex1.

? And just what information does the "1" provide? In fact, how does it hurt to just call this "idx"? Or even, given the small size of the body, "i"?

Because some of us like to be able to read code aloud to the dog. Seriously, I think Richard Harter's style, used on a memory allocation package posted here a while back, was elegant. It used algebraically short identifiers but conscientiously commented every line.

I prefer code that looks like English because this reassures me that someone was programming on purpose. The comments in Harter's code performed this function. In my own code, I would like it to be possible for the code to be spoken or sung in mead-hall like Wiglaf the Geat.

Seriously, every try talking about code over a long distance hookup with a chap from India? If you can say the code and it uses simple international words, the process tends to go smoother.

The "1" means it is the first index and more may be needed.

```
{  
    if (intArray[intIndex1] <= intPivotValue)  
    {  
        intExchange = intArray[intIndex1];
```

Re: "Sorting" assignment

Why don't you declare `intExchange` at this scope? Doing so reduces the chance of it being misused by accident later.

I don't declare anything inside of a loop is why.

```
intArray[intIndex1] = intArray[intStoreIndex];  
intArray[intStoreIndex] = intExchange;
```

If your data type is anything bigger than a machine word, it is worth bracketing this swap with `"if (intIndex1 != intStoreIndex)"`.

Agreed.

```
intStoreIndex += 1;
```

Suddenly scared of the `++` operator?

After learning about sync points, I'm terrified.

```
    }  
}
```

This partitioning algorithm will work and suffices for teaching. However, in practice it's far too inefficient. For example, if the left element of the array is the second largest (e.g. array contents 9 1 2 3 4 5 6 7 8 10), then it does a bubble pass on the array (excluding the last element), moving every single element down by one and shifting the 10 from left to right one cell at a time.

The standard way to implement the partition phase is:

- * Swap the pivot item out of the way.
- * Initialize pointer `pL` to the left element and `pR` to the right element.
- * While `pL` is left of `pR`:
 - * Move `pL` right until it finds an element greater than the pivot.
 - * Move `pR` left until it finds an element less than the pivot.

Re: "Sorting" assignment

* Swap the elements at pL and pR.

This does require careful coding, especially to deal with the boundary cases.

Thanks for your comments. They are illuminating. My copy of Sorting and Searching was left in the United States when I left for Asia, and Knuth conscientiously provides worst case information. Sounds like I need replacement volumes of Knuth and shouldn't have trusted wikipedia to refresh my memory of his discussion.

I agree that quicksort sucks when the array is nearly in order. However, I think it's more useful to start with quicksort than with bubble sort. My first sort program was written in 1401 machine language, and my test data was a list of naughty words which I handed in neatly printed out.

```
intPivotValue = intArray[intStoreIndex];
intArray[intStoreIndex] = intArray[intRight];
intArray[intRight] = intPivotValue;
return intStoreIndex;
}
```

```
void quicksort(int intArray[],
               int intLeft,
               int intRight)
{
    if (intRight > intLeft)
    {
        int intPivotNewIndex = quicksortPartition
            (intArray,
             intLeft,
             intRight,
             intLeft);
        quicksort(intArray,
                  intLeft,
                  intPivotNewIndex - 1);
        quicksort(intArray,
                  intPivotNewIndex + 1,
                  intRight);
    }
}
```

Re: "Sorting" assignment

Again, if this is meant to be teaching code, it suffices. If it's meant to be production code, it doesn't.

Firstly, you should determine which of the two partitions is the smaller. That should be sorted using a recursive call, while the larger one should be sorted in a loop.

```
void quicksort(int intArray[],
               int intLeft,
               int intRight)
{
    while (intRight > intLeft)
    {
        int intPivotNewIndex = quicksortPartition
                               (intArray,
                                intLeft,
                                intRight,
                                intLeft);
        int leftwidth = intPivotNewIndex - 1 - intLeft;
        int rightwidth = intRight - (intPivotNewIndex + 1);

        if (leftwidth < rightwidth)
        {
            quicksort(intArray,
                      intLeft,
                      intPivotNewIndex - 1);
            intLeft = intPivotNewIndex + 1;
        }
        else
        {
            quicksort(intArray,
                      intPivotNewIndex + 1,
                      intRight);
            intRight = intPivotNewIndex - 1;
        }
    }
}
```

Doing this guarantees that, no matter how bad things get, the depth of recursion is never more than $\log N$.

Secondly, when the bins get small the effort of partitioning and recursion overwhelm the benefits of the technique. So it is common to stop when the bin size gets down to around 5 (the exact size needs to be found by experimentation on the platform in use). That is, the outer test in the above code becomes:

```
    if (intRight > intLeft + 5)
or
```

Re: "Sorting" assignment

```
while (intRight > intLeft + 5)
```

Then, after the quicksort completely finishes, do a *BUBBLE SORT* on the

No, dammit. Don't do a BUBBLE SORT. Do an interchange sort:

```
for(i = 0; i < arraySize - 1; i++)  
for(j = i + 1; j < arraySize; j++) if (array[i] > array[j])  
exchange(i, j);
```

entire array. Since every element is within 5 places of where it belongs, this will never need more than 5 ...

..25 iterations...

read more »– Hide quoted text –

– Show quoted text — Hide quoted text –

– Show quoted text –

.