

Forum:[Hack Forums](#)

Thread: [A Challenge for the bored C/C++'er](#)

saltycracker - May 7, 2020, 9:14 am

Here's a simple sounding challenge. Can you sort the array in the following code using recursion?

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

#define ARR_SIZE 10

int main (int argc, char ** argv) {
    int arr[ARR_SIZE] = {2, 9, 5, 1, 8, 10, 3, 6, 4, 7};
    size_t i = 0;

    for (;i < ARR_SIZE; i++) {
        fprintf(stdout, "%d\n", arr[i]);
    }
    return 0;
}
Have fun.
```

Tafelpoot - May 7, 2020, 9:35 am

What exactly do you mean by recursion?

Merge sort can be easily used, but idk if that adheres to your requirement.
Please elaborate.

saltycracker - May 7, 2020, 10:22 pm

(May 7, 2020 09:35 AM)Tafelpoot Wrote: [\[CITE\]](#) What exactly do you mean by recursion?

Merge sort can be easily used, but idk if that adheres to your requirement.
Please elaborate.
Don't use looping constructs.

4Txe4y33C9zvNhMj - May 8, 2020, 1:36 am

Probably not the most efficient way of doing it.

```
Code #include <stdio.h>
#include <stdlib.h>
#include <iostream>
#include <vector>

#define ARR_SIZE 10

int* sort(int* p, size_t size, int i = 0, int j = 0) {
    if (i >= size) { return p; }

    if (j >= size - i - 1) {
        j = 0;
        i++;
    }
}
```

```

    }
    else {
        if (p[j] > p[j + 1]) {
            int temp = p[j];
            p[j] = p[j + 1];
            p[j + 1] = temp;
        }
    }
}

p = sort(p, i, j++, size);

return p;
}

int main() {
    int arr[ARR_SIZE] = { 2, 9, 5, 1, 8, 10, 3, 6, 4, 7 };

    for (size_t i = 0; i < ARR_SIZE; i++) { fprintf(stdout, "%d\n", arr[i]); }

    int* sorted = sort(arr, ARR_SIZE);
    for (size_t i = 0; i < ARR_SIZE; i++) { fprintf(stdout, "%d\n", sorted[i]); }

    std::cin.get();
    return 0;
}

```

micahmonday - May 8, 2020, 4:09 am

bubble sort:

Code #include <stdio.h>

#include <stdlib.h>

#define ARR_SIZE 10

```

void swap(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

```

```

void sort(int *arr, int sz, int iteration) {
    int n = iteration % (sz);

    if (arr[n] > arr[n+1])
        swap(arr + n, arr + n + 1);

    if (++iteration < sz*sz)
        sort(arr, sz, iteration);
}

```

```

int main (int argc, char ** argv) {
    int arr[ARR_SIZE] = {2, 9, 5, 1, 8, 10, 3, 6, 4, 7};
    size_t i = 0;

```

```

sort(arr, ARR_SIZE - 1, 0);

for (;i < ARR_SIZE; i++) {
    printf("%d\n", arr[i]);
}
return 0;
}

```

saltycracker - May 8, 2020, 4:53 am

(May 8, 2020 01:36 AM)4Txe4y33C9zvNhMj Wrote: [\[CITE\]](#) Probably not the most efficient way of doing it.

If you are reaching for recursion in C or C++ then I can guarantee that its not the most efficient solution. The standard doesn't even include tail call optimization so you can't guarantee anything without inspecting the code your implementation produced.

Tafelpoot - May 8, 2020, 1:46 pm

I have chosen to do a lesser known sorting alorithm called interpolation sort.

It was quite a challenge since there is little information available, most available sources talk about the interpolation search instead of sort.

```

Code #include <iostream>
#include <math.h>
#include <vector>
int call_counter = 0;
void interpolation_sort(std::vector<int>& unsorted_number) {
    ++call_counter;
    // 0 and 1 element require no sort
    if (unsorted_number.size() < 2) return;

    // no bucket size stack needed because recursion
    int cur_size = unsorted_number.size();
    std::vector<std::vector<int>> bucket(cur_size);

    int start = 0;
    int min = unsorted_number[start];
    int max = min;

    // start + 1, min and max are already [index=0] so no need to check
    for (int index = start + 1; index < cur_size; ++index)
    {
        if (unsorted_number[index] < min) min = unsorted_number[index];
        else if (unsorted_number[index] > max) max = unsorted_number[index];
    }

    // e.g. 10, 10, 10, 10 or 3, 3
    if (min == max) return;

    // Looping with pointer prob. faster
    for (int index = start; index < cur_size; ++index)
    {
        // Cast to double to prevent int div, ignore C26541 because wrapping can't happen
        int bucket_index_interpolation =

```

```

        (double)(unsorted_number[index] - min) / (max - min) * (cur_size - 1);
        bucket[bucket_index_interpolation].push_back(unsorted_number[index]);
    }

    for (int bucket_index = 0; bucket_index < cur_size; ++bucket_index)
    {
        if (bucket[bucket_index].empty()) continue;

        interpolation_sort(bucket[bucket_index]);

        for (int bucket_value = 0; bucket_value < bucket[bucket_index].size(); ++bucket_value)
        {
            unsorted_number[start++] = bucket[bucket_index][bucket_value];
        }
        //bucket[bucket_index].clear();
    }
}

int main() {
    std::vector<int> unsorted_numbers =
        { 2, 9, 5, 10, 10, 10, 10, 10, 10, 10, 10, 99, 58, 17, 86, 105, 34, 63, 42, 71 };

    interpolation_sort(unsorted_numbers);

    std::cout << "amount of calls: " << call_counter << std::endl;
    for (int i = 0; i < unsorted_numbers.size(); ++i)
        std::cout << unsorted_numbers[i] << ' ';
}

```

While doing this sort I read up on bucket sort and thought of a (new) way to do bucket sort. Can't seem to find a similar solution to what I thought of, I will try to implement it tomorrow and post the solution here.

Gary Oak - May 8, 2020, 4:11 pm

(May 8, 2020 01:46 PM)Tafelpoot Wrote: [\[CITE\]](#) I have chosen to do a lesser known sorting algorithm called interpolation sort.

It was quite a challenge since there is little information available, most available sources talk about the interpolation search instead of sort.

```

Code #include <iostream>
#include <math.h>
#include <vector>
int call_counter = 0;
void interpolation_sort(std::vector<int>& unsorted_number) {
    ++call_counter;
    // 0 and 1 element require no sort
    if (unsorted_number.size() < 2) return;

    // no bucket size stack needed because recursion
    int cur_size = unsorted_number.size();
    std::vector<std::vector<int>> bucket(cur_size);

    int start = 0;

```

```

int min = unsorted_number[start];
int max = min;

// start + 1, min and max are already [index=0] so no need to check
for (int index = start + 1; index < cur_size; ++index)
{
    if (unsorted_number[index] < min) min = unsorted_number[index];
    else if (unsorted_number[index] > max) max = unsorted_number[index];
}

// e.g. 10, 10, 10, 10 or 3, 3
if (min == max) return;

// Looping with pointer prob. faster
for (int index = start; index < cur_size; ++index)
{
    // Cast to double to prevent int div, ignore C26541 because wrapping can't happen
    int bucket_index_interpolation =
        (double)(unsorted_number[index] - min) / (max - min) * (cur_size - 1);
    bucket[bucket_index_interpolation].push_back(unsorted_number[index]);
}

for (int bucket_index = 0; bucket_index < cur_size; ++bucket_index)
{
    if (bucket[bucket_index].empty()) continue;

    interpolation_sort(bucket[bucket_index]);

    for (int bucket_value = 0; bucket_value < bucket[bucket_index].size(); ++bucket_value)
    {
        unsorted_number[start++] = bucket[bucket_index][bucket_value];
    }
    //bucket[bucket_index].clear();
}

int main() {
    std::vector<int> unsorted_numbers = { 2, 9, 5, 10, 10, 10, 10, 10, 10, 10, 10, 99, 58, 17, 86, 105,
    34, 63, 42, 71 };

    interpolation_sort(unsorted_numbers);

    std::cout << "amount of calls: " << call_counter << std::endl;
    for (int i = 0; i < unsorted_numbers.size(); ++i)
        std::cout << unsorted_numbers[i] << ' ';
}

```

While doing this sort I read up on bucket sort and thought of a (new) way to do bucket sort. Can't seem to find a similar solution to what I thought of, I will try to implement it tomorrow and post the solution here.

RIA is indeed useable here, tbh my approach is the most common and I would use the quicksort algorithm to do this... But I am still looking forward to seeing your next sorting method.

Tafelpoot - May 8, 2020, 4:26 pm

What does RIA stand for? Yeah quicksort is great, but I think any sequential will do even better because of the small array in this case.

Alright finished the second sort. I thought of using bitwise checking in combination with bucket sort. I however couldn't figure out how to get it to work. So just did a simpler one.

It starts with the most significant bit and checks for each number if it is on or off.

If it is on it goes into the 1 bucket else into the 0.

Then the function gets called again but now MSB-1 is checked.

It is very slow, takes about 5 seconds to sort 100.000 elements; where interpolation search takes ~0.68 seconds.

The function is used 60.000 times on average in contrast to interpolation which uses the function 30.000 times

```
Code int bit_call_counter = 0;
void bit_sort(std::vector<UINT64>& unsorted_number, int mask_index, std::vector<uint_fast64_t>&
bitmask) {
    ++bit_call_counter;
    if (unsorted_number.size() < 2) return;
    if (mask_index < 0) return;
    int cur_size = unsorted_number.size();
    std::vector<std::vector<UINT64>> bucket(2);

    for (size_t number_index = 0; number_index < cur_size; ++number_index)
    {
        if (bitmask[mask_index] & unsorted_number[number_index])
        {
            bucket[1].push_back(unsorted_number[number_index]);
        }
        else {
            bucket[0].push_back(unsorted_number[number_index]);
        }
    }

    int start = 0;
    for (int bucket_index = 1; bucket_index >= 0; --bucket_index)
    {
        if (bucket[bucket_index].empty()) continue;

        bit_sort(bucket[bucket_index], mask_index - 1, bitmask);

        for (UINT64 bucket_value = 0; bucket_value < bucket[bucket_index].size(); ++bucket_value)
        {
            unsorted_number[start++] = bucket[bucket_index][bucket_value];
        }
        bucket[bucket_index].clear();
    }
}
```

Might try to speed it up next week, i have a feeling bitwise operations must be very fast.

k1r3n - May 19, 2020, 5:56 pm

I did not read the comments, and the author did not specify which algorithm to use, so here:

```
Code void bubble_sort_recursive(int arr[], int index) {
    if (index == 1) return;
```

```

for (int i = 0; i < index - 1; i++)
    if (arr[i] > arr[i + 1])
        std::swap(arr[i], arr[i + 1]);

bubble_sort_recursive(arr, index - 1);
}

```

VureX - May 20, 2020, 6:33 pm

i dont what to do just started trying to learn

Athr - May 21, 2020, 11:11 pm

(May 20, 2020 06:33 PM)VureX Wrote: [\[CITE\]](#) i dont what to do just started trying to learn

Hello. You've posted on the wrong thread. You want to learn C or C++?

I learned a bit of C from 'The C Programming Language' by Brian Kernighan and Dennis Ritchie.

I've heard good things about this playlist for C++

<https://www.youtube.com/watch?v=18c3MTX0PK0&list=PLlrATfBNZ98dudnM48yfGUldqGD0S4FFb>

OP I'll try your challenge today. I'll admit I glanced over the posts above but will try to forget. :P

Clockwork Orange - May 22, 2020, 5:32 am

hey thanks for this my brothers, i will see if i can't crack this chullwock myself !

TheDeceptionist - May 22, 2020, 7:12 am

(May 8, 2020 04:53 AM)saltycracker Wrote: [\[CITE\]](#) If you are reaching for recursion in C or C++ then I can guarantee that its not the most efficient solution. The standard doesn't even include tail call optimization so you can't guarantee anything without inspecting the code your implementation produced.True. But it must also be mentioned that most mainstream C/C++ compilers do support tail call optimization (at least when optimizations are enabled and no language feature such as destructors prevent such an optimization).

Also the standard "doesn't even include tail call optimization" because C/C++ is not a functional programming language. It has language features such as pointers to stack which make guaranteeing such an optimization impossible. The only language standards that guarantee such an optimization are those belonging to functional programming languages such as Scheme.

virtualram - May 23, 2020, 1:18 pm

lol use auto or a static variable and ez? there is nothing hard or a challenge about this..

Litecoin - May 23, 2020, 6:48 pm

thanks i will have a look at this a bit later when i have the time