Lab 6 Daniel Hjelm:

# Task 0:

Memory we recently used will put in the cache-memory and we will reach it faster if we try to reach it again

ModifyWith8Step:

real 0m1.027s

user 0m0.802s

sys 0m0.007s  
  
ModifyLow:

real 0m0.396s

user 0m0.185s

sys 0m0.003s

Same type of result when changing flag on compiler. If we had less data, it would all fit in cache and therefore it would be as fast either way, but for this problem it did not all fit in cache.

hw.l1icachesize: 32768 on my Mac

Task 1:

Mergesort is nlog(n) and bubblesort is n^2, which for large n is a huge difference!

N = 10000000 give us:

MergeSort:

Before sort: the number 7 occurs 10027 times in the list.

Sorting list with length 10000000 took 3.255 wall seconds.

After sort : the number 7 occurs 10027 times in the list.

OK, list is sorted!

BubbleSort: Does not execute in a couple of minutes for N = 10000000 but executes for N = 100000

Before sort: the number 7 occurs 95 times in the list.

Sorting list with length 100000 took 3.586 wall seconds.

After sort : the number 7 occurs 95 times in the list.

OK, list is sorted!

With memory leakage the time went up but not by so much.

It’s slower because we can’t use cache in the same way when we do not free it.

Small memory allocations can also reduce the performance, so sometimes it can be better to use a buffer instead.

When we use buffer the time goes down to approximately 2.1-2.4 seconds which is quite a improvement.

Placing the list on stack (by writing int list1[n1];) is faster but can only handle up to 1e6 elements. When we exceed this we get a segmentation fault 11. But for 1e6 it got twice as fast.

Now, MergeSort is bad when the sublists get too small since it uses recursion and function calls are not that fast. So for small sublists we can use BubbleSort!   
For n\_treshhold = 20 and n = 1e7 we reduce the time in half (from 2.35 to 1.034)

If n\_treshhold is 200: 2.45, so it is slower now.

If n\_treshhold is 50: 1.2

If n\_treshhold = 10: 1.13

So n\_treshhold = 20 seems to be pretty good.

Task -2:

We need to use typedef otherwise we don’t know that intType is a type.

Different types will give different speeds. Let try for n = 1e7:

Long long: 3.791

Long int: 3.874

Int: 3.285

Short int: 3.321

Char: 3.060

This makes sense since we can represent the number with fewer bits so it should get faster.

What would happen if we used intType on every int? – I don’t know, ask professor

Task-3:

When N1 large, longer run time but restrict makes more difference.

Task-4:

Restrict seems to improve the speed!

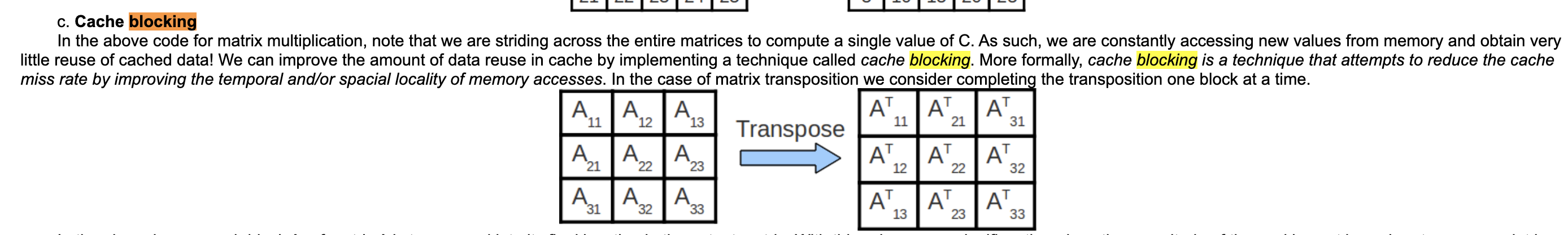
If we use NP instead of np, it becomes faster. So if we already know the value of it, we should not use it as a parameter.

Now let’s use const before NP: Wow!! It became like 5-10 times faster!!

By adding the const we know tell the program that NP will never ever change it value.

Without const, we get approx as fast as when const was outside the function.  
The const only helped when it was declared outside of the function, so if we have it in our function it does not matter.

Task-5:



A high value of blocking seems beneficial but not too high. Blocking is more effective for larger matrices.

Using the transpose\_test\_x instead of transpose\_test seems to improve a lot as well.

Task-6:

In [computer programming](https://en.wikipedia.org/wiki/Computer_programming), a **pure function** is a [function](https://en.wikipedia.org/wiki/Subroutine) that has the following properties:[[1]](https://en.wikipedia.org/wiki/Pure_function#cite_note-1)[[2]](https://en.wikipedia.org/wiki/Pure_function#cite_note-2)

1. the function [return values](https://en.wikipedia.org/wiki/Return_statement) are [identical](https://en.wikipedia.org/wiki/Relational_operator#Location_equality_vs._content_equality) for identical [arguments](https://en.wikipedia.org/wiki/Argument_of_a_function) (no variation with local [static variables](https://en.wikipedia.org/wiki/Static_variable), [non-local variables](https://en.wikipedia.org/wiki/Non-local_variable), mutable [reference arguments](https://en.wikipedia.org/wiki/Value_type_and_reference_type) or [input streams](https://en.wikipedia.org/wiki/Input/output)), and
2. the function [application](https://en.wikipedia.org/wiki/Function_application) has no [side effects](https://en.wikipedia.org/wiki/Side_effect_(computer_science)) (no mutation of local static variables, non-local variables, mutable reference arguments or input/output streams).

In [computer science](https://en.wikipedia.org/wiki/Computer_science), an operation, [function](https://en.wikipedia.org/wiki/Subroutine) or [expression](https://en.wikipedia.org/wiki/Expression_(programming)) is said to have a **side effect** if it modifies some [state](https://en.wikipedia.org/wiki/State_(computer_science)) variable value(s) outside its local environment, that is to say has an observable effect besides returning a value (the primary effect) to the invoker of the operation. Example side effects include modifying a [non-local variable](https://en.wikipedia.org/wiki/Non-local_variable), modifying a [static local variable](https://en.wikipedia.org/wiki/Static_local_variable), modifying a mutable argument [passed by reference](https://en.wikipedia.org/wiki/Evaluation_strategy#Call_by_reference), performing [I/O](https://en.wikipedia.org/wiki/I/O) or calling other functions with side-effects.[[](https://en.wikipedia.org/wiki/Side_effect_(computer_science)#cite_note-Spuler-Sajeev_1994-1)

f(i) inside loop:

real 0m4.026s

user 0m3.846s

sys 0m0.027s

f(i) outside loop:

real 0m0.008s

user 0m0.002s

sys 0m0.003s

Defining the pure function in the header file (int f(int k) \_\_attribute\_\_((const));) speeds up as well.

By defining everything inside the main-file it should also be faster, but I get some duplicate error I don’t understand.

Task-7:

KIJ is the best one.