Lab 4:

Mehtab Singh Jagde – 119003226

Luca Novello – 038515003

***Answers in green**

You may work in a group of up to 3 people if you want. If you choose to do this, ensure that you put the name of all team members into your lab6.md file. Working together can help you understand the content of the videos. You are all expected to watch the video and be part of the discussion. **Each member must submit their own copy of the work but the answers for part A can be the same. Reflection needs to be independently done**

1. Watch the following video: Sorting Algorithms: Speed Is Found In The Minds of People - Andrei Alexandrescu - CppCon 2019

2. Answer the questions in the next section, you can use the internet to help you learn about ideas that you are unfamiliar with, but you will need to link and cite your sources of information. A link is provided that may be useful for you at the bottom of this lab.

3. Write a reflection (this part is individual)

Part A: Questions about video

Do not forget to add names (and email addresses) of all group members to the top of file if you are working in a group.

1. What sorting algorithm was the speaker trying to improve?

He was trying to improve insertion sort, particularly its use as a fallback for small subarrays within hybrid sorting algorithms like introsort.

2. At what partition size does VS perform a simpler sort algorithm instead of continuing to partition?

*Visual Studio uses a threshold of **32** elements.*

3. At what partition size does GNU perform a simpler sort algorithm instead of continuing to partition?

*GNU uses a threshold of **16** elements.*

4. Regular insertion sort does a linear search backwards from end of array for correct spot to insert. According to the speaker, why does switching to a binary search not improve performance?

*Despite reducing comparisons, binary search introduces unpredictable branching that confuses the **CPU's branch predictor**, resulting in **worse performance** due to mispredictions and cache inefficiency.*

5. Describe what is meant by branch prediction? (this may require further research)

Branch prediction is a CPU feature that tries to guess the outcome of a conditional operation to maintain instruction pipeline flow. When the prediction is wrong, the CPU must discard partially completed operations, hurting performance.

Source: Branch Prediction— Algorithmica

6. What is meant by the term **informational entropy**? (this may require further research)

Informational entropy measures uncertainty or unpredictability in a system. In sorting, it reflects how much information each comparison yields. Binary search has high entropy (extracts one bit of info) but is less predictable for CPUs.
s
Source: Entropy — Wikipedia

7. If size == 15, what is size & 1? if size == 16, what is size & 1? Explain how right = first + 1 + (size & 1) avoids a conditional check. Hint:

    - The & is the bitwise AND operator in C/C++. It takes the bit representation of the two operands and performs an AND operation on each of the corresponding bits to form a result
    - To do this question first convert 15, 16 and 1 to base 2 (use 5 digit representation for all of them). Then perform an AND operation of the corresponding bits of the operands… this will get you a 5 digit binary value. Convert the value back to base 10.

- *Binary of 15: 01111*
- *Binary of 16: 10000*
- *Binary of 1: 00001*
    - *15 & 1 = 00001 = 1*
    - *16 & 1 = 00000 = 0*

*Thus, right = first + 1 + (size & 1) **shifts right one more position if size is odd** and avoids an if condition by integrating logic into arithmetic.*

8. Speaker suggests the following algorithm:

    - make_heap()
    - unguarded_insertion_sort()

    He suggests that by doing make_heap() first then you can do something called unguarded_insertion_sort(). Explain what is unguarded_insertion_sort() and why it is faster than regular insertion sort. How does performing make_heap() allow you to do this?

*It is an insertion sort **without boundary checks**. If the smallest element is known to be at the beginning, comparisons can proceed without needing to check if you've reached the start of the array. This removes one conditional check per iteration, speeding up execution.*

9. The speaker talks about incorporate your conditionals into your arithmetic. What does this mean? Provide an example of this from the video and explain how the conditional is avoided.

*Incorporating conditionals into arithmetic means using mathematical operations instead of if/else statements to make decisions. In the video, the speaker shows a **Middle-Out Insertion Sort** that adjusts the midpoint position using the formula "auto right = first + 1 + (size & 1);", where size & 1 checks if the size is odd or even. This avoids an if statement by treating the result as a 0 or 1.*

10. The speaker talks about a bug in gnu's implementation. Describe the circumstances of this bug.

*GNU's sort fails with **rotated sorted arrays**, where a very small element is appended to an otherwise sorted array. This causes quicksort to behave quadratically due to poor pivot choices, but GNU does not switch to heapsort appropriately.*

11. The speaker shows several graphs about what happens as the threshold increases using his new algorithm. The metric of comparison is increased, the metric of moves are increased but time drops... What metric does the author think is missing? Describe the missing metric he speaks about in the video. What is the metric measuring?

The missing metric the speaker refers to is comparisons. While moves and time are shown, comparisons are a key factor in evaluating sorting algorithm efficiency. The metric measures how many element-to-element comparisons are made during sorting, which can significantly impact overall performance.

12. What does the speaker mean by fast code is left leaning?

*Fast code should **flow linearly**, reducing nesting and avoiding conditionals. It tends to have fewer branches and simpler control flow. Code that aligns to the **left margin** is usually more efficient*

13. What does the speaker mean by not mixing hot and cold code?

It means keeping performance-critical code ("hot") separate from less-used paths ("cold"). Mixing them increases branching and pollutes instruction cache, making code slower.

Part B: Reflection

This part must be individually done.

1. What did you/your team find most difficult to understand in the video?

The most challenging part for me to understand was the explanation about branch prediction and how hardware-level predictions affect code execution and optimization.

2. What is the most surprising thing you learned that you did not know before?

The most surprising thing I learned was that small optimization, like avoiding conditional checks and using bitwise operations (e.g., size & 1), can have such a noticeable impact on performance. I didn't realize conditionals affected performance that way.

3. Has the video given your ideas on how you can write better/faster code? If yes, explain what you plan to change when writing code in the future. If no, explain why not.

Yes, the video has definitely given me ideas. I plan to use bitwise operations more often where applicable, avoid unnecessary conditionals, and focus on writing more cache-friendly code.

References:

You may find these articles on branching and cache useful:
https://en.algorithmica.org/hpc/

Submitting your lab

Please provide all your answers in a single file emailed with subject: dsa-456-lab5