Q1:

When a process calls fork(), the operating system needs to create a child process with the same address space as the parent. If the OS immediately copied every single page from the parent's address space into new physical pages for the child, the fork operation would become very slow, especially for large programs.

Copy-On-Write (COW) avoids this upfront cost. Instead of duplicating all physical pages right away, the OS lets the parent and child share the exact same physical pages after the fork. These shared pages are marked read-only and flagged as COW pages. Nothing is actually copied yet.

The important part is what happens when one of the processes actually tries to write to one of these shared pages. Since the page is read-only, this triggers a write-fault. At that moment, the OS allocates a new physical page, copies the data from the old shared page, and then gives the writing process its own private, writable copy. The other process continues using the original page.

This technique improves fork() latency because:

- The OS doesn't copy the whole address space at fork time.
- Most child processes call exec() right after fork, so they never modify the old pages—meaning zero memory copying happens.
- Pages are only copied if they are actually written to, so the system only performs work when necessary.

As a result, COW makes fork() extremely fast, even if the parent process uses a lot of memory.

Sources used for Q1:

1. OS Textbook, Chapter 5
   https://pages.cs.wisc.edu/~remzi/OSTEP/cpu-api.pdf
2. Linux Kernel Documentation – Understanding the Linux Virtual Memory Manager
   https://www.kernel.org/doc/gorman/html/understand/understand013.html

Q2:

fork() using Copy-On-Write

```
function fork(parent):

  create a new child page table

  for each page P in the parent:
    // Child shares the same physical frame as the parent
    child.PTE[P] = parent.PTE[P]

    // Both PTEs become read-only and marked COW
    parent.PTE[P].writable = false
    child.PTE[P].writable = false
    parent.PTE[P].cow = true
    child.PTE[P].cow = true

    // Increase the reference count for that frame
    refcount[P] += 1
  end for

  return child
end function
```

Handling a write to a COW page

```
on write_page_fault(process, page P):
```

```
  if P is a COW page:
    old_frame = P.frame

    // Allocate a new physical page
    new_frame = allocate_frame()

    // Copy data over
    copy old_frame → new_frame

    // Update the process's page table to point to the new frame
    P.frame = new_frame
    P.writable = true
    P.cow = false

    // Decrement refcount on the old page
    refcount[old_frame] -= 1
    if refcount[old_frame] == 0:
      free(old_frame)
    end if
  end if
end handler
```

Freeing pages on process exit

```
function exit(process):
```

```
    for each page P:
        refcount[P.frame] -= 1

        if refcount[P.frame] == 0:
            free(P.frame)
        end if
    end for

    destroy(process)
end function
```