# Week 10 Recitation
# Project Breakdown and File Reading

**Setting Up .cmd File**

Open up run_qtspim.cmd with a text editor and edit the two paths.
One path needs to be pointed to your qtspim simulator program directory, whereas the other should be pointed to your test directory.

**Setting Up w/o .cmd File** (Non-windows users)

Another way of running the project is to first reinitialize and load the file, "main.s".
Then do a simple load file on the rest of the files
    (euclidean_distance, io, sort, find)
Finally, hit run to start your program.

Note: if the script is not being used, the file name in main.s should be an absolute path.

**My Recommendation of Approach**

(1) *load_points and load_points_helper* (File: io.s)

load_points
Input: $a0 - base address of file name string.
Output: $v0 - number of points
    $v1 - base address of the points array (structured (x1,y1,x2,y2, etc.)
Approach:
    Open file using syscall 13 to get a file descriptor.
    Store $ra into some variable or onto the stack to preserve continuity.
    Call *load_points_helper* w/ file descriptor as the input.

load_points_helper
Input: $a0 - file descriptor.
Output: $v0 - number of points
    $v1 - base address of the points array (structured (x1,y1,x2,y2, etc.)
Approach:
    Read file using syscall 14 (reads a certain number of specified bytes)
        First read: 4 bytes → number of points.
        Second read: x bytes → rest of points into some buffer.

    Things to consider: Using a fixed array of size MAX_NUM_POINTS*2+1, declared in .data. You can also use syscall 9 to try to allocate a block of memory.

(2) *euclidean_distance* (File: euclidean_distance.s)

Input: $a0 - x0 | $a1 - x1 | $a2 - y0 | $a3 - y1

Output: $v0 - calculated distance between two points.

Approach:

No need to implement sqrt, instead, use formula (x1 - x2)^2 + (y1-y2)^2

(3) *sort_points_by_x* and *sort_points_by_y* (File: sort.s)

sort _points_by_x:

Input: $a0 - number of points
$a1 - points array base address

Output: none, but array is sorted in memory by x

Approach:

Use any sorting algorithm you wish to implement (bubble sort, merge, etc.)
You can try using a helper function to assist in swapping values.

sort _points_by_y:

Input: $a0 - number of points
$a1 - points array base address

Output: none, but array is sorted in memory by x

Approach: Same as *sort points by x*

(4) *find_closest* (File: find.s)

find_closest

Input: $a0 - number of points

Output: $v0, $v1 - address of 2 points of the closest pair.

Approach:

Brute-force (half credit) → checking every combination of points.
Using sorting for x and/or y to assist somehow.

(5) *output_cloest_pair* (File: io.s)

output_cloest _pair

Input: $a0, $a1 - address of 2 points of the closest pair.

Note: $a0 is the address of x0, 4($a0) is y0,
$a1 is the address of x1, 4($a1) is y1

Approach:

Using the syscall table, use the ones that allow you to print out the integers. You can take additional steps and use fixed strings in memory to try to “pretty-print” the values.

## Example of File Read and Access

```
.data
        file: .asciiz “/absolute/path/to/file”

        .align 2                #do not forget if trying to store words to memory
        buffer: .space 4
.text
main:
# Open File
        li      $v0, 13
        la      $a0, file
        add     $a1, $0, $0
        add     $a2, $0, $0
        syscall                         # Open FIle, $a0<-fd


# Read 4 bytes from file, storing in buffer
        li      $v0, 9                  # Memory allocation syscall
        li      $a0, NUM_BYTES
        syscall

        li      $v0, 14                 # 14=read from  file
        move    $a1, $v0                # $a1 is set to the address of the holder/buffer
        li      $a2, 4                  # $a2 holds the number of bytes to read in
        syscall

        li      $v0, 1                  # 1=print int
        lw      $a0, buffer             # buffer contains the int
        syscall                         # print int
```