

Introduction

As per the lab requirements, I used some C++ code I had previously written to construct an approximate solution to the *n Queens on a $n \times n$ Chessboard problem* with simulated annealing. I ran it on my home computer running Windows 10, in a Docker container using Visual Studio Code and a Virtual Machine running Ubuntu 20, and compared the performances of each run.

Overview

A virtual machine is a virtual environment that acts as a virtual computer system with its own CPU, memory, network interface, and storage, which is allocated upon creation. A hypervisor separates the machine's resources from the hardware and provisions them appropriately so they can be used by the VM, providing an additional layer of security.

Containers are somewhat similar to virtual machines, as they provide isolated environments for running software, however they are less independent than VMs, as they share the same underlying hardware system below the operating system layer, which can be exploited.

Algorithm

```
int f(vector<int> &q, int n) {
    int tot = 0;
    for (int i = 0; i < n; i++) {
        bool no_col = true;
        for (int j = 0; j < i; j++) {
            if (abs(i - j) == abs(q[i] - q[j])) {
                no_col = false;
            }
        }
        tot += no_col;
    }
    return tot;
}

void anneal_queens(int n) {
    for (int i = 0; i < K && ans < n; i++) {
        t *= 0.99;
        vector<int> u = v;
        swap(u[rand() % n], u[rand() % n]);
        int new_val = f(u, n);
        if (new_val > ans || rand() < exp((new_val - ans) / t)) {
            v = u;
            ans = new_val;
        }
    }
}
```

Figure 1: An excerpt of the code used.

The full code can be found at [the Github repo](#).

Method

In the main function of the program, the `anneal_queens` function was called for various values of n . The values started at $n = 10$, in increments of 10 until 1000. The time was recorded before and after each call to `anneal_queens`, which allowed for the calculation of the time taken for the call.

A fixed seed was set initially in the main function, so that the values encountered by all 3 runs (i.e. Windows, VM, Container) would be identical.

My Performance Expectations

I expect the program to run fastest on the regular environment (the base Operating System), as anything run there would directly call to the Kernel, with minimal relative overhead.

Next, I would expect the Docker Container to be the second fastest. It would have a larger overhead than the regular environment, however it would still go through the same Kernel as the host Operating System.

Finally, I would expect the VM to be the slowest, as it runs on top of the host operating system (thus a large overhead) with less hardware resources.

Results & Graphs

The total time taken for each run can be found on the table below.

Environment	Total Execution Time (s)
Windows	35.064
Container	51.0151
VM	90.236

Figure 2: Total time taken for $n = 10, 20, \dots 1000$

The full results can be found at [the Github repo](#).

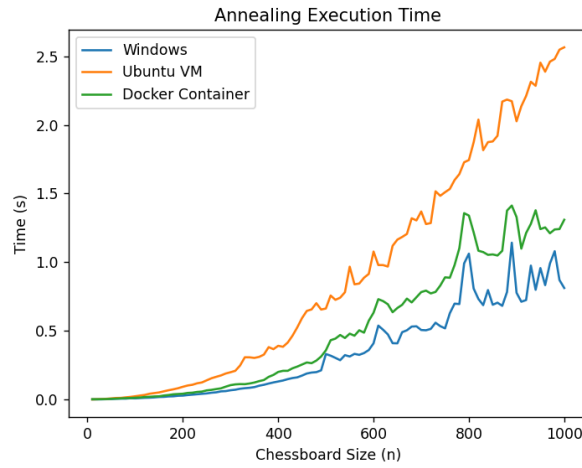


Figure 3: Graph plotting time vs chessboard size.

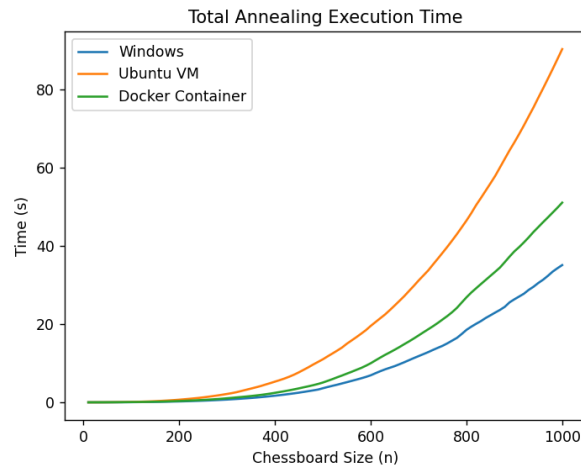


Figure 4: Graph plotting total time for 10, 20, \dots n vs chessboard size (n).

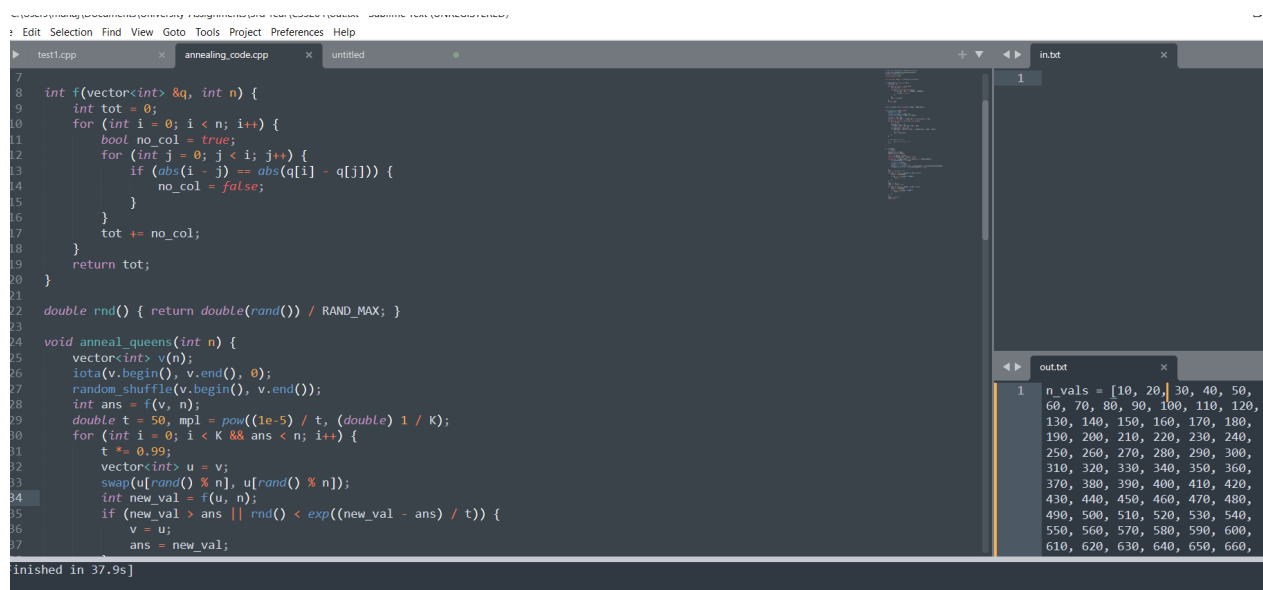
Conclusion

As per my hypothesis, the base/regular environment performed the fastest, owing to its minimal relative overhead, the Docker Container was slower, and the code run on the Virtual Machine performed the slowest. Despite this however, I gained a good insight into how Virtual Machines work and particularly why they would be used. The increased security and freedom as malware cannot spread to the host operating system makes VMs a viable option for many fields, which cannot be achieved on regular environments and via containers.

Reflection/Thoughts

As expected, running the code on my regular environment was the simplest task. Before testing the code on the other systems, I presumed that executing the code on the VM would be more arduous than doing so on the Docker Container, however it ended up being the other way around. Setting up, copying the code onto, and executing the code on the virtual machine was surprisingly simple, whereas I found a lot of obstacles when using Docker. This was in no small part due to my choice of IDE to use Docker (Visual Studio Code) which had to be restarted multiple times, and my choice of language (C++) which there existed fewer resources for using alongside Docker in comparison to a language like Python.

Screenshots



```
7
8 int f(vector<int> &q, int n) {
9     int tot = 0;
10    for (int i = 0; i < n; i++) {
11        bool no_col = true;
12        for (int j = 0; j < i; j++) {
13            if (abs(i - j) == abs(q[i] - q[j])) {
14                no_col = false;
15            }
16        }
17        tot += no_col;
18    }
19    return tot;
20 }
21
22 double rnd() { return double(rand()) / RAND_MAX; }
23
24 void anneal_queens(int n) {
25     vector<int> v(n);
26     iota(v.begin(), v.end(), 0);
27     random_shuffle(v.begin(), v.end());
28     int ans = f(v, n);
29     double t = 50, mpl = pow((1e-5) / t, (double) 1 / K);
30     for (int i = 0; i < K && ans < n; i++) {
31         t *= 0.99;
32         vector<int> u = v;
33         swap(u[rand() % n], u[rand() % n]);
34         int new_val = f(u, n);
35         if (new_val > ans || rnd() < exp((new_val - ans) / t)) {
36             v = u;
37             ans = new_val;
38         }
39     }
40 }
41
42 int main() {
43     int n;
44     cin >> n;
45     anneal_queens(n);
46     cout << ans << endl;
47 }
```

```
1 n_vals = [10, 20, 30, 40, 50,
60, 70, 80, 90, 100, 110, 120,
130, 140, 150, 160, 170, 180,
190, 200, 210, 220, 230, 240,
250, 260, 270, 280, 290, 300,
310, 320, 330, 340, 350, 360,
370, 380, 390, 400, 410, 420,
430, 440, 450, 460, 470, 480,
490, 500, 510, 520, 530, 540,
550, 560, 570, 580, 590, 600,
610, 620, 630, 640, 650, 660,
```

finished in 37.9s]

Figure 5: Running the code on Windows (using Sublime Text).

