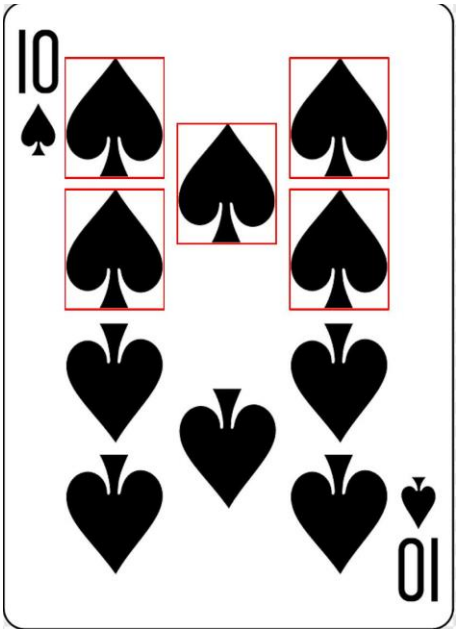
# Problem 1: Single Thread and Multi-Thread Implementation of Template Matching Algorithm

The resulting image from the basic algorithm:



Where the algorithm is written similarly to the given Matlab-like pseudo-code.

```
private static short[][] templateMatching(BufferedImage source, BufferedImage template, int widthSource, int heightSource, int widthTemplate, int heightTemplate, int sectionStart, int sectionEnd) throws IOException {

    double tempSize = heightTemplate*widthTemplate; // Total number of pixels in the template, to normalize the total absolute difference

    double[][] absDiffMat = new double[Math.abs(sectionEnd-widthTemplate)][Math.abs(heightSource-heightTemplate)];

    short[][] coordinates = new short[1][2];

    ArrayList<short[]> coordinatesList = new ArrayList<short[]>();

    for (int i=sectionStart; i<Math.abs(sectionEnd-widthTemplate); i++) {
        for (int j=0; j<Math.abs(heightSource-heightTemplate); j++) {
            double absDiff = 0;
            BufferedImage Nimage = source.getSubimage(i,j,widthTemplate,heightTemplate);

            short[][] grayNimage = convertImage(Nimage);
            short[][] grayTemp = convertImage(template);

            // Compares the total sum of the difference between pixels, when comparing a slice of the source image vs the template.
            for (int a = 0; a < heightTemplate; a++) {
                for (int b = 0; b < widthTemplate; b++) {
                    absDiff += Math.abs(grayNimage[a][b] - grayTemp[a][b])/tempSize;
                }
            }

            // Stores the absolute difference in a matrix at that particular position of the slice; to compare to a threshold value
            absDiffMat[i][j] = absDiff;

            Didn't use threshold to figure out where the matches are; if Nimage happens to be a perfect
            match to the template, then the absolute difference between images should be an 0, or an extremely
            low number, so i chose 5.
            if (absDiffMat[i][j] <= 5) {
                // Adds the specific i and j values of if there is a perfect match. Done by using an ArrayList
                short[] coordinate = new short[] {(short) i, (short) j};
                coordinatesList.add(coordinate);
            }
        }
    }

    coordinates = coordinatesList.toArray(new short[0][]);
    return coordinates;
}
```

Where the method convertImage converts from BufferedImage to short[][] to calculate the absolute difference.

## Multi-threaded implementation

By using a thread class, the algorithm can be done by having multiple concurrent threads scanning the source image for a match with the template image. The class is given by:

```
public class TemplateMatchThread extends Thread{
```

Where the Thread.start(); method is overridden to run the template matching algorithm.

This thread is invoked by the driver class, which creates a for loop and creates a new Thread object for every numOfThreads:

```
for (int i = 0; i < numOfThreads; i++) {
    Thread[i] = new TemplateMatchThread(sourceImage, templateImage, sectionStart, sectionEnd);
    Thread[i].start();

    sectionStart = sectionEnd;
    sectionEnd += sectionWidth;
}
```

A new section of the image Is given to each thread, where it scans for a match between the template and the source.

Single-thread result:

This is done by setting the numOfThreads = 1.

```
Working with 1 thread
Done! Execution time: 37.27224334 minutes
```

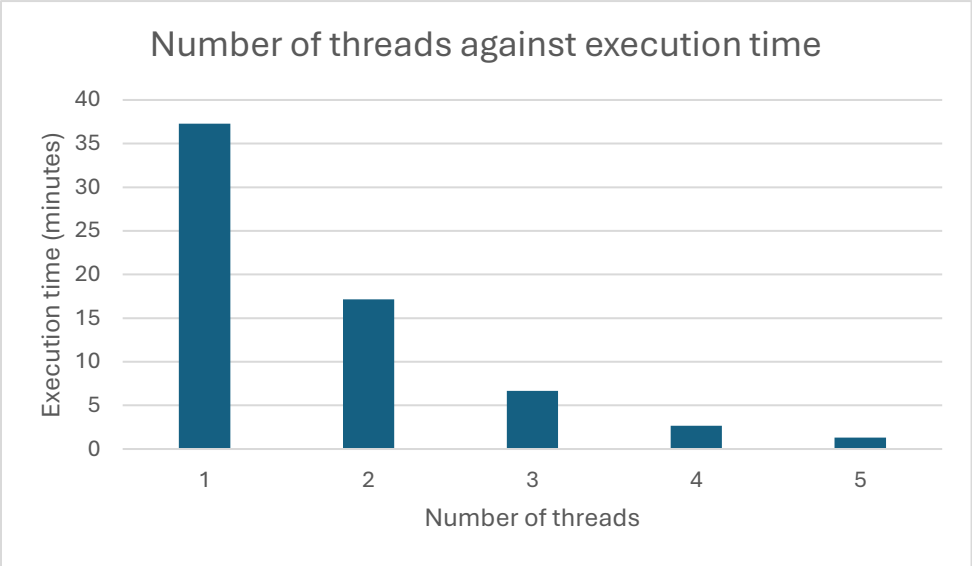
Multi-threaded results:

```
Working with 2 threads
Done! Execution time: 17.154343545000003 minutes
```

```
Working with 3 threads
Done! Execution time: 6.659223501666667 minutes
```

```
Working with 4 threads
Done! Execution time: 2.6495652933333336 minutes
```

Note: With a higher number of threads, the template slice is unable to overlap with potential perfect matches before terminating; due to splitting even slices of the source image by the number of threads.




This information proves that using more threads can drastically cut the execution time down a lot. The higher number of threads, the lower the execution time. This is done in an inverse exponential curve.

The only issue with using more threads is the lesser likelihood of successful matches due to splitting the source image in sections.

Github repo: gp974UOWD/gp974-ECTE331 (github.com)