**Introduction:**

The problem presented to us in lab two was given several images of the sun convert that image into arrays then use those arrays and systematically deduce the brightest region of the picture using different methods, the first version is an implementation that uses 4 for loops, the next is a version that uses 2 loops and array slices, the third version requires the use of 2 for loops an integral image table and 4 array accesses, and the fourth version just access to the integral image table.

**Proposed Solution Design and Implementation:**

Technique 1:

The first version was the 4 for loops variation, a for loop would be used to increment the rows, columns, height of the sub region, and width of the sub region. The method finds the sum of the sub region and compares it to the max sub region finds which is greater then after its done comparing it resets the sub region back to 0 moves the subregion and finds the sum again. This implementation is by far the slowest but could have been made slightly faster by instead of resetting the sum region back to 0 adding the new area of the sub region and subtracting the old area of the sub region. Once the max sub region is found it gives the indexes of those region to a method that draws a rectangle around it on the black and white image

Technique 2:

The second version of the region sum method uses 2 for loops and array slices to find the maximum summed region. This method uses the two for loops to increment the rows and columns which serve as the top left corner of the sub region, then using the numpy .sum() feature gets the sum of array slices of the length and width of the size of the sub region it then compares that number to the greatest sub region and determines which is bigger. Once the method loops through the array, it sends the index of the left corner and uses it on a method that draws a rectangle around the region and displays it on a black and white image.

Technique 3:

The third version implemented was the same thing as the others except it made the use of a integral image table instead of using loops to sum the area. The method called a function called create\_integral\_image that took in an array along with how many rows and columns that array has and returned the proper integral image. It then used 2 for loops to index the top left corner of the region trying to be summed then summed the area using 4 array indexes to the integral table then saved the largest of the index and passed to a function that ploted the region and displayed the black and white image. I originally wanted the integral image to be passed as a parameter but that caused the problem that instead of displaying the image the function would display the integral table so in the end I decided to pass both the integral image and the original image as parameters as it would be easier to more accurately time

**Experimental results**

I tested all three versions of the sum region method, I used the time.time() function in python to time the start and end of each of the implementations then had each of those methods return that running time. Then in the Main method I would call those methods and add all those methods to a list then was able to plot that using the plt.bar function in pyplot. Unexpectantly timing all of these functions took a very long time, so in order to reduce the time taken I decided to take the running time of just a single image rather then every picture in the file here are those graphs:

A screenshot of a cell phone

Description automatically generated

(20,20) (30,30) (30,60)

These numbers are in seconds, there are supposed to be 3 bars but as the 1st primitive method took so long and the fastest version 3 took substantially less time then version 2 and the same thing occurred for version 2 and 1, as seen from the graphs version 1 took upwards of 600 seconds, while version 2 took upwards of 7 seconds, and version 3 being closer to .7 of a second cant even be seen on the graph

A screenshot of a cell phone

Description automatically generated

for the greater sizes of sub region its more of the same with the increasing size of the sub region the longer it takes to run, the 100,100 sun region taking close to 2500 seconds which is around 40 minutes, and the remaining methods hovering around the same time. It obvious to see that the 1st method runs int Big O of n^4. The third method also shouldn’t change very much if any at all, because no matter the size of the array or sub region it only needs to male 4 array calls to compute the sum of the sub region.

**Conclusions**

We take for granted the speed of our computers and graphics processors, but the method implementation is extremely important as well. It is important to always be improving and using techniques to create the fastest methods we possibly can. All the above methods do the exact same thing, but one takes 0.7 of a second and the other takes more them 40 minutes. This is extremely huge change in a seemingly simple problem but without it many of the advance displays used today would be extremely impractical but though the use of computer science clever mathematics and code compiling we can be more efficient and effective.