CSE 331 HW2 Report

1-) Pseudocode

- This pseudocode runs on only 1 array. I have implemented a loop in my original code that the MIPS program runs for 6 predefined arrays. This part of the pseudocode includes the problem solving algorithm as well as printing all found sequences on the screen. The last printed sequence is the answer.

```
int main()
{
   int A[7]={12,7,1,5,7,6,9};   →   This is the predefined array.
   int B[7];   →   This array will hold all found sequences one by one
   int C[7];   →   This array will hold the result sequence
   int size_counter=0;   →   This is the size of array B
   int size=0;   →   This is the size of array C

   for(int i=0; i<6; i++)   →   Here the loop runs 6 times because the array length is
   {                            6. However in my MIPS code the code can run for any
      size_counter=0;           lengths. I cannot show it in the c code.
      B[size_counter]=A[i];
      size_counter++;
      printf("Found sequence: %d\n", A[i]);  }  This is for printing the very first sequence which is
      for(int j=i+1; j<7; j++)               }  the first element of the array
         if(A[j]>B[size_counter-1])
         {
            B[size_counter]=A[j];   }   Here the sequence is being found and stored in B.
            size_counter++;         }   Each element is compared.

            printf("Found sequence: ");          }   This is for printing all sequences that are found
            for(int m=0; m<size_counter; m++)    }
               printf("%d, ", B[m]);             }
            printf("\n");                        }
         }

      if(size_counter>size)                   }
      {                                       }   Here the found sequence size is compared with the
         size=0;                              }   previously found sequence. If the current one's size
         for(int k=0; k<size_counter; k++)    }   is bigger it is stored in array C.
         {                                    }
            C[k]=B[k];                        }
            size++;                           }
         }                                    }
      }

      printf("Current largest sequence: ");   }   This is for printing the result sequence.
      for(int m=0; m<size; m++)               }
         printf("%d, ", C[m]);                }
      printf("\n\n");                         }
   }
}
```

- In this algorithm I have chosen to take one element from the predefined array each time. Compare it with the following elements. If the following element is bigger, I store both of them in array B. The following element is compared with the last stored element and B is rewritten etc. After the process ends for one element, the sequence stored in B is transferred into C only if B's size is larger than C's size. Meaning B is being rewritten each time a comparison happens and a new sequence is found however, it is not the same for C. I have done it this way to prevent storing all found sequences in different storage areas. Because sequences include the ones with 1 element only as well. Meaning there would be many storage needed.

- The space complexity of the algorithm is: O(n). Because there is exactly 3 arrays which have maximum of size n (n being the number of elements in the predefined array) Aside from the 3 arrays there is a few other variables like size_counter, size, i, k, m. They both require only O(1) space. **Therefore Space Complexity: 3xO(n)+5xO(1)=O(n)**

- I will analyze the time complexity in 2 different conditions.

1-) If the print statements were not there, meaning there was not nested 3 for loops. Because printing out all found sequences is actually not a part of the algorithm. Printing out the last found sequence is enough. However, I added extra prints in the loop to have a better idea what is going on during the process. So if there was no print statements:
-First inner for loop runs from i+1 to n. Inside, there are only increment and assign functions which are all O(1). Therefore it would take O(n) time in total when it ran from i+1 to n. This loop will run for all elements in the array that's why it has O(n^2) time complexity when combined with the outer loop.
-Second inner loop runs from 0 to size_counter. size_counter can never be bigger than n, it can be max n. Inside the for loop, there are only increment and assign functions which are all O(1). This for loop depends on an if statement therefore it will not run each time generally. However, in the worst case it might run each time with the size_counter starting with 1 and incremented by one.This still leads to ((k(k-1))/2 runtimes for the loop giving it O(n^2) time complexity when combined with the outer loop.
-All the other statements in the outer loop has O(1) time complexity meaning they have O(n) time complexity combined with the loop.
-The time complexity: O(n^2)
2-) If the print statements stay, there would be one extra loop inside the first inner loop running from 0 to size_counter. This size_counter might be maximum n. In the worst case it will start from 1 and be incremented by one each time. This leads to (((m(m-1))/2)*n runtimes for the loop making the loop have O(n^3) time complexity. Because O(n^3) will be the new biggest growing time complexity in the algorithm the algorithm time complexity would be O(n^3) as well.

2-) General Explanations

- I hold predefined arrays as data. I hold A, B, C array addresses and their sizes in s registers to keep them all the time. I use t registers for all the other variables aside from parameters and returns. I try to reuse t registers as much as possible to avoid using extra space.
- I have implemented the console print parts in a procedure called print_sequence. It takes an array and size of it as parameters. It is used to print both B and C arrays. It works the same as the print parts of my pseudocode. It also includes functions printing to output file as well. I automatically print the Array_outp: [ ] size = parts of the outputs in the output file. The values are extracted from the C array and commas are put accordingly. Size is also extracted from the C array.
- I have implemented the assign parts for the arrays in a procedure called assign_array_values. It takes 2 arrays and their indexes as parameters. It works the same as the assign parts of my pseudocode.

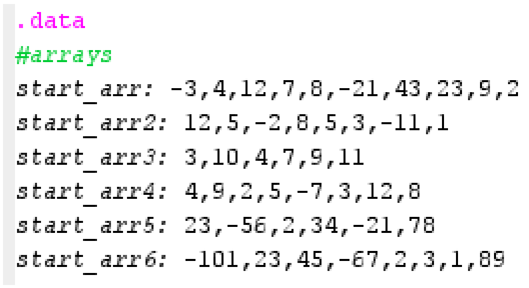
- I have implemented a procedure called itoa. This function works exactly like itoa function of C. It converts an integer to a string and returns the string. I have used itoa implementation of C to write that procedure. This procedure was needed because MIPS do not write integers on files.
- I have implemented a procedure called find_start_arr to maintain the loop of the program. My MIPS program will run exactly 6 times and each time it will process different predefined arrays. This procedure analyzes the current processed predefined array (A in the pseudocode) and switches to the following array when the procedure is called. It also saves the next following array address to later on calculate the current A array's size. This procedure is utilizing mips addressing system.
- To find the array length I substract the following array's address from the current array's address and divide by 4. Because each element takes 4 bytes in the memory.
- I have implemented a procedure called count_digit to determine the digit amount of the elements in C array. It takes the number as a parameter and returns the digit count of it. This was needed because MIPS needs to know the exact size of chars to write them on a file. To count the digits, I firstly check if the number is negative. If so, I increment a counter. Afterwards I keep dividing by 10 until the division result is zero. Each time I increment the counter again. The counter at the end gives the size.
- I have parts in my code where I open&close files. Aside from that all file related functions are done in my print_sequence procedure to print result sequences.
- All the other parts are the exact translation of my pseudocode.

3-) Extra & Missing Parts

- My program can run with any size of arrays. It can work with arrays bigger than size 10 as well. I have shown it in test cases.
- My program can run with negative numbers as well. I have shown it in test cases.
- My program prints each found sequence to the console even if it is not the result. Therefore each step of the process is shown. The result sequence is printed at the end for each array.
- My program does open&close an input.txt and output.txt. However, it does not read from input.txt. It reads the arrays from the data section. Outputing to the file is done properly.
- My program does not print the steps to the outputfile. It only prints the end results to it in a exact Array_outp: [3, 7, 9, 11] size = 4 format like mentioned in the hw pdf.

4-) Results & Test Cases

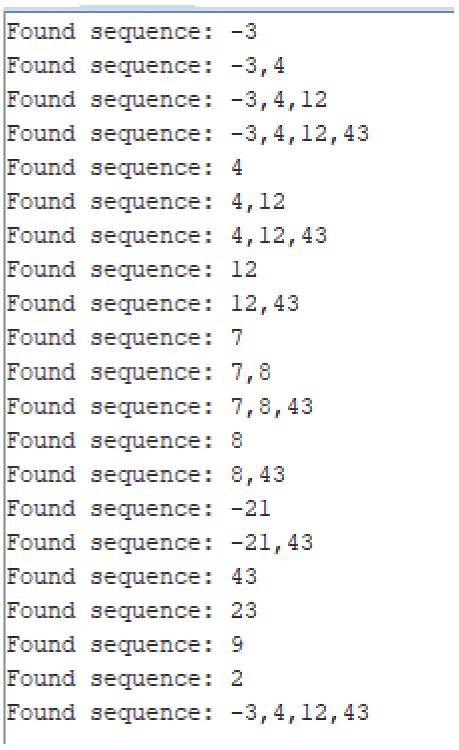
- The 6 arrays I tested are these. You can change these arrays by hand in this format if you want to. However, don't add extra arrays or remove arrays from there otherwise it might not work properly.



```
.data
#arrays
start_arr: -3,4,12,7,8,-21,43,23,9,2
start_arr2: 12,5,-2,8,5,3,-11,1
start_arr3: 3,10,4,7,9,11
start_arr4: 4,9,2,5,-7,3,12,8
start_arr5: 23,-56,2,34,-21,78
start_arr6: -101,23,45,-67,2,3,1,89
```

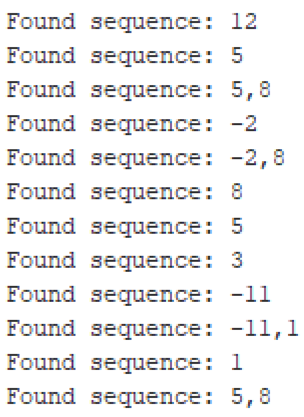
- The test results of these arrays are these. All steps are shown. Array steps and results are separated by 1 blank line. The ones at the end of each section are the results. You can check output.txt file for the exact detailed results. In output.txt file the 1 result sequences are written in ea line.

Array 1



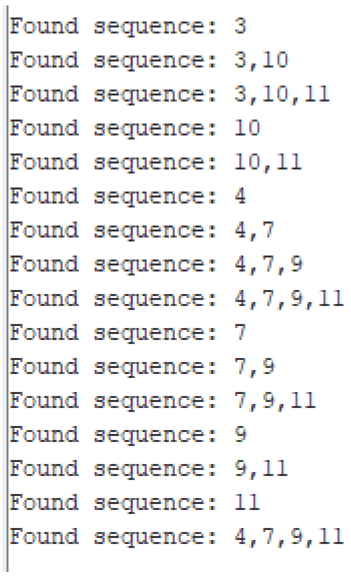
```
Found sequence: -3
Found sequence: -3,4
Found sequence: -3,4,12
Found sequence: -3,4,12,43
Found sequence: 4
Found sequence: 4,12
Found sequence: 4,12,43
Found sequence: 12
Found sequence: 12,43
Found sequence: 7
Found sequence: 7,8
Found sequence: 7,8,43
Found sequence: 8
Found sequence: 8,43
Found sequence: -21
Found sequence: -21,43
Found sequence: 43
Found sequence: 23
Found sequence: 9
Found sequence: 2
Found sequence: -3,4,12,43
```

Array 2



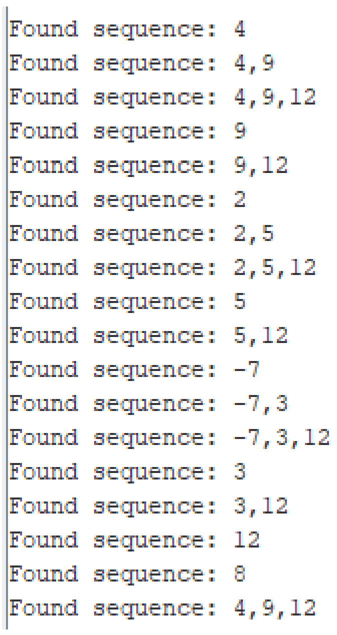
```
Found sequence: 12
Found sequence: 5
Found sequence: 5,8
Found sequence: -2
Found sequence: -2,8
Found sequence: 8
Found sequence: 5
Found sequence: 3
Found sequence: -11
Found sequence: -11,1
Found sequence: 1
Found sequence: 5,8
```

Array 3



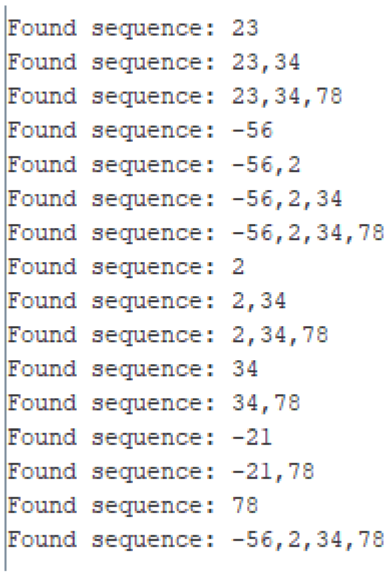
```
Found sequence: 3
Found sequence: 3,10
Found sequence: 3,10,11
Found sequence: 10
Found sequence: 10,11
Found sequence: 4
Found sequence: 4,7
Found sequence: 4,7,9
Found sequence: 4,7,9,11
Found sequence: 7
Found sequence: 7,9
Found sequence: 7,9,11
Found sequence: 9
Found sequence: 9,11
Found sequence: 11
Found sequence: 4,7,9,11
```

Array 4



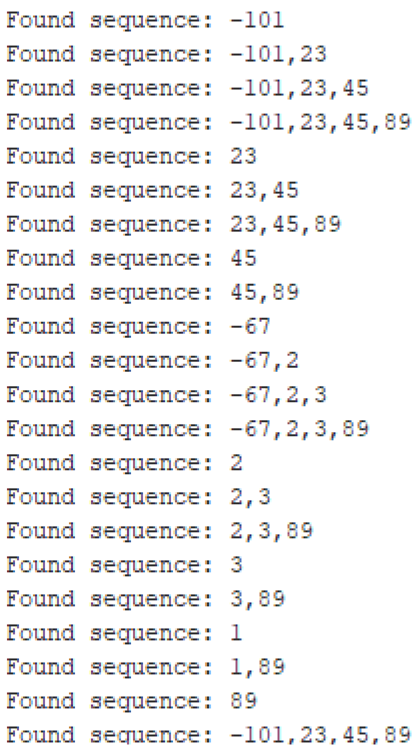
```
Found sequence: 4
Found sequence: 4,9
Found sequence: 4,9,12
Found sequence: 9
Found sequence: 9,12
Found sequence: 2
Found sequence: 2,5
Found sequence: 2,5,12
Found sequence: 5
Found sequence: 5,12
Found sequence: -7
Found sequence: -7,3
Found sequence: -7,3,12
Found sequence: 3
Found sequence: 3,12
Found sequence: 12
Found sequence: 8
Found sequence: 4,9,12
```

Array 5



```
Found sequence: 23
Found sequence: 23,34
Found sequence: 23,34,78
Found sequence: -56
Found sequence: -56,2
Found sequence: -56,2,34
Found sequence: -56,2,34,78
Found sequence: 2
Found sequence: 2,34
Found sequence: 2,34,78
Found sequence: 34
Found sequence: 34,78
Found sequence: -21
Found sequence: -21,78
Found sequence: 78
Found sequence: -56,2,34,78
```

Array 6



```
Found sequence: -101
Found sequence: -101,23
Found sequence: -101,23,45
Found sequence: -101,23,45,89
Found sequence: 23
Found sequence: 23,45
Found sequence: 23,45,89
Found sequence: 45
Found sequence: 45,89
Found sequence: -67
Found sequence: -67,2
Found sequence: -67,2,3
Found sequence: -67,2,3,89
Found sequence: 2
Found sequence: 2,3
Found sequence: 2,3,89
Found sequence: 3
Found sequence: 3,89
Found sequence: 1
Found sequence: 1,89
Found sequence: 89
Found sequence: -101,23,45,89
```