```
#include <cstdio>
#include <cstring>
#define N 23//this means until n = <Your input Number>
using namespace std;
//so for this algorithm we must try every possible difference in
//the array of primes, for each prime and scan O(n/logn) primes
//to arrive tou our conclusion
//we will create the function that will create our prime only
//array here
//
//however here will be  our dynamic programming algorithm
int A[N+1];
const int Asize = sizeof(A)/sizeof(A[0]);//im sorry this is kind of redundant but i
build this really fast without much mind to efficiency or an elegant solution, but should
still be readable
int primeCounter = 0;
int currentLargest =0; //as we create our table this value will change according to the
current largest subsequence of primes
void buildA(){
    cout << "Building A : ";
    for(int i =1;i<=N;i++){
        A[i-1] = i;
        cout << A[i-1]<<",";
    }
    cout << endl;
}
void SieveOfEratosthenes(int n,int primeDictionary[Asize], int primeOnlyArr[Asize])
{
    printf("We will now calculate the primes from 1 to %d\n",n);
    //this will help us keep track of our
    //assume all are prime at first, primes are identified if
    //their index contains 1, not prime = -1
    memset(primeDictionary, 1,sizeof(A[0])* Asize);
    primeDictionary[0] = -1;
    primeDictionary[1] = -1;
    for (int p=2; p*p<=n; p++){
        // If prime[p] is not changed, then it is a prime
        if (primeDictionary[p] != 1){
            // Update all multiples of p
            for (int i=p*2; i<=n; i += p)
                primeDictionary[i] = -1;
        }
    }

    // Print all prime numbers
    for (int p=2; p<=n; p++)
        if (primeDictionary[p])
            cout << p << " ";
    cout << endl;
    //now we index assign the indexes to our array
    for(int p=2;p<=n;p++){
        if(primeDictionary[p] != -1){
            //we found our prime and assign its index
            //so basically if the element as a number greater
            //than zero that means its the index to its prime
            //only array
            primeDictionary[p] = primeCounter++;
        }
    }
    // Print all prime numbers and store in the prime only array
   for (int p=2; p<n; p++)
        if (primeDictionary[p] != -1) {
            printf("The index for prime %d is %d.\n",p,primeDictionary[p]);
            primeOnlyArr[primeDictionary[p]] = p;
        }
    cout <<endl;
}

void buildResult(int resultA[][Asize],int primeIndexDictionary[Asize],int
primeOnlyArr[Asize]){
    for(int i = 0;i<primeCounter;i++){
        for(int j=0;j<Asize; j++){
            int curPrime = primeOnlyArr[i];//this can be thought of as the rows of
result
            if(j == 0 || curPrime == 2)//these col and row will be popuplated with 0
                continue;
            //else
            //fetch previous sequence length
            if(curPrime - j < 2){
                continue;//nothing else to do when we will keep getting differences to <
2
            }
            if(primeIndexDictionary[curPrime-j] != -1){
                resultA[i][j] += 1 + resultA[primeIndexDictionary[curPrime-j]][j];
                if(resultA[i][j] > currentLargest)
                    currentLargest = resultA[i][j];
            }
        }
    }
}
void printArray(int result[][Asize],int primeOnlyArr[Asize]){
    printf("       ");
    for(int i = 0;i<Asize;i++){
        printf("[%5d]",i);
    }
    cout<<endl;
    for(int i = 0;i<primeCounter;i++){
        for(int j = 0;j<Asize;j++){
            if(j==0){
                if(primeOnlyArr[i] == 24)
                    printf("For some reason primeOnlyArr is %d\n",primeOnlyArr[i]);
                printf("[%5d]",primeOnlyArr[i]);
            }
            printf("[%5d]",result[i][j]);
        }
        cout << endl;
    }
}

int main(){
    buildA();
    int primeIndexDictionary[Asize];//i know it could be less but for simplicity sake
    int primeOnlyArr[Asize];//same for this one

    SieveOfEratosthenes(Asize,primeIndexDictionary,primeOnlyArr);
    int result[primeCounter][Asize];
    memset(result,0,sizeof(result[0][0])*primeCounter*Asize+(1*sizeof(result[0][0])));
    cout << "Before building the result " <<endl;
    printArray(result,primeOnlyArr);
    buildResult(result,primeIndexDictionary,primeOnlyArr);
    cout <<"After"<<endl;
    printArray(result,primeOnlyArr);
    printf("The largest PAP sequence is of %d numbers : \n",currentLargest+1);

    return 0;
}
```