```
#include <iostream>
#include <math.h>
#include <tuple>
#include <cstdio>
#define N 3
using namespace std;
int A[N][N]= {{6,3,2},{4,20,5},{10,8,15}};
tuple<int,int> mem[N][N];
bool exists = false;
tuple<int,int> answer;
void fillMem(){
        for(int i = 0;i <N;i++){
                for(int j = 0;j<N;j++){
                        mem[i][j] = make_tuple(-1,-1);
                }
        }
}
tuple<int,int> defaultTuple(int neginf,int posinf){
        return make_tuple(neginf,posinf);
}
tuple<int,int> path(int i,int j){
        if(i == (N-1) && (j == N-1)){
                printf("REACHED i=%d,j=%d\n",i,j);
                tuple<int,int> reacho = make_tuple(0,0);
                return reacho;
        }
        printf("Entering i=%d,j=%d.With val: %d\n",i,j,A[i][j]);
        auto memo = mem[i][j];
        if(get<0>(memo) != -1 && get<1>(memo) != -1){
                printf("Remembered for  i=%d,j=%d. ITS : fac3=%d,fac5=%d\n",i,j,get<0>
(memo),get<1>(memo));
                return memo;
        }
        //else if we havent reached we branch
        int fac3up = (A[i+1][j]%3 == 0)? 1 : 0;
        int fac5up = (A[i+1][j]%5 == 0)? 1: 0;

        int fac3right = (A[i][j+1]%3 == 0)? 1: 0;
        int fac5right = (A[i][j+1]%5 == 0)? 1: 0;

        auto bestup = defaultTuple(-999999,999999);
        auto bestright = defaultTuple(-999999,999999);
        if(i+1 < N){
                 bestup = path(i+1,j);
                 get<0>(bestup) = get<0>(bestup) + fac3up;
                 get<1>(bestup) = get<1>(bestup) + fac5up;
        }
        if(j+1 < N){
                 bestright = path(i,j+1);
                 get<0>(bestright) = get<0>(bestright) + fac3right;
                 get<1>(bestright) = get<1>(bestright) + fac5right;
        }
        int diffUp = get<0>(bestup) - get<1>(bestup);
        int diffRight = get<0>(bestright) - get<1>(bestright);

        auto which = (diffUp >= diffRight)?bestup:bestright;
        printf("At i=%d,j=%d val=%d\nWe get fac3=%d,fac5=%d\n",i,j,A[i][j],get<0>
(which),get<1>(which));
        mem[i][j] = which;
        return which;//TODO change that
}
int main(){

        fillMem();
        int firstFac3 = (A[0][0]%3 == 0) ?1:0;
        int firstFac5 = (A[0][0]%5 == 0) ?1:0;
        auto bestp = path(0,0);
        get<0>(bestp) = get<0>(bestp) + firstFac3;
        get<1>(bestp) = get<1>(bestp) + firstFac5;

        int best3 = get<0>(bestp);
        int best5= get<1>(bestp);
        printf("Our best3=%d,best5=%d\n",best3,best5);
        if(best3>best5){
                printf("Yes there is a monotonic path that visits more 3-entries than
5-entries\n");
                printf("With 3-entries=%d, and 5-entries=%d.\n",best3,best5);
        }


        return 0;
}
```

```
#include <iostream>
#include <math.h>
#include <algorithm>
#include <cstdio>
#define S  18
#define N  6
using namespace std;
int calcs[N+1][S+1];
int num[] = {0,1,2,4,5,6};
int sizeo = sizeof(num)/sizeof(num[0]);
int numsize = sizeof(num)/sizeof(num[0]);
void printArray(){
        printf("       ");
        for(int i = 0;i<S+1;i++){
                printf("[%5d]",i);
        }
        cout<<endl;
        for(int i = 0;i<N+1;i++){
                for(int j = 0;j<S+1;j++){
                        if(j==0)
                                printf("[%5d]",num[i]);
                        printf("[%5d]",calcs[i][j]);
                }
                cout << endl;
        }
}
bool build(){
        for(int i = 0; i < sizeo;i++){
                for(int j = 0;j<18;j++){
                        if(i == 0 || j == 0){
                                calcs[i][j] = 0;
                        }else if(num[i] > j){
                                calcs[i][j] = calcs[i-
1][j];
                        }else{
                                int top = calcs[i-1][j];
                                int possible =num[i]+calcs[i-1][j-num[i]];
                                if(top == possible){
                                        cout << "Two unequeal subsets can equal :
"<<top<<endl;

                                        calcs[i][j] = max(top,possible);
                                        //return true;
                                }else{
                                        calcs[i][j] = max(top,possible);
                                }
                        }
                }
        }
        return false;
}

int main(){
        build();
        printArray();
        return 0;
}
```

```
#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;
}
```

```
#include <iostream>
#include <cstdio>

using namespace std;
int curCount = 0;
int mode = 0;
int getMax(int arr[],int sizeo){//O(n)
        int mx = arr[0];
        for(int i = 1;i < sizeo;i++){
                if(arr[i] > mx)
                        mx = arr[i];
        }
        return mx;
}
void countSort(int arr[],int n, int exp){

        int output[n];//output array
        int i , count[10] = {0};
        //store occurences in our counting arrya
        for(i = 0;i<n;i++){//O(n)
                count[(arr[i]/exp)%10]++;
        }
        //add previous entries
        for(i = 1;i<10;i++){
                count[i] += count[i-1];
        }
        for(i = n-1;i>=0;i--){
                output[count[(arr[i]/exp)%10]-1] = arr[i];
                count[(arr[i]/exp)%10]--;
        }
        for(i = 0;i < n;i++){
                arr[i] = output[i];
        }
}

void radixSort(int arr[],int sizeo){
        //get max number for the largetst number of counting array
        int m = getMax(arr,sizeo);
        for (int exp = 1;m/exp > 0;exp*= 10){//loop through eveyr n umber
                countSort(arr,sizeo,exp);//exp is the current factor of 10 that divides
into digits
        }
}
int main(){

        int amount;
        printf("Please input the amount of numbers youll input: ");
        cin >> amount;
        int * arrayo = new int[amount];

        for(int i =0;i<amount;i++){
                cin >> arrayo[i];
        }

        int sizeo = amount;
        printf("This is our unsorted array : \n");
        for(int i =0;i < sizeo;i++){//output sorted array
                cout << arrayo[i] << " ";
        }
        radixSort(arrayo,sizeo);
        printf("\nThis is our sorted array : \n");

        int prevMode = -1;
        int prevCount =-1 ;
        int possmode = arrayo[0];
        int possCount = 1;
        for(int i = 0;i<sizeo;i++){
                cout << arrayo[i] << " ";
        }
        cout <<endl;
        for(int i =1;i < sizeo;i++){//output sorted array
                printf("prevMode = %d while arrao[%d] = %d\n",prevMode,i,arrayo[i]);
                if(possmode != arrayo[i] || (i ==sizeo-1 && possmode!= arrayo[i])){
                        if(prevCount < possCount){
                                prevMode = possmode;
                                prevCount = possCount;
                        }
                        possmode = arrayo[i];
                        possCount = 1;
                }
                if(possmode == arrayo[i])
                        possCount++;
        }
        printf("This is prevMode : %d\n",prevMode);
        //now that the array is sorted we analyze it
        cout<<endl;
        return 0;
}
```