**Classwork || Practice**

SELECTION SORT

Code:

/\*!

*\* Copyright (c) 2020 Abhishek Srivastava*

\*/

*#include* <*bits/stdc++.h*>

*using namespace* std;

*#define ll long long int*

*int* main(){

ios::sync\_with\_stdio(*false*);

*cin***.**tie(*0*);

vector<*int*> *v* ={*20*, *12*, *10*, *15*, *2*};

*int n* = *v***.**size();

*for*(*int i*=*0*; *i*<*n*; *i*++){

*int k*=*i*, *mn* = *v*[*i*];

*for*(*int j*=*i*+*1*; *j*<*n*; *j*++){

*if*(*mn* > *v*[*j*]){

*mn* = *v*[*j*]; *k* = *j*;

}

}

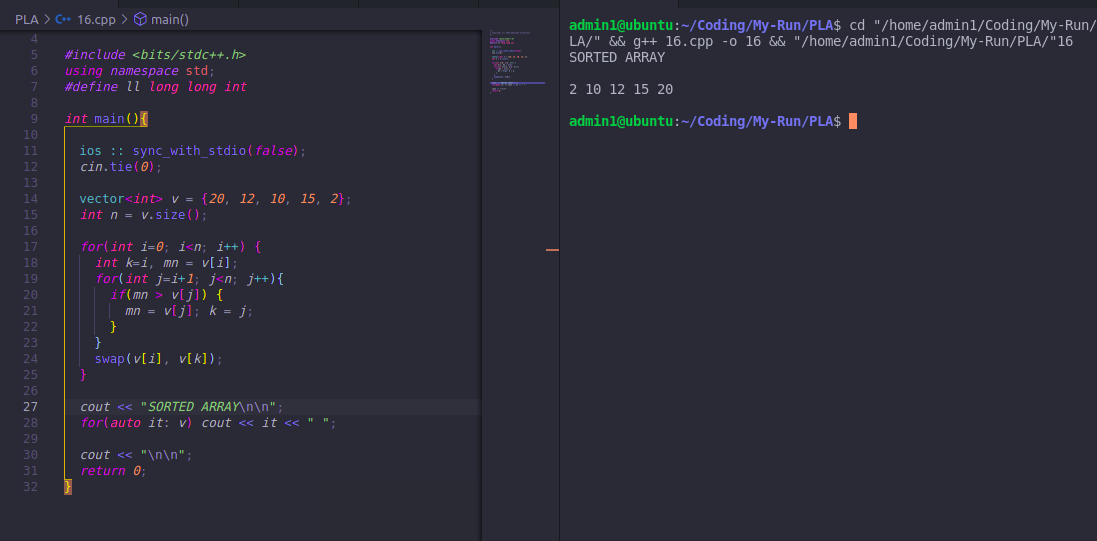
swap(*v*[*i*], *v*[*k*]);

}

*for*(*auto it*: *v*) *cout* << *it* <<"";

*return 0*;

}



MERGE SORT

Code

/\*!

*\* Copyright (c) 2020 Abhishek Srivastava*

\*/

*#include* <*bits/stdc++.h*>

*using namespace* std;

*#define ll long long int*

*void* merge(vector<*int*>& *v*, *int l*, *int m*, *int h*){

vector<*int*> *v1*, *v2*;

*int i*=*0*, *j*=*0*, *k*;

*int n1* = *m* - *l* + *1*;

*int n2* = *h* - *m*;

*for*(*i*=*0*; *i*<=*m*; *i*++)

*v1***.**push\_back(*v*[*i*]);

*for*(*i*=*m*+*1*; *i*<=*h*; *i*++)

*v2***.**push\_back(*v*[*i*]);

*i* = *0*, *j* = *0*, *k* = *l*;

*while*(*i*<*n1* && *j* < *n2*){

*if*(*v1*[*i*]<= *v2*[*j*]){

*v*[*k*]= *v1*[*i*]; *i*++;

}

*else* {

*v*[*k*]= *v2*[*j*]; *j*++;

}

*k*++;

}

*while*(*i* < *n1*){

*v*[*k*]= *v1*[*i*];

*i*++; *k*++;

}

*while*(*j* < *n2*){

*v*[*k*]= *v2*[*j*];

*j*++; *k*++;

}

}

*void* mergeSort(vector<*int*>& *v*, *int low*, *int high*){

*if*(*high* <= *low*) *return*;

*int mid* =(*low*+*high*)/*2*;

mergeSort(*v*, *low*, *mid*);

mergeSort(*v*, *mid*+*1*, *high*);

merge(*v*, *low*, *mid*, *high*);

}

*int* main(){

ios::sync\_with\_stdio(*false*);

*cin***.**tie(*0*);

vector<*int*> *v* ={*20*, *12*, *10*, *15*, *2*};

*int n* = *v***.**size();

mergeSort(*v*, *0*, *n*-*1*);

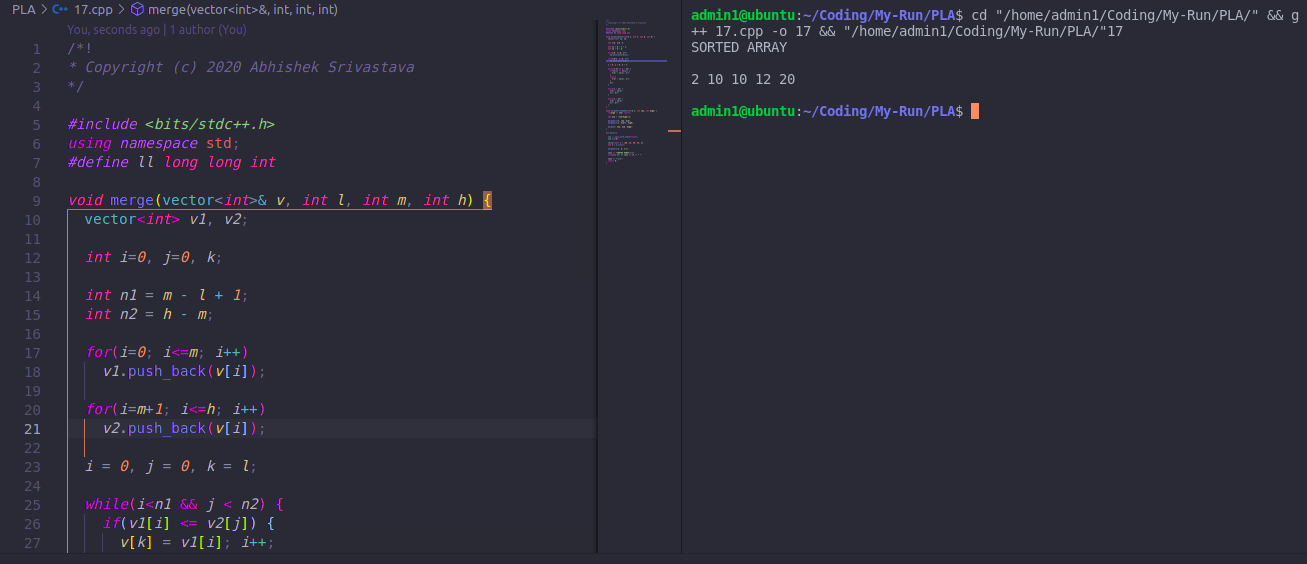
*cout* <<"*SORTED ARRAY*\n\n";

*for*(*auto it*: *v*) *cout* << *it* <<"";

*cout* <<"\n\n";

*return 0*;

}

****

SIMPLE SIEVE

Code

/\*!

*\* Copyright (c) 2020 Abhishek Srivastava*

\*/

*#include* <*bits/stdc++.h*>

*using namespace* std;

*#define ll long long int*

*#define MAX\_RANGE 1000*

*int primes*[*MAX\_RANGE*];

*void* solve(){

memset(*primes*, *1*,sizeof(*primes*));

*primes*[*0*]= *0*; *primes*[*1*]= *0*;

*for*(*int i*=*2*; *i*\**i*<*MAX\_RANGE*; *i*++){

*if*(*primes*[*i*])

*for*(*int j* = *i*\**i*; *j*<*MAX\_RANGE*; *j*+=*i*){

*primes*[*j*]= *0*;

}

}

}

*int* main(){

ios::sync\_with\_stdio(*false*);

*cin***.**tie(*0*);

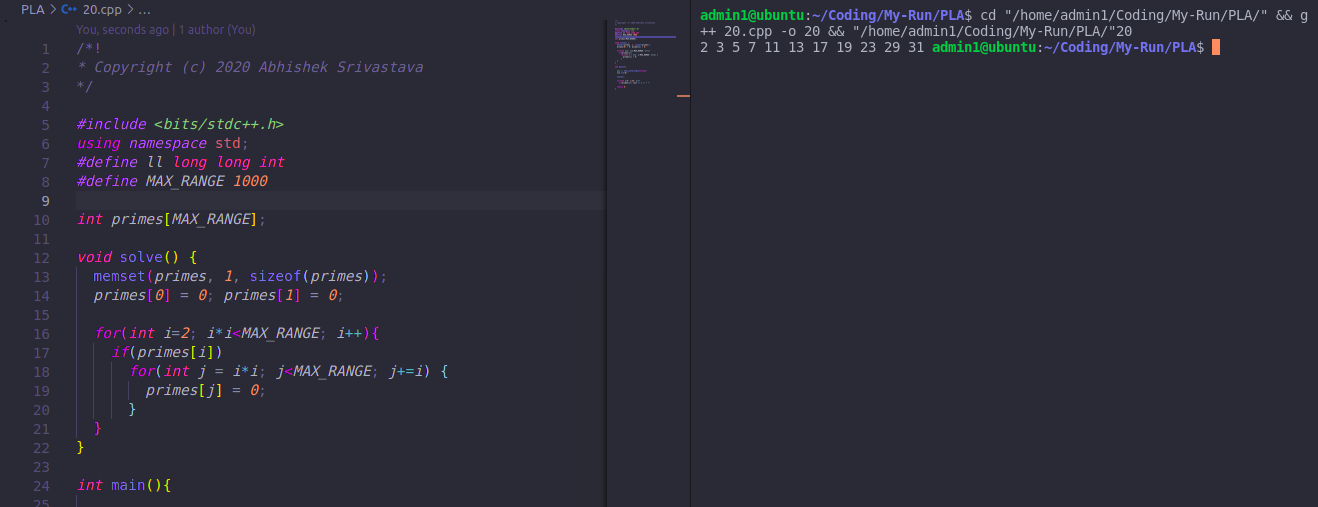
solve();

*for*(*int i*=*0*; *i*<*34*; *i*++)

*if*(*primes*[*i*]) *cout* << *i* <<"";

*return 0*;

}



SEGMENTED SIEVE

Code

*// Segmented Sieve*

#include<bits/stdc++.h>

using namespace std;

#define MAX 100001

vector<int> \_primes;

void sieve() {

vector<int> primes(MAX, 1);

for(int i=2; i\*i < MAX; i++) {

if(primes[i])

for(int j=i\*i; j < MAX; j+=i)

primes[j] = 0;

}

for(int i=2; i < MAX; i++)

if(primes[i]) \_primes.push\_back(i);

}

void primePrimes(long long L, long long R) {

vector<int> isPrime(R-L+1, 1);

for(int i=0; \_primes[i]\*\_primes[i] < R; i++) {

int currPrime = \_primes[i];

long long base = (L/currPrime)\*currPrime;

if(base<L) base += currPrime;

for(long long j=base; j<=R; j+=currPrime)

isPrime[j-L] = 1;

}

for(int i=0; i<=R-L; i++) {

if(isPrime[i])

cout << i+L << " ";

}

cout << "\n\n";

}

int main() {

sieve();

int t; cin >> t;

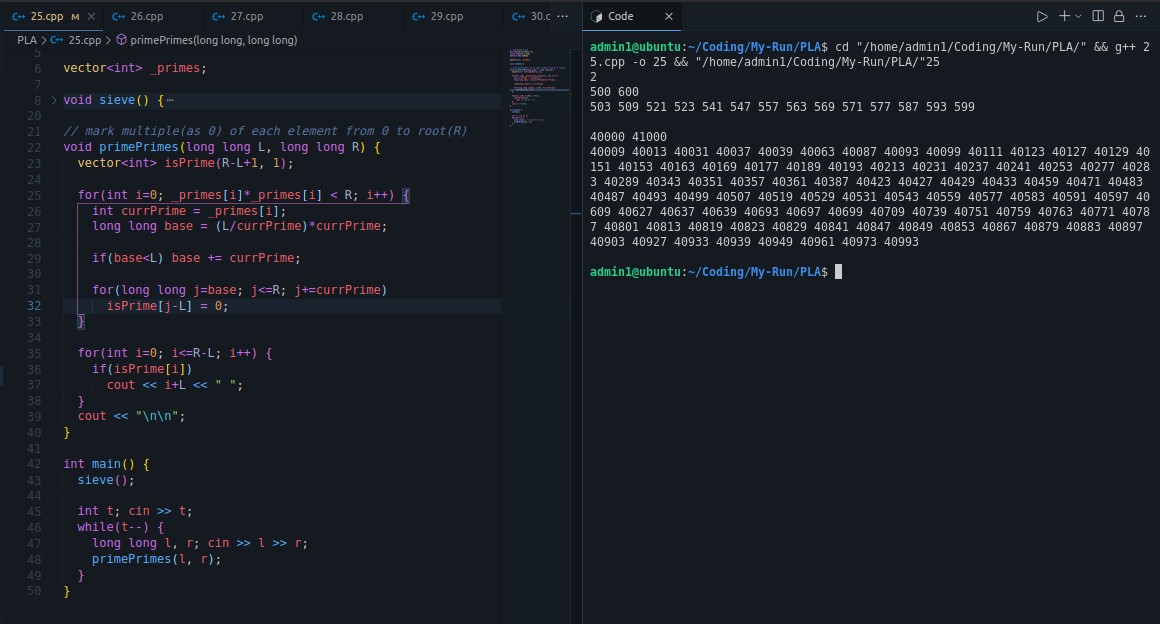
while(t--) {

long long l, r; cin >> l >> r;

primePrimes(l, r);

}

}



INCREMENTAL SIEVE

Code

#include <bits/stdc++.h>

using namespace std;

using vtype = long long;

struct pm {

vtype prime;

vtype multiple;

pm(vtype p, vtype m) :prime(p), multiple(m) {}

};

list<pm> primes = { {3, 3} };

int main() {

for (vtype x = 5; x < 1'000'000; x+=2) {

vtype limit = sqrt(x) + 1;

bool isprime = true;

for (auto &[p, m] : primes) {

if (p > limit) break;

while (m < x) {

m += p;

}

if (m == x) {

isprime = false;

break;

}

}

if (isprime) {

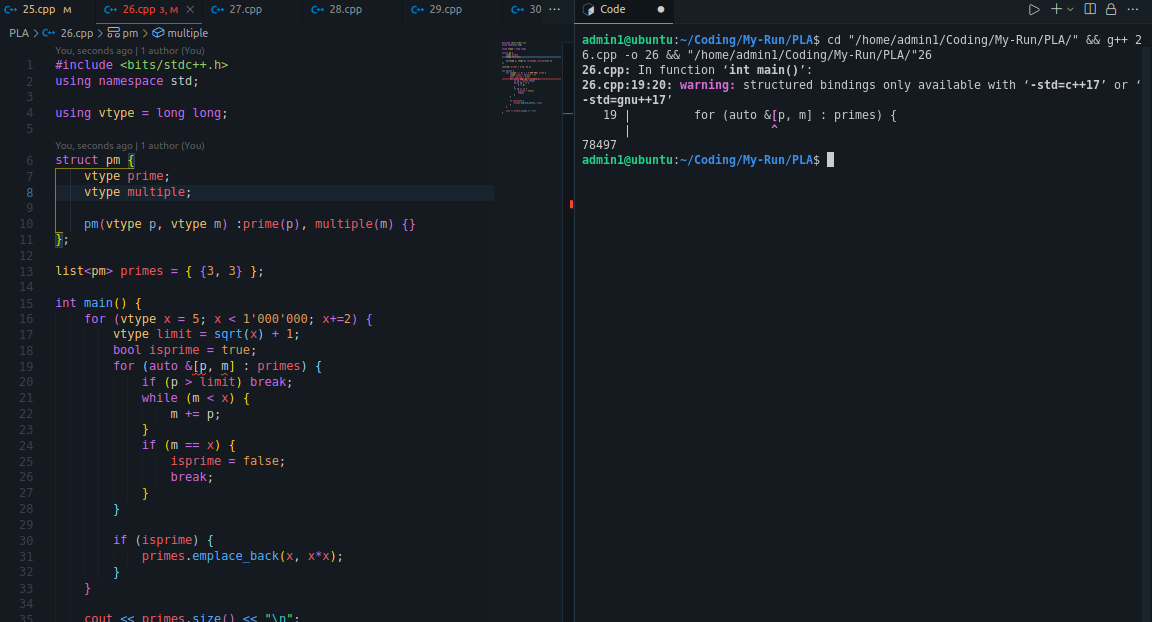
primes.emplace\_back(x, x\*x);

}

}

cout << primes.size() << "\n";

}



**THANK YOU**