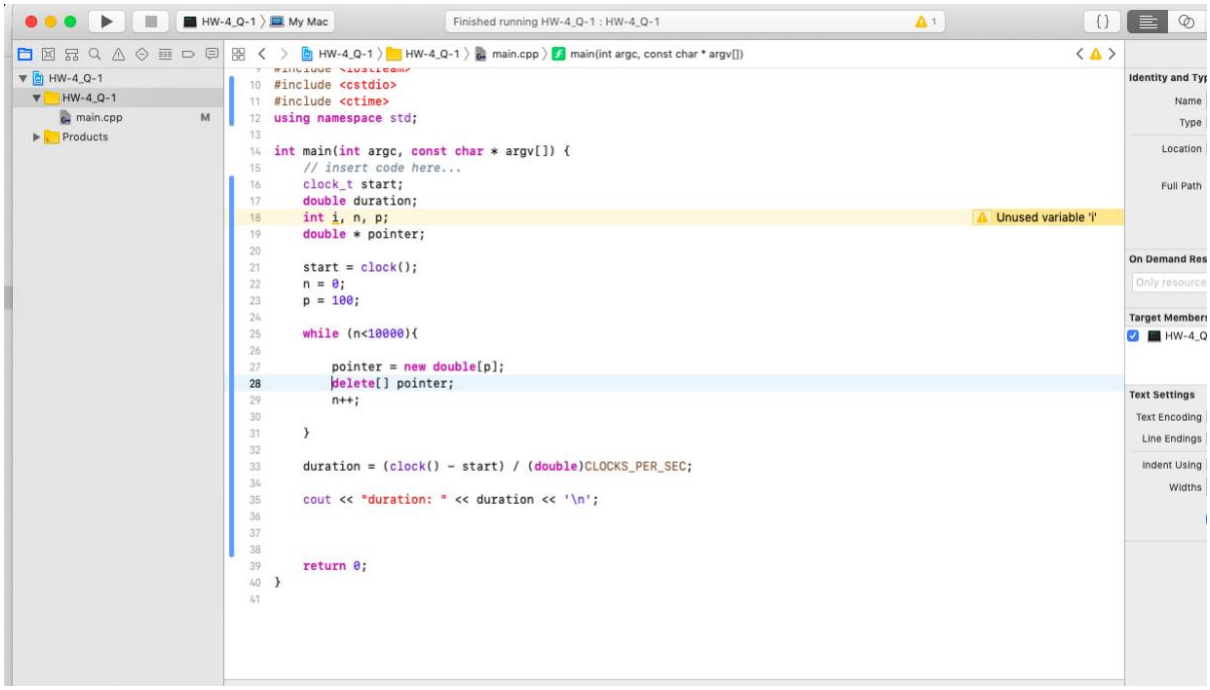
Q1:
Used the Following code for first question , as the array size increases. The code takes more time to execute.
P. ------ Time

100-------duration: 0.001576

1000----duration: 0.001856



P.t.o

Q2

1.) Its not like normal copy because if you copy auto_ptr p1 to auto_ptr p2 then all the control goes to p2 and p1 points to null. This example gives the idea :

```cpp
using namespace std;


class First
{
public:
    First(First& p);

    First(auto_ptr<int> x);
    void print(){
        cout<<*pimpl_;
    }
    auto_ptr<int> pimpl_;
};
First::First(auto_ptr<int> p)
{pimpl_ = p;
}
First::First(First& p)
{pimpl_ = p.pimpl_;
}



int main(int argc, const char * argv[]) {
    int x=9;
    int* a ;
    a=&x;

    auto_ptr<int> p3(a);
    First p1(p3);
    First p2(p1);
    //p1.print(); this will give null because p1 has only one auto_ptr which gave its control to p2
    p2.print();



    return 0;
}
```

P.t.o

2.) You don't need a destructor because auto pointer takes care of freeing that memory area once the encapsulated object goes out of scope & probably not needed anymore because auto pointer takes care of freeing that memory area once the encapsulated object goes out of scope.

Eg:

Hw-4-Q2 〉 main1.cpp 〉 C

```cpp
//
//  main1.cpp
//  Hw-4-Q2
//
//  Created by HIMANSHU KUMAR on 31/10/18.
//  Copyright © 2018 HIMANSHU KUMAR. All rights reserved.
//

#include <stdio.h>
#include <memory>
// file c.h
//
class C
{
public:
    C();
    /*...*/
private:
    class Simple; // forward declaration
    std::auto_ptr<Simple> pimpl_;
};

// file c.cpp
//
class C::Simple { /*...*/ };

C::C() : pimpl_( new Simple ) { }

//Now the destructor doesn't need to worry about deleting the pimpl_ pointer, because the auto_ptr will handle it automatically.
```