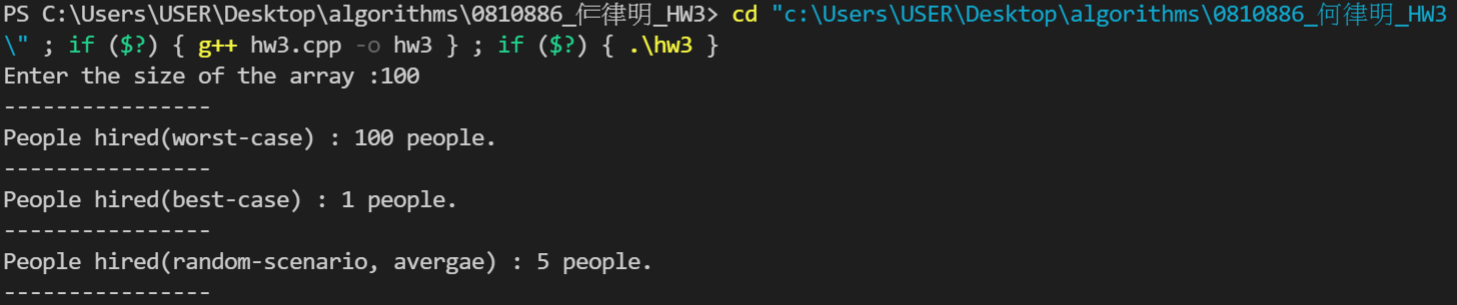
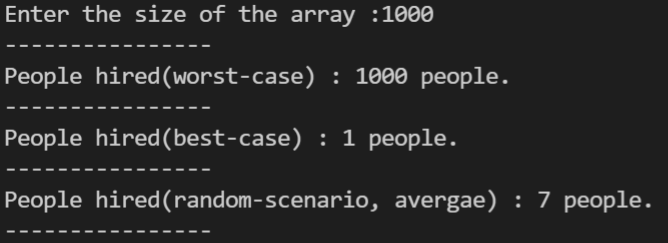
Result:

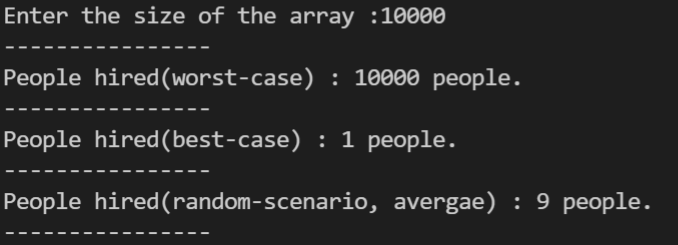
Size 100



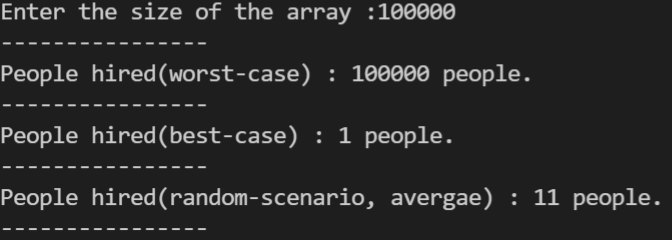
Size 1000



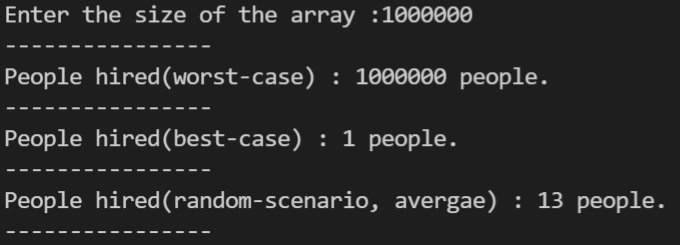
Size 10000



Size 100000



Size 1000000



Code Introduction

User interface and some initialization

    //User interface

    srand(time(0));

    int sz = -1;

    cout<<"Enter the size of the array :";

    cin>>sz;

    vector <int> worst;

    vector <int> best;

Worst case

    //Worst-case

    for(int i=0 ; i<sz ; ++i){

        worst.push\_back(i+1);

    }

    cout<<"----------------"<<endl;

    cout<<"People hired(worst-case) : "<<hire\_assistant(worst)<<" people."<<endl;

Best case

    //Best-case

    for(int i=0 ; i<sz ; ++i){

        best.push\_back(sz-i);

    }

    cout<<"----------------"<<endl;

    cout<<"People hired(best-case) : "

<<hire\_assistant(best)<<" people."<<endl;

Random permutation

    //Build the random vector

    int avg[20];

    int sum=0;

    for (int i=0 ; i<20 ; ++i){

        random\_in\_size(worst);

        avg[i] = hire\_assistant(worst);

    }

    for(int i=0 ; i<20 ; ++i){

        sum += avg[i];

    }

    sum = sum/10;

    cout<<"----------------"<<endl;

    cout<<"People hired(random-scenario) : "<<sum<<" people."<<endl;

    cout<<"----------------"<<endl;

總共取20次random的陣列輸出結果去做平均。

Code Functions

Random-in-place，把原本worst case的vector隨機打亂，共有n!種可能性。

void random\_in\_size(vector<int> &worst){

    int sz = worst.size();

    int part = (sz/20000);

    if(part < sz/20000.0)

        part++;

    for(int i=0 ; i<worst.size() ; ++i){

        int randompart = rand()%(part);

        int remain = sz - (part-1)\*20000;

        int randremain = rand()%(remain);

        int randomnumber = randompart\*20000 + randremain;

        //Swap worst[i] with worst[randomnumber]

        int temp  = worst[i];

        worst[i] = worst[randomnumber];

        worst[randomnumber] = temp;

    }

}

Hire-Assistant

int hire\_assistant(vector<int> v){

    int hire = 1;

    int best = 0;

    for(int i=1 ; i<v.size() ; ++i){

        //Interview candidate

        if(v[i] > v[best]){

            best = i;

            hire++;

        }

    }

    return hire;

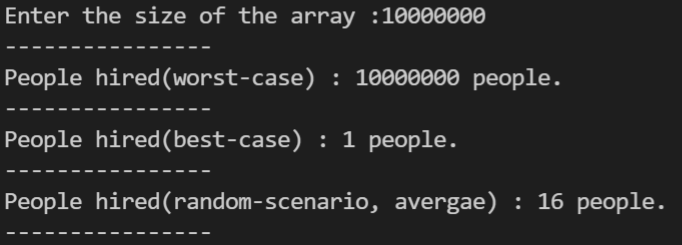
}

返回總共hire的人次，只要陣列中出現了更大的值就hire。Best代表最大的那個值的index，hire代表出現幾次新的值比best更大的次數。

Observation

根據code以及觀察不難發現當worst case時，hire 人次永遠都是input size的大小(後面的員工用遠比前面的員工好)，而當best case時，hire人次永遠都是1，不論input size 為多少，因為第一個已經是最大的了(最好的員工)。至於random permutation case，因為第i個員工被hired的機率為1/i(要在前i個中當作好的員工)，所以hire人次的期望值為1/i從1加到input size (n)，根據前面章節所學big O為log(n)，以下為我額外多做的input size:

Size 10000000



當input size 小於10000000時，的確符合log(n)的成長趨勢，我的code幾乎都是根據課本的pseudo code一模一樣去寫的，至於random的部分，我是用切割的方法去做，每一個切割大小為20000(小於RAND\_MAX的值) 先決定在哪一個區間再決定為區間內的哪一個值。由此一來就可以避免rand()有上限的問題。而最重要的random-scenario，每當input size增加10倍的時候，Expectation hire的人數都增加2人，完完全全符合課本所說的big O為ln(n)的成長趨勢。

5🡪7🡪9🡪11🡪13🡪16，中間只有最後100萬到1000萬時所加為3人，因為我用的average輸出為integer所以其實有可能會有小數點上的差距，但整體而言，不太影響log(n)的結果。

Conclusion

根據課本以及老師上課筆記最終random permutation的結果hire人次的期望值為ln(n)，在input size夠大的時候(但不超過rand()有效範圍)，隨著input size增加，hire人次的確符合ln(n)的成長趨勢(10倍🡪2~3人)。

而best case的最終值永遠為1。

Worst case的最終值則為input size n。

因為overall cost為interview cost c1乘上input size n加上hire cost c2乘上hire人次m，所以overall cost = c1\*n + c2\*m，c2>>c1，當n很大的時候，m為log(n)的數量級，所以說當n還算小的時候，overall cost主要為hire人次m來決定的，當n夠大的時候，m<<n，overall cost主要根據interview 人數n所影響。