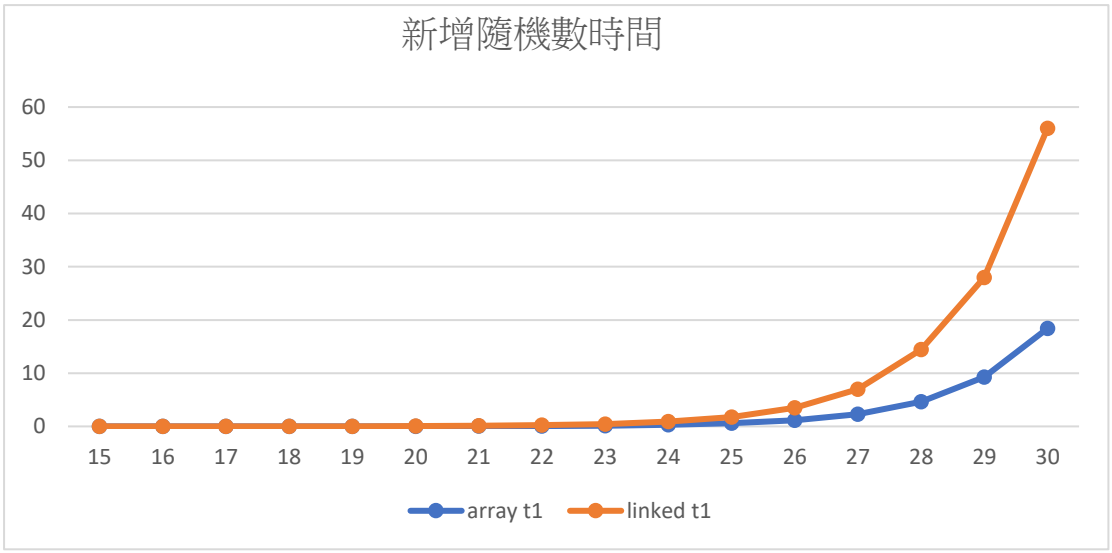
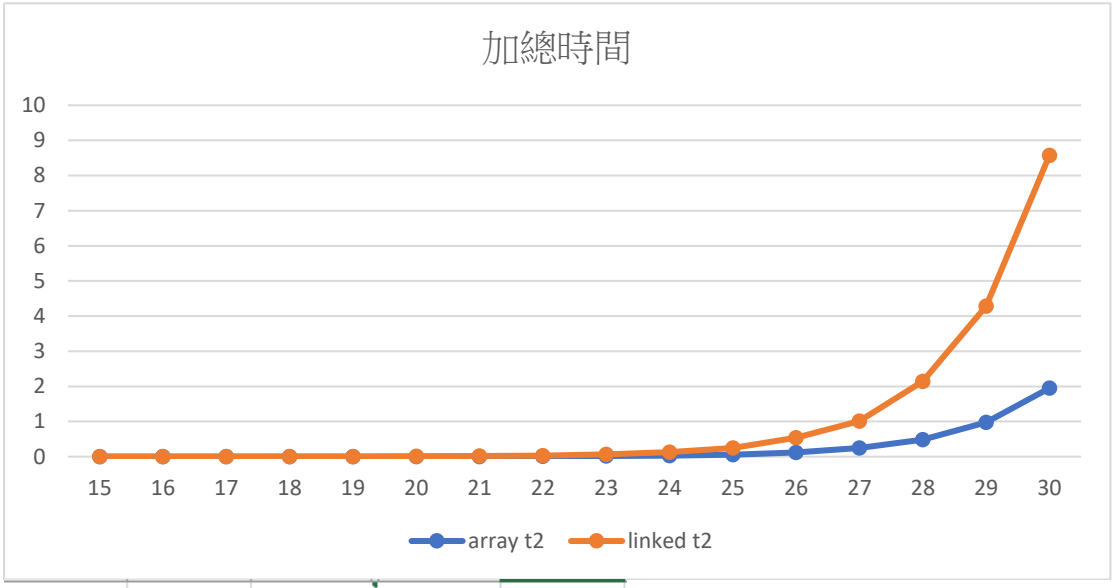
Y 軸時間單位秒 橫軸 k





	array t1	array t2	linked t1	linked t2
15	0.001	0	0.002	0
16	0.001	0	0.003	0.001
17	0.002	0	0.007	0.001
18	0.004	0.001	0.014	0.002
19	0.009	0.001	0.028	0.004
20	0.019	0.002	0.054	0.009
21	0.034	0.004	0.108	0.015
22	0.072	0.008	0.225	0.032
23	0.143	0.016	0.435	0.065
24	0.292	0.032	0.884	0.127
25	0.59	0.059	1.735	0.251
26	1.155	0.116	3.475	0.534
27	2.294	0.243	7.006	1.012
28	4.662	0.484	14.426	2.143
29	9.271	0.974	x	X
30	X	X	x	X

左圖是實際測出來的數據，可以看出當 k+1 時需要的時間大概會是 k 時的 2 倍，因此測不出來的 x 數據皆以前一項的兩倍來作圖。

實驗結果兩個時間都是 dynamic array 比 linked list 更快，dynamic array 加總會比較快是因為 cache 會把陣列附近的一起拿出來 linked list 位置不連續所以會比較慢，另外這次 dynamic array 的程式碼用的是只要陣列大小不夠就開兩倍大小的陣列所以速度會

差距更大而 linked list 每次新增數字就需要跳去找下一個記憶體位置。

2.程式碼來源:

Dynamic array

https://algorithmtutor.com/Data-Structures/Basic/Dynamic-Arrays/

陣列起始大小為 1 且在陣列滿了以後新增數會開一個兩倍大的陣列並複製前面的值，所以是 dynamic array。

Linked list

https://www.codesdope.com/blog/article/linked-list-traversal-using-loop-and-recursion-in-/

每一個資料之間以指標的方法指到下一個資料，所以是 linked list

3.心得:第一次做這種實際的實驗感覺還滿有趣的，以前都只是聽上課講的來知道不同資料結構之間的優劣勢而已，實際寫程式來測試以後印象也比較深刻，感覺寫實驗的程式碼比原本想像中的還要複雜又麻煩，可能因為第一次寫的關係，找到網路上合適的 dynamic array 和 linked list 就花了一些時間，然後還會出現一些原本沒想過的問題在實際做了以後才能夠發現，像是在測試第二個實驗在加總隨機數的時候出現數字 overflow 的問題，後來才想到把隨機數都%100 來解決

疑問:在實驗時用 dynamic array 和 linked list 前者可以做到 k=29 但 linked list 試了 3 次只能做到 k=28，只要用 29 以上跑 linked list 的程式碼電腦就會卡住，自己怎麼想都覺得 linked list 能跑的數量應該要比 array 來的多才對，因為 linked list 儲存數字的記憶體可以分散開來而 array 需要連續的記憶體，就連必須連續的記憶體位置數量都夠用了，沒有限制連續的情況下記憶體應該一定比較多才對，因此不知道為什麼 linked list 能做到的資料大小比 dynamic array 更小

實驗程式碼: dynamic array

```
#include  <iostream>
#include <time.h>
#include <stdlib.h>
using namespace std;

template <class T>
class DynamicArray {
private:
    T *list;
    int max_size;
    int length;
public:
    DynamicArray() {
        // initially allocate a single memory block
        max_size = 1;
        list = new T[max_size];
        length = 0;
    }

    // insert a new item to the end of the list
    void add(T item) {
        if (isFull()) {
            // create temporary list with double size
            max_size = 2 * max_size;
            T *temp_list = new T[2* max_size];

            // move all the elements to the temporary list
            for (int i = 0; i < length; i++) {
                temp_list[i] = list[i];
            }

            // delete the old list
            delete [] list;

            // rename temp list
            list = temp_list;
        }

        // add the new item at the end of the list
        list[length] = item;
        length++;
    }

    int addList() {
        int total=0;
        for (int i = 0; i < length; i++) {
            total=total+list[i];
        }
        return total;
    }

    // check if the list is full
    bool isFull() {
        return length == max_size;
    }
```

```
    ~DynamicArray() {
        delete [] list;
    }

     int addList() {
        int total=0;
        for (int i = 0; i < length; i++) {
            total=total+list[i];
        }
        return total;
    }

};

int main() {
    DynamicArray<int> list;
    int n,k=1;
    cin>>n;
    for(int i=0;i<n;++i)
    {k=k*2;}
    srand(100);
    double start,end;
    start=clock();
    for(int i=0;i<k;++i)
    {list.add(rand()%100);}
    end=clock();
    cout << endl << "1 進行運算所花費的時間" << (end-start) / CLOCKS_PER_SEC << " S" << endl;
    double start2,end2;
    start2=clock();

    cout<<list.addList()<<endl;
    end2=clock();

    cout << endl << "2 進行運算所花費的時間" << (end2-start2) / CLOCKS_PER_SEC << " S" << endl;
    return 0;
}
```

實驗程式碼:linked list

```
#include <iostream>
#include <time.h>
#include <stdlib.h>
using namespace std;

struct node
{
    int data;
    node *next;
};
class linked_list
{
private:
    node *head,*tail;
public:
    linked_list()
    {
        head = NULL;
        tail = NULL;
    }
```

```
    void add_node(int n)
    {
        node *tmp = new node;
        tmp->data = n;
        tmp->next = NULL;

        if(head == NULL)
        {
            head = tmp;
            tail = tmp;
        }
        else
        {
            tail->next = tmp;
            tail = tail->next;
        }
    }

    node* gethead()
    {
        return head;
    }

    void count(node *head,int total)
    {if(head == NULL)
        {
        }
        else
        {
            total=total+head->data;
            count(head->next,total);
        }}
};
int main()
{   srand(100);
    linked_list a;
    int n,k=1;
    cin>>n;
    for(int i=0;i<n;++i)
    {k=k*2;}
    double START,END;
    START=clock();
    for(int i=0;i<k;++i)
    {a.add_node(rand()%100);}
    END=clock();
    cout << endl << "1 進行運算所花費的時間" << (END - START) / CLOCKS_PER_SEC << " S" << endl;
    double START2,END2;
    int total=0;
    START2=clock();
    a.count(a.gethead(),total);

    END2=clock();

    cout << endl << "2 進行運算所花費的時間" << (END2 - START2) / CLOCKS_PER_SEC << " S" << endl
    return 0;
}
```