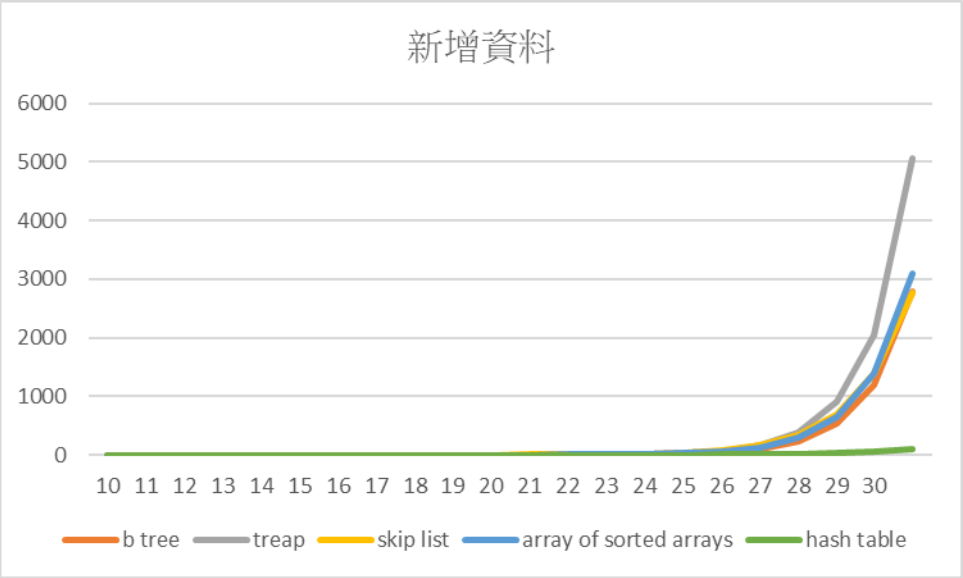
新增 2^k 筆資料:



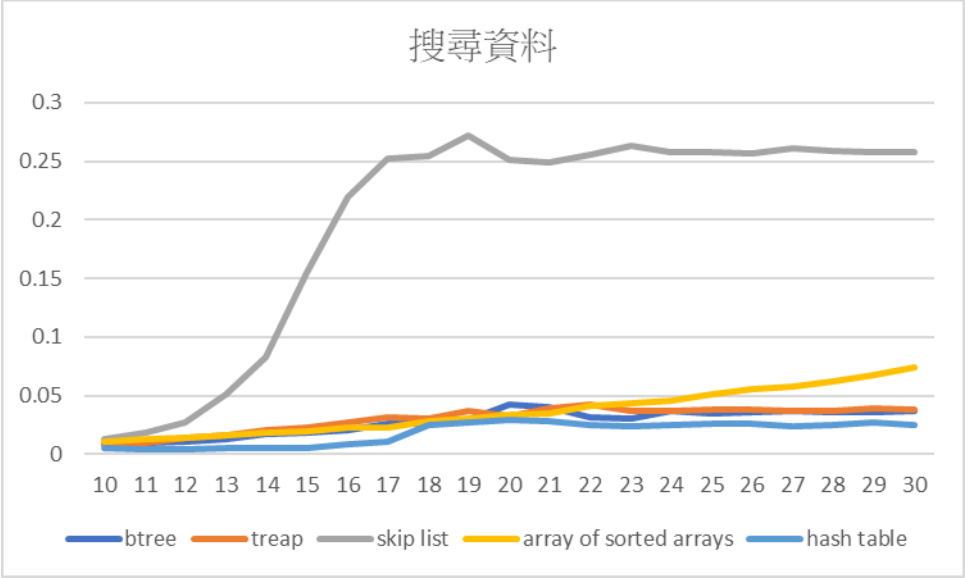
add	B tree	treap	skip list	array of so	hash table
10	0	0	0.001	0.001	0
11	0.001	0	0.001	0.002	0
12	0.001	0.001	0.002	0.003	0.001
13	0.001	0.002	0.004	0.006	0.001
14	0.003	0.005	0.011	0.013	0.001
15	0.008	0.011	0.035	0.029	0.003
16	0.018	0.027	0.102	0.059	0.006
17	0.042	0.062	0.259	0.127	0.013
18	0.096	0.137	0.627	0.283	0.026
19	0.274	0.353	1.399	0.631	0.052
20	0.614	0.819	2.661	1.363	0.101
21	1.68	2.158	5.215	2.881	0.203
22	3.908	5.027	10.805	6.093	0.412
23	8.774	10.878	21.603	12.821	0.839
24	19.952	25.156	43.883	27.397	1.698
25	45.272	62.283	85.075	60.252	3.318
26	100.763	174.469	171.379	132.126	6.605
27	230	382.842	347.17	292.9	13.752
28	541	918.7	691	641	26.411
29	1205	2043	1388	1401	53.878
30	2781	5057	2769	3107	110.12

B tree 資料數*2 時間大約變 2.2-2.3 倍=>O(logn)

Skip list 和 hash table 資料數*2 時間大約變 2 倍=> O(1)

Treap 資料數*2 時間大約變 2.2-2.4 倍=>O(logn)

Array of sorted array 資料數*2 時間大約變 2.1-2.2 倍=>O(logn)



add	B tree	treap	skip list	array of so	hash table
10	0.008	0.009	0.013	0.011	0.005
11	0.01	0.01	0.018	0.013	0.004
12	0.011	0.014	0.027	0.014	0.004
13	0.013	0.016	0.054	0.016	0.005
14	0.017	0.021	0.083	0.018	0.005
15	0.018	0.023	0.155	0.019	0.005
16	0.021	0.027	0.219	0.023	0.008
17	0.026	0.032	0.252	0.023	0.011
18	0.026	0.03	0.255	0.028	0.025
19	0.029	0.037	0.272	0.03	0.026
20	0.042	0.33	0.251	0.034	0.027
21	0.04	0.039	0.249	0.035	0.024
22	0.031	0.042	0.256	0.041	0.023
23	0.03	0.037	0.263	0.043	0.025
24	0.037	0.037	0.258	0.046	0.025
25	0.035	0.038	0.258	0.051	0.026
26	0.036	0.038	0.257	0.055	0.027
27	0.037	0.037	0.261	0.058	0.025
28	0.036	0.037	0.259	0.062	0.024
29	0.036	0.039	0.258	0.068	0.024
30	0.037	0.038	0.258	0.074	0.025

令人疑惑的地方是除了 array of sorted array 以外搜尋資料的時間都會漸漸趨近某個值，推測原因是 data 沒有到一定的數量時容易去搜尋不在資料結構裡的 data 而需要檢查完整的資料結構，但 data 夠齊全以後在搜尋時可能更快找到而省略掉後續的檢查，這種問題發生的原因可能是 data 選取的範圍(int)太小或是 rand()產生的數字不夠隨機

Array of sorted array 在這次的實驗下因為數據都是 2 的 k 次方且我在實作 search 的時候會先檢查如果 array 是空的就跳過 binary search 來減少時間，所以每個實驗到最後其實都會剩下一個最大的 array 在做 binary search 因此結果也大概是 O(log(n)) ，如果可以更隨機的取實驗的 data 數應該可以達到理論上的時間複雜

度 O((logn)^2)

array of sorted array 的實作:

```
#include <iostream>
#include <time.h>
#include <vector>
#include <cmath>
using namespace std;
vector <int* >v;
int pow(int a) //計算2^a
{
    int x=1;
    for(int i=0;i<a;++i)
    {x=x*2;}
    return x;
}
void initialize(int num_of_array)//用vector開好需要的所有array
{
    int k=1;
    for(int i=0;i<num_of_array;++i)
    {
        int *arr=new int[k];
        for(int j=0;j<k;++j)
        {
            *(arr+j)=-1;
        }
        k=k*2;
        v.push_back(arr);
    }
}
void resetvalue(int n)//merge兩個array後用來將用過的array改回空的
{
    for(int i=0;i<=n;++i)
    {
        int *arr=new int[pow(i)];
        for(int j=0;j<pow(i);++j)
        {
            *(arr+j)=-1;
        }

        v[i]=arr;
    }
}
void printarray(int num_of_array)//測試array
{
    int k=1;
    for(int i=0;i<num_of_array;++i)
    {
        for(int j=0;j<k;++j)
        cout<<*(v[i]+j)<<" ";
        cout<<endl;
        k=k*2;
    }
}
bool is_full(int array_num)//檢查array是否滿了
{
    if(*(v[array_num])==-1)return false;
    return true;
}
```

```
58  int* merge_array(int array_num,int *array)//將一個array和vector裡的array merge後回傳結果
59  {
60      int *keep=new int[pow(array_num+1)];
61      int now=0,count1=0,count2=0;//count1=>array count2=>v[array_num]
62      for(int i=0;i<pow(array_num+1);++i)//merge的過程
63      {
64          if(count1>=pow(array_num))
65          {
66              *(keep+now)=*(v[array_num]+count2);
67              ++count2;
68              ++now;
69          }
70          else if(count2>=pow(array_num))
71          {
72              *(keep+now)=*(array+count1);
73              ++count1;
74              ++now;
75          }
76          else if(*(v[array_num]+count2)>=*(array+count1))
77          {
78              *(keep+now)=*(v[array_num]+count2);
79              ++count2;
80              ++now;
81          }
82          else
83          {
84              *(keep+now)=*(array+count1);
85              ++count1;
86              ++now;
87          }
88      }
89      return keep;
90  }
91
92  int binary_search(int* data, int size ,int key)//binary search
93  {
94      if(*data==-1)return -1;
95      int low = 0;
96      int high = size-1;
97      while (low <= high) {
98          int mid = int((low + high) / 2);
99          if (key == data[mid])
100             return mid;
101         else if (key < data[mid])
102             low = mid + 1;
103         else
104             high = mid - 1;
105     }
106     return -1;
107 }
```

```
108 void searchItem(int x,int t)//若array是空的回傳-1 若不是則做binary search找指定的值
109 {
110     for(int i=0;i<t;++i)
111     {
112         int result=binary_search(v[i],pow(i),x);
113         if(result!=-1) {break;}
114     }
115 }
```

```cpp
int main()
{
    int a;
    cin>>a;
    int b=1;
    for(int k=0;k<a;++k)
    {
        b=b*2;
    }
    srand(100);
    initialize(a+1);//初始化空的資料結構
    int count=0;
    double start,end;
    start=clock();
    for(int i=0;i<b;++i)
    {

        if(count%2==0) //若是奇數次的input會放在第0個array 不會merge
        {
            *v[0]=abs(rand());
        }
        else //若是偶數次的input至少會merge一次所以要跑迴圈直到找到空的array
        {
            int k=0;
            int *keep=new int[pow(k)];
            *keep=abs(rand());
            while(1)
            {
                int *keep2=new int[pow(k+1)];
                keep2=merge_array(k,keep);//用keep2暫存merge的結果
                ++k;
                if(is_full(k))//如果下一個array滿了繼續往下做merge
                {
                    keep=new int[pow(k)];
                    keep=keep2;
                }
                else //直到找到沒有滿的array把值放進去vector
                {
                    v[k]=keep2;
                    resetvalue(k-1);//放入後須將前面的array改回空的
                    break;
                }
            }
        }
        ++count;
    }
    end=clock();
    cout<<"執行時間"<<(end-start)/ CLOCKS_PER_SEC <<"s"<<endl;
    start=clock();
    for(int i=0;i<100000;++i)//找十萬筆資料
    {
        searchItem(rand(),a);
    }
    end=clock();
    cout<<"執行時間"<<(end-start)/ CLOCKS_PER_SEC <<"s"<<endl;
    //printarray(a+1); 可檢查array
    return 0;
}
```

實驗程式碼:

B tree: https://onlinegdb.com/5CYYVFDdb

Treap: https://onlinegdb.com/A0lHk6yBo

Skip list: https://onlinegdb.com/jjKviJFmY

Hash table: https://onlinegdb.com/V8ALTLTh

Array of sorted array: https://onlinegdb.com/lX4FKG69N

輸入 k 值，會輸出 n=2^k 的實驗結果(新增資料的時間和搜尋資料的時間)

Ex:b tree 取得兩組實驗數據時間的程式碼部分:

```
int a;
cin>>a;
int b=1;
for(int k=0;k<a;++k)
{
    b=b*2;
}
//cout<<b;
srand(100);
BTree t(3);//不同的建立資料結構函式
double start,end;

start=clock();
for(int i=0;i<b;++i)
{
    t.insert(rand()); //insert b筆資料
}

end=clock();
//printArray(arr, N);
cout<<"執行時間"<<(end-start)/ CLOCKS_PER_SEC <<"s"<<endl;
start=clock();
for(int i=0;i<100000;++i)
{
    t.search(rand()); //search十萬筆資料 不同的search函式
 }
end=clock();
cout<<"執行時間"<<(end-start)/ CLOCKS_PER_SEC <<"s"<<endl;
```

資料結構的程式碼來源:

Btree: https://www.programiz.com/dsa/b-tree

Treap: https://www.tutorialspoint.com/cplusplus-program-to-implement-treap

Skip list: https://www.sanfoundry.com/cpp-program-implement-skip-list/

Hash table: https://www.programiz.com/dsa/hash-table

心得:實作資料結構比預想的難上許多，資料結構在理論上的部分要理解還算滿容易的但真的要自己實作時就會跑出很多細節的問題，可能因為我寫程式的能力不是很好，花了兩三個小時才把程式寫出來。不只實作，真的去做實驗出來的結果也會和上課所學到的理論有不少差異，要能找到原因也是件相當困難的事，但在未來工作或實驗上一定會有更多這樣的問題，所以能有一次作業的練習機會讓自己知道哪裡還不足需要更努力的地方還滿好的。