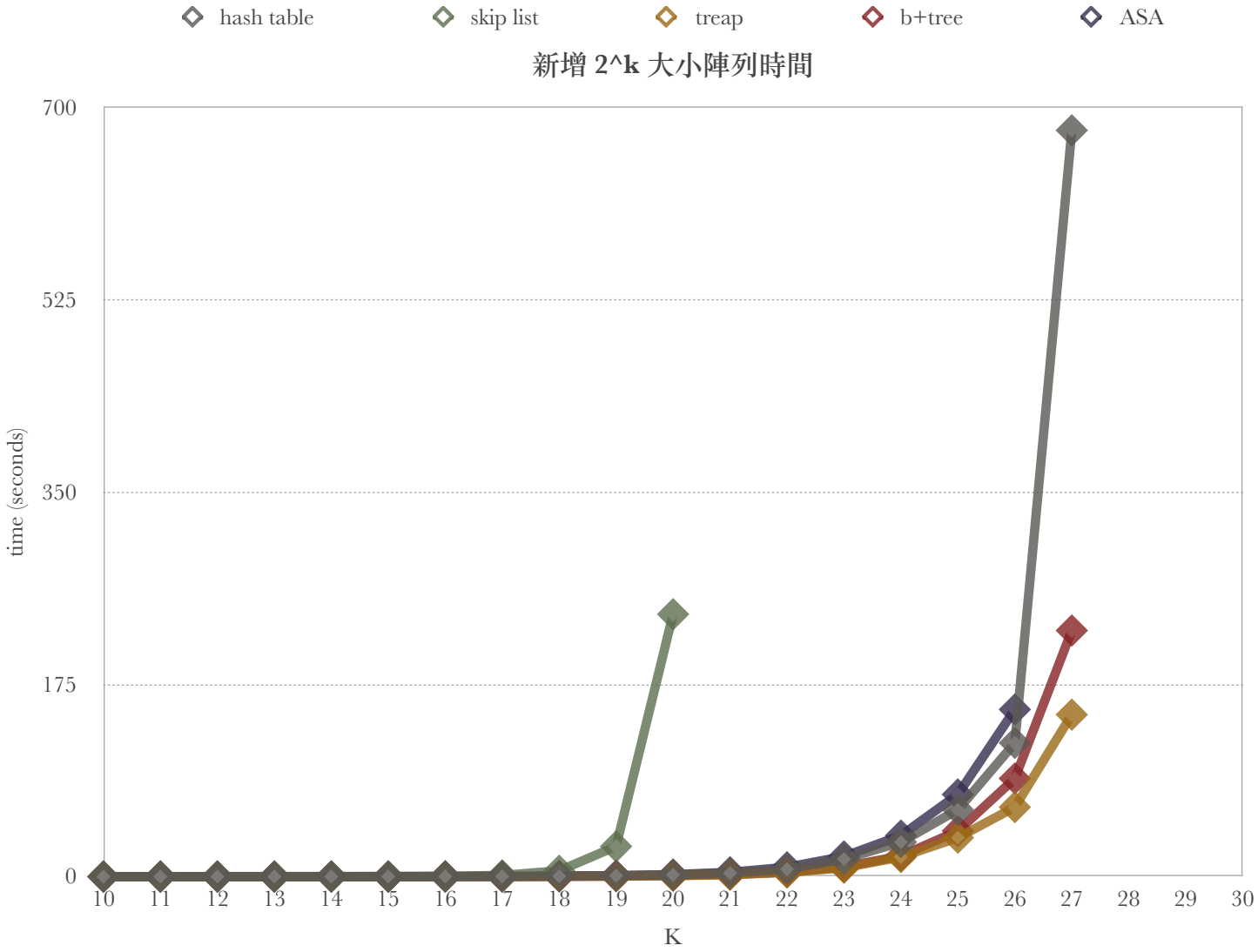
比較 B+tree, treap, skip list, array of sorted arrays and hash table

張芫禎
資科三109102027

折線圖：新增 n=2^k 大小陣列的時間

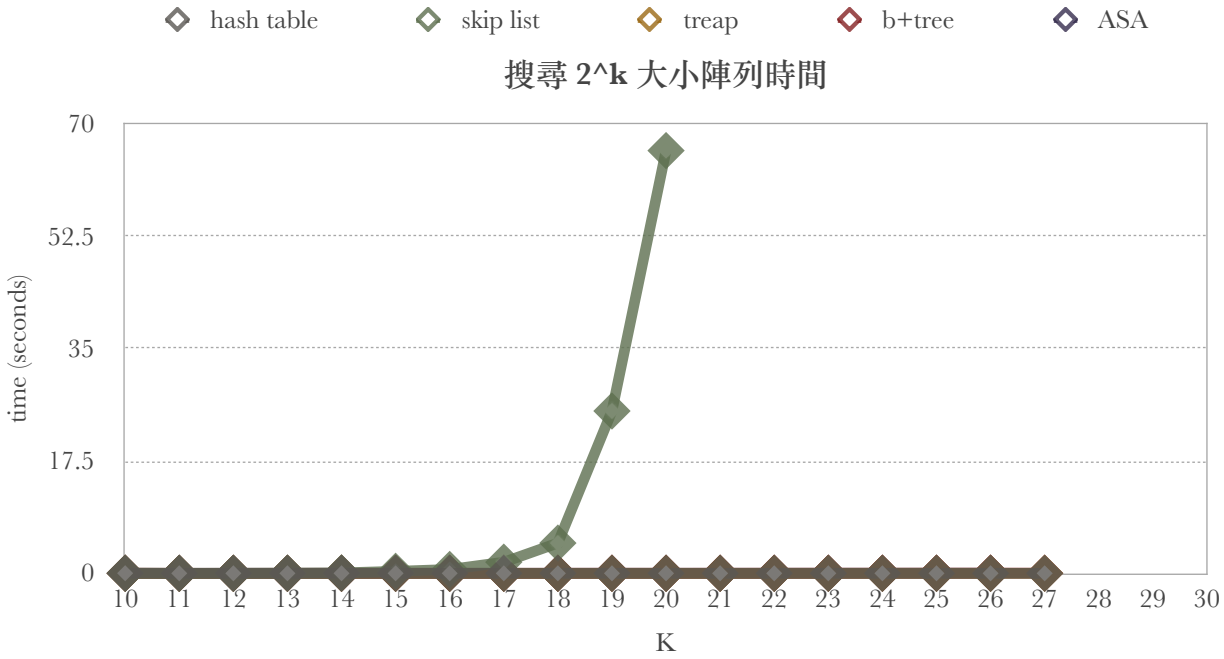


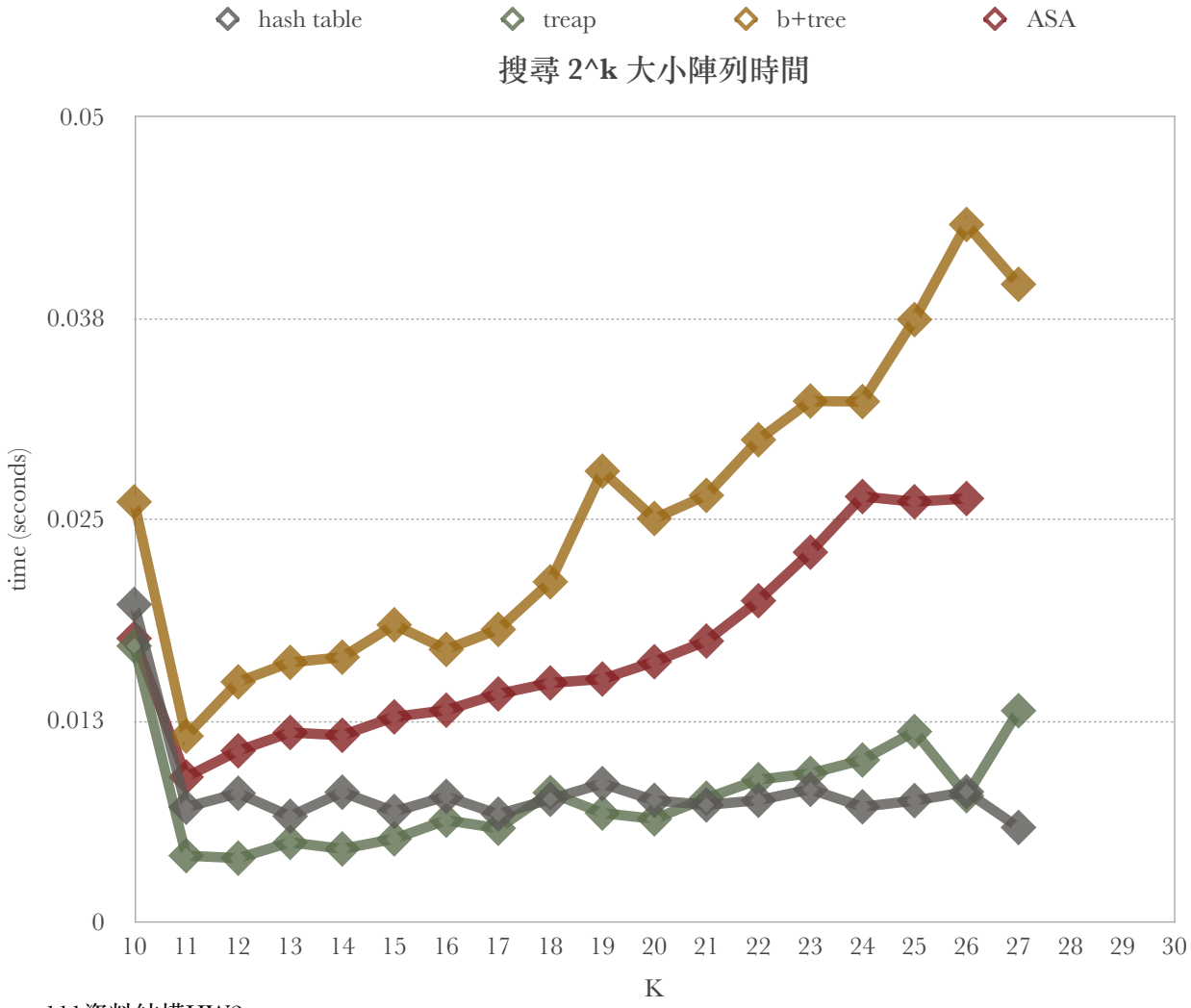
k	Execution time in seconds				
	Hash table	Skip list	treap	b+tree	Array of sorted arrays
10	0.001269	0.0007976	0.0008602	0.0012364	0.0012122
11	0.0021528	0.0014406	0.0011904	0.0012854	0.0026646
12	0.0041626	0.0031666	0.0024046	0.0025726	0.0057112
13	0.008443	0.0075732	0.0048496	0.0053602	0.011609
14	0.0169716	0.019865	0.009958	0.0103688	0.0231994
15	0.0358442	0.0677328	0.021225	0.0231658	0.0481674
16	0.0782438	0.265311	0.0437624	0.0472852	0.1022
17	0.166326	1.12469	0.0868494	0.0970488	0.205727
18	0.377928	5.30766	0.184626	0.213503	0.443275
19	0.80588	27.5154	0.380562	0.447614	0.903745
20	1.65571	239.014	0.793597	0.952765	1.87903
21	3.28642	314.34693576	1.70733	1.92857	3.93337
22	6.76178	1062.4926428688	3.75788	4.10978	8.52093
23	15.9276	2124.9852857376	8.01302	8.83297	18.3767
24	31.403	4249.9705714752	17.795	18.7927	37.3453
25	60.8925	8499.9411429504	35.4211	41.5792	74.8279
26	121.746	16999.8822859008	63.2678	89.3943	152.243
27	679.823	33999.7645718016	147.529	224.161	304.486
28	486.984	67999.5291436032	260.2955	448.322	608.972
29	973.968	135999.058287206	520.591	896.644	1217.944
30	1947.936	271998.116574412	1041.182	1793.288	2435.888

- 紅框代表表現異常，例如：hash table 的平均成長斜率約為 2 ，在 k = 27 時約為 5.5。
- 黑框代表實驗次數不足，紅字為預測的執行時間。
- 預測方式為將第 i 次的執行時間乘以平均成長斜率得到 i+1 的執行時間。

折線圖：搜尋 n=2^k 大小陣列的時間

With and without skip list





k	Execution time in seconds				
	Hash table	Skip list	treap	b+tree	Array of sorted arrays
10	0.0197316	0.018026	0.017154	0.0261104	0.0176198
11	0.0071274	0.0122386	0.0041134	0.011551	0.0090048
12	0.0079932	0.0195246	0.0039598	0.0149198	0.0106324
13	0.0065672	0.0373602	0.0049438	0.0161308	0.0117712
14	0.0080586	0.0745798	0.0045336	0.0164552	0.0115978
15	0.0068596	0.323122	0.0051598	0.0184672	0.0127414
16	0.007795	0.610236	0.0063208	0.016937	0.013159
17	0.0066916	1.74604	0.0058176	0.0181796	0.0141692
18	0.007619	4.72038	0.008032	0.0211412	0.0148522
19	0.0085916	25.3004	0.0067624	0.0280224	0.0150974
20	0.0075532	65.8857	0.0064002	0.0250796	0.0161708
21	0.007267	37.76304	0.0076836	0.0265146	0.0174576
22	0.0075574	75.52608	0.0088366	0.0299728	0.0199644
23	0.0082364	151.05216	0.0092286	0.0323792	0.0229812
24	0.0071518	302.10432	0.0100612	0.0323484	0.0264432
25	0.007536	604.20864	0.0118284	0.0374472	0.0261186
26	0.0080822	1208.41728	0.007842	0.043359	0.0263198
27	0.0058666	2416.83456	0.013126	0.0396426	0.016829
28	0.008127	4833.66912	0.014572	0.025259	0.016829
29	0.008127	9667.33824	0.0082303	0.025259	0.016829
30	0.008127	19334.67648	0.0082303	0.025259	0.016829

• Hash table, treap, b+tree and array of sorted arrays 的成長斜率都約為 1，時間複雜度應該為常數時間，因此預測方式為取現有實驗結果的平均。

• Skip list 去掉極端值後的執行時間的成長斜率約為 2，因此預測方式為將第 i 次的執行時間乘以 2 得到 i+1 的執行時間。

亂數生成程式碼 ：

```cpp
#include <iostream>
#include <fstream>
#include <random>

using namespace std;

int main(){

    ofstream random("random.random", ios::out);

    for(int i=10; i<=30; ++i){
        int size = pow(2, i);

        random_device seed;
        default_random_engine generator(seed());
        uniform_int_distribution<int> uniform(1, pow(2, 30));

        for(int i=0; i<size; ++i){
            random << uniform(generator) << endl;
        }
        random << endl;
    }

    random.close();
    return 0;
}
```

```cpp
#include <iostream>
#include <fstream>
#include <random>

using namespace std;

int main(){

    ofstream random("random.search", ios::out);

    random_device seed;
    default_random_engine generator(seed());
    uniform_int_distribution<int> uniform(1, pow(2, 30));

    for(int i=0; i<100000; ++i){
        random << uniform(generator) << endl;
    }

    random.close();
    return 0;
}
```

B+tree 程式碼 ：from GitHub

```cpp
#include <iostream>
#include <fstream>
#include <cmath>
#include "SourceCode.h"

using namespace std;

int main(){

    ofstream insertRes("insert.result", ios::out);
    ofstream searchRes("search.result", ios::out);
    // random numbers produced in advance
    ifstream insert;
    insert.open("../random/random.insert", ios::in);
    ifstream search;
    search.open("../random/random.search", ios::in);

    for(int i=10; i<=30; ++i){
        for(int time=0; time<5; ++time){
            int size = pow(2, i);

            // degree = maximum number of children = 100
            BPlusTree<int> DS(100);

            clock_t start = clock();
            int tmp;
            // inserting
            for(int j=0; j<size; ++j){
                insert >> tmp;
                DS.insert(tmp);
            }
            clock_t end = clock();
            insertRes << (float)(end - start)/CLOCKS_PER_SEC << endl;

            start = clock();
            // searching
            for(int j=0; j<100000; ++j){
                search >> tmp;
                DS.search(tmp);
            }
            end = clock();
            searchRes << (float)(end - start)/CLOCKS_PER_SEC << endl;
        }
        insertRes << endl;
        searchRes << endl;
    }

    insertRes.close();
    searchRes.close();
    insert.close();
    search.close();
    return 0;
}
```

treap 程式碼 ：from GitHub

```cpp
#include <iostream>
#include <fstream>
#include <cmath>
#include "SourceCode.h"

using namespace std;

int main(){

    ofstream insertRes("insert.result", ios::out);
    ofstream searchRes("search.result", ios::out);
    // random numbers produced in advance
    ifstream insertF;
    insertF.open("../random/random.insert", ios::in);
    ifstream search;
    search.open("../random/random.search", ios::in);

    for(int i=10; i<=30; ++i){
        for(int time=0; time<5; ++time){
            int size = pow(2, i);

            // typedef node* pnode;
            // the root
            pnode DS = NULL;

            clock_t start = clock();
            // inserting
            int tmp;
            for(int j=0; j<size; ++j){
                insertF >> tmp;
                insert(DS, tmp);
            }
            clock_t end = clock();
            insertRes << (float)(end - start)/CLOCKS_PER_SEC << endl;

            start = clock();
            // saerching
            for(int j=0; j<100000; ++j){
                search >> tmp;
                find(DS, tmp);
            }
            end = clock();
            searchRes << (float)(end - start)/CLOCKS_PER_SEC << endl;
        }
        insertRes << endl;
        searchRes << endl;
    }

    insertRes.close();
    searchRes.close();
    insertF.close();
    search.close();
    return 0;
}
```

skip list 程式碼 ：from GitHub

```cpp
#include <iostream>
#include <fstream>
#include <cmath>
#include "SourceCode.h"

using namespace std;

int main(){

    ofstream insertRes("insert.result", ios::out);
    ofstream searchRes("search.result", ios::out);
    // random numbers produced in advance
    ifstream insert;
    insert.open("../random/random.insert", ios::in);
    ifstream search;
    search.open("../random/random.search", ios::in);

    for(int i=10; i<=30; ++i){
        for(int time=0; time<5; ++time){
            int size = pow(2, i);

            skiplist DS;

            clock_t start = clock();
            // inserting
            int tmp;
            for(int j=0; j<size; ++j){
                insert >> tmp;
                DS.insert_element(tmp);
            }
            clock_t end = clock();
            insertRes << (float)(end - start)/CLOCKS_PER_SEC << endl;

            start = clock();
            // saerching
            for(int j=0; j<100000; ++j){
                search >> tmp;
                DS.contains(tmp);
            }
            end = clock();
            searchRes << (float)(end - start)/CLOCKS_PER_SEC << endl;
        }
        insertRes << endl;
        searchRes << endl;
    }

    insertRes.close();
    searchRes.close();
    insert.close();
    search.close();
    return 0;
}
```

hash table 程式碼 ： unordered_multiset from C++ std::unordered_set

```cpp
#include <iostream>
#include <fstream>
#include <cmath>
#include <unordered_set>

using namespace std;

int main(){

    ofstream insertRes("insert.result", ios::out);
    ofstream searchRes("search.result", ios::out);
    // random numbers produced in advance
    ifstream insert;
    insert.open("../random/random.insert", ios::in);
    ifstream search;
    search.open("../random/random.search", ios::in);

    for(int i=10; i<=30; ++i){
        for(int time=0; time<5; ++time){
            int size = pow(2, i);

            unordered_multiset<int> DS;

            clock_t start = clock();
            // inserting
            int tmp;
            for(int j=0; j<size; ++j){
                insert >> tmp;
                DS.insert(tmp);
            }
            clock_t end = clock();
            insertRes << (float)(end - start)/CLOCKS_PER_SEC << endl;

            start = clock();
            // saerching
            for(int j=0; j<100000; ++j){
                search >> tmp;
                DS.find(tmp);
            }
            end = clock();
            searchRes << (float)(end - start)/CLOCKS_PER_SEC << endl;
        }
        insertRes << endl;
        searchRes << endl;
    }

    insertRes.close();
    searchRes.close();
    insert.close();
    search.close();
    return 0;
}
```

Array of sorted arrays 程式碼 ：

ASA.h

```h
#include <vector>

using namespace std;

class ASA{
    public:
    ASA(int cnt=10);

    void insert(int data);
    bool search(int target);

    void display();

    private:
    vector<vector<int> > D, _D;
    // D[0] = sorted array of size pow(2, 0)
    // D[1] = sorted array of size pow(2, 1)
    // ...
    // D[i] = sorted array of size pow(2, i)
    int size[31];
    int level;

    void merge(const vector<int> &, const vector<int> &, vector<int> &);
    bool binarySearch(const vector<int> &, const int);
};
```

ASA.cpp

```cpp
#include <iostream>
#include <cmath>
#include "ASA.h"

ASA::ASA(int cnt) : _D(cnt+1, vector<int>(1)), D(cnt+1, vector<int>(1)) {
// cnt = levels of ASA = k
    level = cnt;
    for(int i=0; i<=cnt; ++i){
        size[i] = pow(2, i);
        // modify the capacities of each vector to pow(2, i)
        D[i].reserve(size[i]);
        _D[i].reserve(size[i]);
        // clear the elements added in initialization list above
        D[i].clear();
        _D[i].clear();
    }
};

void ASA::insert(int data){
    // starting from level 0
    // if the real array of this level is empty
    //      -> add data to the real array
    // if the real array of this level is filled
    //      -> add data to the tmp array
    //       -> real array of the next level = merge(real array, tmp array)
    //        -> clear all elements in both real and tmp arrays of this level
    if(D[0].empty()){
        D[0].push_back(data);
    }else{
        _D[0].push_back(data);

        int index=0;
        while(!_D[index].empty()){
            if(D[index+1].empty()){
                merge(D[index], _D[index], D[index+1]);
                D[index].clear();
                _D[index].clear();
            }else{
                merge(D[index], _D[index], _D[index+1]);
                D[index].clear();
                _D[index].clear();
            }
            ++index;
        }
    }
}

bool ASA::search(int target){
    for(int i=0; i<=level; ++i){
        if(!D[i].empty()){
            return binarySearch(D[i], target);
        }
    }
    return false;
}
```

```cpp
void ASA::display(){
    for(int i=0; i<=level; ++i){
        cout << "level: " << i << endl;
        auto it = D[i].begin();
        for( ; it != D[i].end(); ++it){
            cout << *it << " ";
        }
        cout << endl;
    }
}

void ASA::merge(const vector<int> &v1, const vector<int> &v2, vector<int> &out){
    // use iterator to traverse data stored in the array(in the form of vector)
    auto it1 = v1.begin(), it2 = v2.begin();
    while(true){
        // terminate when both arrays are completely traversed
        if(it1 == v1.end() && it2 == v2.end()){
            break;
        }
        // operate on the other array when either one is completely traversed
        else if(it1==v1.end()){
            out.push_back(*it2);
            ++it2;
        }else if(it2==v2.end()){
            out.push_back(*it1);
            ++it1;
        }
        // add the smaller date to the outcome array
        // sorting executed here
        else if(*it1 <= *it2){
            out.push_back(*it1);
            ++it1;
        }else{
            out.push_back(*it2);
            ++it2;
        }
    }
}

// return true if target value can be found in this array
bool ASA::binarySearch(const vector<int> &v, const int target){
    int low = 0, high = v.size()-1;
    while(low <= high){
        int mid = (low+high)/2;
        if(v[mid] == target){
            return true;
        }else if(v[mid] > target){
            high = mid-1;
        }else if(v[mid] < target){
            low = mid+1;
        }
    }
    return false;
}
```

main.cpp

```cpp
#include <iostream>
#include <fstream>
#include <cmath>
#include "ASA.h"

using namespace std;

int main(){

    ofstream insertRes("insert.result", ios::out);
    ofstream searchRes("search.result", ios::out);
    // random numbers produced in advance
    ifstream insert;
    insert.open("../random/random.insert", ios::in);
    ifstream search;
    search.open("../random/random.search", ios::in);

    for(int i=10; i<=30; ++i){
        for(int time=0; time<5; ++time){
            cout << "done" << endl;
            int size = pow(2, i);

            ASA DS(i);

            clock_t start = clock();
            int tmp;
            // inserting
            for(int j=0; j<size; ++j){
                insert >> tmp;
                DS.insert(tmp);
            }
            clock_t end = clock();
            insertRes << (float)(end - start)/CLOCKS_PER_SEC << endl;

            start = clock();
            // searching
            for(int j=0; j<100000; ++j){
                search >> tmp;
                DS.search(tmp);
            }
            end = clock();
            searchRes << (float)(end - start)/CLOCKS_PER_SEC << endl;
        }
        insertRes << endl;
        searchRes << endl;
    }

    insertRes.close();
    searchRes.close();
    insert.close();
    search.close();
    return 0;
}
```

心得、疑問、與遇到的困難

- Skip list 的實驗d中 k 值變大時，執行時間的確有跟著變長，但是 k 值相同、生成的亂數相同時，多次執行卻會得到差異極大的結果。是因為決定是否將資料放近 upper list 的機率影響了結果嗎？（在我的程式碼中 P = 0.5）

K	第 1 次	第 2 次	第 3 次	第 4 次	第 5 次	average
10	0.053019	0.012641	0.007745	0.004474	0.012251	0.018026
11	0.015154	0.0058	0.014524	0.015797	0.009918	0.0122386
12	0.018071	0.024754	0.00537	0.030633	0.018795	0.0195246
13	0.038804	0.05901	0.025149	0.015334	0.048504	0.0373602
14	0.063469	0.096988	0.070684	0.012447	0.129311	0.0745798
15	0.003785	0.526584	0.352252	0.173182	0.559808	0.323122
16	0.006319	0.897412	1.13244	0.912991	0.102018	0.610236
17	1.86058	1.75286	3.7105	1.40105	0.005219	1.74604
18	0.54896	2.49967	11.7235	1.74005	7.08972	4.72038
19	44.7091	46.3728	19.5285	15.1722	0.719518	25.3004
20	42.7417	1.27871	9.69995	151.667	124.041	65.8857

- 為什麼所有的資料結構在做搜尋時 k=10 的執行時間都遠比 k=11, 12, 13, ... 時長？