

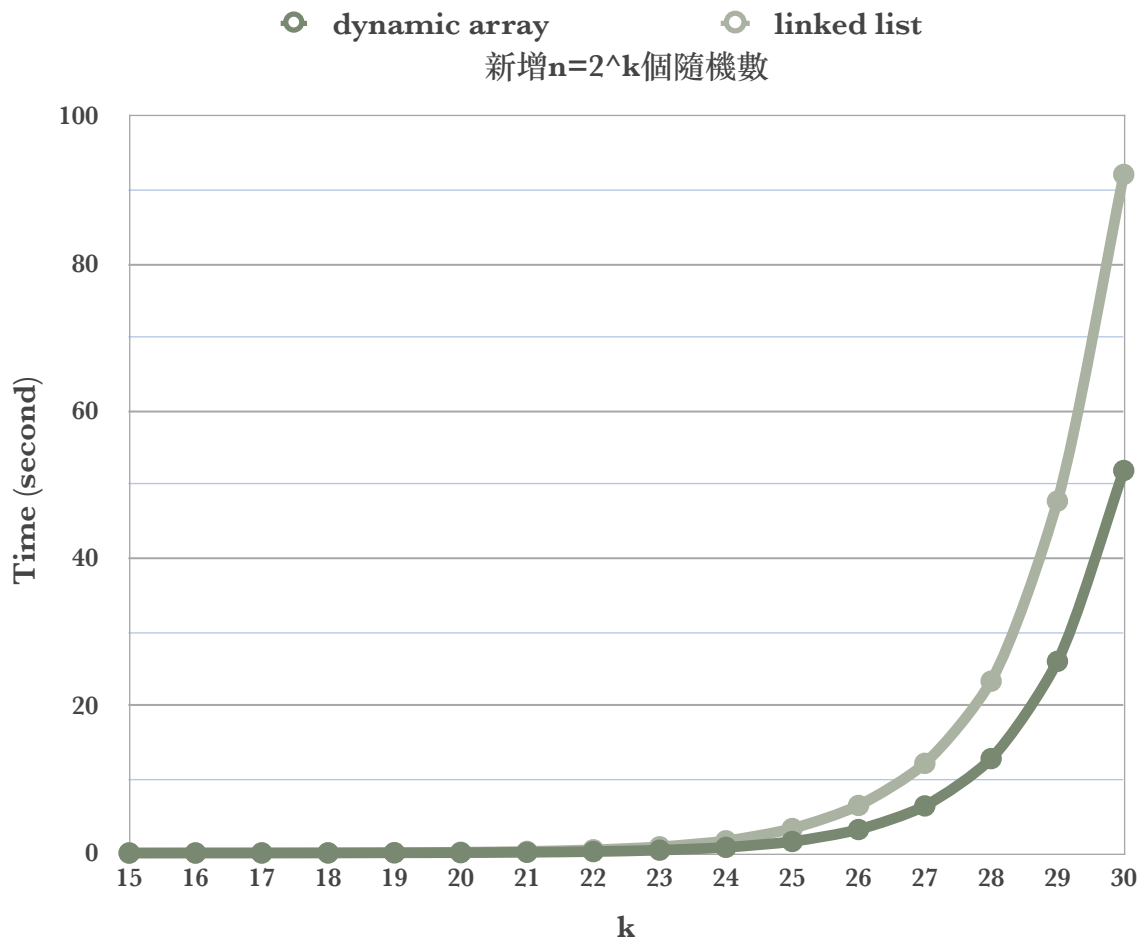
# 比較 dynamic array 與 linked list

張莞禎

資科三109102027

折線圖1：新增 $n=2^k$ 個隨機數

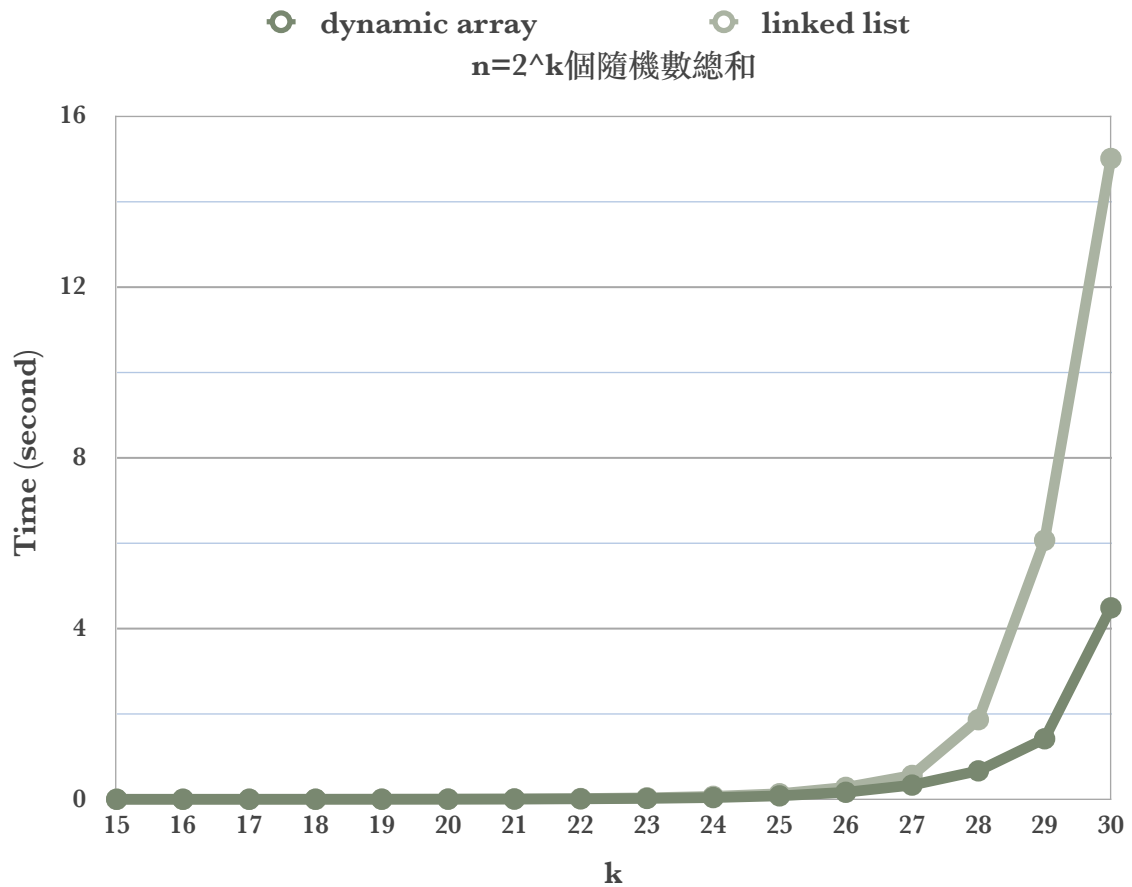
Linked list 所需時間皆比 dynamic array 還要長



k	Dynamic array	Linked list	LL/DA
15	0.0026565	0.003443	1.29606625258799
16	0.00459775	0.00691825	1.50470338752651
17	0.00820875	0.0139688	1.70169636059083
18	0.0153133	0.0273335	1.78495164334272
19	0.0272325	0.0552685	2.02950518681722
20	0.0491272	0.108541	2.20938706052859
21	0.098178	0.223412	2.27558108741266
22	0.19411	0.441382	2.27387563752511
23	0.388396	0.848765	2.18530829359726
24	0.770836	1.66478	2.15970712317536
25	1.56597	3.32459	2.12302279098578
26	3.18971	6.45833	2.02473892610927
27	6.39386	12.1734	1.90392032356042
28	12.8163	23.3231	1.81979978620975
29	26.0169	47.7646	1.83590666067056
30	51.912	92.0982	1.77412159038373

折線圖2：n=2<sup>k</sup>個隨機數總和

Linked list 所需時間皆比 dynamic array 還要長



k	Dynamic array	Linked list	LL/DA
15	0.0001135	0.000162	1.4273127753304
16	0.00020725	0.000401	1.93486127864897
17	0.000364	0.000564	1.54945054945055
18	0.00072575	0.00113575	1.56493282810885
19	0.00134	0.002367	1.76641791044776
20	0.002667	0.00405975	1.52221597300337
21	0.00491125	0.0078375	1.59582590990074
22	0.009835	0.017154	1.74417895271988
23	0.0197523	0.0340167	1.7221640011543
24	0.0393255	0.068871	1.75130640424152
25	0.0795758	0.129835	1.63158900067608
26	0.163393	0.278232	1.70283916691658
27	0.331727	0.557474	1.68052042794225
28	0.664595	1.86087	2.80000601870312
29	1.41465	6.05763	4.28206976990775
30	4.4775	14.9863	3.34702400893356

### 程式碼來源

- Dynamic array : C++ vector library
- Linked list : [from GitHub](#)

### 程式碼 : dynamic array test1

```
1  #include <iostream>
2  #include <fstream>
3  #include <vector>
4  #include <ctime>
5  #include <cmath>
6
7  using namespace std;
8
9  int main(){
10     ofstream result("dynamic1.result", ios::out);
11     for(int k=0; k<4; ++k){
12         for(int i=15; i<=30; ++i){
13
14             clock_t start = clock();
15
16             vector<int> dynamic;
17             for(int j=0; j<pow(2, i); ++j){
18                 dynamic.push_back(rand());
19             }
20
21             clock_t end = clock();
22             result << (double)(end-start)/CLOCKS_PER_SEC << endl;
23
24         }
25     }
26     result.close();
27     return 0;
28 }
```

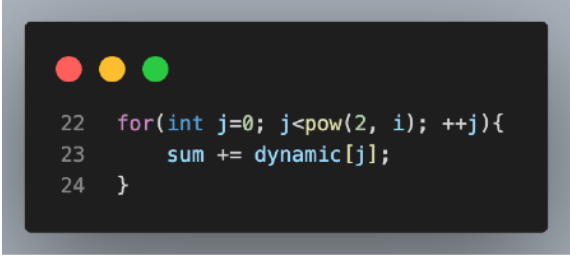
```
1  #include <iostream>
2  #include <fstream>
3  #include <ctime>
4  #include <cmath>
5
6  #include "SourceCode.h"
7
8  using namespace std;
9
10 int main(){
11     ofstream result("LinkedList1.result", ios::out);
12     for(int i=0; i<4; ++i){
13         for(int j=15; j<=30; ++j){
14
15             clock_t start = clock();
16
17             LinkedList list;
18             for(int k=0; k<pow(2, j); ++k){
19                 list.Push_front(rand());
20             }
21
22             clock_t end = clock();
23             result << (double)(end-start)/CLOCKS_PER_SEC << endl;
24
25         }
26     }
27     result.close();
28     return 0;
29 }
```

```
1  #include <iostream>
2  #include <fstream>
3  #include <vector>
4  #include <ctime>
5  #include <cmath>
6
7  using namespace std;
8
9  int main(){
10     ofstream result("dynamic2.result", ios::out);
11     for(int k=0; k<4; ++k){
12         for(int i=15; i<=30; ++i){
13
14             vector<int> dynamic;
15             for(int j=0; j<pow(2, i); ++j){
16                 dynamic.push_back(rand());
17             }
18
19             clock_t start = clock();
20
21             long long sum = 0;
22             unsigned int times = pow(2, i);
23             for(int j=0; j<times; ++j){
24                 sum += dynamic[j];
25             }
26
27             clock_t end = clock();
28             result << (double)(end-start)/CLOCKS_PER_SEC << endl;
29
30         }
31     }
32     result.close();
33     return 0;
34 }
```

```
1  #include <iostream>
2  #include <fstream>
3  #include <ctime>
4  #include <cmath>
5
6  #include "SourceCode.h"
7
8  using namespace std;
9
10 int main(){
11     ofstream result("LinkedList2.result", ios::out);
12     for(int i=0; i<4; ++i){
13         for(int j=15; j<=30; ++j){
14
15             LinkedList list;
16             for(int k=0; k<pow(2, j); ++k){
17                 list.Push_front(rand());
18             }
19
20             clock_t start = clock();
21
22             ListNode *current = list.getFirst();
23             long long sum = 0;
24             while(current != NULL){
25                 sum += current->getData();
26                 current = current->getNext();
27             }
28
29             clock_t end = clock();
30             result << (double)(end-start)/CLOCKS_PER_SEC << endl;
31
32         }
33     }
34     result.close();
35     return 0;
36 }
```

## 心得、疑問、與遇到的困難

- Linked list 提供兩種加入資料的方式，一個是 `push_front()` 放進開頭，和另一種 `push_back()` 放進結尾，依照平常使用 array 的經驗，我以為從後面放進資料會是最自然的選擇，但執行程式後，在新增  $2^{20}$  筆資料所需的時間就超過 24 分鐘，如果以增加一個數量級，執行時間大約以 6 倍成長的趨勢來算，大概永遠無法完成  $2^{30}$  筆資料。在重新看過一遍程式碼後，我發現在每次呼叫 `push_back()` 時都會使用迴圈從第一個節點一路往下一個節點檢查直到最後一個節點，再將新資料節點與結尾節點連結，最耗時間的動作就是尋找結尾結點的位置；而 `push_front()` 則是直接將新資料節點與開頭節點連結，因此才省去搜尋的時間。如果在 `class LinkedList` 中多加一個變數 `ListNode *tail` 來紀錄 list 結尾節點的位置，應該也可以用 `push_back()` 得到差不多的執行時間。
- 在第二個實驗的程式碼發現影響實驗結果的疏失，用 `for` 迴圈操作 dynamic array 資料的加總時，我在終止條件裡放了 `pow(2, i), i = 15, 16, \dots, 30` 的計算，因此將非相關的計算時間加入了實驗結果的統計，而在操作 linked list 時僅用 `while` 迴圈來偵測結尾節點。原本還在疑惑為什麼用 `index` 在 `vector` 裡得到資料只要  $O(1)$  的執行時間，卻還是比 linked list 需要從節點找到下個節點再接觸到資料花的時間更長，在修正後終於得到比較合理的結果。



```
22  for(int j=0; j<pow(2, i); ++j){
23      sum += dynamic[j];
24  }
```