作業二

110753201 資科碩一 曹昱維

- ***作業目的：比較 treap 、skip list_0.1 、skip list_0.5 、skip list_0.9 與 sorted array 。***
    I. ***除原有的 skip list 外，再另外修改 code ，得到兩種變形：第一種變形為正面機率為 0.1 ，第二種變形為正面機率為 0.9 。***
- ***（30%）在報告中請畫出每種資料結構新增資料所需時間：***

I. 圖一為五種資料結構在不同的資料量下所需的插入時間紀錄圖，圖二與圖三為圖一的局部放大圖，圖四是只考慮在最底層插入資料(去除掉向上疊加資料)的情況下，三種 skip list 的插入時間

II. 原始數據(表一)為五種資料結構分別在不同的數據量時新增資料所需時長紀錄，當時長超過 1 個小時的情況，就按其在 EXCEL 的趨勢圖公式來推估後續數據(表格中的藍底白字)
    a. Sorted array insertion 估算公式： y = 5E-08 * x^2.1214
    b. Treap insertion 估算公式： y = 2E-08 * x^1.244
    c. Skip list_0.1 insertion 估算公式： y = 8E-10 * x^1.8768
    d. Skip list_0.5 insertion 估算公式： y = 3E-10 * x^2.2415
    e. Skip list_0.9 insertion 估算公式： y = 2E-09 * x^2.3154

III. 原始數據 (表二)為**只考慮在最底層插入資料(去除掉向上疊加資料)**的清況下，三種 skip list 的插入時間
    a. Skip list_0.1 insertion 估算公式： y = 6E-08 * x^1.3378
    b. Skip list_0.5 insertion 估算公式： y = 2E-07 * x^1.209
    c. Skip list_0.9 insertion 估算公式： y = 2E-07 * x^1.3136

IV. 從圖一，圖二，圖三中的結果可以看出:
    a. Treap 所需的時間最少，理論上 Treap 插入時間複雜度為 O(logn)
    b. Sorted Array 所需的時間遠大於 Treap，因為在這裡採用的排序方法的時間複雜度是 O(nlogn)，其遠大於 Treap 的 O(logn)
    c. Skip list 隨著機率上升而插入資料所需的時間也大幅上升，推估的原因為 Skip list 在插入資料後，還需要向上疊加，而這個**向上疊加的行為主導了大部分的時間**，所以可以看到 Skip list 的**插入時間隨著向上疊加機率上升而大幅提升**

V. 從圖四可以看出，當只考慮將資料插入到 Skip list 的最底層時，而不考慮向上疊加時，**反而是 Skip list_0.5 所需的插入時間最少** ，而這也符合我們的理論，因為在插入資料前要先搜索整個 skip list，而當上升機率太低(0.1)的時候，就沒有發揮出向上疊加可以令搜尋時間減少的優勢，但是當上升機率太高(0.9)的時候，卻又增加了不必要的搜索時間

- ***（30%）在報告中請畫出每個資料結構搜尋資料所需時間：***
  I. 圖五為五種資料結構在不同的資料量下所需的搜尋時間紀錄圖，圖六為圖五的局部放大圖
  II. 原始數據(表三)為五種資料結構分別在不同的數據量時搜索資料所需時長紀錄，但是在搜索之前筆需要先插入資料，當插入時長超過 1 個小時的情況，搜索時長就按其在 EXCEL 的趨勢圖公式來推估後續數據(表格中的藍底白字)
     a. Sorted array insertion 估算公式： y = 0.0112*x^0.1023
     b. Treap insertion 估算公式： y = 0.0013*x^0.2895
     c. Skip list_0.1 insertion 估算公式： y = 0.0003*x^0.4957
     d. Skip list_0.5 insertion 估算公式： y = 0.0012*x^0.3312
     e. Skip list_0.9 insertion 估算公式： y = 0.0044*x^0.2702
  III. 可以從圖五觀察出，五種資料結構的搜索時間是符合 O(logn)的趨勢
  IV. 從圖六可以看出 **Skip list_0.5 在所有 skip list 中所需的時間最少**，理由可以參考上述

- ***（5%）在報告中請畫出三種 skip list 在不同的 n 值時，平均每個 data 的 additional copy 個數。請解釋你得到的結果。n = 2^k (k = 10, 11, 12, …直到 $T_{insert}(n)$超過一小時或是 k=30)***
  I. 平均每個 data 的 additional copy 個數也同時是 skip list 的平均層數，而我們的平均層數又受到初始設定的機率(0.1, 0.5, 0.9)影響
  II. 圖七為三種 Skip list 在不同資料量下平均每個 data 的 additional copy 個數的紀錄圖
  III. 原始資料(表三)是三種 skip list 的平均每個 data 的 additional copy 個數紀錄
  IV. 從圖七可以看出：**平均每個 data 的 additional copy 個數並不會隨著資料量變化**
  V. 平均層數的期望值： $\sum_{l=1}^{\infty} l(p)^l$, l是層數, p是上升機率
  VI. 我們的實驗結果也符合上述算式

- ***（10%）在報告中請畫出三種 skip list 在不同的 n 值時的 list 個數。請解釋你得到的結果。***
  n = 2^k (k = 10, 11, 12, …, 直到 $T_{insert}(n)$超過一小時或是 k=30)
  I. 圖八為三種 Skip list 在不同資料量下的 list 數量記錄圖，圖九是圖八的放大圖
  II. 原始資料(表四)為為三種 Skip list 在不同資料量下的 list 數量記錄
  III. 從圖八與圖九可以看出，三種 Skip list 的線條幾乎疊在一起，這代表 **list 數量與上升機率沒什麼關聯**
  IV. 從圖八也可以看出 Skip list 的 **List 數量與資料量呈線性關係**

- ***（5%）解釋如何修改 skip list 程式碼（以註解呈現）***
  I. 決定是否要上升的隨機函數:
     a. 這個隨機函數被修改成可以設定小於一定數值才會回傳 true

```
/* 產生隨機值 */
bool randomVal(float proba){
    if (seed == 0)
        seed = (unsigned)time(NULL);

    srand(seed);
    float K= (float)rand()/RAND_MAX;  // 得到一個0~1 之間的數值
    seed = rand();
    if (K < proba) // 當小於proba時，就增加一層，proba 可以在建立skip list 的時候另外設定
        return true;
    else
        return false;
};
```

  II. Insert method 中的則是藉由執行上述的隨機函數來決定是否上升
     a.

```
/* 根據設定好的機率函數所產生的隨機值決定是否將該元素添加至跳躍層 */
while (randomVal(p)){
```

  III. 另外新增 Skip list 的上升閾值 p，預設值為 0.1，在使用前另外設定即可
     a.

```
template<typename T>
class SkipList{
    public:
        float p = 0.1; // 預設為0.1，在使用之前要另外設定
        int seed = 0;
```

- ***（10%）實驗程式碼（含新增與搜尋的程式碼範例）與使用說明。***
  I. 使用說明:
     a. 執行 test_skip_list() 就會在工作目錄產生 Skip List 的 insert time 記錄，search time 記錄，list number 記錄，average list layer 記錄
        - 這些紀錄都會以 csv 檔儲存
        - 使用 test_skip_list() 需要輸入:
          - float prob : 上升閾值
          - string file_name_it : insert time 記錄檔名
          - string file_name_st : search time 記錄檔名
          - string file_name_l : list number 記錄檔名
          - string file_name_al : average list layer 記錄檔名
          - string type_record : 記錄檔中的註記文字

b. 執行 test_treap()就會在工作目錄產生 Treap 的 insert time 記錄，search time 記錄
   - 這些紀錄都會以 csv 檔儲存
   - 使用 test_treap () 需要輸入:
     - string file_name_it : insert time 記錄檔名
     - string file_name_st : search time 記錄檔名
     - string type_record : 記錄檔中的註記文字

c. 執行 test_sorted_array ()就會在工作目錄產生 Sorted Array 的 insert time 記錄，search time 記錄
   - 這些紀錄都會以 csv 檔儲存
   - 使用 test_sorted_array () 需要輸入:
     - string file_name_it : insert time 記錄檔名
     - string file_name_st : search time 記錄檔名
     - string type_record : 記錄檔中的註記文字

- ***（10%）心得、疑問、與遇到的困難***

I. 疑問 1 : 圖四是只考慮在最底層插入資料(去除掉向上疊加資料)的情況下，三種 skip list 的插入時間，雖然這三條線的趨勢符合 0.5 的機率應該是插入時間最少的，但是這三條線的趨勢卻沒有符合 **O(logn)的趨勢(理論上來說應該要符合)**，這點我還不清楚為什麼，不太確定是程式碼的問題還是有其他的因素影響

II. 疑問 2 : 圖五為五種資料結構在不同的資料量下所需的搜尋時間紀錄圖，雖然五種資料結構的搜索時間都符合 O(logn)，但還是有一些常數倍的差距，我還不太確定原因是什麼

III. 遇到的困難 1 : Skip list 程式碼來源並沒有發揮 skip list 的優勢，原來的程式碼在插入資料時，是直接走到最底下，在往右搜索然後才插入，為了找出這個問題花費不少時間

- ***報告請另外註明：***

I. 作業程式碼 :
https://github.com/theabc50111/nccu_cs_hw/blob/main/DataStructure_HW/HW2/hw2.cpp

II. 資料結構程式碼來源 :
   a. Skip list: https://www.twblogs.net/a/5d09490abd9eee1e5c813476
   b. Treap : https://www.geeksforgeeks.org/treap-set-2-implementation-of-search-insert-and-delete/
   c. Sorted array : https://www.cplusplus.com/reference/algorithm/sort/
   https://www.cplusplus.com/reference/algorithm/binary_search/

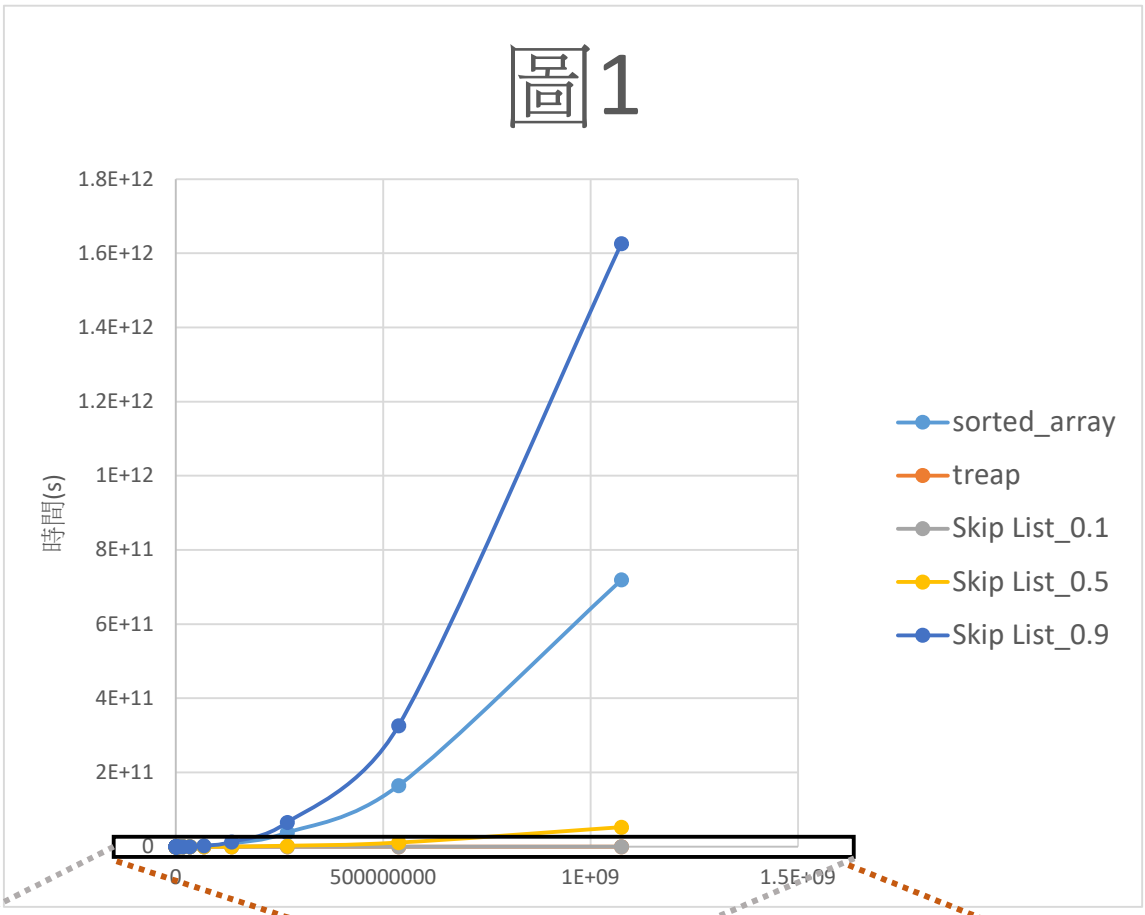


圖1
1.8E+12
1.6E+12
1.4E+12
1.2E+12
1E+12
8E+11
6E+11
4E+11
2E+11
0
時間(s)
0
500000000
1E+09
1.5E+09
sorted_array
treap
Skip List_0.1
Skip List_0.5
Skip List_0.9

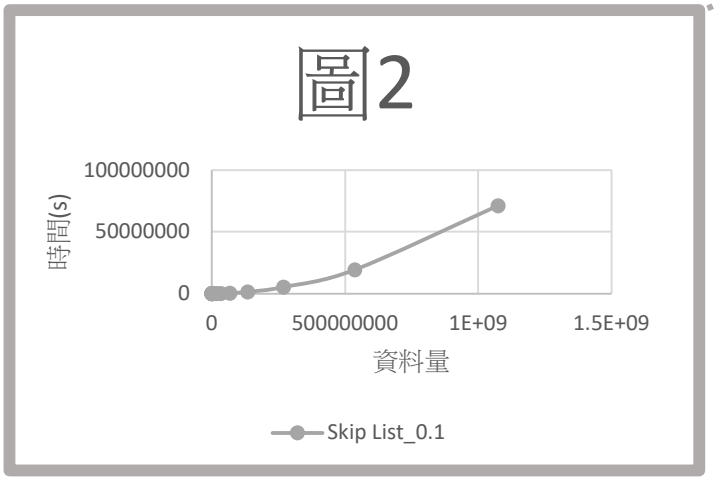


圖2
100000000
50000000
0
時間(s)
0
500000000
1E+09
1.5E+09
資料量
Skip List_0.1

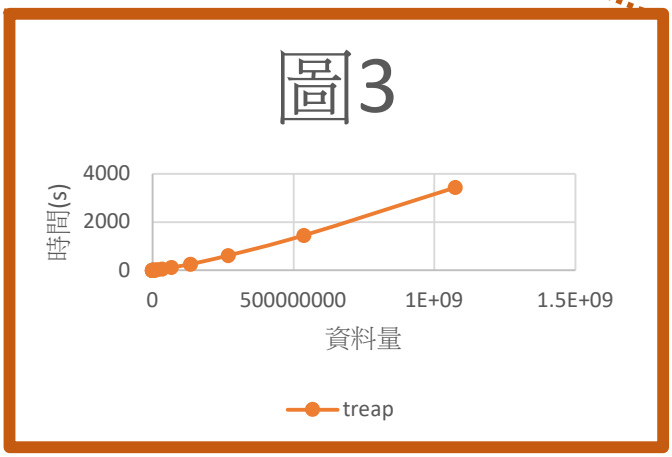
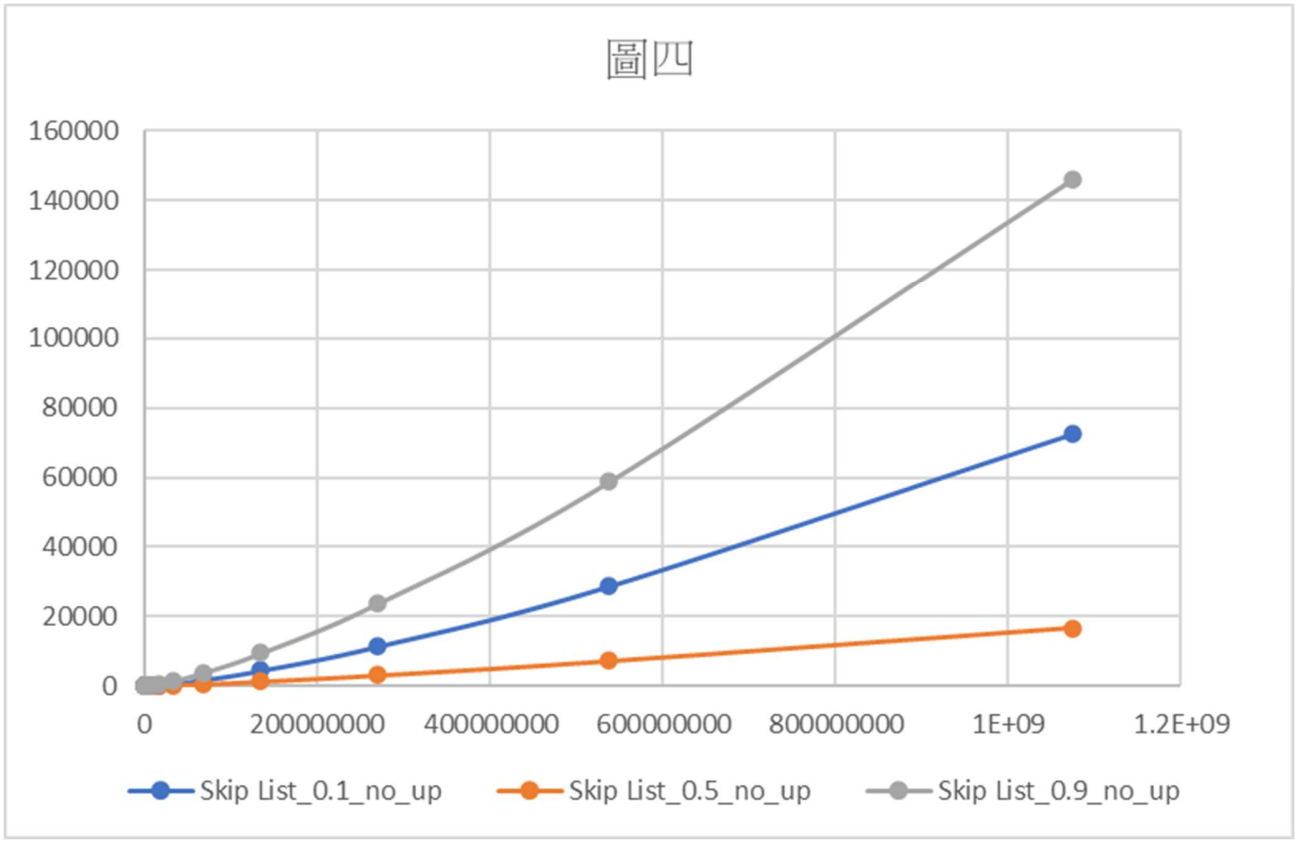
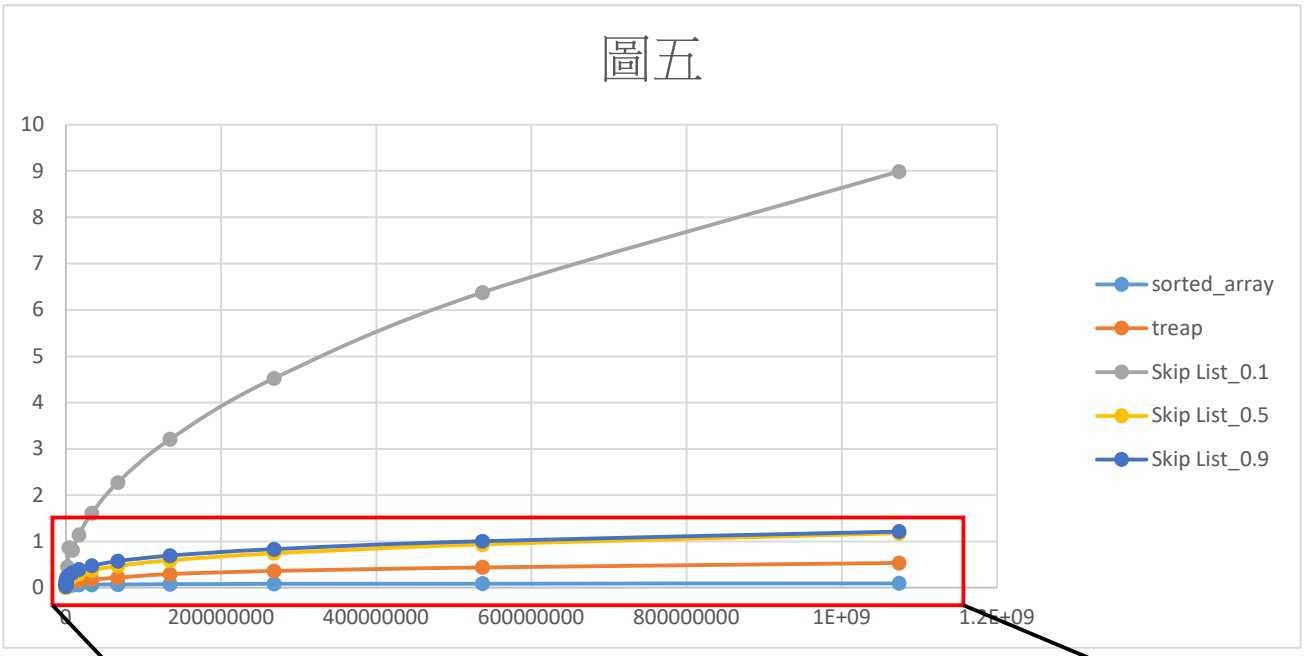
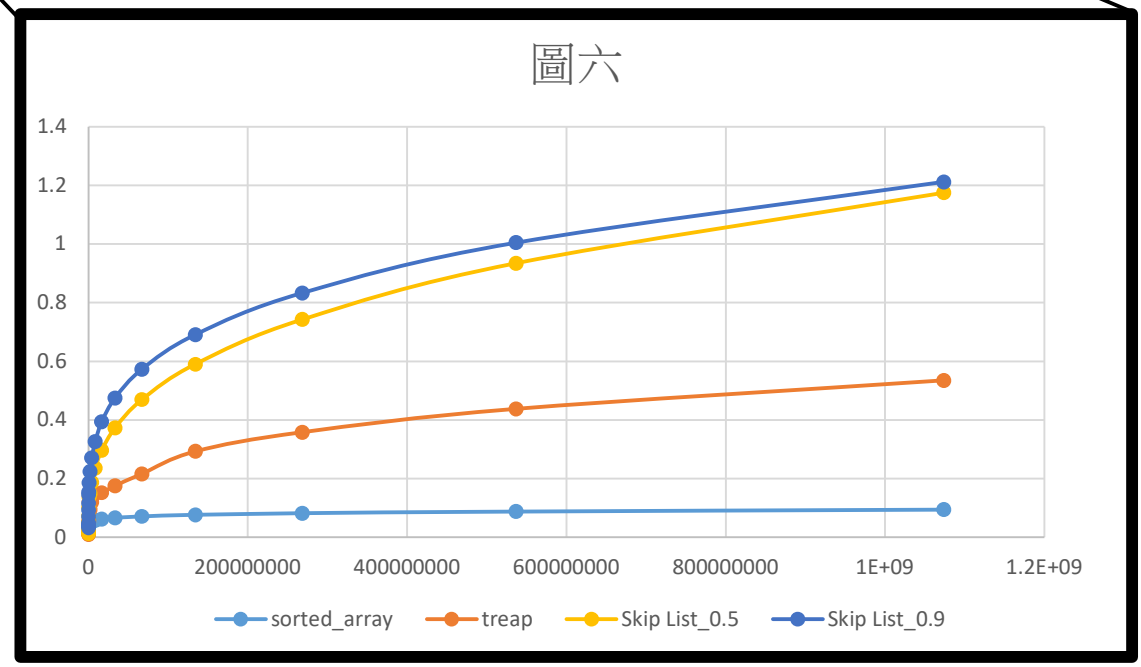
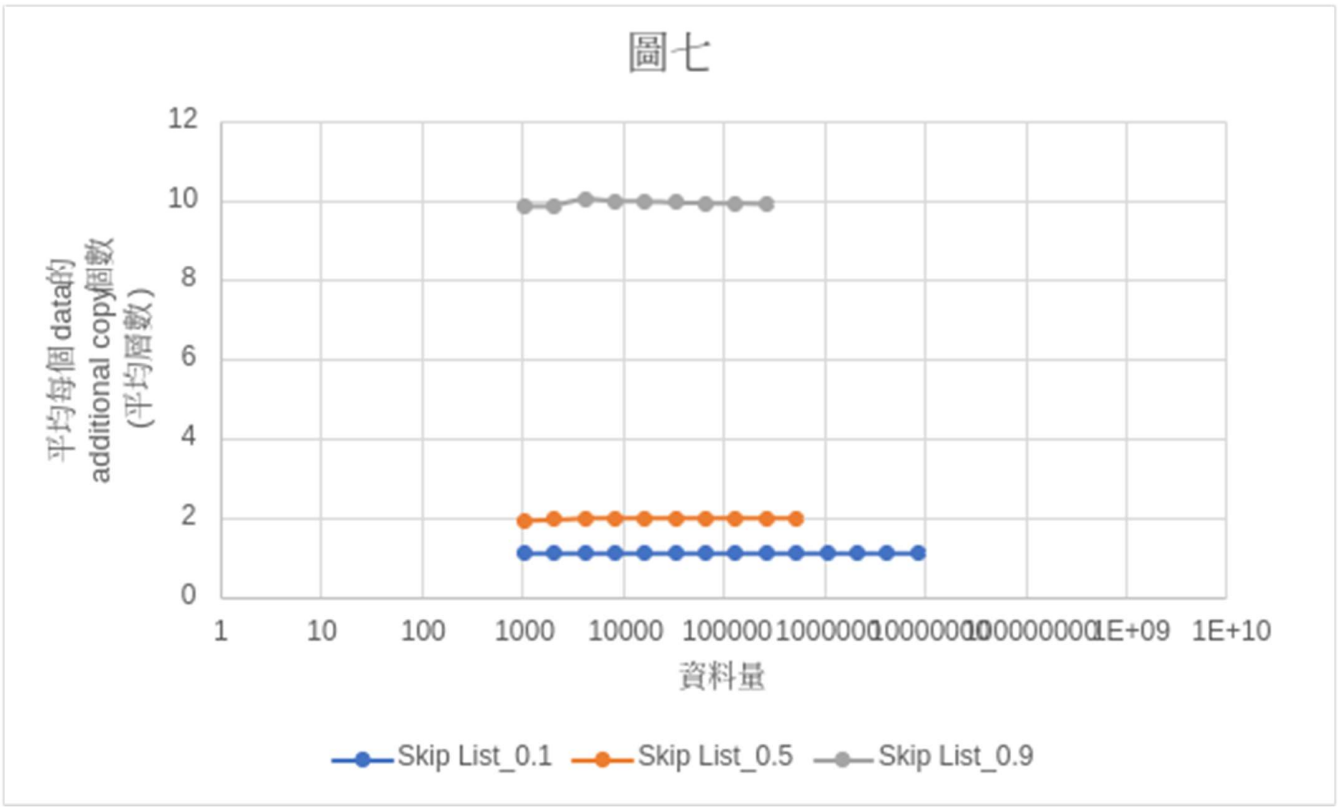


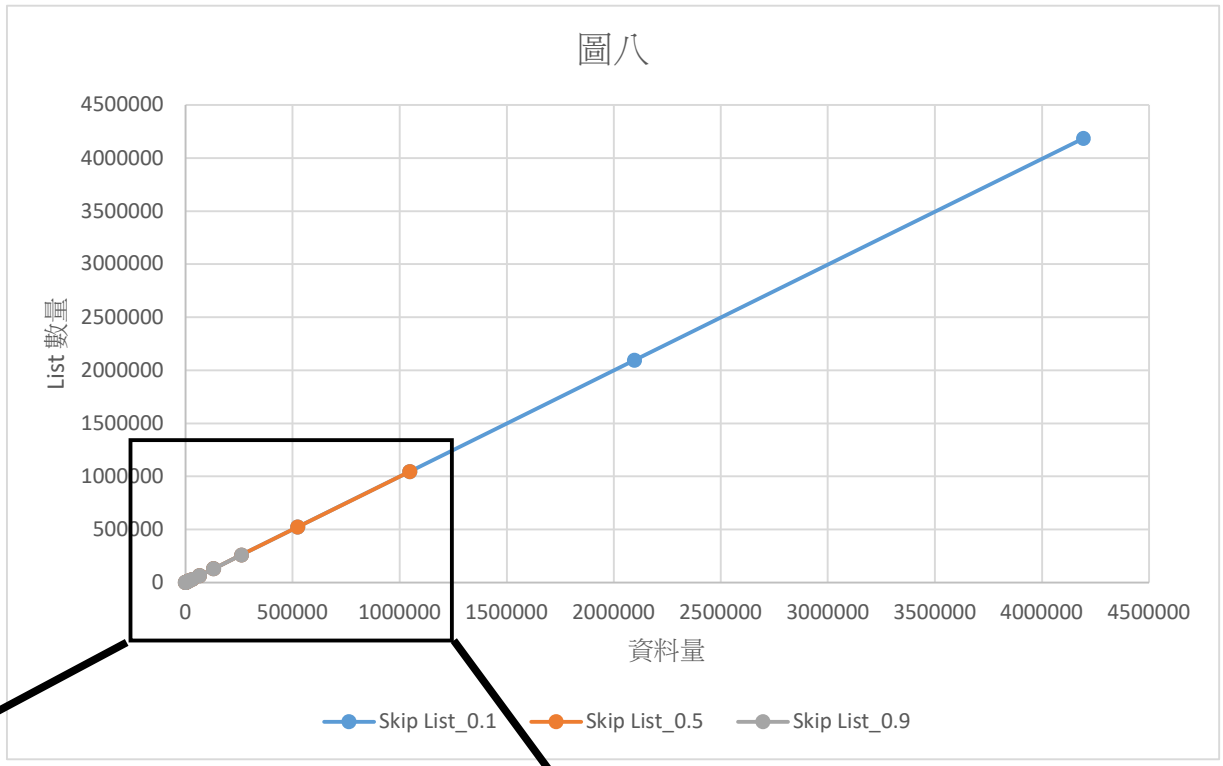
圖3
4000
2000
0
時間(s)
0
500000000
1E+09
1.5E+09
資料量
treap

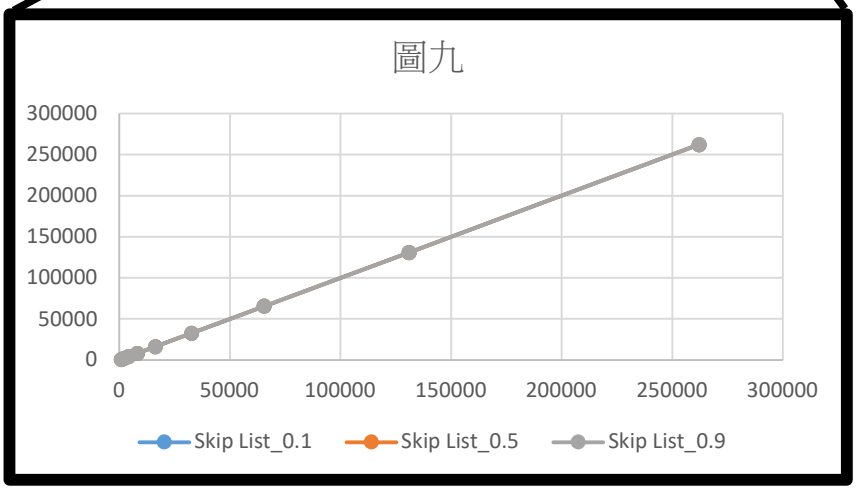

圖四
160000
140000
120000
100000
80000
60000
40000
20000
0
0
200000000
400000000
600000000
800000000
1E+09
1.2E+09
Skip List_0.1_no_up
Skip List_0.5_no_up
Skip List_0.9_no_up


圖五
10
9
8
7
6
5
4
3
2
1
0
0
200000000
400000000
600000000
800000000
1E+09
1.2E+09
sorted_array
treap
Skip List_0.1
Skip List_0.5
Skip List_0.9


圖六
1.4
1.2
1
0.8
0.6
0.4
0.2
0
0
200000000
400000000
600000000
800000000
1E+09
1.2E+09
sorted_array
treap
Skip List_0.5
Skip List_0.9


圖七
12
10
8
6
4
2
0
平均每個 data的
additional copy個數
(平均層數)
1
10
100
1000
10000
100000
1000000
10000000
100000000
1E+09
1E+10
資料量
Skip List_0.1
Skip List_0.5
Skip List_0.9


圖八
4500000
4000000
3500000
3000000
2500000
2000000
1500000
1000000
500000
0
List 數量
0
500000
1000000
1500000
2000000
2500000
3000000
3500000
4000000
4500000
資料量
Skip List_0.1
Skip List_0.5
Skip List_0.9


圖九
300000
250000
200000
150000
100000
50000
0
0
50000
100000
150000
200000
250000
300000
Skip List_0.1
Skip List_0.5
Skip List_0.9

表一					
資料量	sorted_array	treap	Skip List_0.1	Skip List_0.5	Skip List_0.9
1024	0.116163	0.000158	0.001347	0.00339	0.025146
2048	0.516636	0.000304	0.002895	0.007824	0.101677
4096	2.42442	0.000748	0.006278	0.02906	0.433132
8192	10.4029	0.001899	0.014581	0.127707	1.89453
16384	46.3294	0.003843	0.034601	0.563815	7.3325
32768	191.083	0.008394	0.113346	2.35801	38.1586
65536	819.387	0.023576	0.367627	9.1506	259.826
131072	3708.27	0.058011	1.41725	47.033	1716.52
262144	14847.9	0.108416	6.77385	472.308	9328.04
524288	67993.22293	0.300291	30.2674	2866.61	35004.36332
1048576	295849.3779	0.906438	251.04	13439.7	174231.6441
2097152	1287287.918	1.69457	1452.94	44366.45161	867225.1953
4194304	5601195.432	4.25569	6972.47	209803.8391	4316549.632
8388608	24371696.36	11.7082	7897.164632	992138.1879	21485308.34
16777216	106045145.3	23.9074	29003.09116	4691707.207	106941542.2
33554432	461419372.6	53.2171	106516.6216	22186542.94	532293661.5
67108864	2007709422	123.156	391192.4636	104917605.9	2649452554
134217728	8735864516	258.2411756	1436691.675	496143273	13187455240
268435456	38011142448	611.6553864	5276387.358	2346204387	65639588620
536870912	1.65393E+11	1448.732221	19378036.38	11094930290	3.26716E+11
1073741824	7.19649E+11	3431.384885	71167688.91	52466647318	1.62621E+12

表二			
資料量	Skip List 0.1 no up	Skip List 0.5 no up	Skip List 0.9 no up
1024	0.000806	0.000959	0.001551
2048	0.00205	0.002725	0.004476
4096	0.005328	0.00631	0.010452
8192	0.011735	0.013016	0.024366
16384	0.024824	0.028065	0.051083
32768	0.052085	0.061243	0.130241
65536	0.114321	0.129423	0.365587
131072	0.256397	0.300593	1.0285
262144	0.639474	0.876259	2.49592
524288	1.64184	2.52199	6.520141779
1048576	5.13809	3.801327529	16.20653016
2097152	16.5588	8.787807917	40.2831148
4194304	49.919	20.3154207	100.1281164
8388608	296.281	46.96464945	248.8794559
16777216	277.5757533	108.571628	618.6172855
33554432	701.6159532	250.9930029	1537.641363
67108864	1773.443609	580.2389511	3821.97688
134217728	4482.654962	1341.38098	9499.944277
268435456	11330.60866	3100.968887	23613.1573
536870912	28639.8783	7168.737428	58693.10192
1073741824	72391.75349	16572.49659	145888.166

表三			
資料量	Skip List_0.1	Skip List_0.5	Skip List_0.9
1024	1.1123	1.93555	9.86328
2048	1.11182	1.95654	9.87402
4096	1.11182	1.9895	10.0405
8192	1.10938	2.00195	9.98792
16384	1.10974	2.00061	9.98297
32768	1.1106	1.99893	9.94913
65536	1.11103	2.00035	9.93525
131072	1.11163	1.99744	9.92722
262144	1.11159	1.99462	9.92066
524288	1.11148	1.9948	
1048576	1.1115		
2097152	1.11159		
4194304	1.11157		
8388608	1.11156		

表四			
資料量	Skip List_0.1	Skip List_0.5	Skip List_0.9
1024	1024	1024	1024
2048	2048	2048	2048
4096	4096	4096	4096
8192	8192	8192	8192
16384	16384	16384	16384
32768	32768	32768	32768
65536	65533	65535	65534
131072	131067	131069	131066
262144	262115	262109	262110
524288	524151	524171	
1048576	1048080	1048080	
2097152	2095020		
4194304	4186020		