Dsnp HW5 Report

- 報告撰寫人：R05943092 曾育為
- 實驗設計
  - 設計 Array、Dlist、Bst 三者資料結構，進行比較，下表為三者實作方法

	Array	Double link list	Binary search tree
Insert	分成三種情況： 1. 當_capacity 為空，建立一個_capacity 為 1 的 array 2. 當_capacity=_size 時，將_capacity 放大二倍，將原先的值放入 3. 當_capacity<_size，直接放入	分為兩種情況： 1. 當 empty()時，直接取代插入_head 2. 當不是 empty()時，插入至_head 前面的前面	會先原先_root 的左子樹建一個真實的_root，而原先的__root 就等於 iterator end()結束的地方 分成二種情況： 與真實的__root 比較 1. 小者往左走 2. 大者往右走 以此遞迴，找到正確位置，插入此點
erase	用 iterator 指向預刪除的點，用 array 最後一個值補上，並將_size-1	分成二種情況： 1. 刪除_head，刪除後並將_head 指標指向下一個 2. 不是刪除_head，刪除後將前後指標補上即可	我會先判斷預刪除的點是屬於其父點的左還是右子點 再分四種情況討論： 刪除的點 1. 沒有左右子點 2. 只有有左子點 3. 只有有右子點 4. 有左子點和右子點
pop_front	將 array 最後一個值取代第一個值，並將_size-1	將_head 刪除，並將_head 指標指向下一個	刪除_root 最左邊的點
pop_back	直接將_size-1	將_head 前一個刪除	刪除_root 最右邊的點
clear	直接將_size=0、_data=0	將_head 的前面及後面接等於_head	將_root 的左子數等於零即可
sort	無	用 insertion sort 的方法完成	無

- 實驗預期
  - 跟據 Time Complexity 進行判斷

	Array	Double link list	Binary search tree
Insert	0(1)	0(1)	O(logn)
erase	0(1)	0(1)/ 0(n) [不知 iterator 情況下]	O(logn)
pop_front	0(1)	0(1)	O(logn)
pop_back	0(1)	0(1)	O(logn)
clear	0(1)	0(1)	O(1)
sort	0(n*n)	0(n*n)	無

- 結果比較與討論
  - 輸入以下指令，記錄結果(單位：second)

	Array	Double link list	Binary search tree
Insert 100000 筆資料	0.03	0.02	0.19
刪除時，皆有 100000 筆資料情況下			
隨機 delete 100000 筆資料	0.01	20.93	229.9
pop_front 100000 筆資料	0.01	0.02	0.24
pop_back 100000 筆資料	0.02	0.02	0.23
clear 100000 筆資料	0.33	0.01	0.19
sort 100000 筆資料	0.07	120.7	無

  - 用作業所提供的測資 do2 進行比較(單位：second)

	Array	Double link list	Binary search tree
do2	0.54	20.31	21.64

  - 討論
    - Array 為連續記憶體，除了在 clear 外，其餘皆為最快
    - Binary search tree 本身建樹時，已經將資料排序好，但如果隨機刪除資料，必須每筆搜尋相當慢
    - Double link list 如果改用 merge sort 或是 quicksort，O(log n)的演算法，結果會更好
    - 結論，就實驗數據而言，不考慮其他用途，Array 為最好的方式