一、資料結構的實作

1. Dlist: 使用每個 Node 連到前一個與後一個的方式建構 Node 的關係，並第一個 Node 的前面與最後一個 Node 的後面加上一個 dummy Node，以此定義 iterator 的結尾。

   Array: 使用動態陣列方式存取 Node，可以使 Node 存在一塊連續性的記憶體，記憶體的存取依照二的次方數來呼叫。

   Bst: 使用每個 Node 有_parent, _left, right 來建構 Tree, Parent 值必大於等於_left, 小於等於_right，並在最大的 Node 的_right 增設一個 dummy Node 以定義 iterator 的結尾。以此來建構可以快速找到特定 Node 的結構。

2. Dlist: 在空的時候先設 root 為 dummy Node，若要增加 Node，只需將遇加入的位置前後的 Node 的 Pointer 接上新 Node、新 Node 的 Pointer 接上前後的 Node 即可，若要刪除 Node，只需將前後 Node 的 Pointer 互相接上即可。Sort 因為不是 random access，所以使用 insertion sort 以 n^2 time complexity 實作。

   Array: 額外儲存一個長度變數為目前已經儲存的 Node 數，以此判斷是否還有剩下的可使用記憶。增加 Node 時，將 Array 的最後一個可使用的區塊儲存新 Node 即可。若 Node 數已到記 憶體上限，則重新要求兩倍的記憶體，先將原來的所有 Node 複製上去，再新增 Node。 Sort 時可以直接使用 STL 的函式，以 nlogn time complexity 實作。

   Bst: 每個 Node 除了_left, _right 外，多儲存一個_parent 以方便回溯。若要增加 Node，從_root 一路比較值的大小，若較小則往_left 走，反之則往_right。因使用此方式增加 Node，無 需執行 Sort，但是在找下一個 Node 時較麻煩：先找_right 存在與否，若是，則再往_left 走到不能走為止；若否，則往_parent 找到一個右上的 Node 為止，若找不到，則代表已 經是最後一個 Node(dummy)。

3. 三種資料結構的優缺點於 2-2 討論。

   Dlist, Array 格式大致被固定，解釋 Bst：相較於在每個 iterator 儲存_root 與 trace(從_root 走到現在的點的走法)，我認為在 Node 較多時，++iterator 與--iterator 以此方式較容易，因為可能只是往上走一個 Node，以此方式為 O(1)，以 trace 方式為 O(logn)，雖然在增加與減少 Node 時會稍微麻煩一點，但我覺得是可以接受的 tradeoff。至於 dummy Node 則只是純粹的讓 end()較好做。

二、實驗比較

1. 實驗設計

   do6: adtadd -r 5000
   adtprint
   adtprint -r
   adtsort
   adtreset 3
   adtadd -r 50000
   adtd -r 10000
   (中間皆加入 usage 看執行時間)

2. 實驗預期

	Dlist	Array	Bst
Adtadd -r 100000	O(n)	O(n)	O(n log n)
Adtprint	O(n)	O(n)	Ave: O(n) ~ O(n log n)
Adtprint -r	O(n)	O(n)	O(n) ~ O(n log n)
Adtd -r 10000 (k=10000, n=100000)	O(k*n)	O(k*n)	Ave: O(k*n) ~ O(k*n log n)
Adtsort	O(n^2)	O(n log n)	0
Adtreset 3	O(n)	O(1)	O(n)
Adtadd -r 50000	O(n)	O(n)	O(n log n)

3. 結果比較與討論

	Dlist	Array	Bst
Adtadd -r 100000	0.03 s	0.06 s	0.10 s
Adtprint	0.03 s	0.02 s	0.04 s
Adtprint -r	0.03 s	0.01 s	0.04 s
Adtd -r 10000 (k=10000, n=100000)	3.31 s	4.34 s	34.56 s
Adtsort	29.47 s	0.03 s	0.00 s
Adtreset 3	0.00 s	0.00 s	0.01 s
Adtadd -r 50000	0.01 s	0.01 s	0.03 s

結果大致符合實驗預期的假設。乍看之下似乎 Array 是最好的選擇，但因此次實驗只有測試基本功能，實際使用時各個指令的使用次數可能差異很大。例如若是在排序過的資料結構中持續小數目增加減少物件的同時維持排序，那 Bst(O(log n))的效率就會遠高於 Dlist(O(n))與 Array(O(n))；而在不管排序的情況下持續增減物件，Dlist 的效率也會高於 Bst 與 Array。並且因 Array 為避免經常重新要記憶體，需要較多的記憶體來增加效率。