一、

Array：

以一連串連續的記憶體位置來存取資料，不需要一直用到pointer來做各個資料間的處理。而實作Array的部份，一個動態陣列所包含的部份有_data(各記憶體位置內存的資料)，_size(用來讓使用者知道目前動態陣列的大小)、_capacity(實際上跟系統取要的記憶體空間，大小為2的次方)。

再透過iterator可以方便的對Array內的每個資料做處理，像是erase、insert、sort等等。使用Array的方式存取資料可以相當快速，且寫code部份也容易，缺點是消耗較多的記憶體空間，因為通常_capacity會超過_size。

DList：

以pointer將每個資料間雙向的聯結起來，形成一個串流。實作的部份，DList除了有_data之外，還有用來指向前一個資料的pointer
_prev和指向下一個資料的_next，再使用一個dummy的節點來聯結頭尾的資料。

同樣的DList也有iterator，方便做各個資料的操作(insert、erase、sort等)。以DList的方式操作使用到的記憶體空間比較少，比較節省資源，但相對的速度較慢。

BSTree：

同樣是以pointer連接各個資料，以樹狀的方式存，右側(_right)的資料一定比自己大，左側(_left)的比自己小，不斷延伸下去。我的作法是先建造一個dummy的pointer
_root，第一個放進去的object放在_root->_left，每個Object都會有_left、_right、_parent。

而iterator的使用也增加很多便利性，erase、insert等function都比較好寫也方便，且BST的特點是他不需要sort，從begin()到end()就已經是從小到大排序好的。效率上速度也算快，使用的記憶體也大約在Array和DList之間，算是很好的資料結構方式。

二、

1.實驗設計：

同樣依序執行atda -r 100000，usage，adtp，usage，adts，usage，adtd -r 50000，usage這些指令，看三種不同結構所需的時間及記憶體空間，來評比效能。

2.實驗預期：

所需時間：Array < BSTree < DList

所需記憶體： DList < BSTree < Array

3.結果比較：

Array：

Insert time： 0.05 s
Delete time：30.73 s
Print time： 0.05 s
Sort time：0.06 s
Total time used：30.89 s
Total memory used：9.195 MB

DList：

Insert time：0.04 s
Delete time：16.83 s
Print time： 0.05 s
Sort time：216.5 s
Total time used：233.5 s
Total memory used：6.352 MB

BSTree：

Insert time： 0.13 s
Delete time：133.3 s
Print time： 0.06 s
Sort time：0 s
Total time used：133.5 s
Total memory used：7.906 MB

綜合來看，和實驗預期的差不多，不過由於DList所使用的sort是Bubble sort，再處理大量資料時時間會大幅成長，時間幾乎都花在sort上，若換成merge sort或其他sort的演算法應該會有顯著的速度提升。BSTree在一開始insert資料時就已經是sorted的，因此之後不必sort，很有效率，但delete的話就需要比較麻煩的運算，所需時間就較長，時間也幾乎都耗在delete上。