1.

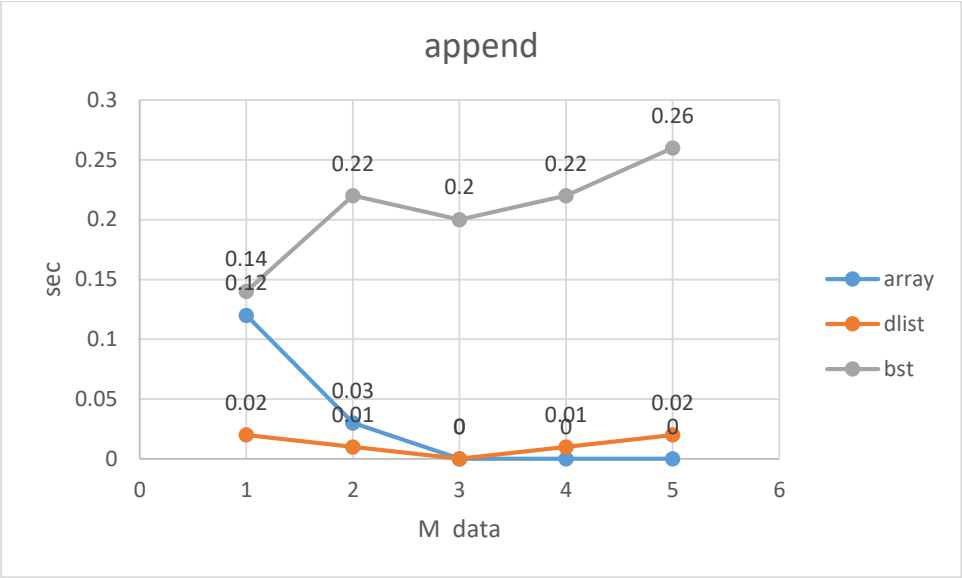
binary tree 設計結構: tree node 只存 right left child 不存 parent

所以在設計 iterator 時需要有一個 stack 裡面存著由 root 到現在走到的 node 的路徑(一連串 node) 每++ -- 更新 stack。

dummy node 的 left child 接著 root ，dummy node 會放在 iterator 中的 stack 的底部，這樣走到最後，也就是 end()就會是 dummy，

另外程式中盡量不使用 iterator，因為不知為何 iterator 的執行效能不佳，所以程式中 iterator 和 其他 function 如 pop insert delete 是盡量被我分開。

實驗:



測試方法

adta -r 1000000　　(base) 1M

usage

adta -r 100000　　(append)

usage　　➔　紀錄

adtd -a

adta -r 2000000　2M

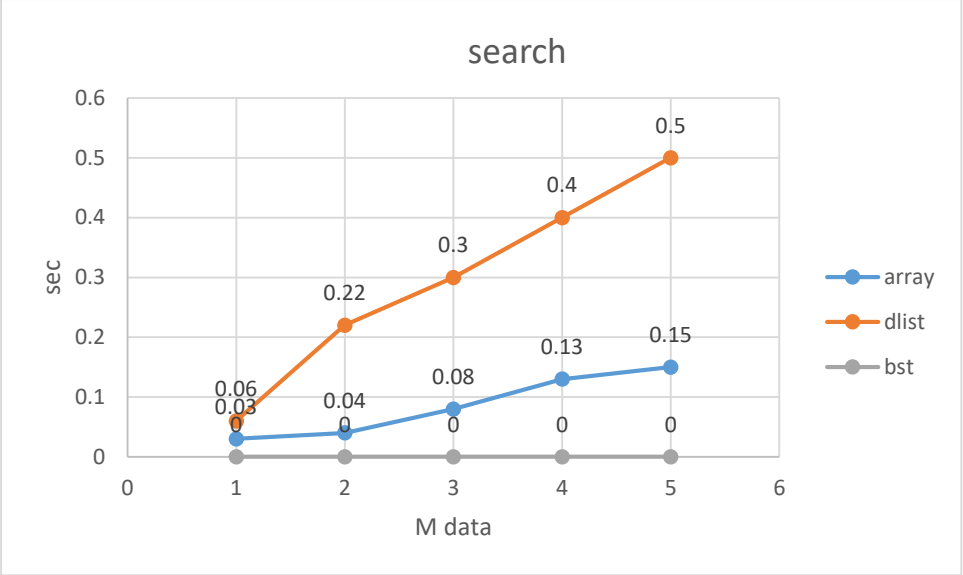
usage

adta -r 100000(append)

usage ➔　紀錄

解釋:array 在 n 小時反而比較大的原因可能是源自頻繁的擴大 array (size*2)

可以預期 pop 和 append 不會差很多(除了 array)



array　dlist 是 O(n)

bst 是 log(n) 遠小於 n

測試方法

adta -r 1000000

usage

adtq aad

adtq sad

adtq aad

adtq fad

adtq sed

adtq had

adtq sad

adtq kad

adtq shd

adtq had

adtq sad

usage ← 紀錄

我們可以預期 delete 不會跟 find 差很多

沒有運用 adtd –r 測試 delete 的原因是 adtd –r 中

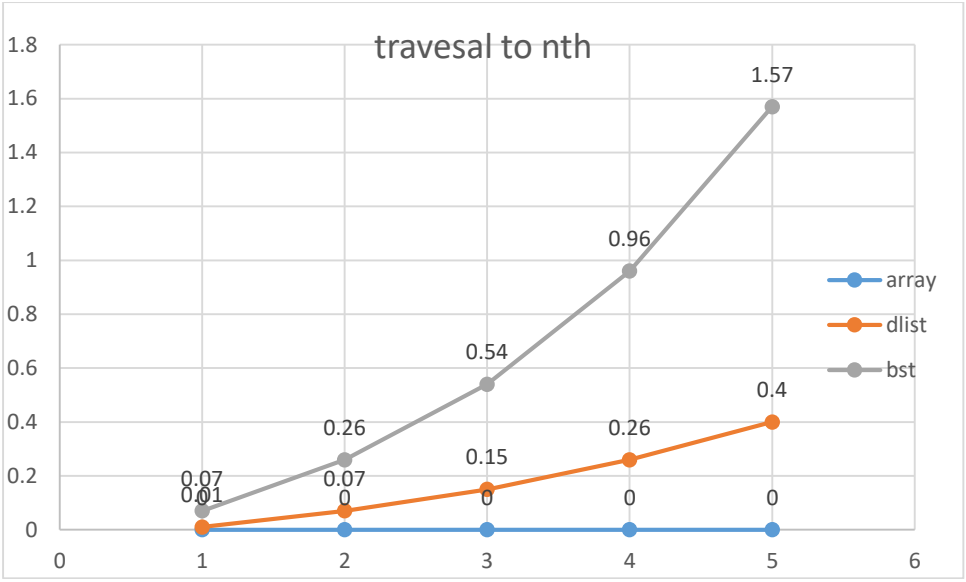
要先將 iterator traversal 到第 n 個再 delete

array → O(1) traversal + delete O(1)

dlist → O(n*1)traversal + delete O(1)

bst → O(n*log(n)) traversal + delete O(log(n))

測到的並不會是 delete，會被 travesal 蓋過去



adta -r 10000

usage

adtd -r 1000

usage🡐 紀錄

adtd -a

adta -r 20000

usage

adtd -r 1000

usage🡐 紀錄

adtd -a

adta -r 30000

usage

adtd -r 1000

usage

adtd -a

adta -r 40000

usage

adtd -r 1000

usage

adtd -a

adta -r 50000

usage

adtd -r 1000

usage