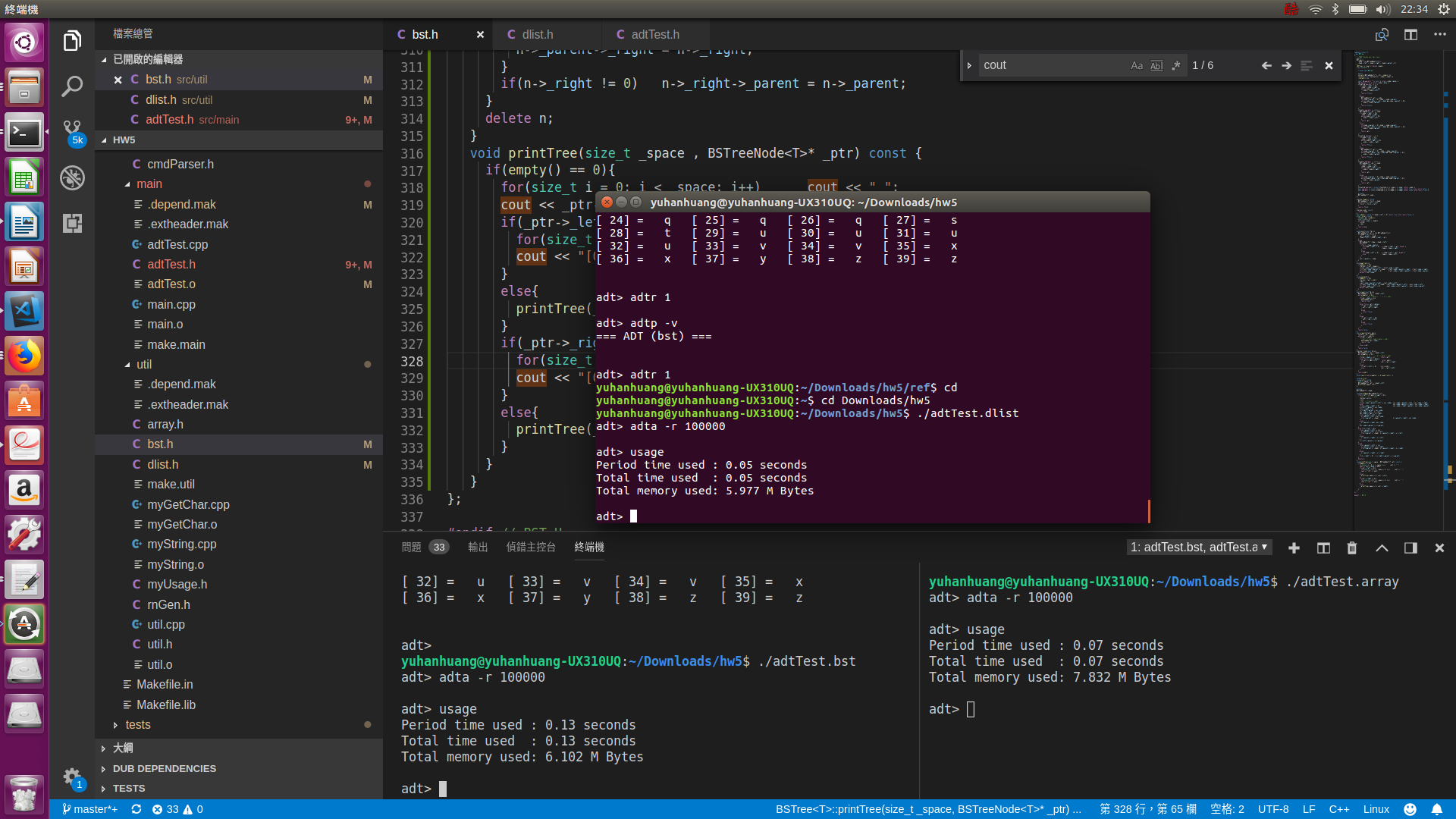
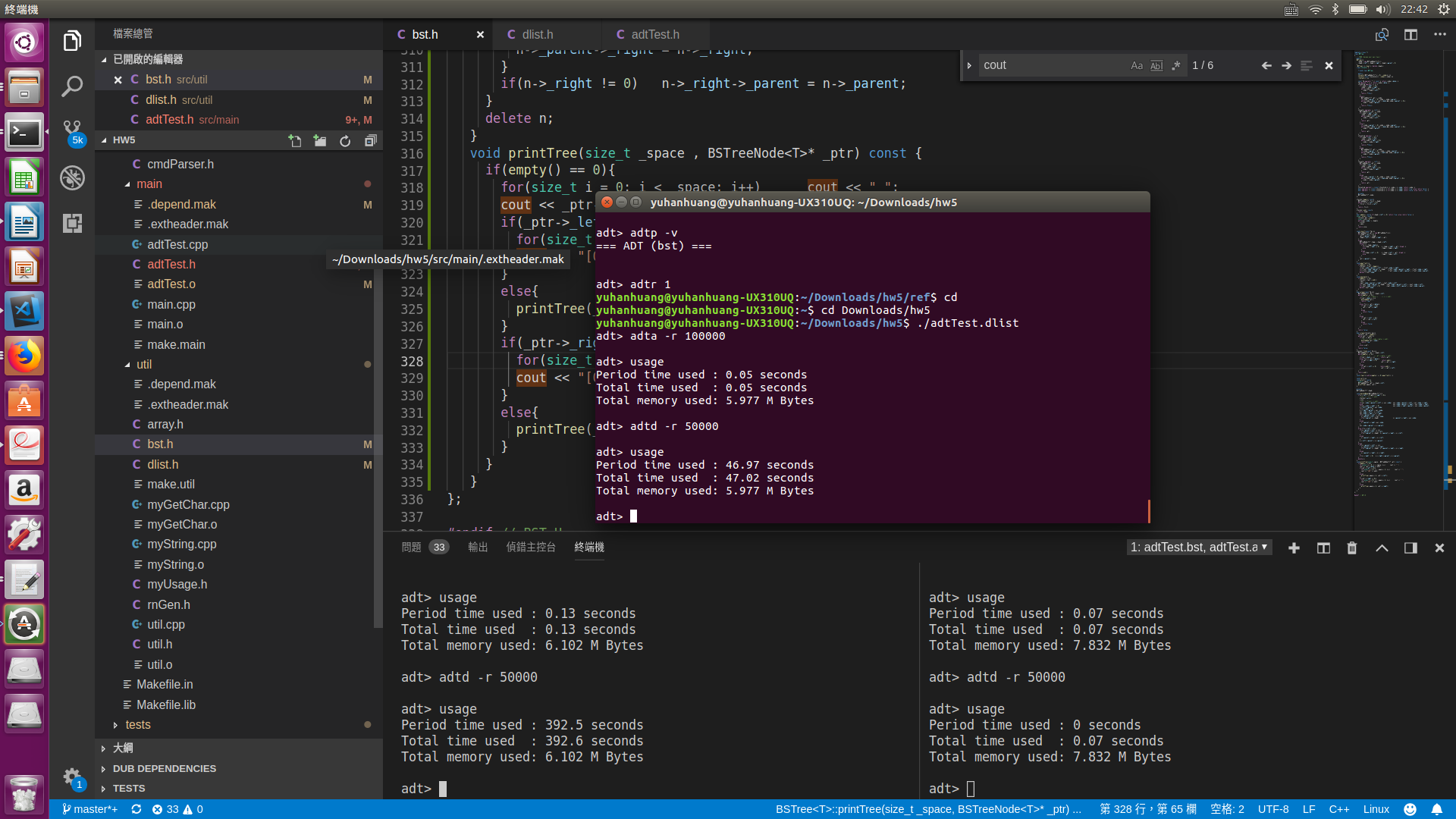
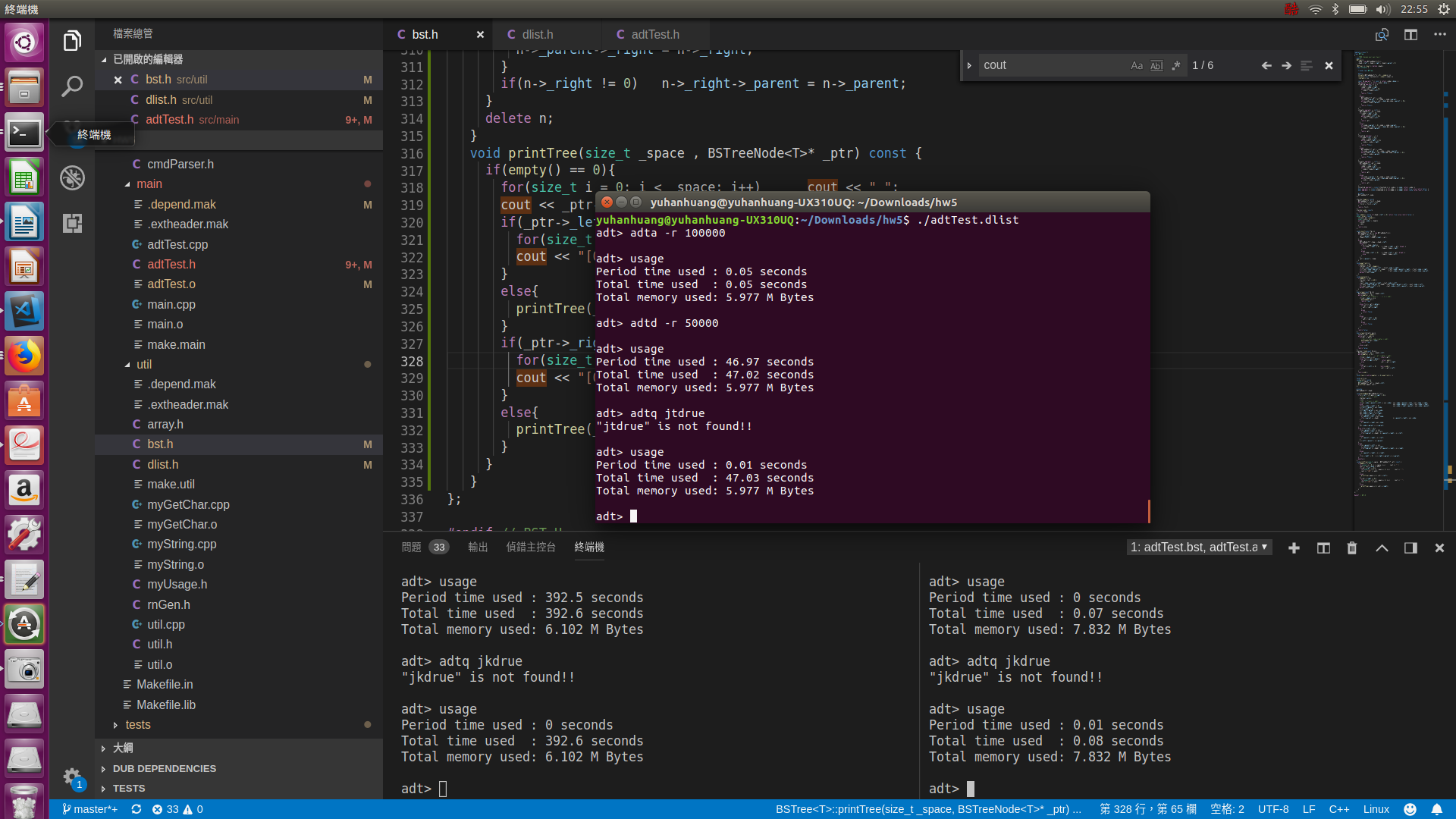
對於加入element來說，我的doubly linked list最快，array和他差不多，但是bst因為時間複雜度為n（前兩者為1），慢了兩個整整兩倍。



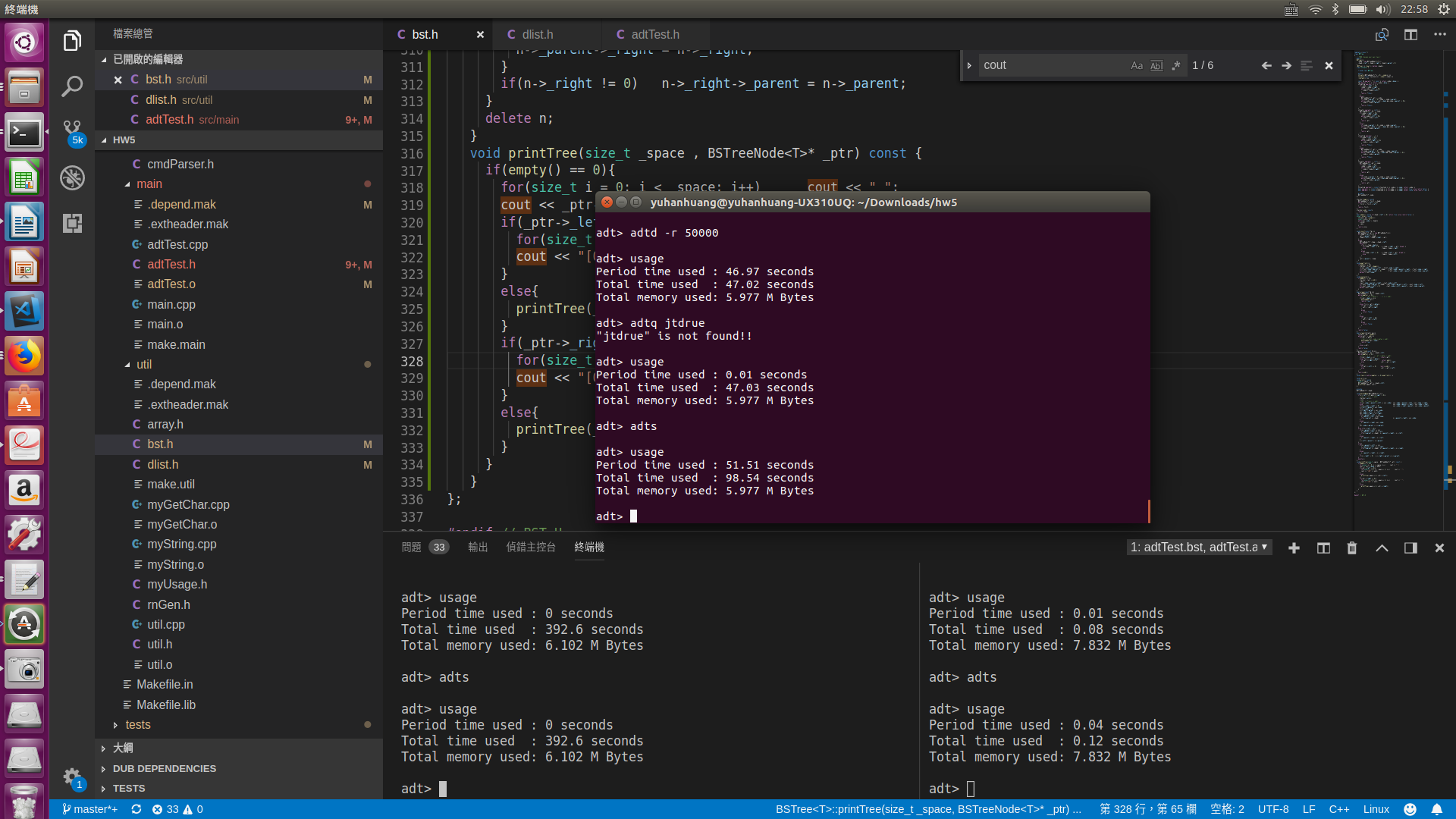
對於刪除element（erase(iterator pos)）而言，array相比其他adt快了非常多，因為時間複雜度為1，而doubly linked list 尚可，因為時間複雜度為n，差不到一百倍，bst由於deletion 需要找successor比較麻煩 加上用iterator找的話不能使用像binary tree的find(const T& x)來找元素，所以雖然時間複雜度同為n，相對慢了許多，速度差了array快千倍。



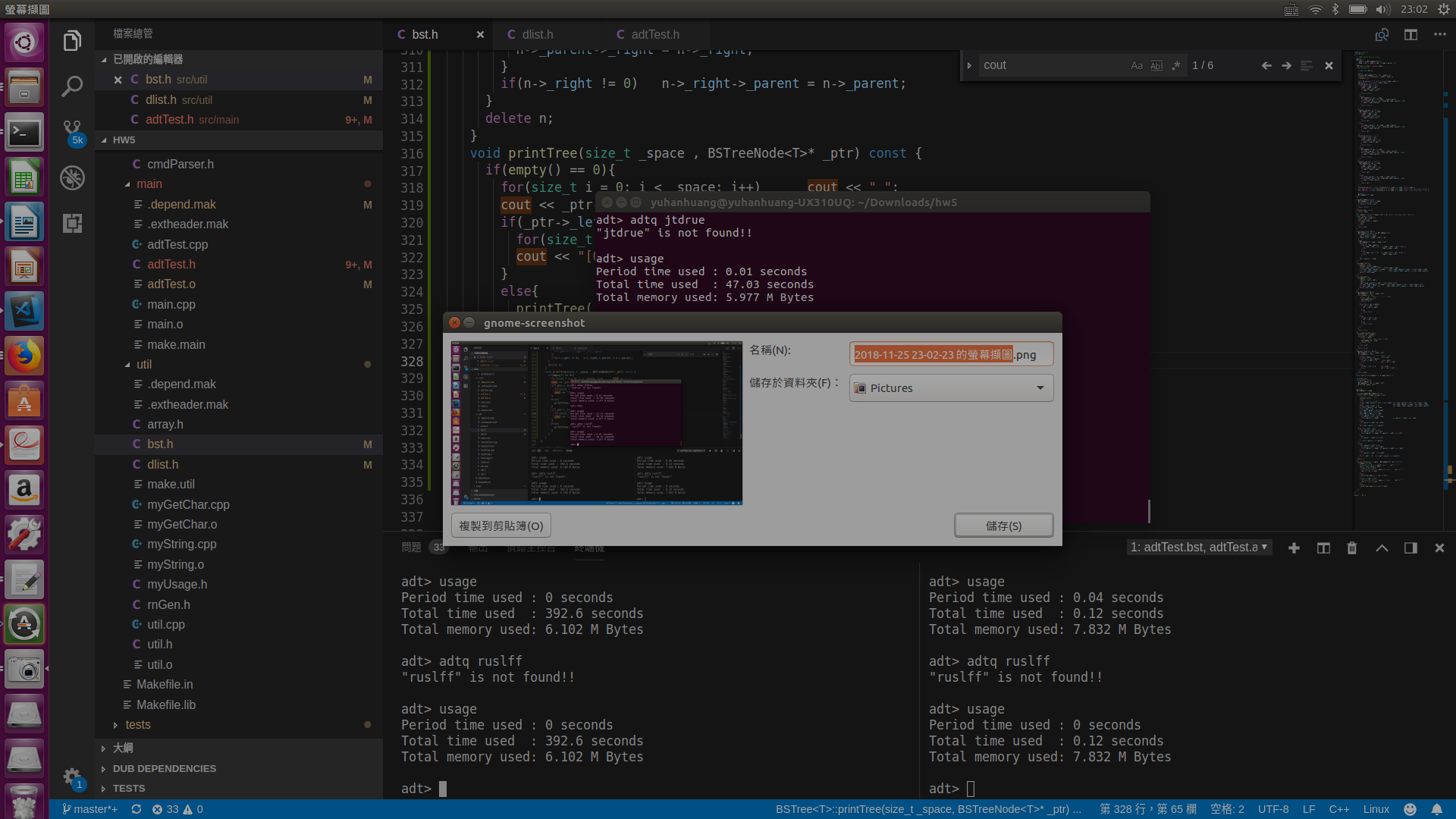
對於尚未sort()的find(const T& x)和erase(const T& x)而言，bst因為時間複雜度是logn，所以最快，array和dlist時間複雜度為n，所以慢了一點。



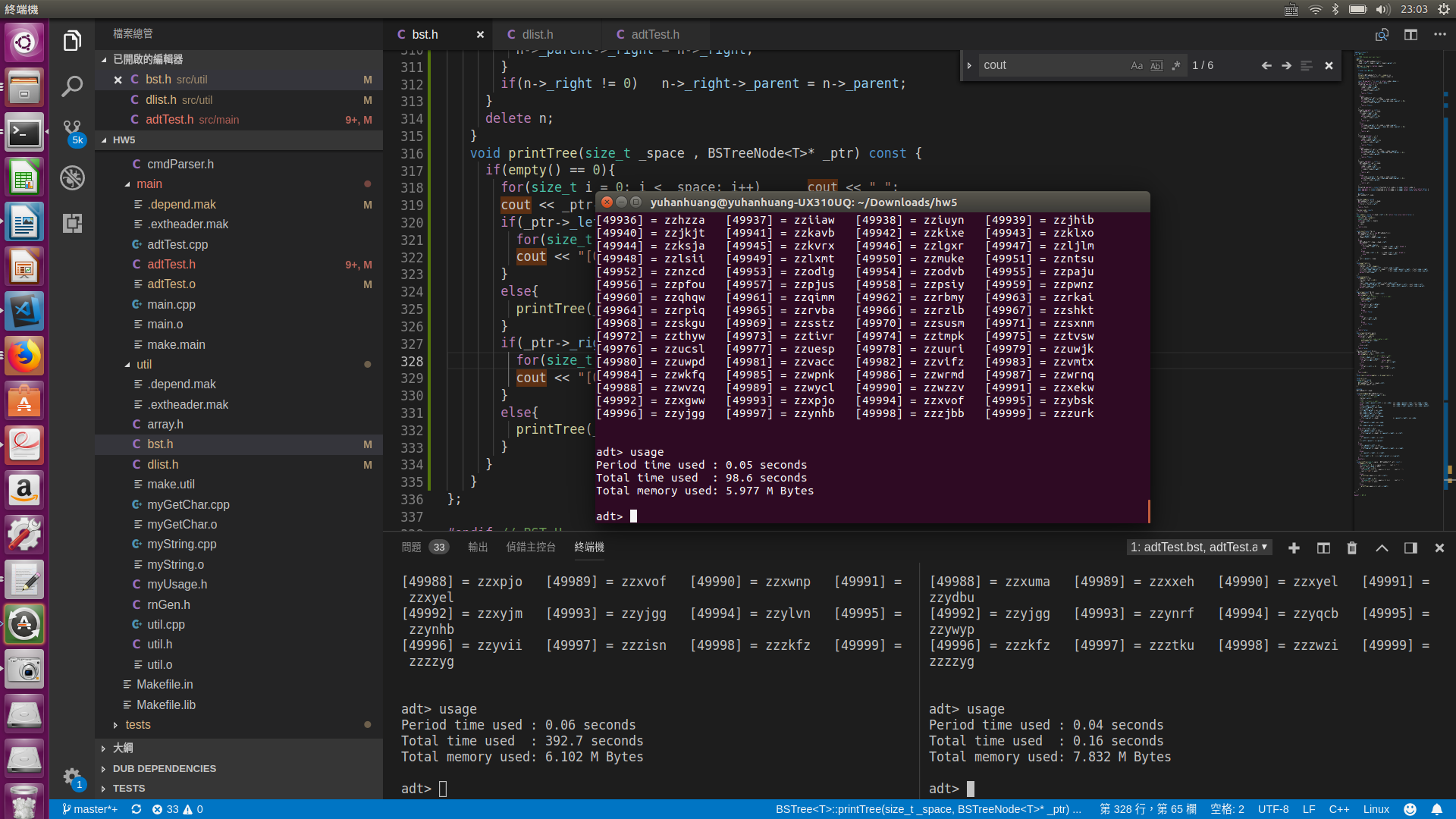
對於sort() 而言，bst因為一直維持著排序的關係，所以不用sort()最快，array 的sort()第二快因為其實間複雜度為nlogn，而dlist最慢，因為時間複雜度為n平方。



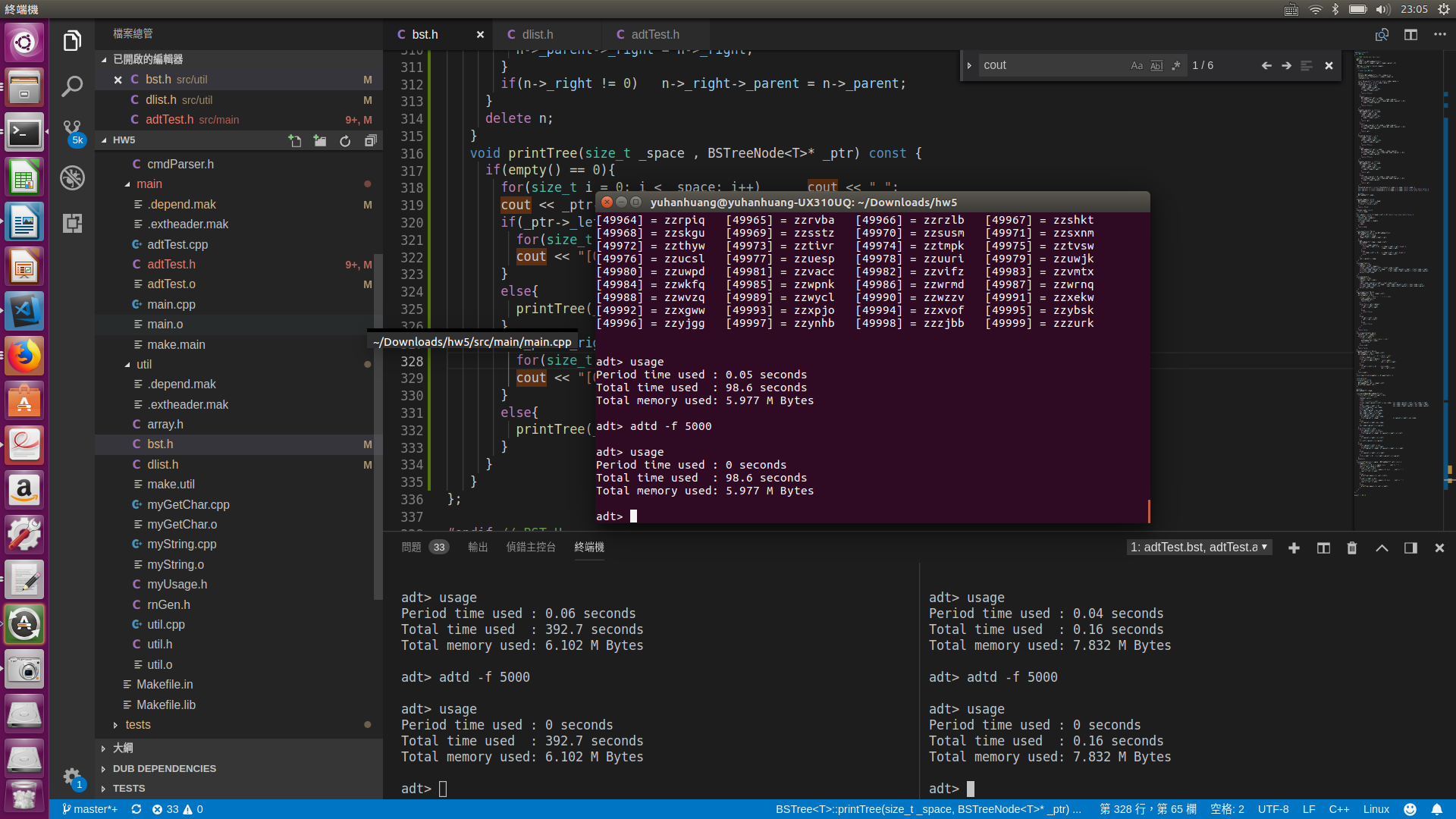
對於sort()後的find()和erase(const T& x)而言，由於array和bst都用時間度為logn的binary search，所以最快，dlist仍維持在時間複雜度為n的linear search，所以慢。



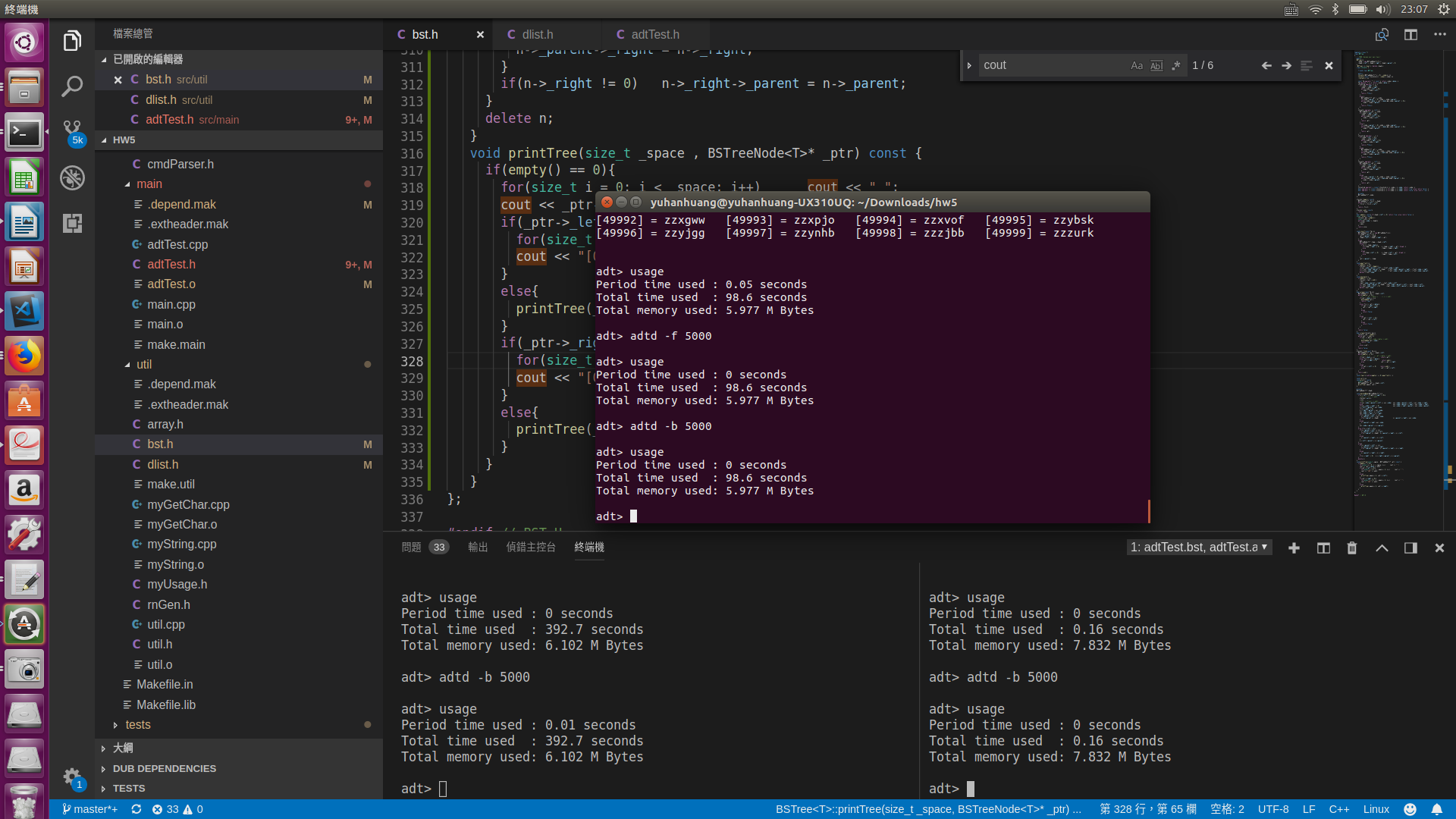
對於print() 而言，三者時間複雜度為n，故都差不多快。



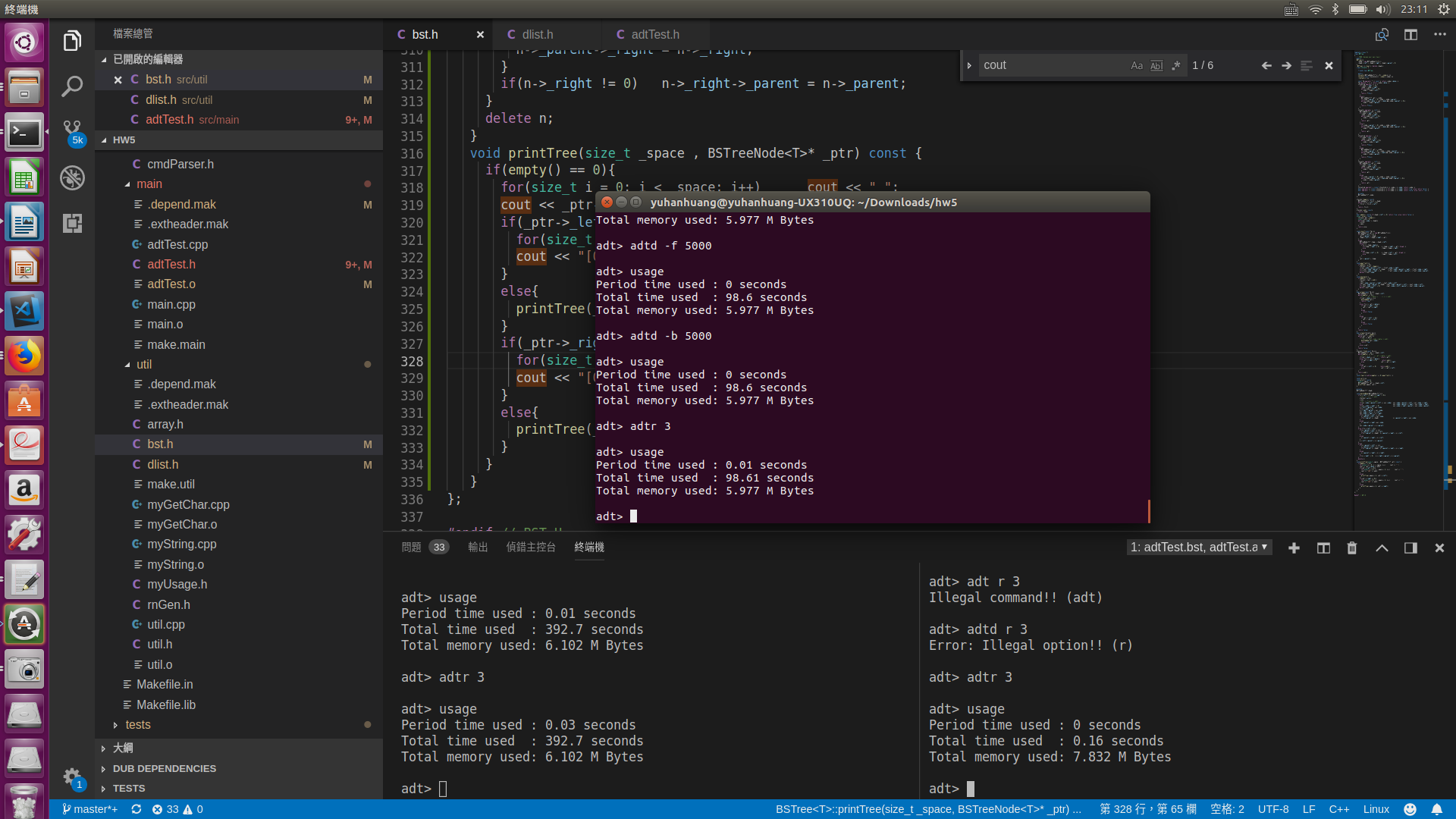
對於pop\_front()而言，dlist和array時間複雜度為1，bst為n，故差不多快。



對於pop\_back()和上述同



對於clear()而言，array時間複雜度為1最快，其次是dlist，其時間複雜度為n，bst最慢因為deletion要處理successor，即便時間複雜度也為n



以上便是我的觀察