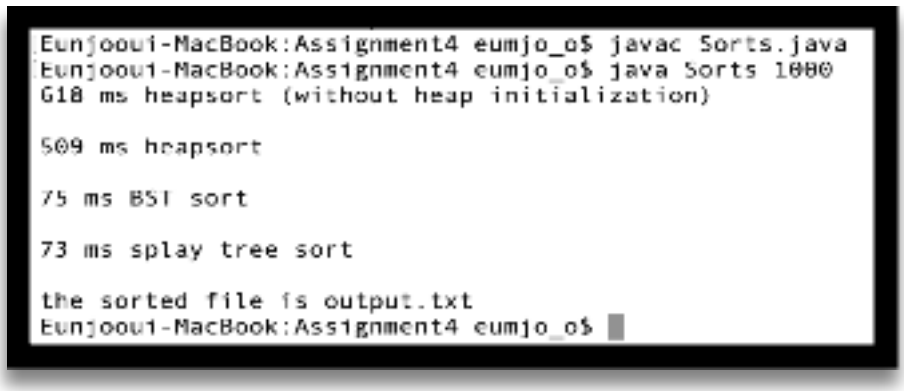
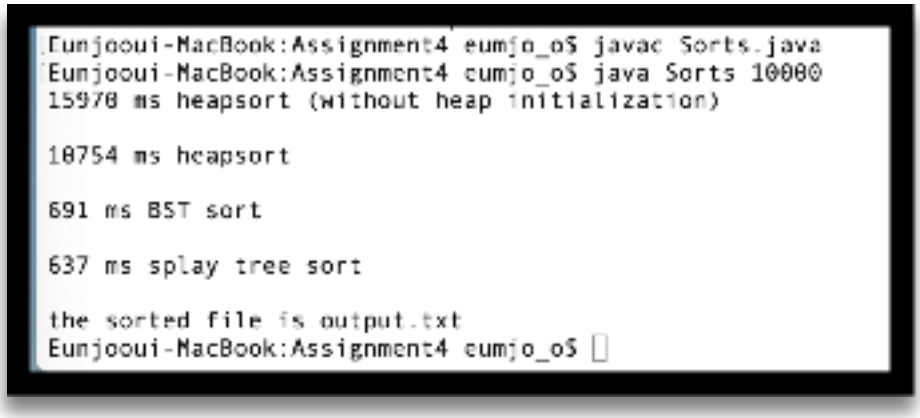
2014-19498 German Education
Jung, Eunjoo

Assignment 4 - Time Complexity Comparison.





1) Heapsort without heap initialization - > O(n log n)
2) Heapsort -> O(log n) <but average still O(n log n)
3) BST sort -> O(log n)
4) Splay sort -> O(log n)

Heapsort takes normally **O(n logn) time complexity** for the best and worst case. Both heap sorts are represented based on the array. For the 1)**heap sort without Initialization**, it takes about **O(nlogn)** time complexity and the 2)**heap sort with Initialization** average **O(logn)** time complexity.

For the first one, which is heap sort without heap initialization. First of all, I made an unsorted array with randomly generated Integers with an input Number(1000 <= 10000). And then with a Buildheap method I made a heap array(which is heap initialization).

For the second one, which is heap sort with heap initialization, we can simply add the elements repeatedly to the empty heap. When the elements inserted, then they should follow the heap property, so until every parent node

has its children nodes, whose keys are smaller than parents's. Next, for the heap sort with initialization, it occurs with an unsorted array, which has randomly assigned elements.

So for the creating and building heap it takes about O(n) time complexity, and for the Heap sort it takes about O(logn). The process is followed by 1) replacing the root item with the last index node. 2) I removed the last node(which was originally root node and the biggest Integer, but currently located at the last of the tree) and reduced the int size variable. Since the root key(which was at the last of the tree, but currently located at root) does not satisfy the heap array property, 3) it should be perculated down, until it satisfies the heap array property, so it keeps perculating down. When is satisfies the heap property, then keep doing the same precess(replacing the root key with the last key and perculate down -> 1),2),3) process) until the size variable becomes one. As a result, we can get the sorted (increasing order) array. When doing the Perculatedown with n elements the time complexity is O(log n), but when the size reduced, it gets n-1 elements, and consequently, O(log n!) time complexity and then O(n log n) time complexity.

For the **Binary Search tree** it takes about **O(log n)** time complexity. For the **Splay tree sort** it takes about **O(log n)** time complexity. Binary Search tree is the tree, in which left Child's value is smaller than parent's and right child's value is bigger than parent's value. For the inserting operation, it takes about O(log n) time complexity, and also for the searching operation, it takes about O(log n) time complexity. So, total O(2 log n) time Complextity for the 3) **BST sorting** algorithm. And averagy **O(log n) time Complexity**. For the splay tree, the Difference between the BST and the Splay tree is that the inserted node becomes the root of the Tree. It follows the same rule with BST. So it takes about **O(log n) time Complexity** for the searching and inserting opearations, in other words, for 4) **Splay sorting** algorithm.

For the searching operation, it takes O(log n) time Complexity, because the possibility that the tree can make the permutaions of n keys is n!. So, the Internal Path length can be devided by the nodes numbers and as a result O(log n) time complexity. But the worst case of Search operation time Complexity is O(n) and averagy O(log n). For the insertion operation, it takes O(n) time Complexity for the worst case and the average O(log n). As search operation, insertion operation it takes O(log n) because, the tree search occurs until it finds the right place for the Insertion.

Beacuse the splay tree does the self-optimizing making the inserted Node as root and for that operation doing zigzaging, it takes slightly better time Complexity than normal BST. So, frequently assecced node becomes the root and, that means it is easier to do other operations. The worst case Time Complexity is O(n) but, it is actually unlikely and, the average is O(log n) time Complexity.