Name: Kuan Jun Hao Jason
Admin No.: 201484K
Group: 2
PEM Group: SF2102

Sorting Algorithms

Introspective Sort

Introspective sort, also known as introsort or std::sort(); (C++'s in-built sort function that uses introsort), is a hybrid sorting algorithm with a time complexity of $O(n \times log_2(n))$ that combines the efficiency of quicksort, heapsort, and insertion sort.

This was challenging to implement this in Python as most of the Python codes available had some performance flaws such as doing a heap sort on the entire array instead of a sub-array. Hence, I had to refer to a popular GitHub repository and the GNU C++ stl_algo.h source code.

Introsort will be useful in sorting records as the number of records grows in size and does not need to maintain the exact order for duplicate package names as it is not stable. The program will then use introsort to pre-sort the array before doing a Fibonacci search when the staff user wants to see all the records by package name.

Heap Sort

Heap sort, which has a time complexity of $O(n \times log_2(n))$, works by building a max-heap (sorting in ascending order) which is a complete binary tree with its own property and keeps calling the function to build a max-heap and swapping the element at index 0 and the tail of the array as it reduces the heap size.

This was quite challenging to implement as I had to learn what is a binary tree, a complete tree, the property of a min and max heap, and the type of traversals in the tree represented in an array. Furthermore, it was difficult to visualise what was happening.

Heap sort will be useful if there are many records and can sort the array faster. The application will do a heap sort before doing a binary search when the staff user wishes to edit/delete a record by package name.

Tree Sort

Tree sort that has a time complexity of $\Theta(n \times log_2(n))$ requires a binary search tree and doing an inorder traversal (left→visit→right) to sort the array. By using object-orientated programming, I implemented a type of binary search tree, specifically an AVL tree, to avoid the worse case of $O(n^2)$ in an unbalanced tree when executing the tree sort algorithm.

It was difficult to understand the inorder traversal as it uses recursion but as I implemented and kept testing it with demo codes, I got comfortable with recursion and understood how tree sort works.

In my program, whenever the staff user wants to delete a record, it will use this algorithm, tree sort, to pre-sort the array by customer name before doing an exponential search which is useful if the number of records grows.
Furthermore, in my program, its time complexity is actually $O(n)$ as I stored the AVL tree as well instead of inserting all the elements before doing the inorder traversal.

Radix Sort

Radix sort is a type of non-comparison sort that uses counting sort but some modifications to it. Its usefulness comes in its linear time complexity of $O(d(n + b))$ where b is the base number 10, and d is the number of digits in the largest number, which can be significantly faster than $\Theta(n \times log_2(n))$ sorting algorithms for larger arrays.
The implementation was not too difficult but I had a hard time visualising how it worked.

I have implemented radix sort such that whenever the staff user wants to search for records within a specified cost per pax range, it will pre-sort the array using radix sort before doing a binary search.

Shell Sort

Shell sort is a sorting algorithm that has a time complexity of $\Theta(n \times log_2(n))$ but can have the worst case of $O(n^2)$. It is similar to insertion sort but with some modifications.
The implementation was not too bad after I understood what were the differences between insertion and shell sort.

It sorts the elements that are far apart from each other and progressively reduces the interval gap between the elements to be compared. This effectively means that shell sort has to do less shifting as compared to your normal insertion sort which makes it faster and useful for sorting large arrays.
Since the minimum requirements of this assignment did not have a sort option for pax number, I added shell sort to sort by pax number if the staff user wishes to do so.

Search Algorithms

Fibonacci Search

Fibonacci search, similar to binary search, uses the Fibonacci sequence to decide which sub-arrays to look in as it uses the divide and conquer strategy. It was quite easy to implement but I did take some time to understand it because the logic behind it was not as vivid to me.

The advantage of this search algorithm is that it does not use the division operator which can be costly on the CPUs on older computers when doing a binary search.
However, with faster CPUs nowadays, the performance difference can be considered negligible but it is still interesting to use the Fibonacci number to solve a problem.

I implemented Fibonacci search in my program whenever the staff user wants to see all records by package name after pre-sorting the array with the introsort algorithm.

Exponential Search

Despite its name, exponential search is basically binary search but with a modification to it to improve binary search's efficiency. It was quite easy to implement as I reused my binary search code.

Its advantage is that if the element to be found is at the front of the array, the exponential search algorithm will outperform your typical binary search algorithm, especially if the array is large in length.

When the staff user wants to delete a record by customer name, instead of just using linear search, the program will do an exponential search after pre-sorting the array using tree sort.

Data Structures

AVL Tree

When I first implemented a binary search tree, I realised that it can be skewed either to the left or right depending on the value of the root node and its child nodes. This will then make the insertion, deletion, and searches have the worst time complexity of $O(n)$ instead of $O(log_2(n))$. Hence, I implemented an AVL tree to make my tree balanced and to ensure $O(log_2(n))$ operations and to avoid Python's max recursion depth error if the number of records grows a lot.

How an AVL tree balance itself is by doing rotations which is like cutting and pasting nodes to balance itself whenever we insert or delete a node which was the only challenging part when implementing it. The rotations were difficult to visualise but after looking at visualisations and videos explaining about it, I now know how the rotations generally work.

The AVL tree is used when the staff user wants to see all records by customer name or for the tree sort algorithm to avoid its worst time complexity. My program will store the array that holds the records and also the AVL tree which has all the nodes ordered by customer names.
This allows $O(n)$ time complexity for the tree sort algorithm since there is no need to put all the elements into the tree again. Additionally, it allows the staff user to search for customer names in $log_2(n)$ time.

Doubly Linked List

A doubly linked list is similar to a normal array but the elements are linked together like a chain.
In a doubly-linked list, each node has a pointer to the next and the previous node which allows a bidirectional traversal.
Since I have implemented a doubly linked list in C++ before, it was relatively easy to implement it in Python.

For my program, I also needed some way to keep records of the same key in the AVL tree as a binary search tree cannot have duplicate nodes of the same key. Hence, using a doubly-linked list to allow $O(1)$ deletion of any element in the linked list.

Bad Sorting Algorithms

Implemented to show how these sorting algorithms below are "bad" and are meant as a joke.
It did not take too long to implement these sorting algorithms as each of them was simple to understand.

Bogosort/Bozosort

These sorting algorithm basically just does permutations of the array and stops whenever the array is sorted. This means that it can take hours, months, or even years to sort a large array.

Sleep sort/Stalin sort

Both sleep sort and stalin sort has its unique way of sorting.
Sleep sort uses multiple threads that append to the array after sleeping for a time that is proportional to the element's value.
Stalin sort on the other hand sorts by removing elements that are not in order.
However, this is not what we usually want as data can be lost which is where it got its name from.

Pancake sort/Gnome sort/Slow sort

Sorting algorithms that works but are very slow compared to bubble, insertion, and selection sort.
From implementing these three sorting algorithms, I got to know why they are bad on purpose and a rough idea on how to avoid them if I were to develop my own algorithms.
Generally, they have too much shifting/swapping and slow sort is bad on purpose as it does the opposite of the divide and conquer strategy.