**Data Structures**

**And**

**Algorithms**

**3D5A**

**Assignment 2**

Sorting

**By**

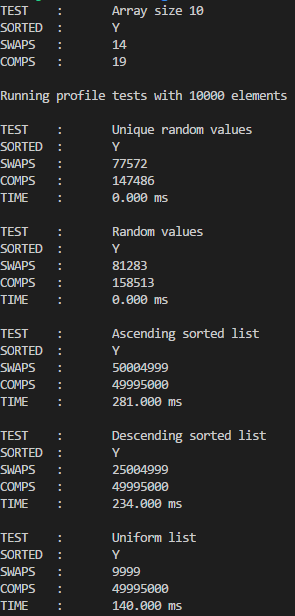
Robert Trew

15315527

**Lab Date**

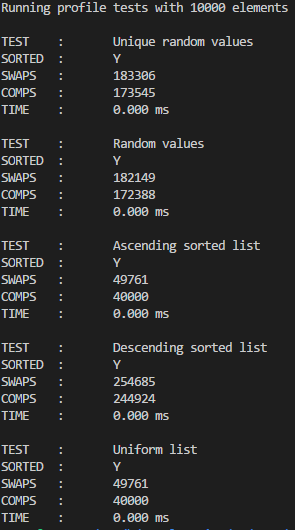
17/11/2017

**Task 1**



Using quicksort, it can be seen that Quicksort performs best in unordered arrays (randomly filled), best and average cases, O(n log(n)). In fully sorted arrays (i.e. ascending and descending) it experiences its worst performance O(n2). The uniform list is almost an in between case, but tends towards the worst case, namely in terms of comparisons made.

**Task 2**

I chose to implement Tim Sort, which is a hybrid of merge and insertion sort.

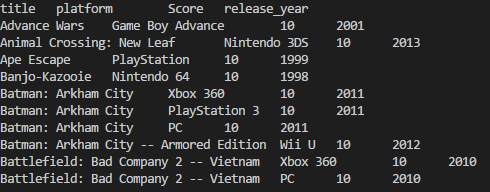
The algorithm works by splitting the array into many small “runs” and performing insertion sort on these. Insertion sort is used as it reduces overhead (compared to fancier algorithms), performs well on small sets of data and is a stable sort.

These runs are then passed through to merge sort, which will then merge the sorted runs into a fully sorted array. Merge sort works well as it pairs nicely with the work already done by insertion sort. My choice of merge sort is stable, so the overall Tim Sort is a stable sort.

When comparing the values from Tim Sort and Quicksort, it can be seen that Tim Sort is superior in both speed and computations, Tim Sort has a best case O(n), while worst and average cases are O(n log(n)).

However, Tim Sort has larger space (memory) requirements than Quicksort, in particular, due to the creation of arrays in the merge sort. Also, it must be mentioned that quicksort is easier to implement. In conclusion, as long as there is not too much restriction on available memory, Tim Sort is the superior sort in terms of performance.

**Task 3**



Using Tim Sort, the above is the list of the 10 most popular games. As Tim Sort is stable, the alphabetical order is preserved. Popularity was determined by score (i.e. 10/10).

To sort the games based on score and year, I would first sort the games by year (to group all the same years together), then within that I would sort by score. So that all the same years are grouped together, however, this may prove tricky to implement.

An alternative may be to sort all the games by year, then split the full array into smaller arrays of the specific year, performing insertion sort on these would lead to the desired result, these “year” arrays could then be recombined (as the order is known), or kept separate. This would be easier to implement, but less elegant.