TASK 1

Bubblesort is a sorting algorithm that compares two consecutive elements in a data structure and manipulates them by swapping the two elements around if the first element is larger than the second element (assuming a list is being sorted in ascending order). This will continue up to the last two consecutive elements; this is considered one “passing” after the last two are compared. This guarantees that the largest element is at the very end of the list at the first passing since the bigger element always gets passed in the direction of the end of the structure. On the second passing, the consecutive elements can be compared up to the second-last and the third-last elements; the iteration of this algorithm continues until it either it has executed (n-1) passings (where n is the size of the data) or if it cannot detect any swaps made in a particular passing.

1. “Bubble Sort.” Internet: <http://www.algolist.net/Algorithms/Sorting/Bubble_sort>, [October 5, 2012].
2. “Bubble Sorting in Java” Internet: <http://www.roseindia.net/java/beginners/arrayexamples/bubblesort.shtml>, May 28, 2007 [October 5, 2012].

**Psuedo-code.**

Let “list” be an array with elements that have a conventional ordering to them.

Let n be the size of the list (positive integer)

FOR(initialise i at 0, run it as long i<n, increment i by 1)

Let swapCheck be equal to the count of swaps in a pass

FOR (initialise ‘j’ at 1, run it as long j < (n-i), increment ‘j’ by 1)

initialise a variable ‘t’ for temporary storage of a value

IF spot in (j-1) of list > spot in (j) of list THEN

SET t= list[j],

SET value of spot (j) equal to the value in spot (j-1)

SET value of spot j-1 equal to the value of t

INCREMENT swapCheck by 1

END IF

IF swapCheck is equal to zero break out of the loop

ENDFOR

ENDFOR

|  |  |  |
| --- | --- | --- |
|  | Comparisons | Swaps |
| i=0 | 4 | 4 |
| i=1 | 3 | 3 |
| i=2 | 2 | 2 |
| i=3 | 1 | 1 |

So there are 10 comparisons made and incidentally 10 swaps made for the set {10, 9, 8, 3, 1}.

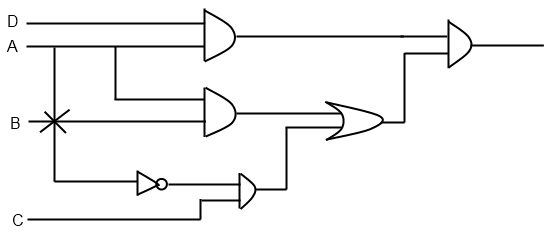
TASK 2

|  |  |  |
| --- | --- | --- |
|  | Quicksort | Insertion Sort |
| How objects with the same power value are handled | Quicksort chooses an arbitrary “pivot” point where anything less than the value of the pivot is moved to the left of the pivot (and the same for greater values to the right). This is done by swapping values that doesn’t satisfy those conditions with each other. After that is done, the list splits into two lists, none containing the pivot and restarts the algorithm. | This starts with the first two elements and compares the two and swaps them accordingly; after that is done, the third element is compared to predecessor elements until it hits a value that is less than it and then inserted in that spot. This continues until the last element is checked. |
| Space requirements | Since quicksort is defined recursively, it would have to allocate memory for each of the methods called within itself until they are finished. | Insertion sort compares elements on an iterative basis, which means it does not use as much memory as quicksort. |
| Worst case number of comparisons (of size 10) |  | The number of comparisons done would be 1+2+3+...+(n-1) = 45 |
| Worst case number of swaps (of size 10) |  | The number of swaps would also be 45. |
| Ease of programming | Recursive algorithms are harder to program than iterative ones. | Iterative algorithms are easier to program than recursive ones. |

Overall, the insertion sort should be used. It is easy to program into the game and there is always going to be the situation where you only have to deal with sorting one object at a time as opposed to many things at the same time. You would have a chance of not having to sort the list at all when a new object is acquired (highest stats). For the worst case, you would have to iterate (n-1) times to place the new object (lowest stats) in the already sorted list.

TASK 3

**Logic Gate Diagram**



**Truth Table**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| A | B | C | D | A^B | C^(NOT-A) | (A^B) OR (C^ (NOT-A)) | A^D | **[**(A^B) OR (C^ (NOT-A))**]**^**[**A^D**]** |
| T | T | T | T | T | F | T | T | T |
| T | T | T | F | T | F | T | F | F |
| T | T | F | T | T | F | T | T | T |
| T | T | F | F | T | F | T | F | F |
| T | F | T | T | F | F | F | T | F |
| T | F | T | F | F | F | F | F | F |
| T | F | F | T | F | F | F | T | F |
| T | F | F | F | F | F | F | F | F |
| F | T | T | T | F | T | T | F | F |
| F | T | T | F | F | T | T | F | F |
| F | T | F | T | F | F | F | F | F |
| F | T | F | F | F | F | F | F | F |
| F | F | T | T | F | T | T | F | F |
| F | F | T | F | F | T | T | F | F |
| F | F | F | T | F | F | F | F | F |
| F | F | F | F | F | F | F | F | F |

TASK 4



John Horton Conway is a mathematician from the U.K., known for his achievements in many branches of mathematics; one of his famed accomplishments was the creation of “The Game of Life”, a model that replicates bacterial growth (or decay) in a Cartesian graph under certain parameters. The game was based on John von Neumann’s work, which mathematically models a hypothetical machine that replicates copies of itself on a Cartesian graph, but with more parameters.

1. “John Horton Conway.“ Internet: <http://www-history.mcs.st-andrews.ac.uk/Biographies/Conway.html>, 2004 [October 5, 2012].
2. G. Martin (1970,Oct.). “The fantastic combinations of John Conway's new solitaire game life.“ Scientific American. [Online]. 223, pp. 120-123. Available: [http://web.archive.org/web/20090603015231/http://ddi.cs.uni-potsdam.de/HyFISCH/Produzieren/lis\_projekt/proj\_gamelife/ConwayScientificAmerican.htm](http://web.archive.org/web/20090603015231/http:/ddi.cs.uni-potsdam.de/HyFISCH/Produzieren/lis_projekt/proj_gamelife/ConwayScientificAmerican.htm) [October 5, 2012]