Assignment 9

Exercise 1

a)

The list of numbers in table a) shows our example unsorted list. The first thing we do is select an arbitrary pivot value which for this example I will use the first element 5 shown in red in table b). Next, we move from the left of list (immediately right of the pivot) searching for the first number larger than the pivot and from the right looking for the first number lower than the pivot. These are represented in bold in b); 6, 2. In c) we swap these two numbers and repeat the searching process from either side starting from the numbers just swapped (2, 6) as we know we have already gone through the preceding numbers on the first iteration. We find 4 and 7 in the c) table but we have also crossed over for the first time whilst searching which means after we have swapped these two items we have been through all of the values on either side that are eligible for sorting with the current pivot.

a)	5	1	3	6	7	4	8	2
b)	5	1	3	**6**	7	4	8	**2**
c)	5	1	3	2	**7**	**4**	8	6
d)	5	1	3	2	**4**	**7**	8	6
e)	5	1	3	2	**4**	7	8	6
f)	4	1	3	2	5	7	8	6
g)	4	1	3	**2**	5	7	**8**	**6**
h)	2	1	3	4	5	7	**6**	8
i)	2	**1**	3	4	5	6	7	8
j)	1	2	3	4	5	6	7	8

In d) we swap the 4 and 7 and now either side of these numbers the list is sorted in relation to the pivot. Finally, we swap the 4 with the pivot (5) which is now in its sorted position as shown in f) in green; all the values to the right are larger and all the values to the left are smaller. In the next step, g), the same logic is applied to either side of the previous pivot using a new pivot in each division of the list i.e. either side of the 5. Selecting the first value again provides us with the pivot 4 in the lower subsection and 7 in the upper subsection. Searching through the values as we did previously we find no values higher than 4 (this is an example of worst-case scenario for this step) so it is swapped with the rightmost value i.e. 2. For the upper list we find 6 and 8 so we swap them and that is all the values of this subsection searched through so we swap the 6 and the 7 to put the pivot in its final sorted position. We finish by performing the final iteration in the lowest subsection using 2 as the pivot and then the list is sorted.

b)

The stability of a sorting algorithm is concerned with how it treats equal or repeated elements. Stable sorting algorithms preserve the original order of such elements equal to

each other despite the sorting of the surrounding elements. Unstable sorting algorithms do not maintain this order of equal elements.

5	1	3	1'	7	4	8	2
5	1	3	1'	7	4	8	2
5	1	3	1'	**7**	4	8	**2**
5	1	3	1'	**2**	4	8	**7**
4	1	3	1'	2	5	8	**7**
2	1	**3**	**1'**	4	5	7	8
2	1	**1'**	3	4	5	7	8
1'	1	2	3	4	5	7	8

This diagram follows almost the same sequence but with two cells with the value of 1. To signify the difference, I have marked the first **1** as normal and the second as **1'.** If this algorithm was stable the order of these two values would be maintained i.e. leftmost 1 and rightmost 1' but as we can see the 1' ends up on the left of the 1. This is because of the second to last stage where the pivot is 2 (marked in red). The 3 and the 1' have just been swapped and that is the last of the search using 2 as the pivot. Therefore, following the logic from the previous examples, we swap the pivot with the rightmost value which is lower than the pivot. In this case that is 1' bringing it to the other side of the 1, upsetting the original order of these values. This capacity makes quick sort an unstable algorithm.

Exercise 2b)

With quick sort the choice of pivot determines what kind of time complexity we will achieve. If we pick a pivot that is either the smallest or largest element in a sub-list we have to go through the entire length of that list for the iteration. This might be more likely if the list is partially ordered and we are selecting the first element for instance. Random shuffling, on the other hand, makes the chance of choosing the worst-case pivot far less likely on average.

Exercise 3a)

In the worst-case scenario, we would choose a pivot that splits the list into two subsections of lengths 1 and n-1. Then we would do the same for the remaining subsection, which is now length n-1, and this would result in two lists of length 1 and n-2. We would end up with a search tree of n levels. For each of these levels we are searching through each element resulting in n levels of searches of size n, n-1, n-2, n-3.....2, 1. Expressed as an equation this will include several coefficients the dominant of which will be $n^{2.}$ due to the multiplication of n coefficients. As we know from big-*O* notation the dominant form is what establishes the time complexity of an algorithm so quicksort has big-$O(n^2)$.

Exercise 3b)

In the best-case scenario, we would partition the sequence so that they were as evenly balanced as possible i.e. either perfectly evenly or, in the case of an odd number of elements, with a difference of one. This would result in a perfectly balanced tree of subsequences. As each list would be evenly split and as this is done recursively this would only require $\log_2(n)$ steps to get to the smallest subsections. For each of these divisions we know that there would be approximately n comparisons as each element of every subsection is compared against the pivot value of that subsection. So as there will be roughly n comparisons for $\log_2(n)$ steps we know that the time complexity for the best-case scenario is $O(n\log_2(n))$. This is the best-case scenario as there is no more efficient way of dividing the subsections that in 2 resulting in the $\log_2 n$ value and that even without switching the values they have to be compared for each subsequence resulting in n comparisons for each recursive step.