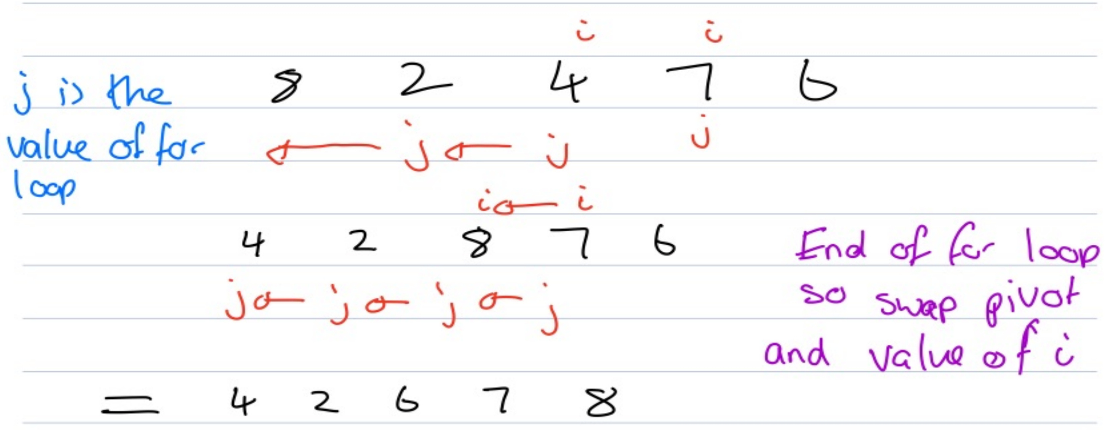
Assignment 2 report – Sorting – 3D5A

Part1

In question 1 it asks us to use implement quicksort and on array on size of 10 values. Before I did any coding, I had to decide whether my pivot was going to be the first value or last value. I choose the last value in the array to be my pivot point. Originally when I started coding this, I thought why not use 2 for loops to compare the values inside the array and get a suitable place for the array to be partitioned. However, this way would be very inefficient. So, I thought is there any way that I can just use one for loop. How it works, is that it takes the last element in the array as the pivot and then two counters will then take the value of (last_element-1). One of these counters will be the for loop. Then every time the array[counter] is more than the pivot, you swap the values of array[loop_value] and array[counter] and then counter will have a value of 1 taken away. This will be repeated until the loop reaches the end at which point the pivot and array[counter+1] will be switched. Hopefully this diagram will help explain it better:


j is the value of for loop
i i
8 2 4 7 6
j j j
i i
4 2 8 7 6
j j j j
End of for loop so swap pivot and value of i
= 4 2 6 7 8

Then to compare each of the different arrays, we are using swap and comparisons. I have also incorporated the time it took for the computer to sort the array[1]. I think adding this function will help gave greater inside into how fast quicksort is at sorting the array. When looking at this and the number of swaps to sort the array, the uniform array both had the lowest number of swaps and the lowest time to sort. However, on the other hand the array which had no duplicates and not sorted took the longest time to sort in a time of 0.000121 seconds. Then the array descending took the most swaps to sort however it did not take the longest, it had a total of 34 swaps. I also tested for the amount of comparisons it took to sort the array, with the uniform, descending and ascending arrays taking the most with 45 comparisons and with the array no duplicates taking the least with 22. The results that I found would tend to agree with the discussions that we had in class.

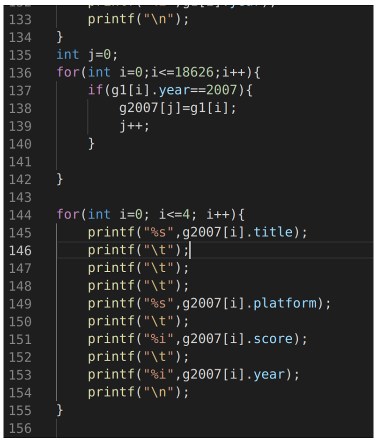
As part of the assignment we also had to test our quick sort on a size of 10,000 array. However, the results drastically change as the duplicates array now takes the least time to sort. However, the uniform array and ascending array share the same lowest number of swaps which is 9999. With the descending array taking the greatest number of swaps (25009999) and it is also joint top of most comparisons with ascending and uniform array with a total of 49995000 comparisons. The descending array also took the longest time to sort taking 0.231559 seconds to sort.

Part2

For part 2 of the assignment I decided to use bubble sort. I choose bubble sort because quicksort is a very efficient way of sorting arrays and it would be very hard to implement something that would be better. So, using bubble sort would give me an insight into how good quicksort is. So again, using the 10,000 arrays and using bubble sort, my results confirmed how bad bubble sort is compared to quicksort. In terms of time it was taking a least 0.28 seconds to sort the arrays and the comparisons and swaps where all about 1000 times bigger. However, there was on improvement, this was that both the ascending and uniform arrays had no swaps but yet in part 1 they both had 9999 swaps. However, in general terms bubble sort preformed worse in every case compared to quicksort. I also had to use insertion sort in this part and it was very easy to implement.

Part 3

In part 3 I used quicksort to sort the struct of data. Most of this part of the assignment was done in assignment 1. However, the one thing that I found difficult was allocating all the memory for the struct of 19,000. So, used the word static in front of the declaration [2] and this solved my memory issue. Then I just had to change my swap function to incorporate the struct. Apart from that, part 3 was straight forward. For the second part of part3 it asks how you could get the top 5 games from every year. My solution to this would be: have a struct for every single year and then using if loops pass the values into in the correct struct and then simply print out the first 5 elements of each struct to get the best 5 games of that year. Here is some sample code below:



```
        printf("\n");
    }
    int j=0;
    for(int i=0;i<=18626;i++){
        if(g1[i].year==2007){
            g2007[j]=g1[i];
            j++;
        }

    }

    for(int i=0; i<=4; i++){
        printf("%s",g2007[i].title);
        printf("\t");
        printf("\t");
        printf("\t");
        printf("%s",g2007[i].platform);
        printf("\t");
        printf("%i",g2007[i].score);
        printf("\t");
        printf("%i",g2007[i].year);
        printf("\n");
    }

```

References:

[1] http://www.cplusplus.com/reference/ctime/clock/

[2] https://stackoverflow.com/questions/571945/getting-a-stack-overflow-exception-when-declaring-a-large-array