Q1.

The worst time complexity of the function is O(n), when the target is not found and we have to go through the loop completely. Space complexity is O(1), because it gets a fixed amount of memory regardless of the input size.

The best time complexity is $\Omega(1)$, because it happens when the target is found in the first iteration .

The space complexity is same, because it is not allocating more memory location or any new changes in the input size.

Q2.

a.

The time complexity 0f this code is O(n).

Because the total number of iterations of inner loop is equal to the sum of powers 2 from 1 to n. in other words: 2^0 +2^1 + 2^2 +…. 2 ^log2(n) and The outer loop iterates for log2(n) times.

b.

the time complexity of this code is O(n^2).

Because the outer loop iterates n times and in each iteration of outer loop, the inner loop iterates i times. The total number of iterations is equal to the sum of integers from 1 to n, which is equal to the formula n/2(1+n) = n/2 + (n^2)/2, which makes the time complexity of O(n^2) (because n^2 is bigger and more affective than n)

c.

the time complexity of this code is O(n log(n)).

The number of iterations of outer loop is log2(n), because i is doubled each time. Also, inner loop is also iterating for n times (from 1 to n) in each iteration of outer loop. So, the total number of iterations is equal to n*log2(n), so the time complexity is O(n log(n)).

Q3.

	1 Second	1 Hour	1 Month	1 Century
log n	≈10^{300000}	$3600*10^{6}$	$2592*10^{9}$	$31536*10^{11}$
√n	∞ (infinity)	∞	∞	∞
n	10^{300000}	$3600*10^{300000}$	$2592000*10^{300000}$	$3153600000*10^{300000}$
nlogn	10^{300006}	$3600*10^{300006}$	$2592000*10^{300006}$	$3153600000*10^{300006}$
n^2	$(10^{300000})^2$	$3600* (10^{300000})^2$	$2592000* (10^{300000})^2$	$3153600000* (10^{300000})^2$
n^3	$(10^{300000})^3$	$3600* (10^{300000})^3$	$2592000* (10^{300000})^3$	$3153600000* (10^{300000})^3$
2^n	2^10^{300000}	3600*2^10^{300000}	2592000* 2^10^{300000}	3153600000*2^10^{300000}
n!	12/3600	12	12*2,592,000	12*3,153,600,000

Explained my solutions for all questions in another file.

Q4.

The algorithm is similar to the quick sort, that we are trying to partitioning the array in 3 parts: red, blue, white.

1. Initialise three variables which points to the low, middle and high part of the array.
    - 'Low' point to the first (beginning) part of the array , in other words index 0
    - 'mid' also points to the beginning of the array (at start of the process), index 0
    - 'high' points to the ending of the array, index n-1 (n=number of elements in array)

2. Keep repeating below steps until ‘middle’ <= ‘high’ (this is a loop)
   We have three different cases (a,b,c), need to take different actions in each case.
   a. If the color of the key at ‘middle’ is red:
      - Swap key at ‘low’ with key at ‘middle’
      - Increment ‘low’ and ‘middle’ by 1
   b. If the colour of the key at ‘middle’ is blue:
      - Only Increment ‘middle’ by 1

   c. If the colour of the key at 'middle' is white:
      - Swap key at ‘high’ with key at ‘middle’
      - Decrement ‘high’ by 1

3. When the loop ends, everything is sorted. All the reds are before blues and all the blues are before the whites.

This is a linear algorithm, and the time complexity is O(n), n is the number of the keys in array.

Also the space complexity of this algorithm is O (1).

I guess this design is with the best space complexity, since it is O (1), so it is constant (algorithm has constant space)

The pseudo-code of this algorithm:

```
Keys (is the array of keys with values of red, blue and white)

 rearrange (keys){

            low =0

            middle =0

            high = length(keys)-1

      while middle <= high:

             if keys[middle] == “red”:

                 swap (keys, low, middle)

                  low = low +1

                   middle = middle +1

              else if keys[middle] == “blue”:

                      middle = middle +1
            else if keys[middle] == “white”:
                    swap(keys,middle,high)
                     high = high -1

return keys
```