P1: (1) D (2) 2 (3) 5

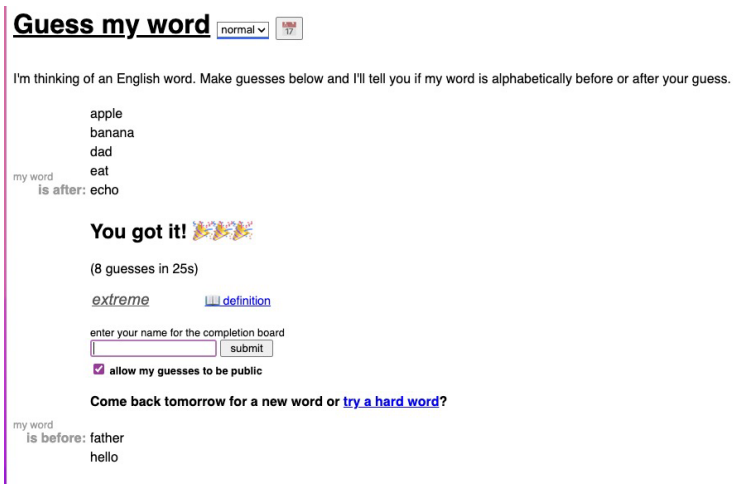
(4) 1536

(5). total swaps: 8

(6). 76

(a).

P2



(b) We can use binary search and set a pivot as a mid index, then we get the value of mid index and compare with our findword to check if this word is before or after our mid word. We can execute repeatedly until we find our word.

```
// Pseudocode using Binary search
findSercertWords(words, k){
    int left = 0, right = 2^k - 1;
    int mid = left + ((right - low) / 2);
    midword = words[mid];

    while(left <= right){
        if(wordfind < midword){
            right = mid - 1;
        }else if (wordfind > midword){
            left = mid + 1;
        }else{
            return wordfind;
        }
    }
}
```

Because we implement this question as a Binary search, so, we will guess the time is $\log(2^k - 1) = k$.

so, we can get k time guess at most.

(c).

As we know, the maximum guess number is $T(n)$, we can apply for the binary search algorithm, we will get $T(n) = T(n/2) + O(1)$

then, we applying the Master Theorem, we can know $a=1$, $b=2$, and we also have

$$n^{(\log_b^a)} = n^0 = 1.$$

so, we can get $T(n) = \theta(\log n)$.

(d). Because the average guessing time would be

$$\log_2^{(26751)} \approx 18$$

And $18 \cdot \$1 > \$15$.

so, you will lose the money.

second solution:

$$E(x) = P(\text{win}) \cdot 15 - P(\text{lose}) \cdot 1$$

$$= \frac{1}{267751} \times 15 - \frac{267750}{267751} \times 1$$

$$= -0.9999$$

so, we can get the same conclusion, lose the money.

P3

(a). Insertion sort will be faster.

Because all the elements in array are equal, we can consider it has been sorted.

So, time complexity will be $O(n)$.

(b). Heapsort is faster, and time complexity is $O(n \log n)$.

Because for quick, we always pick up the right-most element as our pivot, and Quicksort always put elements smaller than pivot in front of pivot, and put elements larger than pivot behind the pivot. and repeatedly execute smilar progress, so it's $O(n^2)$.

So, Heapsort is faster.

(c). Bucket sort is faster, time complexity $O(n+k)$.

Because the Bubble sort always compare adjacent elements, if the first is bigger, then swap, for this question, it will be $O(n^2)$, But Bucket sort will work best when the data are more or less uniformly distributed.

For the bucket sort, it depends on the time of sorting elements between buckets, and smaller the buckets are divided, the less elements there is between the buckets and the less time it takes to sort.

So, Bucket sort is $O(n+k)$.

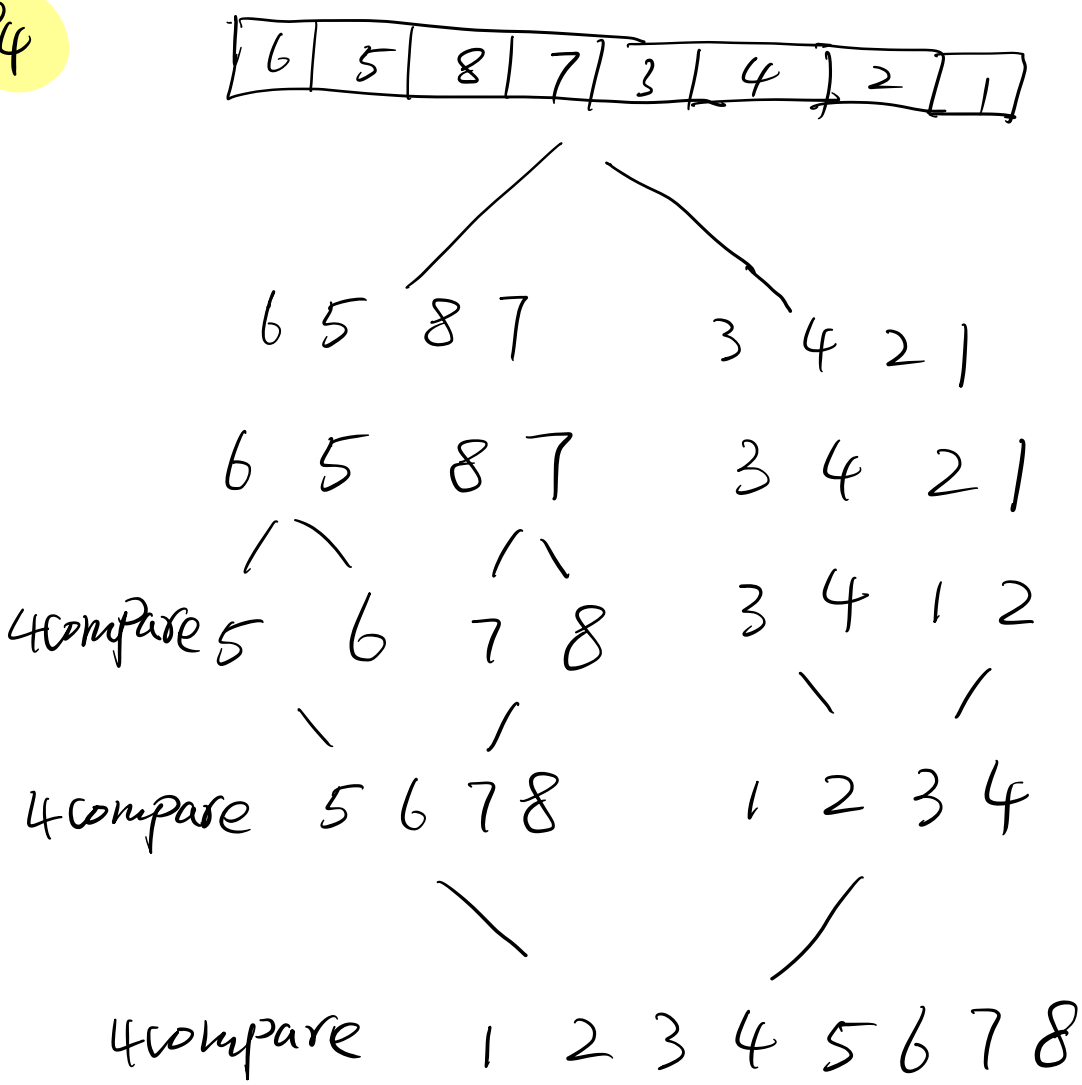
(d). Counting Sort is faster.

Because the input elements are fixed and it's n integers between 0 and k, so the running time is $O(n+k)$.

So, time complexity is $O(n+k)$ for Counting Sort.

(a). A=[6,5,8,7,3,4,2,1].

P4



So, we have 12 comparsion in total.

(5-6, 7-8, 3-4, 1-2)

(5-7, 6-7, 1-3, 2-3)

(1-5, 2-5, 3-5, 4-5).

(b). when an array has even number of elements. it will be divided into two equal length subarrays, and each of them will have minimum comparsion $M(n/2)$, and the elements of two subarrays will be compared between themselves.

when they will compare between themselves, the minimum comparsion will happen when all the elements of one array is either less or greater than each element of the other array, the number of comparsions between themselves will be $n/2$, and because they are $n/2$ length each.

so, the total number of comparsions using Merge Sort will be $M(n) = 2M(n/2) + n/2$.

(C). We can try to prove it by induction.

First, we assume $M(n) = \frac{n \log n}{2}$,

we can take for $n = d$, d is a constant.

so we have $M(d) = \frac{d \cdot \log d}{2}$.

We can also let $n = 2d$,

so, we have

$$M(2c) = 2 \cdot M(c) + c$$

$$= 2 \cdot \frac{c \cdot \log c}{2} + c$$

$$= c \cdot \log c + c$$

$$= 2c\left(\frac{\log c + 1}{2}\right)$$

$$= 2c \cdot \frac{(\log c + \log 2)}{2}$$

$$= 2c \cdot \frac{\log 2c}{2}$$

$$= 2c \cdot \frac{\log 2c}{2} = \frac{2c \cdot \log 2c}{2}$$

then, we can find $M(n) = \frac{n \log n}{2}$, takes for $n = 2c$.

so, we can conclude that

$M(n) = \frac{n \log n}{2}$ is proved.

(d). From the question, we know total numbers of input = 8.

P(exactly 12 comparsions) = 1.

Probability of getting single comparsion = $\frac{1}{2}$

So, we can apply for binomial distribution:

$$P(X=x) = n \cdot C_x p^x q^{n-x}$$

$$= 12 \cdot C_8 \left(\frac{1}{2}\right)^8 \cdot \left(\frac{1}{2}\right)^4$$

$$= 12 C_8 \left(\frac{1}{2}\right)^{12}$$

$$= 495 \cdot \frac{1}{4096}$$

$$= 0.1208$$

So, the probability proof getting exactly 12 comparsions in merge sort will be 0.1208.

(a). Greedy algorithm:

PS

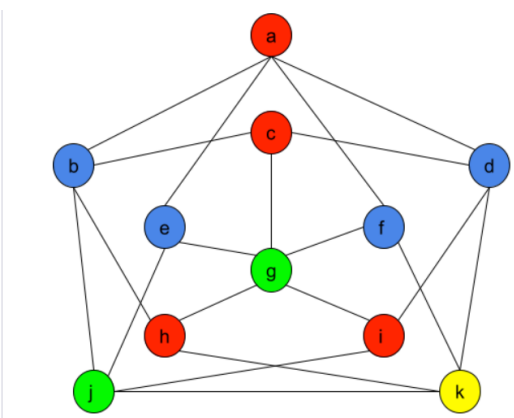
① Color first vertex like vertex a with first color;

② Do following for remaining V-1 vertex, considering the currently picked vertex, then color it with the lowest numbered colour that has not been used on any previously colored vertices adjacent to it. If all previously used color appear on vertices adjacent to V, which means we have to assign a new color to it.

③ Repeat the previous step until our all vertices are colored.

By applying the Greedy algorithm, we can get:

$\chi(G) = 4$.



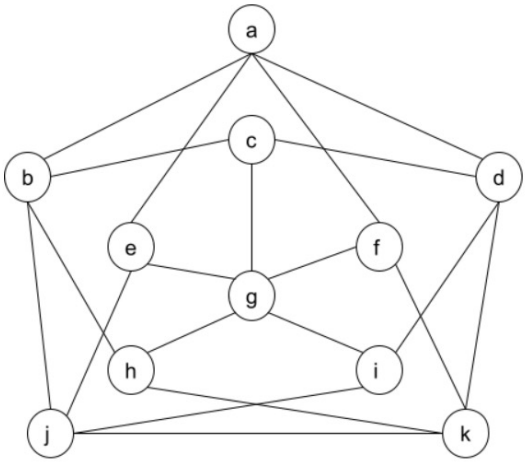
(b). Greedy Algorithm:

① Color a vertex with first color;

② Picked an uncolored vertex V. Color it with the lowest numbered color that has not been used on any previously colored vertices adjacent to V. If all previous used colors appear on vertices adjacent to V, we have to assign a new color to it.

③ Repeat the previous step until all vertices are colored.

I will take an example to show the color order and how my algorithm works.



① Consider the given graph, apply the graph coloring from vertex a, color the vertex to Red.

② Consider the next vertex b, as vertex b is directly connected with vertex a, so color the vertex b to Blue.

③ Consider the next vertex c, since c is directly connected with vertex b, so color vertex c to Red.

④ Consider the next vertex d, which is directly connected with a and c, so color the vertex d to Blue.

⑤ Consider the next vertex e, which is directly connected with a, so color vertex e to Blue.

⑥ Consider next vertex f, same as vertex e, we color it to Blue.

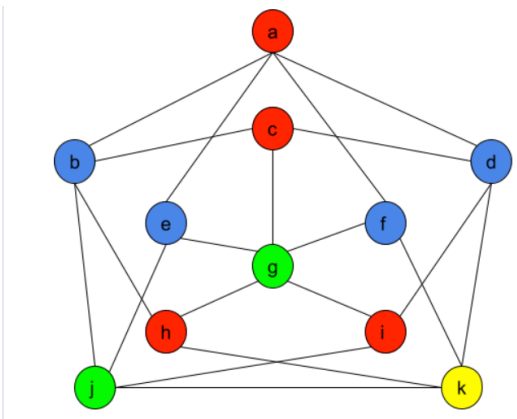
⑦ Consider next vertex g, since g is directly connected with c, e, f, which have been colored, so we assign a new color Green to g.

⑧. Consider the next vertex h, which is directly connected with b and g, so we color the vertex h to Red.

⑨. Consider the next vertex i, which is directly connected with d and g, so we color vertex i to Red.

⑩ Consider the next vertex j, which is directly connected with vertex b, e and i, which have been colored before, so we color vertex j to Green.

(iv) Consider the next vertex K, since K is directly connected with vertex d, f, h and j, so the color of vertex K will not be Blue, Red, Green. So, we will assign a new color Yellow to vertex K.

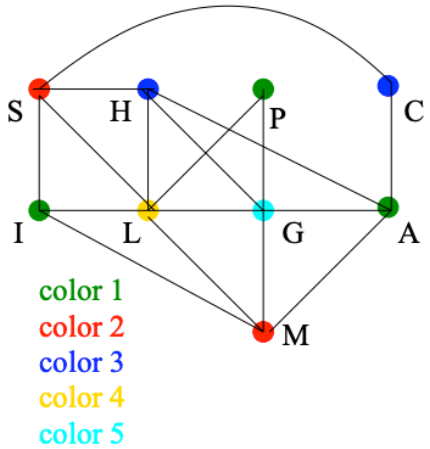


So, from the above graph, it is clearly that the greedy algorithm needs 4 colors to color this graph.

(c). NO.

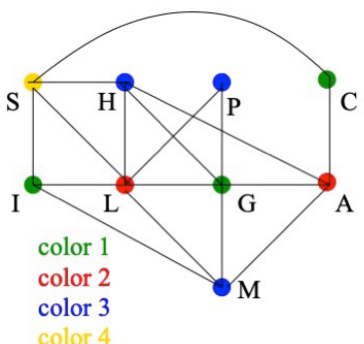
For some graphs, the algorithm does not always use exactly $\chi(G)$ colors.

Suppose we choose to color the vertices in order A, I, P, M, S, C, H, L, G. First we will color A with color 1 (green) and also I and P color green. finally, we can get the following graph color:



So, we can see that the quality of our coloring from Greedy algorithm is dependent on the order in which we color the vertices.

If we decide to color the vertices in order G, L, H, P, M, A, I, S, C. we will only use 4 colors to color all vertices.



(d). If a chromatic number is 2, this graph is known as Bipartite graph. we can use a similar BFS to tell.

so, the algorithm is as follows:

① Creat 2 sets, set A and set B. initially both are empty.

② Start with any vertex and perform BFS:
add first vertex into set A, and add all its neighbors into set B. And we make sure that this vertex is not present in set B already. As it happens, we will return false if is present in both sets.

③ If we are able to seperate all the vertices into two sets, then we will return true.

The total time complexity is $O((V+E)V)$, where E is number of edges, and V is number of vertices.

so, we taking n as number of vertices edges, the total complexity is $O(n^3)$.