Problem 1

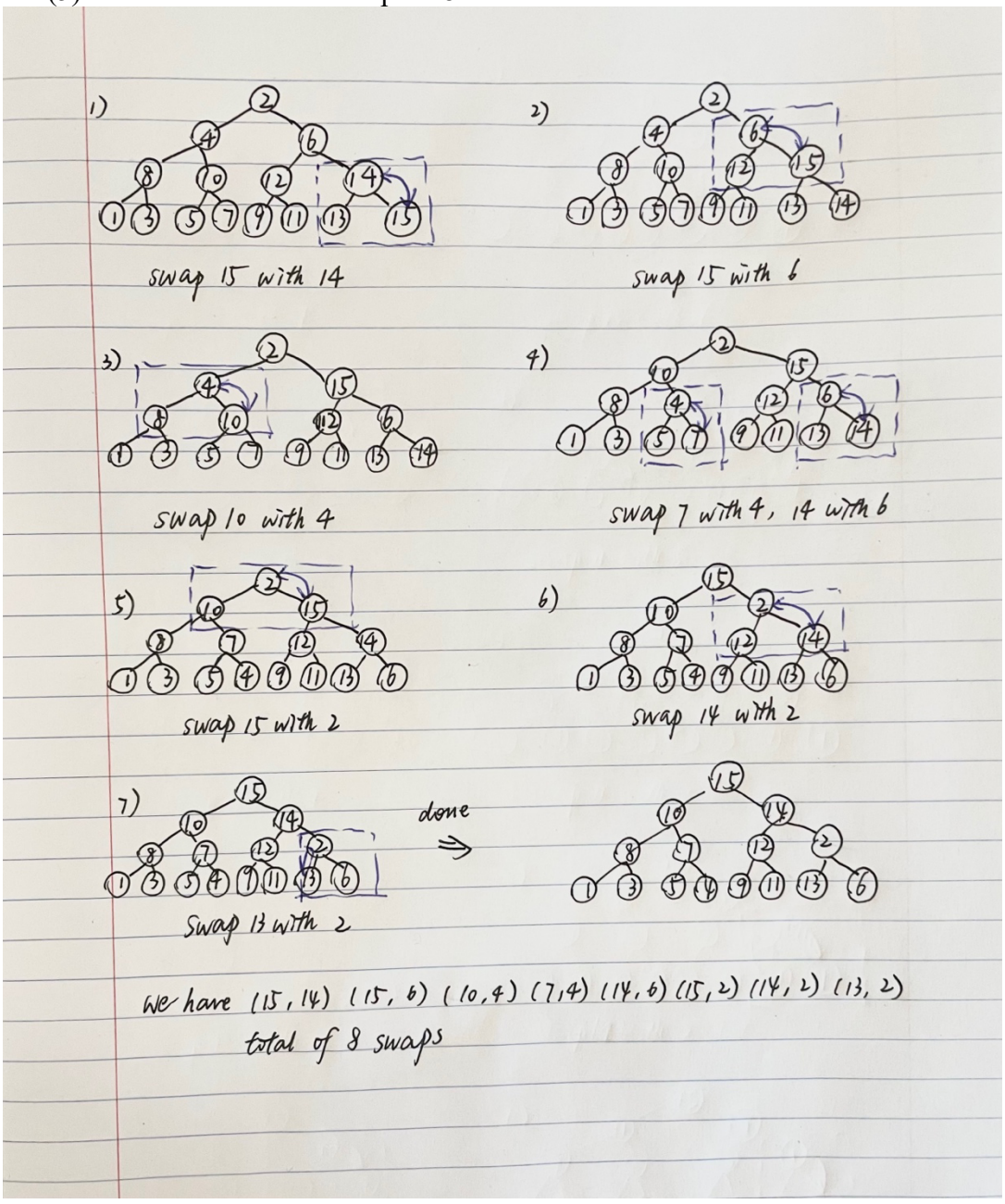
(1) D

(2) iii

(3) k = 5

(4) 512 + 1024 = 1536

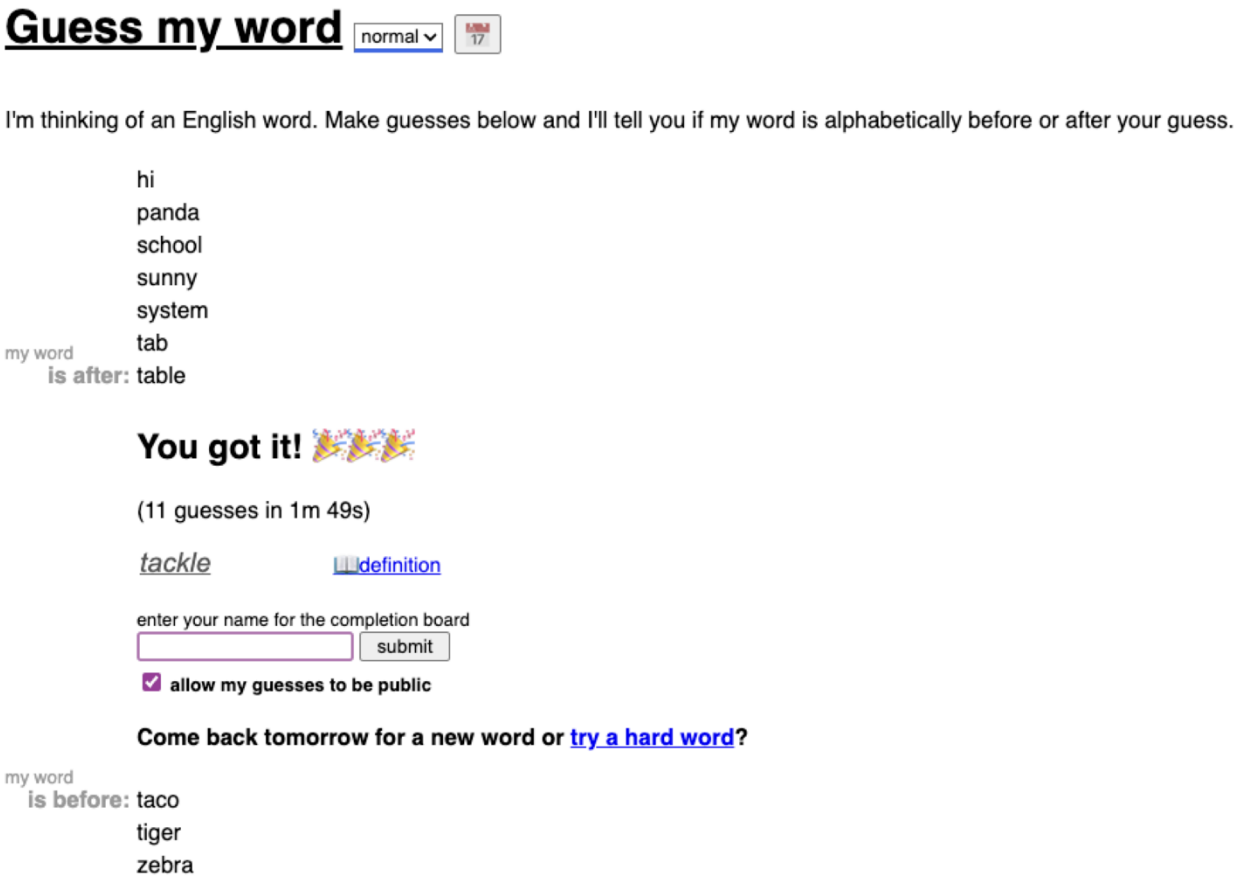
(5) The total number of swaps is 8



(6) B + C + D + 0.2E = 15 + 24 + 30 + 7 = 76

Problem 2

(a)



(b)

We find the mid word and compare it with the search word to check if this word is before or after the mid word. We perform the task repeatedly until we find the exact word.

And this will be a good application for binary search problem, since log (2^k – 1) = k guesses.

```
function identifySercertWords (wordslist, k):
    l = 0, r = 2^(k) + 1
    mid = (r - l) / 2
    midWord = wordsList[mid]
    while (l < r) :
        if (searchWord < midWord):
            r = mid
        if (searchWord == midWord):
            return searchWord
        if (searchWord > midWord):
            l = mid + 1
```

(c)
Since T(n) be the maximum number of guesses, by applying binary search algorithm, we get
$T(n) = T(n/2) + O(1)$

By using Master theorem, we have $a = 1$, $b = 2$, $n \wedge (\log_b a) = n \wedge 0 = 1$,
Therefore, we can get that $T(n) = \Theta(\log n)$.

(d)
The average guessing times = $\log_2(267{,}751) = 18.030532438$
Since $18.03 * 1 > 15$, I expect to lose money.

Problem3

(a)
Insertion sort will be faster since it's O (n) if all elements are equal.

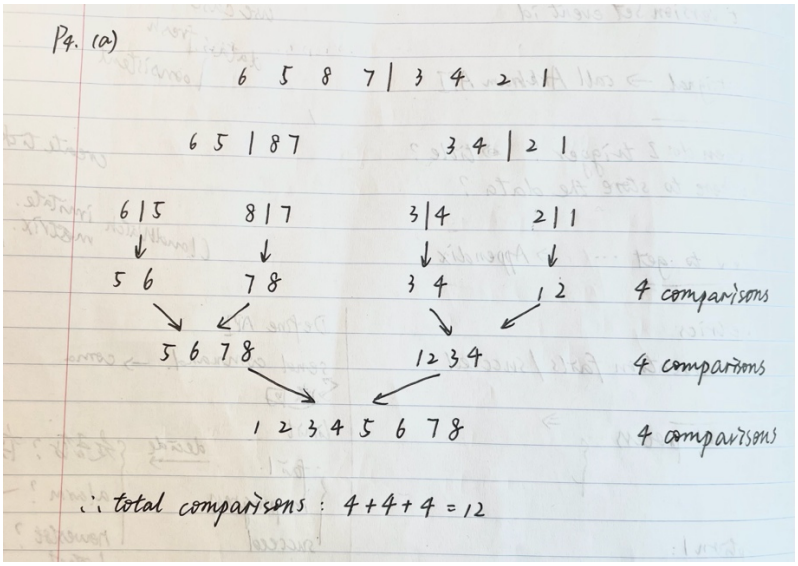
(b)
Quicksort is faster, since it doesn't do unnecessary element swaps. For Heapsort, even if all data is already ordered, you are going to swap 100% of elements to order the array.

(c)
Bucket sort is faster, because it works best when the data are more or less uniformly distributed. The average time complexity for Bucket Sort is O (n + k).

(d)
Counting sort is faster. Since the input range is fixed, time complexity is O (9 + n)

Problem4

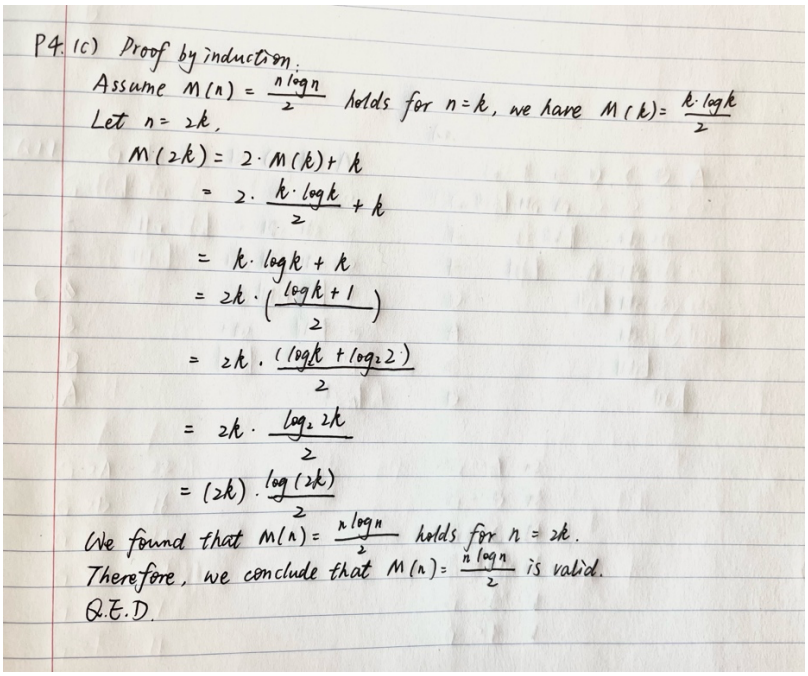
(a)



(b)

For merge sort, we recursively sort each halt, and merge it. In Master's Theorem, a = b = 2 then we have M(n) = 2 M(n/2) + f (n). Given M(4) = 4, M(2) = 1, M(1) = 0, we use substitution and get f(n) = n/2.

(c)



(d)

Total number of inputs = 8,
P (exactly 12 comparisons) = 1,
Probability of getting single comparisons = ½ (since divide the array into two halves)
By using binomial distribution:

$$P\ (X = x) = \binom{n}{k} p^k q^{n-k}$$

$$= \binom{12}{8} 0.5^4 0.5^8$$

$$= 495 * (1 / 4096)$$

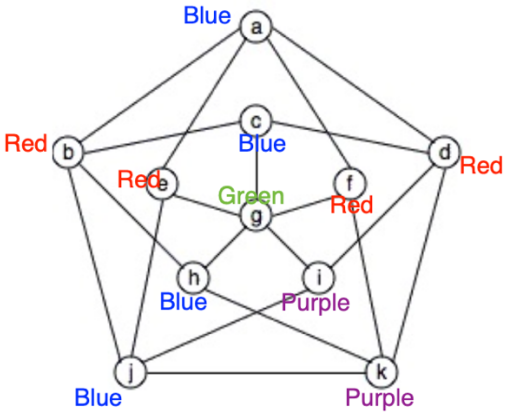
$$= 0.1208$$

Therefore, the probability pf getting exactly 12 comparisons in merge sort will be 0.1208

Problem 5

(a)

According to the graph marked by colors: We know that the graph is 4-colorable.
However, we can't mark the graph by using only 3 colors.



(b)

Greedy algorithm:

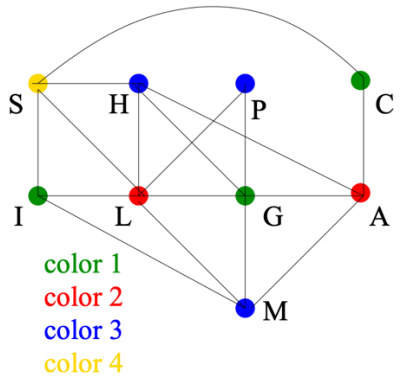
1. Color a vertex with color 1
2. Pick an uncolored vertex v. Color it with the lowest-numbered color that has not been used on any previously colored vertices adjacent to v. (If all previously used colors appear on vertices adjacent to v, this means that we must introduce a new color and number it.)
3. Repeat the previous step until all vertices are colored

By applying this algorithm, we can get that $\chi(G) = 4$

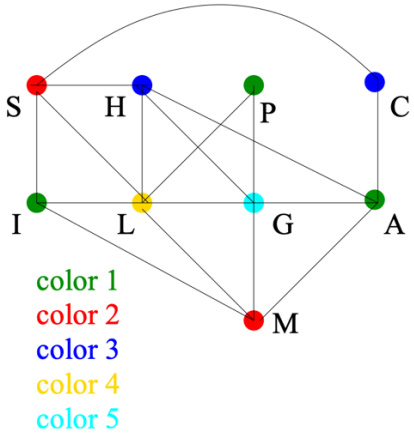
(c)

No. The provided algorithm in (a) will not always use exactly $\chi(G)$ colors.

Examples:
For the following graph, if decide to color the vertices in order G, L, H, P, M, A, I, S, C. Then we can mark the graph with 4 colors.



However, if decide to color the vertices in order A, I, P, M, S, C, H, L, G. We have to mark the graph with 5 colors.



(d)

If the chromatic number is 2, then the graph is called Bipartite graph. We can use BFS to tell whether the graph is Bipartite or not.

The algorithm is as follows:

1. Create 2 sets: Set A and Set B initially both are empty.
2. Start with any vertex and perform BFS algorithm: add first vertex into Set A, and all its neighbors into Set B. We need to make sure that this vertex is not present in Set B already. By chance if is present in both sets, we return false.
3. If we can separate the vertices into two sets, then we return true.

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

So, the taking n as number of vertices edges, the total complexity if O (n^3).