Solver: Dinh Ngoc Hai

Email Address: ngochai001@e.ntu.edu.sg

1.

a.

i) Best case: when the fake coin is the first coin in the array, it is found immediately, number of comparison = 1.

e.g: [1, 2, 2, 2, 2, 2]

Worst case: when the fake coin is in the middle of the array, particularly the (n/2+1)-th coin, we need 2 comparisons for every pair of coins, even the last 2 middle coins. The number of comparison = n

e.g: [2, 2, 2, 1, 2, 2]

ii) We have f = [1, 3, 5,.......,6, 4, 2] where f[i] is the number of comparisons needed when the fake coin is in position i. The average case is the mean of the above array.

For odd n, the answer $= 1 + 2 + \cdots + (n-2) + (n-1) + (n-1) = \frac{n\times(n+1)}{2} - 1$

(notice that the middle fake coin need n-1 comparisons instead of n since when we don't compare when only 1 coin left)

For even n, the answer$= 1 + 2 + \cdots + (n-2) + (n-1) + n = \frac{n\times(n+1)}{2}$

b.

Worst case occurs when we have to search until the last division. In term of number of comparisons:

$$O(n) = O(n/2) + 1$$

Solve the above recurrence equation with assumption of $O(1) = 0$

$$O(n) = O(2^k) = O(2^{k-1}) + 1 = O(2^{k-2}) + 2 = \cdots = k = \log_2 n$$

c.

An improved algorithm: instead of dividing the pile into 2 piles, we divide it into 3 piles. If the 2 compared piles have equal weight, then the fake coin is in the remaining pile. Hence, we reduce the size of the problem by 3 instead of 2 after each comparison. The pseudo-code is as follow:

```
searchFakeCoin2 (Coins pile[], int n)
{
    Divide pile into 3 piles: pile1, pile2, pile3;
    switch (compare (pile1, pile2)) {
```

```
                case 1: // pile1 < pile2
                        if (sizeof(pile1)==1) return pile1;
                        else return searchFakeCoin2(pile1);
                case 2: // pile1 > pile2
                        if (sizeof(pile2)==1) return pile2;
                        else return searchFakeCoin2(pile2);
                case 1: // pile1 == pile2
                        if (sizeof(pile3)==1) return pile3;
                        else return searchFakeCoin2(pile3);
        }
}
```

d.

Similar to part b, we calculate the complexity: $O(n) = O\left(\frac{n}{3}\right) + 1 = \log_3 n$

The speedup is $\frac{\log_2 n}{\log_3 n} = \log_2 3 \approx 1.58$

2.

a.

i. $\lim_{n\to\infty} \frac{nlg^2(n)}{\lg(n^2)\lg(n^3)} = \lim_{n\to\infty} \frac{nlg^2(n)}{2\lg(n)3\lg(n)} = \lim_{n\to\infty} \frac{n}{6} = \infty$

Therefore, f is not in O(g), f is not in θ(g), f is in Ω(g)

ii. $\lim_{n\to\infty} \frac{2^n}{2^{1.1n}} = \lim_{n\to\infty} \frac{2^n}{2.2^n} = 0$

Therefore, f is in O(g), f is not in θ(g), f is not in Ω(g)

b.

The given statement is false.

To justify, we only need to show 1 contradict example: $f(n) = n, g(n) = n^2$

c.

i. (note that new key are added to the front)

Hash	Key

0	
1	
2	
3	
4	<- 6582 <- 6791 <- 2567 <- 1643
5	
6	<- 1865
7	
8	
9	
10	

ii.

Hash	Key
0	
1	
2	
3	
4	1643
5	6791
6	1865
7	6582
8	
9	
10	2567

c.

For hash table in i):

6791: 2

6582: 1

6584: 1

For hash table in ii):

6791: 3

6582: 4

6584: 1

3.

a.

Best case is when all 3 numbers of A is less than B[0], we only need 3 comparisons.

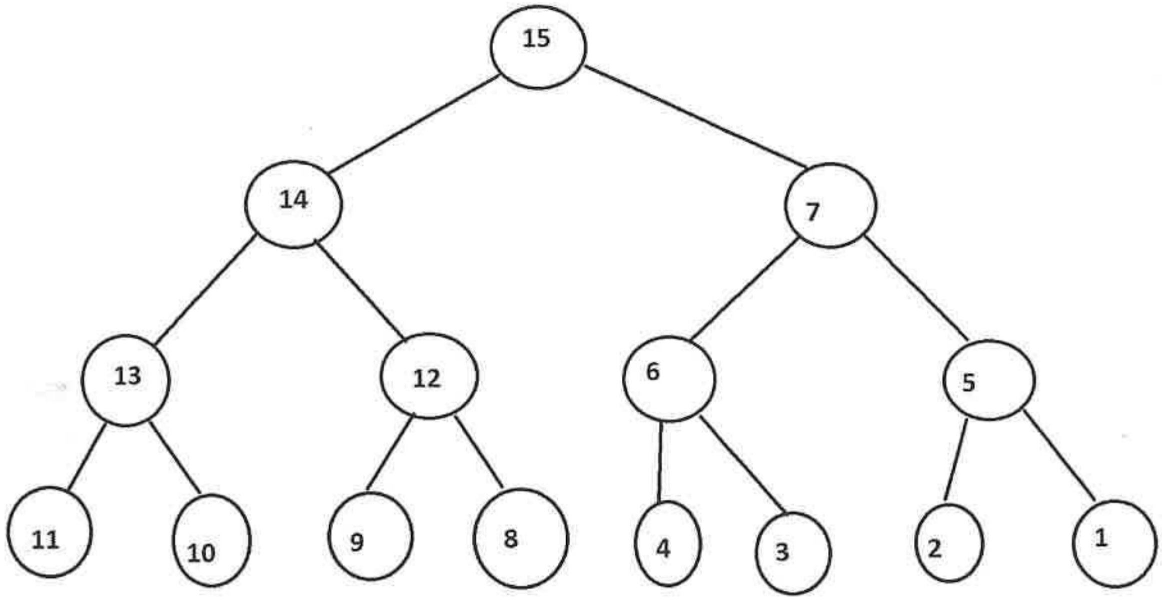
e.g: A=[1, 1, 1], B=[2, 2, ..., 2]

Worst case is when we have to compare numbers from A and B until the end of the 2 arrays, we need 1002 comparisons.

e.g: A=[1, 1, 1000], B=[2, 2, ..., 2]

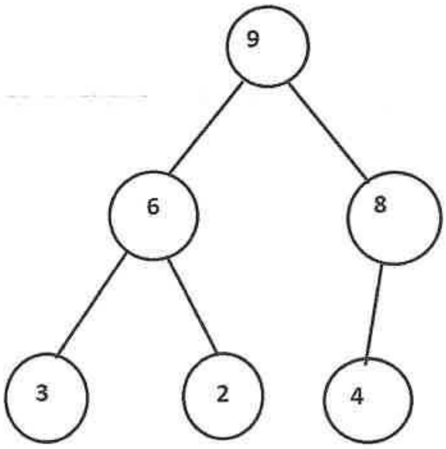
b.

In order to maximizing the number of comparisons in deleteMax, we aim to make the right-most element of the bottom level to go farthest after fixHeap, i.e. to go to the left-most of the bottom level. One possible arrangement of the initial heap is as follow:



c.

We need 7 comparisons to construct the heap:



Then, we need 8 comparisons to sort the array: 2, 3, 4, 6, 8, 9.

d.

We can use quick sort for large input and then use insertion sort for reasonably small array size, e.g: 10.

4.

a.

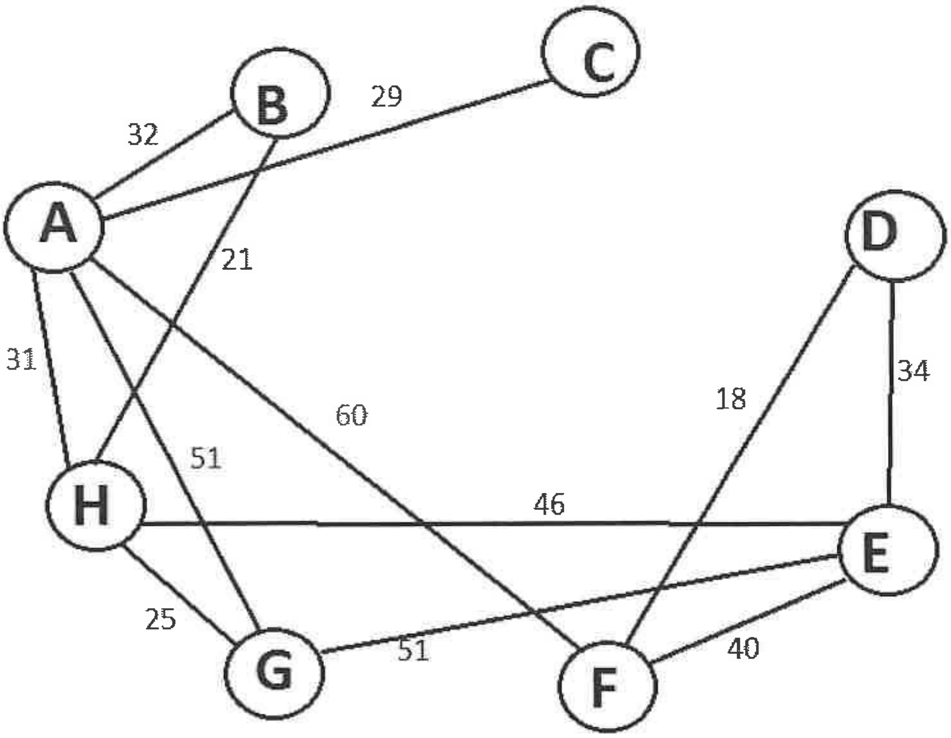
There exist 2 solutions for this 4-queens problem:

		Q	
Q			
			Q
	Q		

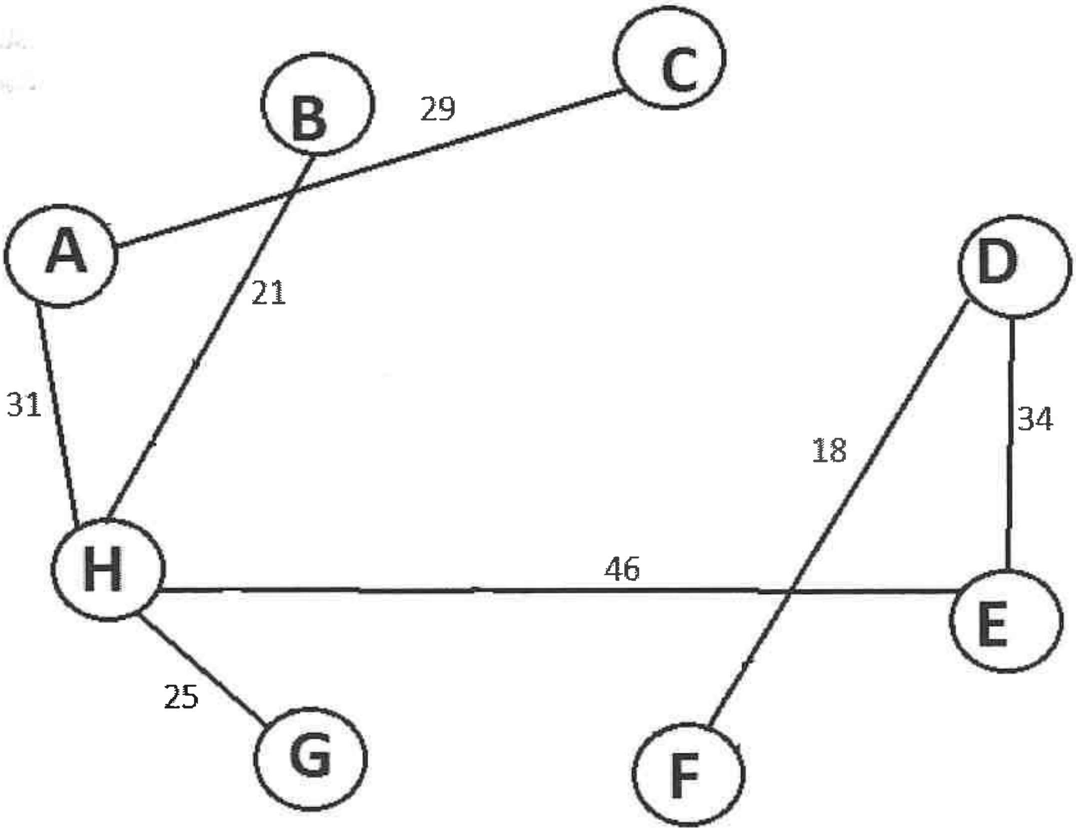
	Q		
			Q
Q			
		Q	

b.

i.



ii.



Sum of edge weights = 204

c.

Let (c,d)=-10 (in fact, every number smaller than or equal to -2 will works too)

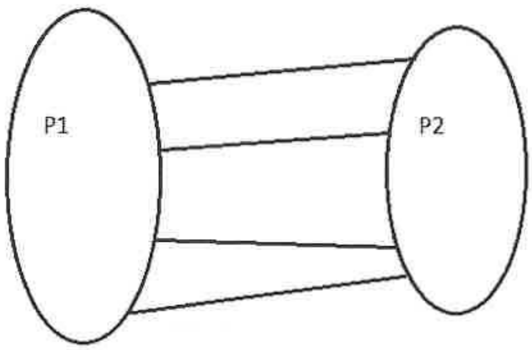
The shortest path is [s,b,c,d]=-4 (if we prevent cyclic path) but Dijkstra's algorithm will output [s,a,d]=5

d.

The given statement is true. Here is 2 way to prove:

Proof 1: Call X the smallest edge in the graph. We will prove X is in the MST

Arbitrarily divide the graph into 2 parts P1 and P2 such that X connect them



The MST of the graph is MST of P1 + MST of P2 + smallest edge of those connect P1 and P2. Since X is smallest in graph, it should be the smallest among the edges connecting P1 and P2. Therefore, X is in MST.

Proof 2:

We already know the following characteristic of MST: when an edge is added and it form with some edges of MST a cycle such that the added edge is not largest, MST has to be constructed so that the new edge is included.

Apply this characteristic: if we build an MST without smallest edge X, when we add X to the MST, we can always find a cycle such that X is not largest (and indeed it is smallest). Hence, an MST without X is not correct.

Therefore, X is in MST.

For reporting of errors and errata, please visit pypdiscuss.appspot.com
Thank you and all the best for your exams! ☺