# COMP 2119A: Solution for Test 1

## 1

i) True.

Let $c = 2000$ and $n_0 = 1$. For any $n \geq n_0$, we have $20n \leq 20n^{1.01} = 2000 * 0.01n^{1.01}$.

Therefore, $20n = O(0.01n^{1.01})$

ii) False.

Let $f(n) = n^2$ and $g(n) = n$, then it is easy to see $f(n) = \omega(g(n))$. And $\log f(n) = 2\log n, \log g(n) = \log n$

But let $c > 2$, for any $n > 1$, $2\log n < c\log n$, so $\log f(n) \neq \omega(\log g(n))$

iii) False.

Let $f(n) = 0.5, g(n) = n^{0.001}$, then we can see $f(n) = O(0.5)$ and $g(n) = O(n^{0.001})$.

Assume that $g(n) + f(n) = O(1000000)$, then $\exists n_0, c > 0$, such that $\forall n \geq n_0, 0.5 + n^{0.01} < c * 1000000$

So we get $n \leq (c * 1000000)^{1000}$, which is contradicted with $\forall n \geq n_0$

So $O(0.5) + O(n^{0.001}) \neq O(1000000)$

iv) True

$f(n) = f(2^k) = f(2^{k-1}) + 2^k = f(2^{k-2}) + 2^{k-1} + 2^k = ... = f(1) + 2 + 2^2 + ... + 2^k = 10 + \frac{2(1-2^k)}{1-2} = 10 + 2(2^k - 1) = 10 + 2(n-1) = 2n + 8$

Let $c = 3, n_0 = 8$, $\forall n \geq n_0$, we have $f(n) = 2n + 8 \leq c * n$. So $f(n) = O(n)$

## 2

1. Divided 27 coins(c1,c2,...,c27) into three groups, A1=c1,...,c9, A2=c10,...,c18, A3=c19,...,c27. Choose A1 and A2 to weigh. If $A1 > A2$, then A1 contains counterfeit coin. Else if $A1 < A2$, then A2 contains counterfeit coin. Else if $A1 = A2$, then A3 contains counterfeit coin.

2. Divided the group Ai contains counterfeit coins into three groups, B1, B2, B3, each of which has 3 coins. Choose B1 and B2 to weigh. If $B1 > B2$, then B1 contains counterfeit coin. Else if $B1 < B2$, then B2 contains counterfeit coin. Else if $B1 = B2$, then B3 contains counterfeit coin.

3. Divided the group Bi contains counterfeit coins into three groups, C1, C2, C3, each of which has 1 coin. Choose C1 and C2 to weigh. If $C1 > C2$, then C1 is counterfeit coin. Else if $C1 < C2$, then C2 is counterfeit coin. Else if $C1 = C2$, then C3 is counterfeit coin.

## 3

```
Algorithm F(n){
    if(n=1) return 5;
    if(n=2) return 6;
    else{
     return F(n-1)+f(n-2)
     }
    }
```

Time Complexity: $T(n) = T(n-1) + T(n-2) + C \leq 2T(n-1) \leq 4T(n-2) \leq 2^{n-2}T(2) = 2^{n-2}C = O(2^n)$

## 4

```
Algorithm Find(A[1,...,n],low,high) {
    middle = (low+high)/2;
    if(A[middle] = middle) return middle;
    else if (A[middle] > middle) {
     low = middle+1;
     }
    else high = middle-1;
    Find(A[1,...,n],low,high);
    }
```

Time Complexity: $O(\log n)$

## 5

Idea: we use another stack T to store (sum,num), which includes the current sum of all elements and the number of elements. And stack S store the origin elements. When we want to find the average, we visit the top element of T and return sum/num. Time complexity of each operation is $O(1)$.

```
void Push(element a) {
(sum,num) = T.top();
sum = sum+a;
```

```
num ++;
S.push(a);
T.push((sum,num));
}
void Pop() {
T.pop();
S.pop();
}
element Top(){
return a=S.top()
}
element FindAve() {
(sum,num)=S.top();
return ave=sum/num;
}
```

## 6

We design a n*n matrix $A = [a_{ij}]_{n*n}$, and let $a_{ij}$ = the number of edges from $i$ to $j$. $a_{ij} = 0$ if no edge from $i$ to $j$.

a) Bool Source(A){
for i = 1 to n {
 bool isSource = true
 for j = 1 to n {
  if($i \neq j$){
   if($a[ij] = 0$ or $a[ji] > 0$) isSource=false;
  }
 }
 if(isSource = true) return true;
 }
return false;
}

Time Complexity: $O(n^2)$

b) Pair MaxNumEdge(A) {
max=0;
(v1,v2)=(0,0);
for i=1 to n {
 for j=1 to n {
  if($a[ij] > max$) {

```
      max = a[ij];
      (v1,v2)=(i,j);
      }
     }
    }
  return (v1,v2);
  }
```

Time Complexity: $O(n^2)$

c) Vertex MaxNumNeighbors(A) {

```
max = 0;
v=0;
for i=1 to n {
 sum = 0;
 for j=1 to n {
  if(a[ij] > 0 or a[ji] > 0) sum++;
 }
 if(sum > max){
  max=sum;
  v=i;
 }
}
return v;
}
```

Time Complexity: $O(n^2)$

# 7

a)See Table 1.

b) Assume that $\exists i, j \in [0, m-1], i > j$, such that $h(k,i) = h(k,j)$, i.e., $h(k,i) - h(k,j) = (i-j)h_2(k) = 0 \mod m$. Since $h_2(k)$ is relatively prime to $m$, we have $(i-j) = 0 \mod m$. So $i-j = t*m, t \in Z$. Since $i, j \in [0, m-1]$, $i-j \leq m-1 < m$. So $t = 0$, that is to say $i = j$, which is contradicted with $i > j$. Therefore, the probe sequence is a permutation of [0,...,m-1].

# 8

Set two queues Q1 and Q2 to perform a stack.

```
void Push(element a){
 Enqueue(Q1,a);
```

```
}
Time Complexity: O(1);
Void PoP() {
 while(!(Q1.head+1=Q1.tail)) {
  x=Dequeue(Q1);
  Enqueue(Q2,x);
 }
 Dequeue(Q1);
 While(!(Q2.head=Q2.tail)) {
  x=Dequeue(Q2);
  Enqueue(Q1,x);
 }
}
```

Time Complexity: $O(n)$

```
Element Top() {
 while(!(Q1.head+1=Q1.tail)) {
  x=Dequeue(Q1);
  Enqueue(Q2,x);
 }
 x=Dequeue(Q1)
 While(!(Q2.head=Q2.tail)) {
  x=Dequeue(Q2);
```

| | linear probing | quadratic probing | double hashing |
|---|---|---|---|
| 0 | 52 | 52 | 52 |
| 1 | 118 | 118 | 118 |
| 2 | | | 58 |
| 3 | | | |
| 4 | 30 | 30 | 30 |
| 5 | | | |
| 6 | 45 | 45 | 45 |
| 7 | 58 | 58 | 47 |
| 8 | 47 | 47 | 35 |
| 9 | 61 | 61 | 61 |
| 10 | 35 | 35 | |
| 11 | 89 | 89 | 89 |
| 12 | | | |

Table 1: Q7 a)

```
  Enqueue(Q1,x);
 }
 return x;
}
```
Time Complexity: $O(n)$