Tutorial-2

Ques1. What is the time complexity of below code and how?

```
void fun(int n)
{ int j=1 ; i=0;
  while(i<n)
  { i=i+j;
    j++;
  }
}
```

$i = 0, 1, 3, 6, 10, 15$ let say k terms.

So general formal would be $\frac{k(k+1)}{2}$

k^{th} term $= n \Rightarrow \frac{k(k+1)}{2} = n$

$k^2 + k = 2n$

$k^2 = n \Rightarrow k = \sqrt{n}$

$$\boxed{T(n) = O(\sqrt{n})}$$

Ques2. Write Recurrence relation for the recursive function that prints fibonacci series. Solve the recurrence relation to get the time complexity of this program and why

Recursive function

```
int fib(int n)
{ if(n<=1)                    → O(1) = c
    return n;
  return fib(n-1) + fib(n-2)  → T(n-1) + T(n-2)
}
```

Recurrence Relation $T(n) = T(n-1) + T(n-2) + c$

Now $T(n-1) \simeq T(n-2)$

$T(n) = 2T(n-1) + c$

By Backward Substitution

$T(n-1) = 2T(n-1-1) + c \Rightarrow 2T(n-2) + c$

$T(n) = 2[2T(n-2) + c] + c$

$= 4T(n-2) + 3c$

Now $T(n-2) = 2T(n-2-1) + c$

$= 2T(n-3) + c$

$T(n) = 4T(n-2) + 3c$

$= 4(2T(n-3) + c) + 3c$

$T(n) = 8T(n-3) + 7c$

Generalizing $2^k T(n-k) + (2^k - 1)c$

assume $n-k=0 \Rightarrow n=k$

$2^n T(0) + (2^n - 1)c$

$= 2^n + (2^n - 1)c$

$= 2^n(1+c) - c$

$= 2^n$.

Time Complexity $= O(2^n)$

Space Complexity

For fibonacci recursive implementation, the space required is directly proportional to the maximum depth of recursion tree, since maximum depth is directly proportional to numbers of elements so $O(n)$.

Q3. Write program which have complexity $n(\log n)$.

(i)
```
for (i=1; i<=n; i++)
{
    for (j=1; j<=n; j=j*2)
    {
        sum = sum+i;
    }
}
```

(ii) n^3
```
for (i=0; i<n; i++)
{   for (j=0; j<n; j++)
    {
        for (k=0; k<n; k++)
        {  sum = sum+k;
        }
    }
}
```

(iii) $\log n(\log n)$
```
for (i=1; i<=n; i=i*2)
{  for (k=1; k<=n; k=k*2)
   {  sum = sum+i;
   }
}
```

Q4 Solve the Recurrence relation $T(n) = T(n/4) + T(n/2) + Cn^2$

$T(n/4) \simeq T(n/2)$

$T(n) = 2T(n/2) + Cn^2$

as $a \geq 1$ and $b \geq 1$

By using master's method

$T(n) = aT(n/b) + f(n)$

$C = \log_b^a = 1$

$f(n) > n^c \Rightarrow Cn^2 > n^c$

$T(n) = O(f(n))$

$= O(n^2)$

Q5 What is the time complexity of following func().

```
int fun (int n)
{     for (int i=1 ; i<=n ; i++)
      {  for (int j=1; j<n ; j+=i)
         {  O(1)
         }
      }
}
```

for $i=1 \rightarrow 1+2++3+ \cdots (n+1) = n$

for $i=2 \rightarrow 1+3+5+ + \cdots n \Rightarrow n/2$

for $i=3 \rightarrow 1+4+7+ - - - n \Rightarrow n/3$

$\therefore n + n/2 + n/3 + \cdots +1$

$\Rightarrow n\left(1 + 1/2 + 1/3 + \cdots + 1/n\right)$

Now we know that $n\left(1+\frac{1}{2}+1/3+\cdots+1/n\right) \leq n\left(1+\frac{1}{2}+\frac{1}{2}+\frac{1}{4}+\frac{1}{4}+\frac{1}{4}\cdots\right.$

$n\left(1+1/2+1/3+\cdots+1/n\right) \leq n\left(1+0.5+0.5+\cdots\right)$

$O(n\log n)$

Q6 What should be the time complexity of

```
for (int i=2; i<=n ; i=pow(i,k))
{ //some O(1)
}
```

for first iteration $i=2$

for second iteration $i = 2^k$

for third iteration $i = (2^k)^k = 2^{k^2}$

⋮

n^{th} iteration, loop ends when $2^{k^i} = n$

Take log on both sides

$\log n = \log_2 2^{k^i}$

$\log n = k^i$

$i = \log(\log n)$

$O(\log(\log n))$

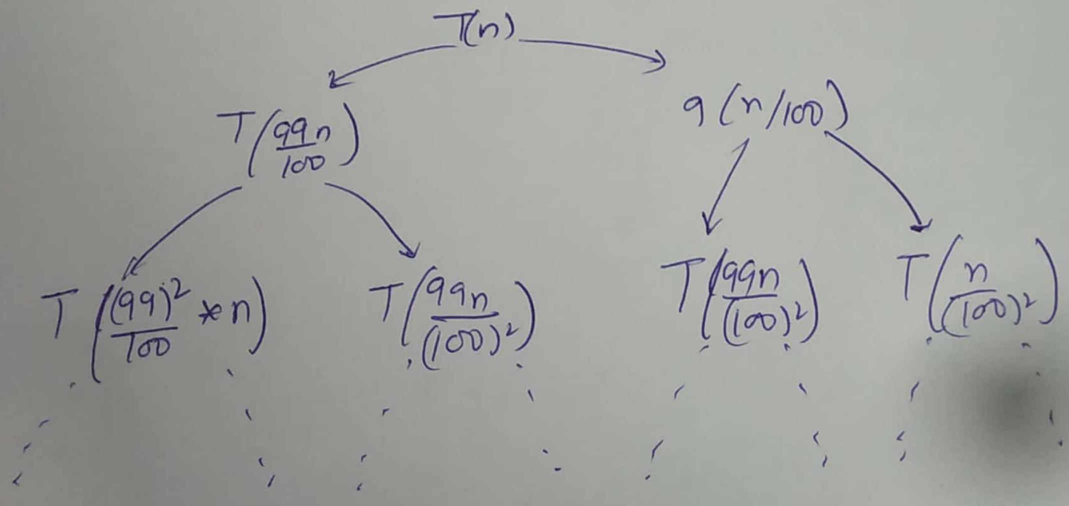
Q7 Write a recurrence relation when quick sort repeatedly divides the array in two parts of 99% and 1%

99 to 1 in quicksort

When pivot is where from front or end always.

so

$$T(n) = T(99n/100) + T(n/100) + O(n)$$



$$n = \left(\frac{99}{100}\right)^k$$

$$\log n = k \log \frac{99}{100}$$

$$k = \log n \frac{100}{99}$$

Time Complexity $= n * \log\left(\frac{100}{99} n\right)$