Tutorial - 2

(1)

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

i	j
0	1
1	2
2	3
3	4
.	.
.	.
.	.

j
1
1
1 + 2
1 + 2 + 3
.
.
.

So, T.C = O(√n)

Ans

$j \le k$

$1 + 2 + 3 + 4 - - - k$

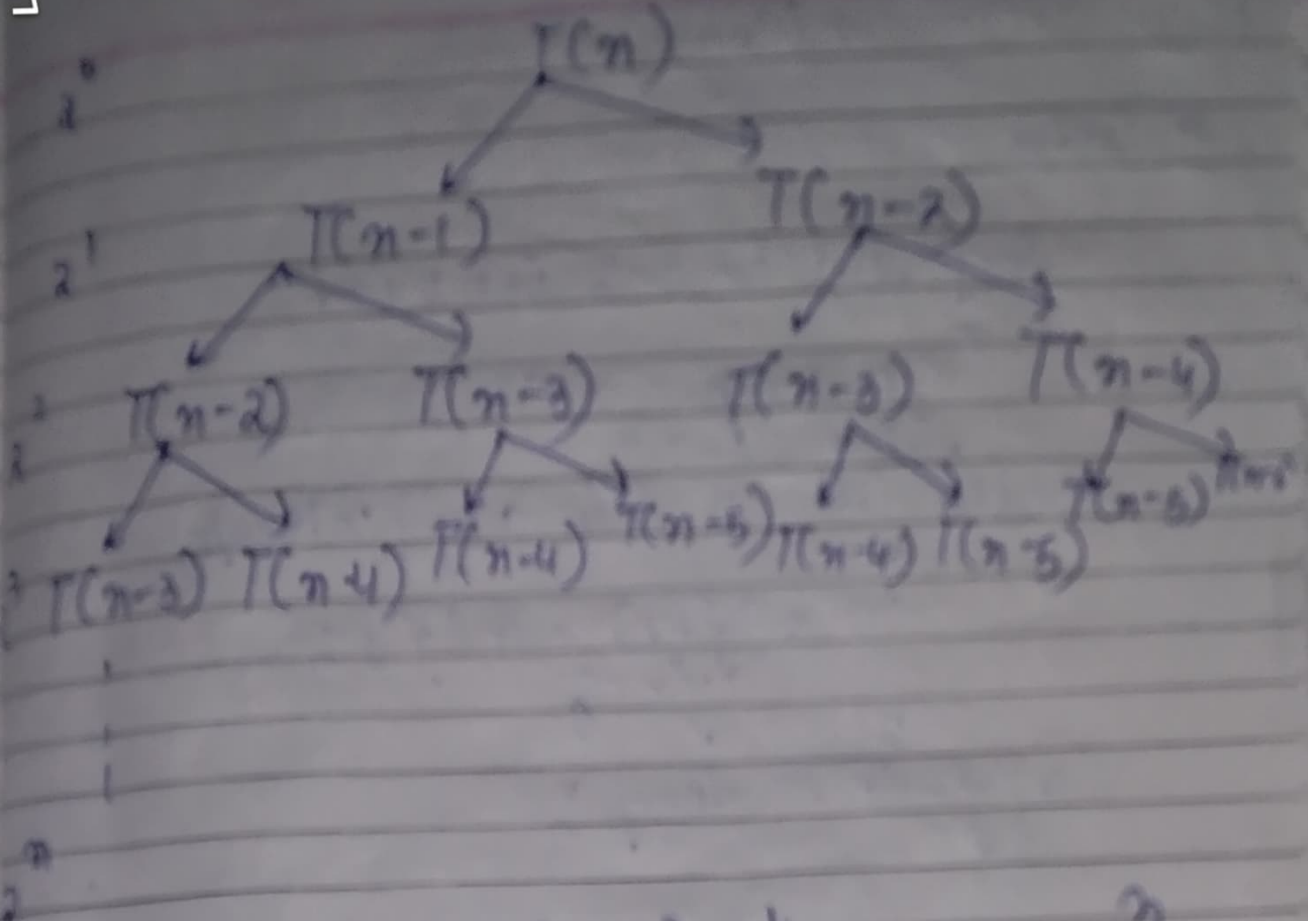
$1 + 2 + 3 + 4 - - - k \propto n$

$\frac{k(k+1)}{2} \propto n \Rightarrow k^2 \propto n$

$k = \sqrt{n}$

Ans 2:- Recurrence Relation of fibonacci series.

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



$$T.C = 2^0 + 2^1 + 2^2 + 2^3 + 2^4 - - - 2^n$$

$$sum = \frac{a(r^n - 1)}{r - 1}$$

$a = 1$

$r = 2$

$$\frac{1(2^n - 1)}{1} = 2^n - 1$$

$$T.C = O(2^n)$$ Ans

T.C = time S.C depends on the maximum depth of the tree so

$$S.C = O(n)$$ Ans

(2) TOC = $O(n^3)$

```
int target = 8
vector<vector<int>> res;
for(int i=0; i<n; i++)
{
  for(int j=i+1; j<n; j++)
  {
    for(int k=j+1; k<n; k++)
    {
      if(nums[i]+nums[j]+nums[k]==target)
      {
        vector<int> ds;
        ds.push_back(nums[i]);
        ds.push_back(nums[j]);
        ds.push_back(nums[k]);
        res.push_back(ds);
      }
    }
  }
}
// find triplets whose sum is 0.
```

(2) (N log N)

(3)

```
void merge(int * a, int * temp, int lb,
           int mid, int high)
{
  int i = lb;
  int j = mid + 1;
  int k = 0;
  while( i <= mid && j <= high)
  {
    if( a[i] < a[j])
      temp[k++] = a[i++];
    else if(a[j] > a[j])
      temp[k++] = a[j++];
  }
while (i <= mid)
  temp[k++] = a[i++];

while( j <= high)
  temp[k++] = b[j++];

for( i=0; i<k; i++)
{
  a[lb+i] = temp[i];
}

void merge sort (int * a, int * temp,
                 int lb, int high)
if (lb < high)
{
  int mid = lb + (high - lb)/2;
```

```
    merge_sort (a, temp, mid);
    merge_sort (a, temp, mid+1, ub);
    merge(a, temp, lb, mid, ub);
}

int main()
{
    int a[5] = {17, 9, 16, 23, 5};
    int temp[5] = {0};

    merge_sort(a, temp, 0, 4);
    for(int i=0; i<4; i++)
        cout << a[i] << " ";
}
```

Q 70 C = log(log n)

```
for(int i=1; i<n; i=i*2)
{
    for(int j=1; j<n; j/=2)
    {
        // O(1);
    }
}
```

Page:
Date: / /

(1) $T(n) = T\left(\frac{n}{4}\right) + T\left(\frac{n}{2}\right) + cn^2$

Apply master theorem

$T(n) = a_1 T\left(\frac{n}{b_1}\right) + a_2 T\left(\frac{n}{b_2}\right) + f(n)$

$a_1 = 1$
$b_1 = 4$

$c = \log_b a = \log_4 1 = \log_2 2 = 0$

Compare n_1^c & $f(n)$, $n_1^0 \propto cn^2$

$1 \propto n^2$

$T.C = O(n^2)$ Ignoring constant

Now

$a_2 = 1$
$b_2 = 2$

$c = \log_b a = \log_2 1 = 1$

Compare n^c & $f(n)$

$n^1 \propto (n)^2$

$T.C = O(n^2)$ [∵ Ignoring constants]

∴ Overall Time complexity

$= O(2n^2) = O(n^2)$

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

i	j
1	1, 2, 3 — — — n times
2	1, 3, 5, 7 — n/2 times
3	1, 4, 7, 11 — n/3 times
:	:
:	:
n	j=1 — 1 time

$$T.C = n + \frac{n}{2} + \frac{n}{3} + \frac{n}{4} - - - 1$$

$$n\left[1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} - - - \frac{1}{n}\right]$$

$a = 1$

$r = \frac{1}{2}$

$$T.C. = n(\log n)$$

$$Sum = \frac{a(1 - r^n)}{1 - r}$$

(6)
```
for( i = 2 ; i <= n ; i = pow(i,k)
{
    // O(1)
}
```

where k is constant.

$T.C = 2, 2^k, 2^{k^2}, 2^{k^4}, \ldots, 2^{k(\log k(\log n))}$

Since $2^{k \log k(\log n)} = 2^{\log n} = n$.

So, there are total $\log_k(\log(n))$ iterations.

(8) a) $100 \propto \log_2(\log n) \propto \log(n) \propto \log^2 n$
$\propto \sqrt{100}(n) \propto n \propto n \log n \propto n^2 \propto 2^n \propto$
$4^n \propto 2^{2^n} \propto \log(n!) \propto n!$

b) $1 \propto \log(\log(n)) \propto \sqrt{\log n} \propto \log n \propto \log 2n$
$\propto 2 \log n \propto n \propto 2n \propto 4n \propto n \log n \propto n^2 \propto$
~~$2(2^n n)$~~ $\propto \log(n!) \propto n! \propto 2(2^n)$

(c) $96 \propto \log_8(n) \propto \log_2(n) \propto 5n \propto n \log_6 n$
$\propto n \log_2 n \propto n! \propto \log n! \propto 8^{2n}$