$T(n)-1+c$ $\qquad$ $T(n-1)+c$

$T(n-2)+c$ $\qquad$ $T(n-2)+c$

$T(n)\cdot 1+2c$

Master's Theorem.

for divide and conquer recurrences

bigo nikalna

$100n^2 + 20n + 5$ is $O(n^2)$

$100n^2 + 20n + 5 \leq 100n^2 + 20n^2 + 5n^2$

$100n^2 + 20n + 5 \leq 125n^2$

$f(n)$ C $g(n)$

$C = 125$ $n_0 = 1$ exclude lowerorder.

$g(n) = \Omega(f(n))$

$g(n) \leq C \cdot f(n)$

n = no. of size
no. of input

searching   sorting

$f(n) = O(g_1(n)) + O(g_2(n))$

$f(n) = O \cdot \max(g_1, g_2(n))$

geeks for geeks
notations

function swap (a, b)
c = a
a = b   → 3 swaps.
b = c

call by value
call by reference

function max (A, n)
m = A[0];
for (i = 1; i < n; i++)
{ if (m < A[i])
m = A[i]
}
return m

recurrence relation

iteration
recursive

recursion &
time compl. O(n)
nikalne ko.

⇒ n-1

nlogn.
searching & sorting
best ki value

reverse(A[ ], l, r)
swap (A[l], A[r])
reverse (A, l+1, r-1)
if (l >= r) return
base case

bigo

linear recurence relation

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

time complexity

binarySearch (A[ ], l, r, target)
- mid = l + (r-l)/2
if (A[mid] == target) return mid
if (A[mid] < target) binarySearch (A, l, mid-1, target)
if (A[mid] > target) binarySearch (A, mid+1, r, target)
(base condition) if (l > r) return -1

$T(n) = T(n/2) + O(1) = T\left(\frac{n}{2}\right) + C$

divide ~~&~~ con conquer relation

Recurrence Relation.

linear recurrence reln.

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

$n > 0 \quad T(0) = 1$

$T(n-2) + C$

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

$T(n) = T(n-2) + 2n - 1$
$T(n-2) = T(n-3) + n - 2$
$T(n) = T(n-3) + 3n - 3$
$T(n-3) = T(n-4) + n - 3$

$T(n) = T(n-4) + n - 3 + 3n - 3$
$= T(n-4) + 4n - 6$
$: K = 4$

$= T(n-K) + Kn - \frac{K(K-1)}{2}$
$= T(0) + n^2 - \frac{n(n-1)}{2}$
$= 1 + n^2 - \frac{n(K-1)}{2}$

Time complexity $O(n^2)$

binary sort
quick sort

Master's Theorem

$T(n) = a\,T\left(\frac{n}{b}\right) + f(n)$

$a \geq 1$
$b > 1$
$k \geq 0$
Condition satisfy karni chahiye

a = no. of subproblem
$\frac{n}{b}$ = size of subproblem
n = size of problem.
$f(n)$ = extra work done

Ye pehele
then
ye

Range nikalte he

$n^{\log_b a}$

$T(n) = aT\left(\frac{n}{b}\right) + \theta(n^k \log^p n)$

1) $a > b^k$ $\quad T(n) = \theta(n^{\log_b a})$

2) $a = b^k$
$p > -1 \rightarrow T(n) = \theta(n^{\log_b a} \log^{p+1} n)$
$p = -1 \rightarrow T(n) = \theta(n^{\log_b a} \log\log n)$
$p < -1 \rightarrow T(n) = \theta(n^{\log_b a})$

3) $a < b^k$
$p \geq 0 \quad T(n) = \theta(n^k \log^p n)$
$p < 0 \quad T(n) = \theta(n^k)$

log.

$T(n) = 1\left(\frac{n}{2}\right) + c$
$a = 1 \quad b = 2 \quad k = 0 \quad p = 0$

$T(n) = \cancel{T(\frac{n}{2})}$
$\theta(n^{\log_2 1} \log n)$
$\theta(\log n)$

$T(n) = 2T\left(\frac{n}{2}\right) + cn$
$a = 2 \quad b = 2 \quad k = 1 \quad p = 0$
$T(n) = \theta(n \cdot \log n)$
$T(n) = \theta(n^{\log_2 2})$

$a = 1 \quad b = 2 \quad k = 0 \quad p = 0$

$b^k = 2^0 = 1$
$f(n) = \theta(n\log_2 \log n)$

$$\boxed{T(n) = T(\sqrt{n}) + 1}$$

Value substitution.

Substitution

Master's Theorem.

$\frac{1}{2}$ $\quad n = 2^m$

$$T(n) = T(\sqrt{n}) + 1$$

$$n = 2^m = \log n = m \log 2$$

$\cancel{S=}\frac{4}{1}$

$\frac{1}{2}$

$$T(2^m) = T(2^{m/2}) + 1$$

$$S(m) = T(2^m)$$

$$S(m) = S\left(\frac{m}{2}\right) + 1$$ — Master Theorem

$$S(m) = \Theta(\log m)$$

$$S(m) = \Theta(\log \log m)$$

$$S(m) = \Theta(\cancel{\log_2} \log m)$$

$$S(m) = \Theta(\log m)$$

$$\log n = \log_2^m$$

$$\log n = m \log 2$$

$$\log_2 n = m.$$

$$T(n) = \Theta(\log \log_2 n)$$

Tower of annoy

1) $T(n) = 2T(n-1) + C$ $\quad \{ n>1 \quad T(1)=1$ $\quad \boxed{O(2^n)}$

2) $T(n) = 3T\left(\frac{n}{2}\right) + n^2 \longrightarrow \boxed{\Theta(n^2)}$

3) $T(n) = 3T\left(\frac{n}{2}\right) + \log^2 n \longrightarrow \boxed{\Theta(n^{\log_2 3})}$

4) $T(n) = 2T\left(\frac{n}{2}\right) + n\log^2 n \longrightarrow \Theta(n \log^3 n)$

5) $T(n) = 2^n T\left(\frac{n}{2}\right) + n^n \longrightarrow \log n$. ye mam ne Jonbujke galat bataya tha

$T(n) = 3T$

$n-1$

$n-1$

$n-1$

$n-1-1$

$n-2$

substituting $T(n-1)$ from the recurrence.

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

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

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

$T(n-2) = 2^2[2T(n-3) + c] + 3c$

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

$T(n) = 2^3 T(n-3) + 7c$

4. k-th Expansion.

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

Base case

$k = n-1$, reduce $T(n-k)$ to the base case $T(1)$:

$T(n) = 2^{n-1} T(1) + (2^{n-1} - 1)c$

Since $T(1) = d$, we have

$T(n) = 2^{n-1} d + (2^{n-1} - 1)c$

$T(n) = 2^{n-1} d + 2^{n-1} c - c$

$T(n) = 2^{n-1}(d+c) - c$

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

$T(n) = 3T\left(\frac{n}{2}\right) + n^2$

~~$T(n) = 3T(\frac{n}{}$~~

$T(n) = aT\left(\frac{n}{b}\right) + f(n)$

where

$a = 3$
$b = 2$
$f(n) = n^2$

Calculate $\log_b a$ Compute $f(n)$ with $\log_b a$, which is

$\log_2 3$.

$\log_b a = \log_2 3$ ✓ $a < b^k =$

Compare $f(n)$ with $n^{\log_b a}$

$F(n) = n^2$ with $n^{\log_b a}$

$T(n) = \Theta(n^2)$

(3) $T(n) = \Theta(n^{\log_b a})$
$= \Theta(n^{\log_2 3})$

(4) $a = 2 \quad b = 2 \quad k = 1 \quad p = 2$

$a = b \quad T(n) = \Theta(n^{\log_b a} \log^{p+1} n)$
$= \Theta(n \log^3 n)$

(5) $T(n) = 2^n T\left(\frac{n}{2}\right) + n^n$

$O(n^n)$

$T(n-1) = 2^{n-}$

$2^n \geq n^n$

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

$n_0 = 2$

$T(n) = 2^n \left[2^{n-1} T\left(\frac{n-1}{2}\right) + n-1^{n-1}\right] + n^n$

Ceil = Upper limit floor = Lower limit

Q.1) $T(n) = 2T(\text{ceil}(\sqrt{n})) + 1 \quad T(1) = 1$

Q.2 $T(n) = 2T\left(\frac{n}{2}\right) + n - k. \quad \log\ p = 0 \quad T(0) = T(1) = 1$

(a) $T(n) = O(n^2)$ (c) $T(n) = \Omega(n^2)$

(b) $T(n) = \Theta(n \log n)$ (d) $T(n) = O(n \log n)$

which isn't correct

Q.4 Procedure $A(n)$
if $(n <= 2)$ return 1
else return $A(\sqrt{n})$

$T(n) = ?$

Q.3 If running time of an algorithm is represented by following recurrence relation if $(n \leq 3)$ then $T(n) = n$
else $T(n) = T\left(\frac{n}{3}\right) + Cn$ $\qquad T(n) = 2$

Q. T

$T(n) = 2T(\text{ceil}(\sqrt{n})) + 1 \qquad T(1) = 1$

$n = 2^m \qquad m = \log_2 n$ substitution.

$T(2^m) = 2(T(2^{m/2}))$

$T(2^m) = S(m)$

$S(m) = 2S\left(\frac{m}{2}\right) + 1$

$a = 2 \quad b = 2 \quad k = 0 \quad p = 0$

$b^k = 1 \quad a > b^k$

$S(m) = \Theta(m^{\log_2 2})$

$= \Theta(m)$

$= \Theta(\log_2 n)$

$T(n) = 2T\left(\frac{n}{2}\right) + n \qquad T(0) = T(1) = 1\ \Omega(n)$

$a = 2 \quad b = 2 \quad k = 1 \quad p = 0$

$b^k = 2 \quad a = b^k$

$O(n\log n)$ $\quad n^2$

(c) ×

if $(n <= 3)$ then $T(n) = n$
else $T(n) = T\left(\frac{n}{3}\right) + Cn$

$a = 1 \quad b = 3 \quad K = 1 \quad p = 0$

$b^k = 3 \quad a < b^k \quad p >= 0$

$T(n) = \Theta(n^k \log^p n)$
$= \Theta(n)$

Procedure $A(n)$
if $n <= 2$ — recursively call hoga
return 1
else return $A(\sqrt{n})$ $\quad T(n) = ?$

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

$n = 2^m \quad a = b^k \quad m = \log n$

$= T(2^{\frac{m}{2}})$

$S(m) = S\left(\frac{m}{2}\right)$

$a = 1 \quad b = 2 \quad k = 0 \quad p = 0$

$a = b^k$

$S(m) = \Theta(m^{\log_2 1} \log$
$= \Theta(\log m$
$= \Theta(\log \log n)$

---

int recursive (int n)
if $(n = 1)$ — $T(1) = 1$
return 1
else return(recursive $(n-1)$

$T(n) = T(n-1) + T(n-1) = 2T(n-1)$
$T(n-1) = 2T(n-2)$
$T(n) = 2^2 T(n-2)$
⋮
$T(n) = 2^K T(n-K)$

$T(n) = 2^{n-1}(T(1))$

$T(n) = 2^{n-1}$

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

$2^n \leq n^n$

$O(n^n)$ $\quad n > 2$

```
int a=0 , b=0;
for (i=0 ; i<n; i++)
{ a=a + rand ();
}
for (j=0; j<m; j++)
{ b = b+ rand ( );
}
```

Using loop

```
int a=0;
for(i=0; i<N; i++)
{ for (j=N; j<i ; j--),
{ a = a + i + j;
}
}
```

```
int i=0, l = N;
  while (l > 0)
{
  a += i;
  l = i/2;
}
```

$\Rightarrow O(\log N)$

```
for(int i=1; i<n; i++)
{
   i *= k;
}
```

$\Rightarrow O(\log_k n)$

```
int value =0;
for (int i=0; i<n; i++)
   for (int j=0; j<i; j++)
      value += 1;
```

$\Rightarrow n(n-1)/2$ Or $O(n^2)$

1) Selection Sort
2) Insertion Sort } → $O(N^2)$ $O(N*N)$
3) Merge sort
4) Quick sort } → Divide & Conquer
$O(N \log N)$

1) Stack (LIFO)
2) Queue (FIFO)

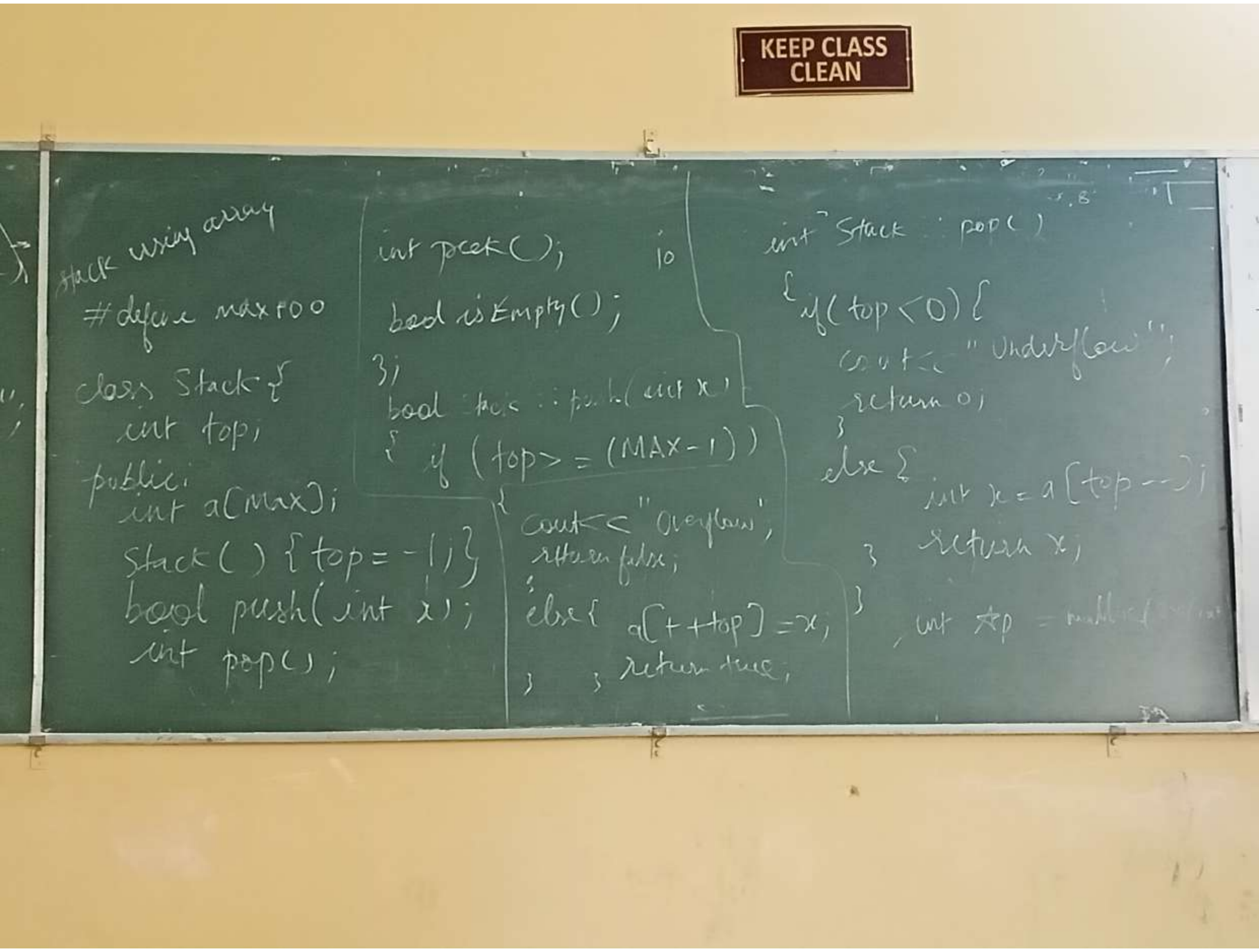
→ using array
→ Using linked list — Singly, doubly, Circular. linked list
Ad / disadvantages
→ Using Queue

push( ), pop( )

Recursion
Pollish Rotation conversion

```
bool stack::isEmpty()
{
  return (top<0);
}
```

```
int Stack::peek()
{
  if(top<0)
  { cout<<"Empty";
    return 0;
  }
  else {
    int x=a[top];
    return x;
  }
}
```


KEEP CLASS CLEAN
Stack using array
#define MAX 100
class Stack {
int top;
public:
int a[MAX];
Stack() { top = -1; }
bool push(int x);
int pop();
int peek();
bool isEmpty();
};
10
bool Stack::push(int x)
{ if (top >= (MAX-1))
{ cout << "Overflow";
return false;
}
else { a[++top] = x;
return true;
} }
int Stack::pop()
{ if (top < 0) {
cout << "Underflow";
return 0;
}
else {
int x = a[top--];
return x;
}
}
int *p