UNIT-II

Disjoint Sets

=>

$S_1 = \{1,7,8,9\}$, $S_2 = \{2,5,10\}$, $S_3 = \{3,4,6\}$

Number node	1	2	3	4	5	6	7	8	9	10
root	-1	5	4	-1	-1	4	1	1	1	5

=> Algorithm Simple union(i,j)

```
{
    p[i] = j;
}
```

p[i]=j

=> Algorithm Simple find(i) | find root

```
{
    while(p[i] > 0)
        p[i] := j;
    return i;
}
```

while (p[i] >0)
p[i]=i
return i

Element	1	2	3	4	5
P	-1	1	1	1	1

1, 2, 3, 4, 5 --- n

Degeneration Tree

1. union(1,2), union(2,3)
2. Union(3,4), union(n-1,n)

n-1. Find(1), Find(2), --- Find(n-1), Find(n).

n

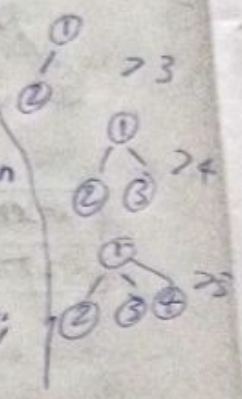
$$O\left(\sum_{i=1}^{n} i\right) = O(n^2)$$

Union performed between

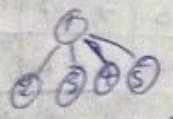
1, 2, 3, 4, 5

Weighting Rule:-

```
{
    temp = p[i]+p[j];
    if(p[i]<p[j]) then
    {
        p[i] = j;
        p[j] = temp;
    }
    else
    {
        p[j] = i;
        p[i] = temp;
    }
}
```




```
t = p[i]+(p[j])
if( p[i]<p[j])
p[i]=j
p[j]=t
else
p[j]=i
p[i]=t
```



Final tree

$\Rightarrow O(1)$

Collapse Find:-

```
{
    r := i;
    while (p[r] > 0)
    do
        r := p[r];
        while (i != r) do
        {
            s := p[i];
```

```
r = i
while(p[r] > 0)
    r = p[r]
while(i != r)
    s = p[i]
    p[i] = r
    i = s
return r;
```

```
P[i] = m;
i = 5;
}
return m;
}
```

T(1) 3 nodes | T(2) 2 nodes

2 nodes / 3 nodes

element	j	i	i/j
r	i	-2	i/2

Divide & conquer:-

If problem is Small, don't divide

If problem is Big, divide it.

Only 2 operations:-

Small(), combine()

$1 < k < n$ / K-Sub processes / n - Inputs

```
Algorithm DAC(P)
{
    if problem is Small then Small(P)
    else
    Divide Sub Sub problems (P) into Smaller ones P1, P2, P3 -- Pk, k≥1
        Small(P1), Small(P2), Small(P3), Small(P4) ... Small(Pk)
    Combine them combine(DAC(P1)) combine(DAC(P2)) -
        combine(DAC(P3)) combine(DAC(P4)) -
}
```

$\Rightarrow$ If Small, $T(n) \Rightarrow T(1)$ (Known) $n = 1$

If Big, $aT(\frac{n}{b}) + f(n)$, $n > 1$

$g(n)$ — Small ✓ (Constants) (Function) Input

~~$aT(\frac{n}{b})$~~ $T(n_1), T(n_2), T(n_3)$ — $+ f(n)$ — Big ✓

$T(n_1) \rightarrow$ Time taken for Divide & Combine

$n \rightarrow$ Inputs

$g(n) \rightarrow$ For Small Functions

$f(n) \rightarrow$ Additional time taken for divide & combine etc.

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

$a = 2, b = 2,$ $\quad 2(T(\frac{n}{2})) - T$

$$T(n) = 2T(\frac{n}{2}) + 2$$

$2T(\frac{n}{2}) + 2$

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

$2^i T(\frac{n}{2^i}) + 2^i + 2^{i-1}$

$2^i T(\frac{2(n)}{2^i}) + 2^i + 2^{i-1}$

$2^i + 2(2^i - 1)$ / $(2-1)$

$2^i + 2(2^i - 1)$

$2^i + 2^{i+1} - 2$

$\frac{n}{2} + n - 2$

$\frac{3n}{2} - 2$

$T(\frac{n}{2}) = 2T(\frac{n}{4}) + 2$

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

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

$T(n) = 2^i T(\frac{n}{2^i}) + 2^i + 2^{i-1} + \cdots 2$ — ①

$T(n) = 8T(\frac{n}{8}) + 8(n)$

$8(T(\frac{n}{2^3}) + 1)$

$2^3(T(\frac{n}{2^3}) + 1)$

$n = 2(2^i)$

$\frac{n}{2} = 2^i$ / $n = 2^{i+1}$

Stops, $\frac{n}{2^i} = 2$, $n = 2^{i+1}$ — ②

② in ①

$T(n) =$

$T(n) =$

$T(n) =$

$T(n) =$

$T(n) =$

$T(n) =$

$T(n) =$

T

T

② in ①

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

$$T(n) = 2^i + (2) + 2^i + 2^{i-1} + \dots + 2$$

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

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

$$T(n) = \frac{n}{2} + n - 2$$

$$T(n) = \frac{3n}{2} - 2 = \frac{3n-4}{2}$$

$a^n + a^{n-1} + a^{n-2} + \dots + a$

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

$n = 2(2^i)$

$n = 2^{i+1}$

$2^i = \frac{n}{2}$

$2 = \frac{n}{2^i}$

$\Rightarrow$

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

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

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

$$T(n) = 8T\left(\frac{n}{8}\right) + 3n$$

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

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

$2\left(2T\left(\frac{n}{4}\right) + 2\right) + 2n$

$2\left(2\left(2T\left(\frac{n}{8}\right) + 2\right) + 2\right) + 2n$

$8T\left(\frac{n}{8}\right) + 8 + 4 + 2 + 2n$

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

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

$8T\left(\frac{n}{8}\right) + 8(n)$

Binary Search:- $1 < i < n$ Elements in Sorted order

x = Searching Element in

$S(p)$ = True, if $n = 1$

$S(p)$ takes i/o

$g(1) = \theta(1)$.

$x = aq$, Found

$x < aq$, Search left

$x > aq$, Search right

$p > 1$, DAC

q in $[i, 1]$

=) Iterative (used in dividing Arrays only)

Algorithm BinSearch(a, n, x)

// $a[1:n]$

// $n \geq 0$, x is present/not

// return j such that, $x = a[j]$

else return 0.

```
{
  low = 1; high := n;
  while(low ≤ high) do
  {
    mid := [(low+high)/2];
    if(x < a[mid]) then high := mid-1;
    else if(x > a[mid]) then low := mid+1;
    else return mid;
  }
  return 0;
}
```

=) Recursive

Algorithm B

```
{ if(l=
  {
  }
  else
}
```

Binary search in Java / Binary search in Java

}

=) Recursive (In dividing of arrays)

Algorithm BinSrch(a, i, l, x)

// a[i:l], $1 < i \le l$, x is in/not,

// return j, such that x = a[j],
else return 0;

```
{ if(l = i) then
  { if(x = a[i]) then return i;   // If Small(p)
      else return 0;
  }
  else
    { // Reduce p into a Smaller Subproblem.
        mid := [(i+l)/2];
        if (x = a[mid]) then return mid;
        else if (x < a[mid]) then
            return BinSrch(a, i, mid-1, x);
            else return BinSrch(a, mid+1, l, x);
    }
}
```

Binary Decision Tree:-

	Successful Search	unSuccessful Search
Best	$O(1)$	$O(\log n)$
Average	$O(\log n)$	$O(\log n)$
worst	$O(\log n)$	$O(\log n)$

=> Finding min. and max.

Time complexity of Algorithm depends on the number of Element Comparison.

```
Algorithm StraightMinMax(a,n,max,min)
  // Set max. & min. to the min. of a[1:n]
  {
    max:= min:= a[1];
      for i:=2 to n do
      {
        if(a[i]>max) then max:=a[i];
      else if(a[i]<min) then min:=a[i];
      }
  }
```

Comparisons: $2(n-1)$.

=>
```
if(a[i]>max)
{
    max=a[i];
}else if(a[i]<min)
```

```
{ min = a[i];
}
```

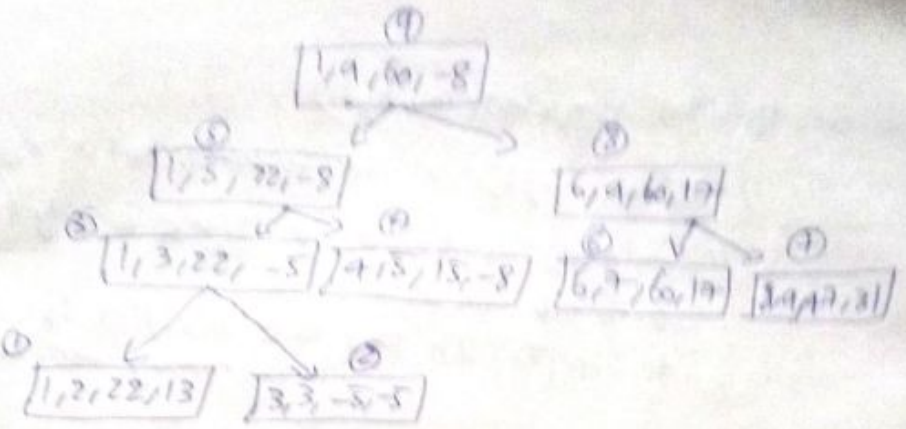
If $a[i] < min$ we want to check then $a[i] > max$ is false. (want to be)

	Best	worst	Average
Comparisons	$n-1$	$2(n-1)$	$2(n-1)$
Numbers in	Inc. order	Dec. order	Normal

=> min. & max. ~~& Algorithm~~:-

```
Algorithm maxmin(i, j, max, min)
// The
// a[1:n] is a global array. parameters
    // i & j are integers.
// 1<i≤j≤n, The Effect is to set max
// & min to the largest & Smallest
    // values in a[i:j] respectively.
{
  if(i=j) then max := min := a[i];
                          //Small(p)
  else if(i=j-1) then //Another case of
                          small p.
    {  if(a[i]<a[j]) then
       {  max := a[j]; min := a[i];
       }
```

i=j
i=j-1



into
oblems.

=) Time Complexity:

$$T(n) = \begin{cases} T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + 2 & n > 2 \\ 1 & n = 2 \\ 0 & n = 1 \end{cases}$$

n is a power of two, $n = 2^k$, k (+ve) Integer,

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

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

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

$$= 2^{k-1}T(2) + \sum_{1 \le i \le k-1} 2^i$$

$$= 2^{k-1} + 2^k - 2 = \frac{3n}{2} - 2$$

$$\Rightarrow \frac{2^k}{2} + 2^k - 2, \Rightarrow \frac{n}{2} + n - 2, \Rightarrow$$

The comparisons,

$\frac{3n}{2} - 2$ is better than $2(n-1)$

max;
min;

⇒ Algorithm mergesort(low, high)

```
// a[low : high] to be sorted (Global array)
// Small(p) is true, if there is only 1
   element.
// to sort. In this case the list is
                already sorted.

{ if(low < high) then //If >1 element
  {
      // Divide 'p' into sub-problems
      // Find where to split the
         mid :[(low+high)/2];

      //Solve the subproblems
         mergesort(low, mid);
         mergesort(mid+1, high);

      //Combine the Solutions.
         merge(low, mid, high);
  }
}
```

merge (low, mid, high)
merge (low, mid, high);

$$T(n) = \begin{cases} a & n = 1, a \text{ (a constant)} \\ 2T(n/2) + cn & n > 1, c \text{ (a constant)} \end{cases}$$

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

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

$$= 4\left(2T\left(\frac{n}{8}\right) + \frac{cn}{4}\right) + 2cn$$

$$= 8T\left(\frac{n}{8}\right) + 3cn \quad / \; 2^3 T\left(\frac{n}{2^3}\right) + 3cn$$

$$\vdots$$

$$= 2^k T(1) + kcn$$

$T(n) = O(n \log n)$ (high)

P = × * ×
Q = + * ·
R = · * ↓
S = · * ↑
T = + * ·
U = ↑ * +
V = ↓ * +

2) $\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$

Normal,

$$T(n) = \begin{cases} b & n \le 2 \\ 8T(n/2) + cn^2 & n > 2 \end{cases}$$

$8(4) + 4(4)$

$\begin{bmatrix} 8 & 8 \\ 8 & 8 \end{bmatrix}$

P = × × ×
Q = + × ·
R = · × ↓
S = · × ↑
T = + × ·
U = ↑ × +
V = ↓ × +

$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$

P = 2
Q = 0
R = 0
S = 0
T = 2
U = 0
V = 2

$(-1)(-1) = 1$

$P = (A_{11} + A_{22})(B_{11} + B_{22})$

$Q = (A_{21} + A_{22})(B_{21})$

$R = A_{11}(B_{12} - B_{22})$

$S = A_{22}(B_{21} - B_{11})$

$T = (A_{11} + A_{12}) B_{22}$

$U = (A_{21} - A_{11})(B_{11} + B_{12})$

$V = (A_{12} - A_{22})(B_{21} + B_{22})$

7 *
6 +
8 −

$C_{11} = P+S-T+V$

$C_{12} = R+T$

$C_{21} = Q+S$

$C_{22} = P-Q+R+U$

$\left.\right\}$ 6 + 2 −

$\boxed{\Theta(n^3), \Theta(n^2)}$

$$T(n) = \begin{cases} b & n \le 2 \\ 7T(n/2) + an^2 & n > 2 \end{cases}$$

$$T(n) = an^2\left(1 + \frac{7}{4} + \left(\frac{7}{4}\right)^2 + \cdots + \left(\frac{7}{4}\right)^{k-1}\right) + 7^k T(1)$$

$$\le cn^2\left(\frac{7}{4}\right)^{\log_2 n} + 7^{\log_2 n} \quad / \text{ c, a constant}$$

$$= cn^{\log_2 4 + \log_2 7 - \log_2 4} + n^{\log_2 7}$$

$$= \Theta(n^{\log_2 7}) \approx \Theta(n^{2.81}).$$

$$\begin{bmatrix} 7 & 6 \\ 1 & 5 \end{bmatrix}\begin{bmatrix} 2 & 3 \\ 6 & 1 \end{bmatrix}$$

(24)(−16)

$C_{11} = 36 + 20 - 13 + 7$

$C_{12} = +14 + 13$

$C_{21} = 12 + 20$

$C_{22} = 36 + 12 - 14 - 30$

$$\begin{bmatrix} 30 & 27 \\ 32 & 32 \end{bmatrix}$$

$7 \cdot 6 + 22$

$P = (12)(3)$ / $R = 7(2)$

$Q = 6(2)$ / $S = 5(4)$

$T = 13(1)$

$U = (-6)(5)$ / $V = (1)(7)$

=) Defect

Board =

=)

=) Defective chess Board problem:-
(Tiling)

Board => n by n , $n = 2^k$
$k >= 1$

=) Use Trionimos
2x2 size, 1x1 size

=) Divide & Conquer used,
Divided into Sub-problems
Each is an Exact copy of the larger one.

Squares => $n^2 / 4^k$
Removing the defect, $4^k - 1$ (multiple of three)

=) Every Recursive Algorithm to must have base case, to ensure that it will terminates.

=) n, 2^k is 2.
-) Defect identified.
-) Divide into 4 x4
~~-) Divide into 2x2~~
-) Fill with trionimo (grey)
-) Divide into 2x2
-) Color all tiles ~~except trionimo~~
-) ~~Divide into 2x2~~ Continue the procedure till all tiles covered in sub board.
-) Every 2x2 will have defect and that are created by us.
-) Color all trionimo which are inserted by

Divide into Sub problems

Solve the Sub problems

Combine the Sub problems.

as as defective.

we will Find the defective left in the board. (only one)

parts half of original Board

$T(n) = 4T(\frac{n}{2}) + c$ → coloring last tile.

$O(n^2)$,

Scanned By Camera Scanner

Increment binary counter

=>

```
Inc(A)
{ i = 0;
  while(i<n and A[i]==1)
  {
    A[i] = 0;
    i++ = 1;
  }
  if(i<n)
    A[i] = 1 => ②
}
```

Inc(A)
i = 0
while i<n && A[i] = 1
A[i] = 0
i++ = 1
if i<n
A[i] = 1

A[3	2	1	0]	
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7

For every n (call) flips

For every $\frac{n}{2}$ (calls) flips

For every $\frac{n}{4}$ flips (4 calls)

1 + 1
Actual cost Credit

Setting -> ② = 1 + ①
Actual cost Credit

reset -> 0 + ①

O(n)

Potential method:-

D_0 – Data Structure Initial

c_i – Actual cost of ith operation.

D_i – D.S. after i^{th} operation. to D_{i-1}.

ϕ – A potential fn, $\phi(D_i)$ – potential of D_i

$\hat{c}_i$ – Amortized cost of ith operation,

$$\hat{c}_i = c_i + \phi(D_i) - \phi(D_{i-1})$$

Actual cost Potential Difference

$$\hat{c}_i = c_i + \phi(D_i) - \phi(D_{i-1})$$

$$\hat{c}_i = c_i + \phi(D_i) - \phi(D_{i-1})$$

Total amortized cost:-

$$\sum_{i=1}^{n} \hat{c}_i = \sum_{i=1}^{n} (c_i + \phi(D_i) - \phi(D_{i-1}))$$

$$= \sum_{i=1}^{n} c_i + \phi(D_n) - \phi(D_0)$$

Telescopic series:-

$$a_0, a_1 \ldots a_n$$

$$\sum_{k=1}^{n} (a_k - a_{k-1}) = a_n - a_0$$

$$\phi(D) \geq \phi(D_0)$$

$$\sum_{i=1}^{n} \hat{c}_i \geq \sum_{i=1}^{n} c_i$$

Stack:-

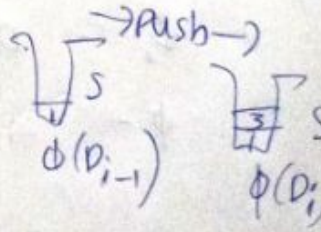
$$D_0, \ \phi(D_0) = 0$$

$$\phi(D_i) \geq 0, \quad i = 0, 1, \ldots n$$

$$\hat{c}_i = c_i + \phi(D_i) - \phi(D_{i-1})$$

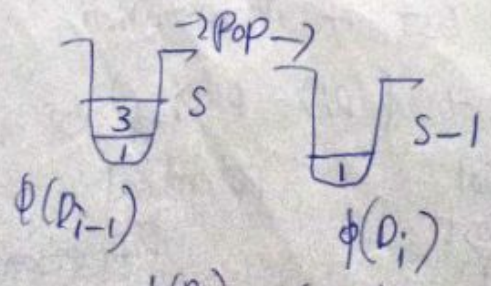
$$= 1 + 1$$

$$= 2$$



$$\Rightarrow (S+1) - S = 1$$

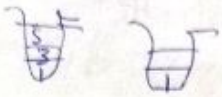
$$\cancel{O(n) = 1}$$



$$\phi(D_i) - \phi(D_{i-1}) = (S-1) - S = -1$$

$$\hat{c}_i = c_i + \phi(D_i) - \phi(D_{i-1})$$
$$= 1 - 1$$
$$= 0$$

multipop



'k' popped

$\phi(D_{i-1})$ $\phi(D_i)$

$$\phi(D_i) - \phi(D_{i-1}) \Rightarrow -K$$

$$\hat{c}_i = c_i + \phi(D_i) - \phi(D_{i-1})$$
$$= k - K = 0$$

Aggregate

Total Cost of operation's $= T(n)$

Cost/operation $= T(n)/n$

push(x) – $O(1)$

pop() – $O(1)$

multipop() – $\min(n, k)$

one (single pop) – $O(n)$

many (multiple pop)^n – $O(n^2)$

Average – $O(1)$

Binary Count

A => 3 2 1 0

0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7

worst-case $O(nk)$ size

Increments

$A[i]$ flips 2^i i.e. times, $\left\lceil \frac{n}{2^i} \right\rceil$ "

$i > \log n$ bit $A[i]$ never flips at all.

$$\sum_{i=0}^{\log n} \left\lceil \frac{n}{2^i} \right\rceil < n \sum_{i=0}^{\infty} \frac{1}{2^i}$$

$$\sum_{i=0}^{\infty} \left(\frac{1}{2}\right)^i = \frac{1}{1-\frac{1}{2}} = 2$$

$$\leq 2n$$

$$\Rightarrow O(n)$$

Average $\Rightarrow \frac{O(n)}{n} = 1$,

```
Inc(A)
{
  int i=0
while(i<n && A[i]==1)
{ A[i] = 0
    i+=1
}
if(i<n)
    A[i]=1
}
```

3	2	1	0	
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7