24/9/18

UNIT-1

Steps involve in problem solving

1. You should clearly define the problem statement
2. Design the algorithm
3. Algorithm Analysis (Basically for complexity)
4. Implementation
5. Test and fix the bugs.

It is a finite ~~step~~ set of instructions if followed accomplishes a particular task. — Def of algorithm.

Properties of Algorithm

→ It should have zero or more externally provided quantity as inputs.
→ It should have some outputs (o/p can be single or more than 1)
→ It should have clarity and precision, in short should be definite.
→ It should have finite no:- of steps → finiteness
→ Effectiveness → Every instruction should be effective/useful.

Analysis of Algorithm.

There are two types of complexity
→ Space complexity → Hardisk, RAM, cache, Bandwidth.
→ Time complexity → Fixed part and Variable part of prog

Assignment S1 Assignment Q1

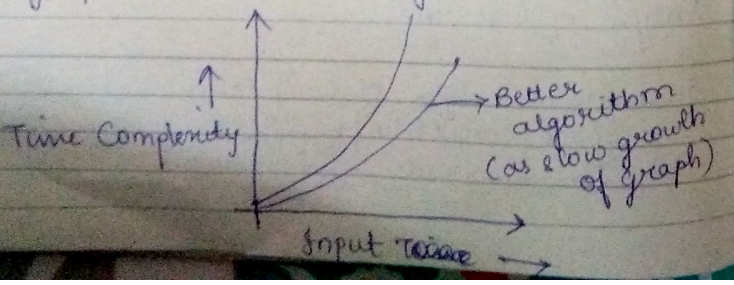
Q → You are given an array X of n elements. You want to compute the array A, such that A[i] is the average of elements X[0], X[1], X[i]; i = 0, 1, N.

Sol 1 → At each step i, compute the element A[i] by traversing the array X and determining the sum of the elements average respectively.

Sol 2 → At each step i, update the sum of the elements then compute the elements A[i] as sum/i

Two approach to analyse the algorithm, theoretical and experimental approach.

→ In theoretical analysis, we find the upper bound.
→ We are interested in growth graph of time complexity. Slower growth of the graph → means better algorithm




```
i = 0;
① while (i < n)
  {
    sum = 0
    for (j = 0; j <= n; j++)
    {
      sum = sum + X[j];
    }
    A[j] = sum/(i+1);
  }
```

```
i = 0;
② while (i < n)
  {
    sum = sum + X[i];
    A[i] = sum/(i+1);
  }
```

X = [35, 12, 3, 8, 25]
A [35, 23.5, 16.66, 14.5, 16.6]

Problems with experimental approach analysis

① Implementation of that problem.
② Not coverage of all I/Ps. (Skip those I/P that impact the sol° of the problem).
③ Hardware and S/W should be same.
④ Even if you are using the same H/W and S/W then processor load matters.
⑤ Sharing of resources.
⑥ No. of background processes.
⑦ Memory
⑧ Compiler.
⑨ N/W architecture
⑩ Prog. language.

So, we switched to theoretical approach/analysis.

→ Asymptotic Notation (for theoretical approach)

Let us say we have a sequential search of the unordered array where items are randomly distributed.

```
for (i=1; i<=N; i++)
    if (a[i] == x)
        Successful
    else
        Unsuccessful.
```

We want to find the complexity of worst and average case

Worst case - O(N)
Average case - O(N/2) = O(N)

★ Binary search (Divide & conquer strategy)

```
if (ME > Item)
    UB = ME - 1;
else
    LB = MD + 1;
```

Worst case - $O(\log_2 N)$
Average case - $O(\log_2 N)$

★ Selection sort

Worst case - $O(N^2)$

$O(1) < O(\log_2 N) < O(N) < O(N \log_2 N) < O(N^2)$
$O(N^3) < O(2^N) < O(N!)$

26/7/18

```
sum(a,n)
{ sum = 0; → 1
  for(i=0; i<n; i++) → n+1
  {
    sum = sum + a[i]; → n
  }
  return s; → 1
}
```

TUT - 1

Total steps = 2n+3

Recursion

```
sum(a,n)
{
  if (n==0) → 1+1
  {
    return 0; → 1
  }
  return (sum(a,n-1) + a[n]) → n
}
```

2n+2

$$t_{Rsum}(n) = \begin{cases} 2 & \text{if } n=0 \\ 2 + t_{Rsum}(n-1) & \text{if } n>0 \end{cases}$$

$t_{Rsum}(n) = 2 + t_{Rsum}(n-1)$

$= 2 + 2 + t_{Rsum}(n-2)$

$= 2(2) + t_{Rsum}(n-2)$

$\vdots$

$= n(2) + t_{Rsum}(0)$

$= 2n + 2$.

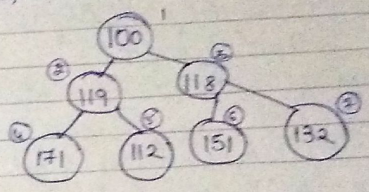
Assignment

Write an algorithm for fibonacci series & count the no: of steps it required to execute

27/7/15

To represent a binary tree, we need a Sequential Representation.

A = [100, 119, 118, 171, 112, 151, 132]



→ Parent of ith node = $i/2$ if $i \neq 1$.
→ Left child of ith node = $2i$
→ Right child of ith node = $2i+1$

Heapsort

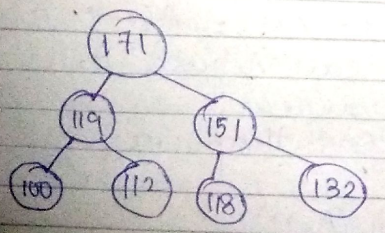
```
Heapsort (A, n)
  call Heapify (A, n)
  for i ← n to 2 by -1 do
    t ← A(i) ; A(i) ← A(1) ; A(1) ← t;
    call Adjust (A, 1, i-1)
  repeat
end Heapsort
```

```
Heapify (A, n)
  for i ← [n/2] to 1 by -1 do
    call Adjust (A, i, n)
  repeat
end Heapify
```

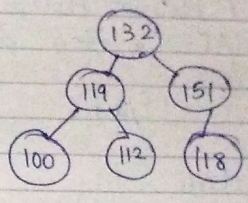
```
Adjust(A,i,n)
j ← 2*i ; item ← A(i)
while j≤n do
{
  if j<n & A(j) < A(j+1)
      then j ← j+1
  end if

  if item ≥ A(j) then exit.
  else A([j/2]) ← A(j)
       j ← 2*j
  repeat
  A([j/2]) ← item
end Adjust
```

After heapify (creating max heap)



After 1 heapify when we call adjust

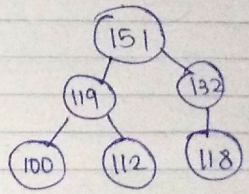


① i=1 ; j=2 ; item = 132

j=3 ; A[1] = 151

j=6

A[3] = 132.



151	119	132	100	112	118

after swapping.

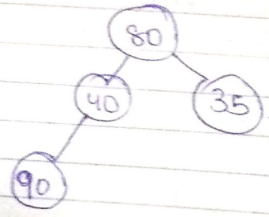
118	119	132	100	112	151

Now apply same on this.

Insert (A, n)

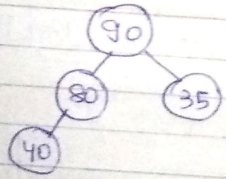
```
j ← n  i ← [n/2]  item ← A[n]
while i > 0 & A[i] < item do
        A[j] ← A[i]
        j ← i ;  i ← [i/2]
        repeat
        A[j] ← item
end insert
```

Eg:-

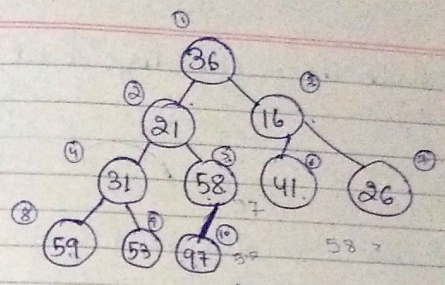


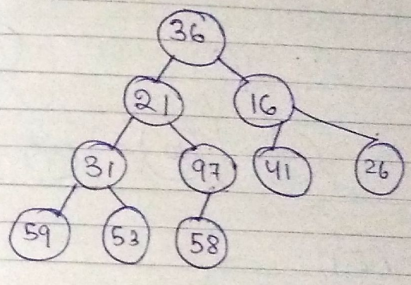
Insert 90.

Max heap.



Q 36, 21, 16, 31, 58, 41, 26, 59, 53, 97.
Create a maxheap.





31/7/18

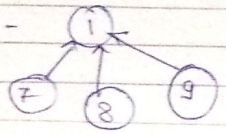
→ Let us say we have three sets (disjoint set)

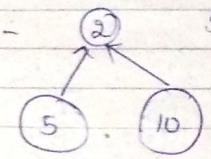
$S_1 = \{1, 7, 8, 9\}$

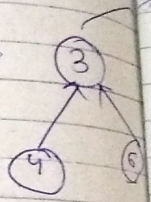
$S_2 = \{2, 5, 10\}$

$S_3 = \{3, 4, 6\}$

We will represent this set with the help of tree, all the child will point towards the node of the tree

S_1 - 

S_2 - 

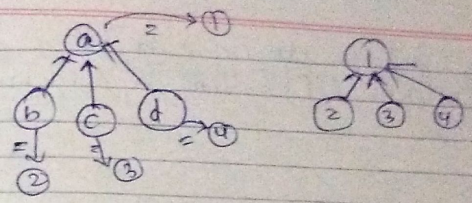
S_3 - 

Parent array. P[]

index	1	2	3	4	5	6	7	8	9	10
P	-1	-1	-1	3	2	3	1	1	1	2

At Root node index we will have value -1 and all the child nodes index will have the value of its parent.

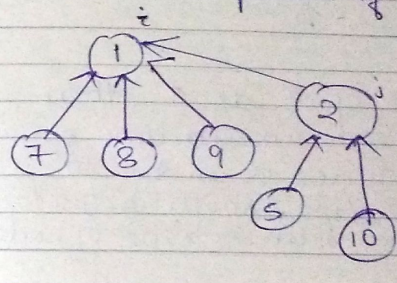
When we have set, containing letters or symbols not numbers; then we will assign letters as number; then we will make a symbol table; and then we can make a parent



Symbol table

1	a
2	b
3	c
4	d

→ Now if we want to do union of these 2 sets; S_1 and S_2 then we can either make parent of 1 as 2 or vice-versa




```
union(i,j)
{
 P[j] = i
}
```

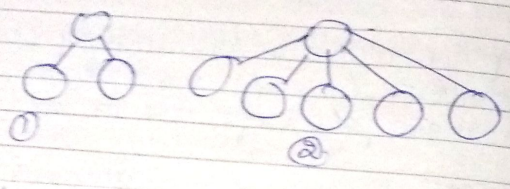
index	1	2	3	4	5	6	7	8	9	10
P	-1	1	-1	3	2	3	1	1	1	2

Find(i) [We have to find element i; to which set belong basically we have to find the parent, grand-parent, and so on till we reach root node]

```
{
while P[i] != 0 do
  i = P[i]
  return i;
}
```

→ Suppose if we have ith node at ith level, it requires O(i) time to reach the root node.

→ Suppose we have 2 sets.

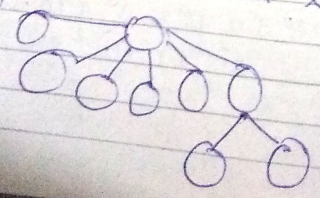


If I make parent of 2 as 1, then here we will need maximum time as more no. of nodes is in 2. So we will require 2 steps; so its better to make parent the tree with max. no. of nodes

→ Weighing rule :-

If no. of nodes in a tree i is less than no. of nodes in a tree j then make j the parent of i.

So, in the above case make tree 2 the parent of ①



* For weighing rule, we have to keep count of the no. of nodes of a tree. So, we have to store count to a memory location, so that we don't require extra memory.
We will store count at the parent index of the array with a -ve sign.

We Union(i,j)
x ← P[i] + P[j]

→ Now taking the same eg. of S_1, S_2 and S_3 sets.

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

We Union(i,j)
x ← P[i] + P[j]
If P[i] > P[j]
P[i] ← j
P[j] ← x
else
P[j] ← i
P[i] ← x

Now,

index	1	2	3	4	5	6	7	8	9	10
p	-7	1	-3	3	2	3	1	1	1	2

Q Suppose we have 8 sets like
{1} {2} {3} {4} {5} {6} {7} {8}

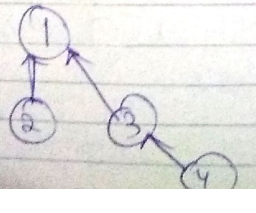
(i) U(1,2) =

(ii) U(3,4) =

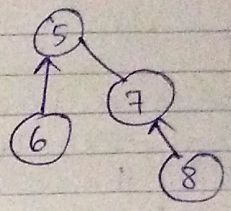
(iii) U(5,6) =

(iv) U(7,8) =

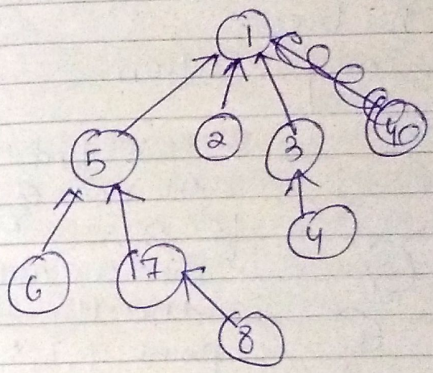
(v) U(1,3)



(vi) U(5,7)



(vii) U(1,5)



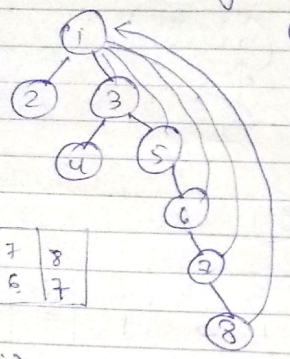
index	1	2	3	4	5	6	7	8
p	-1	1	1	3	1	5	5	7

Q Suppose for the above problem we have perform Find(8) ... Find(8) 9 times; how many steps we require?

→ To find one time 8 → it require 3 steps
Total steps: 3×9 = 27 steps.

Collapsing rule

→ If we have a tree with a depth very large, then once we find the root node of the leaves of that branch, we will directly point every other node of that brach to the root node of the tree.
→ This is a temporary solution.



1	2	3	4	5	6	7	8
-1	1	1	3	3	5	6	7

Once we find root node of 8; then we will directly point 7, 6, 5, 3 to the root node (i.e we will collapse the whole branch) Now each node of that branch will have parent = root node and their own connection with parent (like of 5 and 6) will collapse

```
Co Find(i)
    j ← i
    while P(j) > 0 do
        j ← P(j)
    repeat
        k ← i
        while k ≠ j do
            t ← P[k]
            P[k] ← j
            k ← t
        repeat
        return j
```

1/8/18

Divide and Conquer.

Break the problem into several sub problems that are similar to the original prob. but smaller in size. Solve the sub problems recursively and then combine these solutions to create a solution to the original problem.

① Finding maximum and minimum in an array.

```
Straight maxmin (A, n, max, min)
    max ← min ← A(1)
    for i ← 2 to n do
        if A(i) > max
            then max ← A(i)
        end if
        if A(i) < min
```

then min ← A(i)
end if
repeat.

Total comparisons — 2(n-1)

We can reduce the comparisons little bit by a small change in code.

if A(i) > max
then max ← A(i)
else if A(i) < min
then min ← A(i)
end if

Best comparison → when array is in ascending order.
Total comparison = n-1

Worst comparison → when array is in descending order.
Total comparison = 2(n-1)

is useful when ↑ elements is > 2

② Same question using divide and conquer.

MaxMin (i, j, fmax, fmin)
// where i = first element index
// j → last element index

Case:
: i = j ; fmax ← fmin ← A(i)
: i = j-1 ; if A(i) < A(j) then
fmax ← A(j)
fmin ← A(i)
else
fmax ← A(i)
fmin ← A(j)
: mid ← (i+j)/2

Call MaxMin (i, mid, lmax, lmin)
Call maxMin (mid+1, j, rmax, rmin)

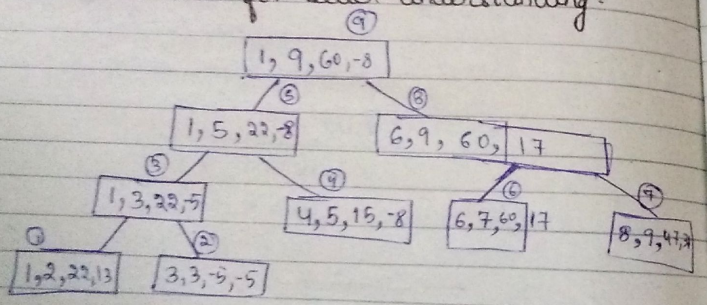
if (lmax > rmax) then fmax ← lmax
else fmax ← rmax

if (lmin < rmin) then fmin ← lmin
else fmin ← rmin

Let us say we have an array of 9 elements.

[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
ele- 22	13	-5	-8	15	60	17	31	47

To understand recursion, we should make recursion tree for better understanding.



$$T(n) = \begin{cases} T(1) & n=1 \\ aT(n/b)+f(n) & n>1 \end{cases}$$

→ a, b are known constants
→ f(n) is the function which took the time to divide the prob into sub problems & combining the sub problem

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

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

Assume $n = 2^k$

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

$= 4T(n/4)+4+2$ (At 3rd step)

$= 4\left(2T(n/8)+2\right)+4+2$

$= 8T(n/8)+8+4+2$ (At 4th step)

⋮

(At kth step)

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

$= 2^{k-1}+2^k-2$

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

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

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

* Although no:- of comparisons is less than that of straight method because it require stack space for all the variables and for given n elements it requires $[\log_2 n] + 1$ level of recursions.

In short, overhead of stack space or only overhead increases.

— x —

3/8/18 Merge sort using divide & conquer strategy

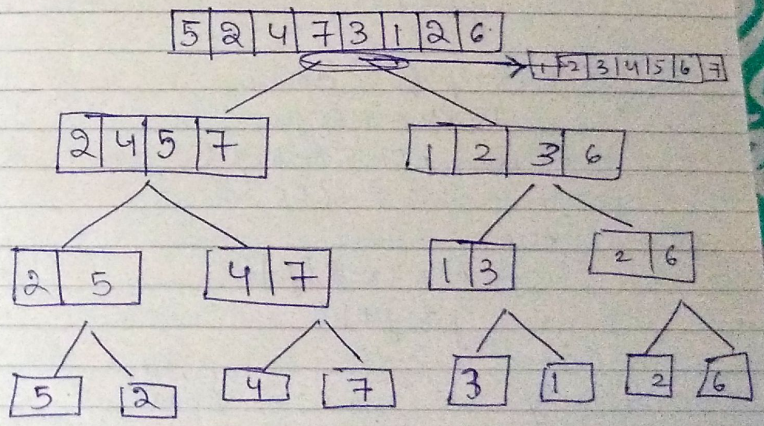
p	10	11	12	13	14	15	q
5	2	4	7	3	1	2	6

MergeSort (A, p, r)

if p < r
then q ← (p+r)/2

MergeSort (A, p, q)
MergeSort (A, q+1, r).

Merge (A, p, q, r).



Merge (A, p, q, r)

$n_1 \leftarrow q - p + 1$

$n_2 \leftarrow r - q$

```
for i ← 1 to n1
    L[i] ← A[p+i-1]

for j ← 1 to n2
    R[j] ← A[q+j]

L[n1+1] ← ∞ → (Sentinel in prog. lang)
R[n2+1] ← ∞

i ← 1   j ← 1

for k ← p to r do
    if L[i] ≤ R[j]
    then A[k] ← L[i]
         i ← i+1
    else
         A[k] ← R[j]
         j ← j+1.
```

Complexity

$T(n) \leftarrow 2T(n/2) + Cn$ → dividing into 2 parts; → cost of merging.

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

$= 4T(n/4) + 2Cn$ (2nd step)

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

$= 8T\left(\frac{n}{8}\right) + 3Cn$. (3rd step)

$\boxed{2^K T\left(\frac{n}{2^K}\right) + KCn}$

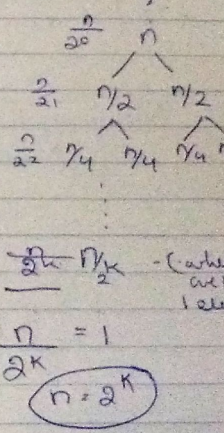
$= 2^K T(1) + KCn$.

Assume $n = 2^K$.

$= n + n\log_2(n)$

$= n + n\log n$

$O(n) = n\log_2 n$.

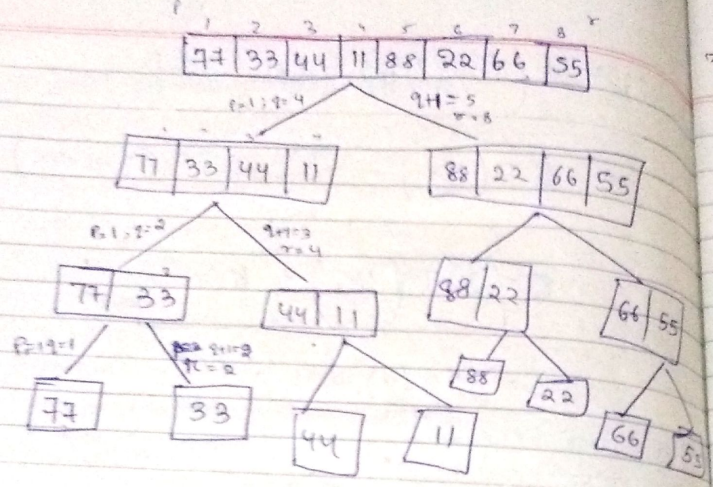


$\frac{n}{2^K} = 1$

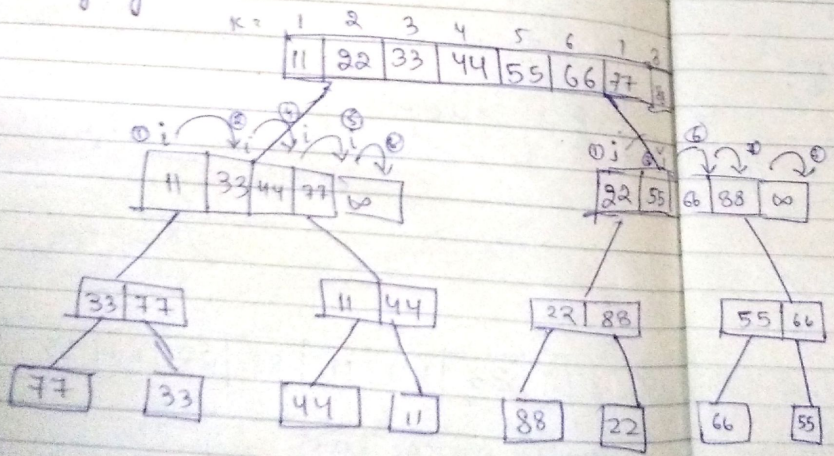
$n = 2^K$

Q | 77 | 33 | 44 | 11 | 88 | 22 | 66 | 55 |

Apply merge sort.

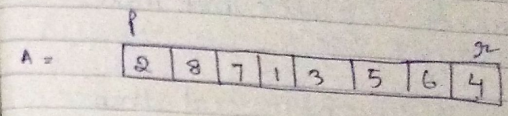


Merging



7/8/18

Quick-sort



QuickSort (A, P, r)

if P < r // more than one element

then q ← Partition (A, P, r)

Quicksort (A, P, q-1)

QuickSort (A, q+1, r)

Partition (A, P, r)

x ← A[r]

i ← P-1

for j ← P to r-1

if A[j] ≤ x

then i ← i+1

swap A[i] ↔ A[j]

}

swap A[i+1] ↔ A[r]

return i+1

→ Worst case partitioning → Left side will be present as a whole and there will be no right side.

→ If the array will be balanced, then it will act as a merge sort and time complexity - O(nlogn)

→ But if there will be worst case partioning then

$$T(n) \rightarrow T(n-1) + T(0) + cn \quad (cn \rightarrow \text{average case})$$

$$\vdots$$

$$+ 1 + n \quad (n \rightarrow n-1 \rightarrow \cdots \rightarrow 1)$$

$T(1) = 1$

$$n + (n-1) + \cdots + 1$$

$$\geq \frac{n(n+1)}{2} = O(n^2)$$

Assignment

→ Derive the complexity of worst case partioning of quick sort

→ Simulate merge sort & quick sort and evaluate the performance of both.

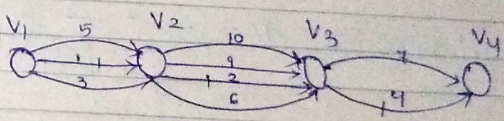
(Random generation of numbers in an array & apply quick & merge sort)

Greedy Algorithm

Suppose a problem can be solved by sequence of decisions then greedy method says that each decision is locally optimal.
These locally optimal solutions will finally add up to a globally optimal solution.

Eg:- Pick K numbers out of N numbers such that sum of these K numbers is the largest.

Eg2



Locally find the minimum distance b/w V_1 and V_2 and locally b/w V_2 and V_3 and V_3 and V_4. Finally summing up will give a globally optimal soln to find minimum distance b/w V_1 & V_4

Optimal Storage on Tape

You are given n programs that are to be stored on a computer tape of length

b. associated with each program i is length l_i. Assume that each time tape is positioned at front, if programs are stored in order $I = i_1, i_2, \ldots i_n$. Then time t_j required to retrieve a program i_j is given by

$$t_j = \sum_{K=1}^{j} l_{iK}$$

Then expected minimum retrieval time is given as

$$MRT = \frac{\sum_{j=1}^{n} t_j}{n}$$

The problem is to find the minimum MRT

$$\text{minimize MRT } d(I) = \sum_{j=1}^{n} \sum_{K=1}^{j} l_{iK}$$

Eg:- $n = 3 \quad (l_1 l_2 l_3) = (5, 10, 3)$

1, 2, 3 = (5) + (5+10) + (5+10+3) = 33
1, 3, 2 = 5 + (5+3) + (5+3+10) = 31
2 1 3 = 10 + (10+5) + (10+5+3) = 43
2 3 1 = 10 + (10+3) + (10+3+5) = 41
3 1 2 = 3 + 3+5 + 3+5+10 = 29

* Here greedy approach says arrange length in ascending order and retrieve program in that manner only.

Q We have n prog. and m tapes.

n	m	Find the minimum retrieval time.
↓	↓	
13	3	

$l_1 = 12 \quad l_2 = 5 ; \; l_3 = 8 ; \; l_4 = 32 ;$
$l_5 = 7 ; \; l_6 = 5 ; \; l_7 = 18 ; \; l_8 = 26 ;$
$l_9 = 4 ; \; l_{10} = 3 ; \; l_{11} = 11 ; \; l_{12} = 10 ;$
$l_{13} = 6.$

Step 1 Sort l_i in ascending order

3, 4, 5, 5, 6, 7, 8, 10, 11, 12, 18, 26, 32

Step 2.

$T_0 =$ 3 5 8 12 32
$T_1 =$ 4 6 10 18
$T_2 =$ 5 7 11 26

→ Ans

This will give us the minimum time

generalized: ith program should go to Timodm tape

where $i = 0, 1, 2, \ldots$
$m = 0, 1, 2, \ldots$

8/8/18

Knapsack problem

You are given some +ve integers, let us say, $P_1, P_1, \cdots P_n$.

You have n items and profits attached $P_1, P_2 \cdots P_n \rightarrow P_i$ is the profit of the ith item.

$w_1, w_2 \cdots w_n \rightarrow$ wt of each item

$w_i \rightarrow$ weight of the ith item.

$M \rightarrow$ integer M which is the capacity of knapsack

Find $x_1, x_2, x_3 \cdots x_n$, $0 \le x_i \le 1$ such that

$$\sum_{i=1}^{n} P_i x_i \text{ is maximized.}$$

subject to $\sum_{i=1}^{n} w_i x_i \le M$.

If $x_i = 0$ that means you haven't picked ith item at all.

$x_i = 1 \rightarrow$ picked ith item fully.

$0 < x_i < 1 \rightarrow$ means fraction of the ith item.

Eg:- $M = 20$
$n = 3$ items.
$(P_1, P_2, P_3) = (25, 24, 15)$
$(w_1, w_2, w_3) = (18, 15, 10)$

→ Want to find x_1, x_2 and x_3.

x_1 x_2 x_3	$\sum w_i x_i$	$\sum P_i x_i$	
$\frac{1}{2}, \frac{1}{3}, \frac{1}{4}$	$9+5+2.5 = 16.5$	$12.5+8+3.75 = 24.25$	
$1, \frac{2}{15}, 0$	$18+2+0 = 20$	$25+3.2+0 = 28.2$	(according to profit)
$0, \frac{10}{15}, 1$	20	31	(according to weight)

→ One greedy approach is arrange in decreasing order the ~~weights~~ profit and fill the sack fast (I was unable to pick some items)

→ 2nd greedy approach is arrange ~~profit~~ weight in ascending order so that our sack fill slowly.

→ Better approach is in which profit is more & weight is less. ie, considering P/w. (This is our best approach)

So now, our third approach is sort P/w in decending order.

1	2	3	$x_1 x_2 x_3$	$\sum w_i x_i$	$\sum p_i x_i$
P/w	P/w	P/w			
$\frac{25}{18}$	$\frac{24}{15}$	$\frac{15}{10}$	0, 1, $\frac{2}{3}$	20	31.5
= 1.38	$= \frac{8}{5}$ = 1.6	= 3/2 = 1.5			Maximum

Greedyknapsack (m, n)

// P[1:n] & w[1:n] such that $\frac{P[i]}{w[i]} \geq \frac{P[i+1]}{w[i+1]}$

Now: P = [24, 15, 25]
w = [15, 10, 18]

// m is the size of the knapsack, X[1:n] → solution vector

```
for i=1 to n do X[i] = 0
    U = m;
for i=1 to n do
{
    if w[i] > U then break;
    else
        x[i] = 1.0;
        U = U - w[i];
}
if (i ≤ n) then x[i] = U/w[i];
```

Q. P = (27, 22, 16)
w = (18, 14, 10)
M = 40.

1	2	3
$\frac{27}{18}$	$\frac{22}{14}$	$\frac{16}{10}$
1.5	1.57	1.6

P = [16, 22, 27]
w = [10, 14, 18]
$x_{[]}$ = [0, 0, 0]
U = 40.

① $x_{[1]}$ = [1, 0, 0]
U = 30

② $x_{[2]}$ = 1
x = [1, 1, 0]
U = 16

③ $x_{[3]} = \frac{16}{18} = \frac{8}{9}$

$X = [1, 1, \frac{8}{9}]$

Profit = 1×16 + 1×22 + 27×8/9
= 16 + 22 + 24
= 62

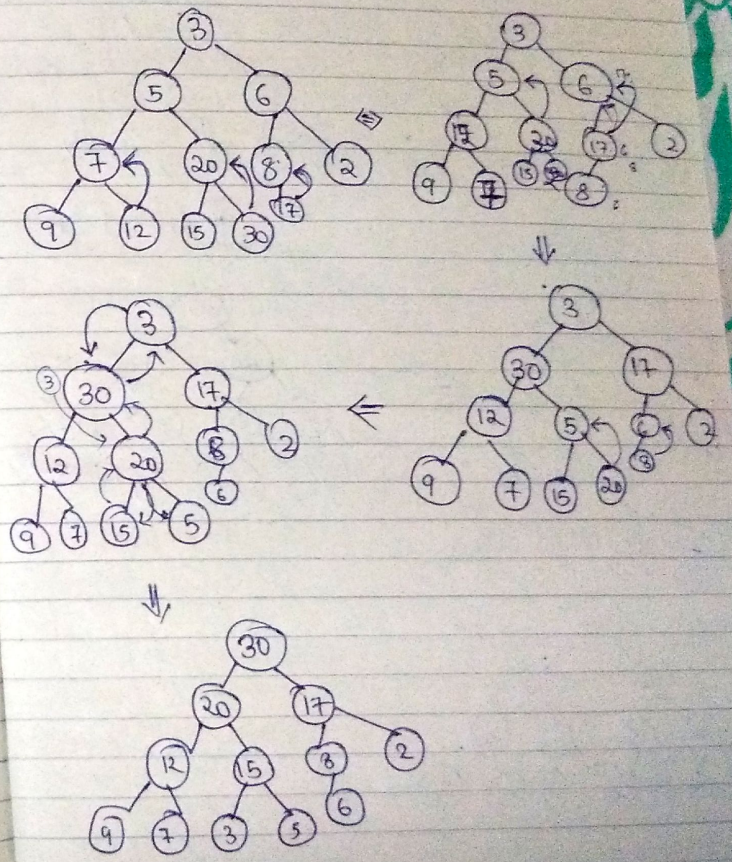
Algorithm of storing n program in m tapes

```
store (n, m)
// n is no. of prog. & m is no. of tapes
{
  j = 0;
    for i = 1 to n do
    {
      write {"append program", i,
                "to tape", j);
      j = (j+1) mod m;
    }
}.
```

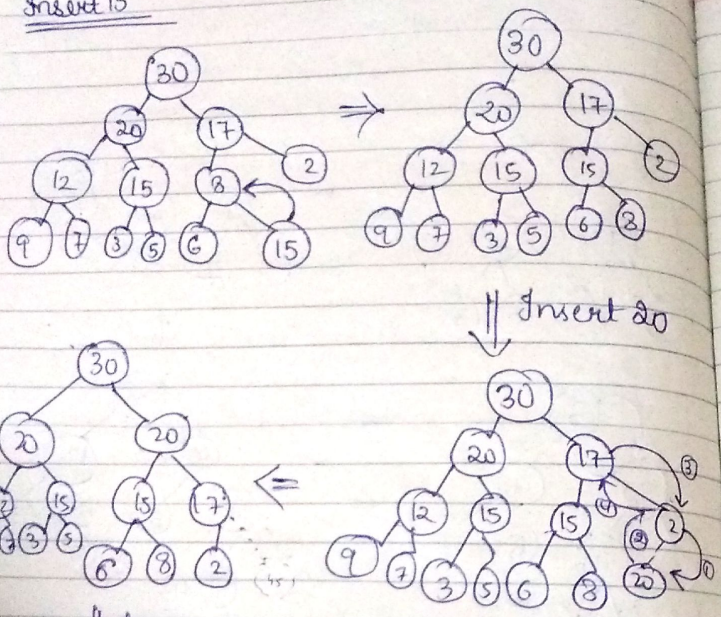
9/8/18

TUT-2

Q 3, 5, 6, 7, 20, 8, 2, 9, 12, 15, 30, 17

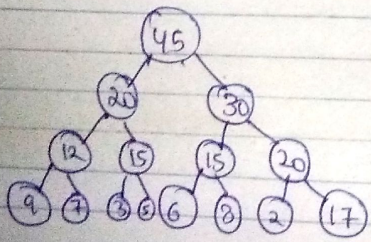


Insert 15



Insert 20

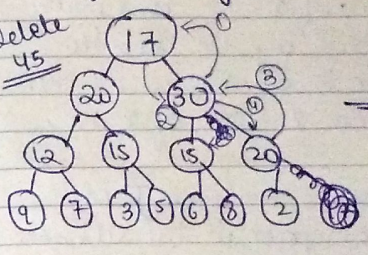
Insert 45.

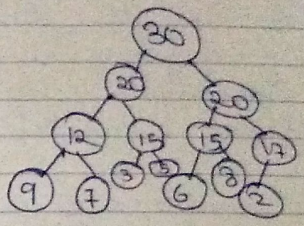


Deletion.

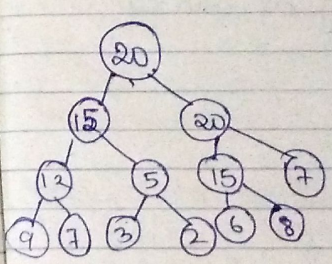
If root node is deleted then last node will be shifted to root node and then heap is formed again.

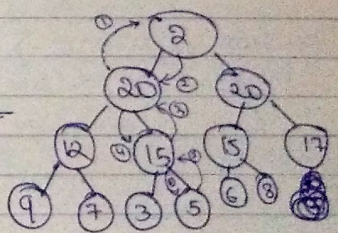
Delete 45



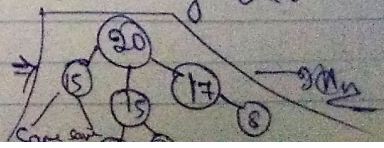


delete 30

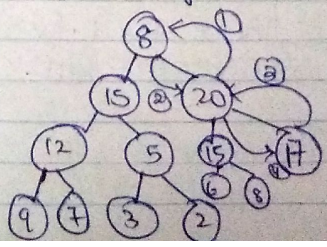




(In this case j and j+1 are equal so we will take the left tree) as j = 2*i



delete root again



10/8/18

Job sequencing with deadline (Imp for simulation)

Problem → We are given n jobs associated with any job i, there is integer deadline (d_i) > 0; for any job i the profit P_i is earned if job is completed by its deadline. To complete a job one has to process the job on a machine for 1 unit of time.

Feasible solution is a subset J of jobs such that each job in the subset can be completed by its deadline.

Objective function → $\max \sum_{i \in J} P_i$

Eg:- $n = 4$. $P_1, P_2, P_3, P_4 = (100, 10, 15, 27)$

$d_1, d_2, d_3, d_4 = 2, 1, 2, 1$.

Here at max we can process 2 jobs as after 2 seconds; there will be jobs missing it's deadline after that.

Feasible sol.	sequence	$\sum P_i$
(1,2)	2, 1	110
(1,3)	1,3 or 3,1	115
(1,4)	4,1	127 → maximum
(2,3)	2, 3	25
(3,4)	4,3	42
{1}		100
{2}		10
{3}		15
{4}		27

→ greedy approach → Pick maximum profit's job first.

Q1. P = 10, 34, 67, 45, 23, 99.

d = 3, 4, 3, 1, 5, 2.

→ Sort according to descending profit.

P =	99	67	45	34	23	10
d =	2	3	5	4	5	3

→ Maximum deadline = 5; that
so maximum job will be finished = 5.

1	2	3	4	5
99	67			

now 3rd in order have 1 as deadl & we have space so we can shift the job to right.

1	2	3	4	5
45	99	67	34	23

→ final soln.

Q2. P = 20, 15, 10, 5, 1.

d = 2, 2, 1, 3, 3.

1	2	3
20	15	5

One solution is place every job in its deadline slot
Sol² see in Q2. and now when same deadline job comes then we will keep it "hold and then we

Assignment → Simulate this algo

will compare it with next job that fit the deadline.

1	2	3
15	20	5

Sol3 start with descending deadline. ie, start backward, first find the position of the job with maximum deadline.

Q1 →

1	2	3	4	5
45	99	67	34	23

Q2 →

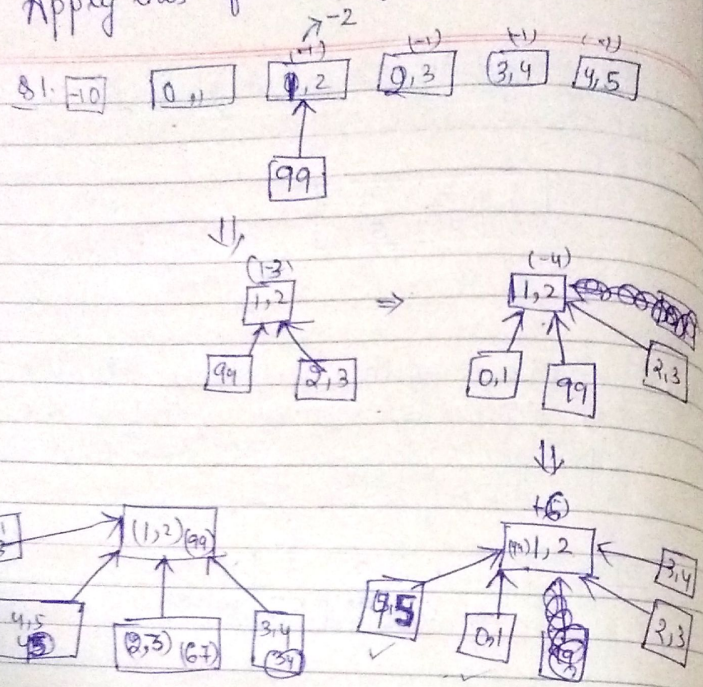
1	2	3
15	20	5

Sol4 → The fastest sol^n is with union & find and it never fails.

If job i has not been assigned a processing time then assign it to slot $[\alpha-1, \alpha]$ where α is the largest integer x such that it lies b/w $1 \le x \le d$ and provided slot α & $\alpha-1$ is free

Apply this after sorting

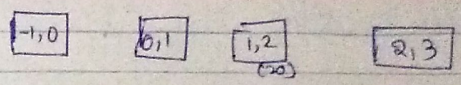
Q1.



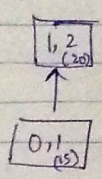
Final answer →

1	2	3	4	5
23	99	67	34	45

Q2

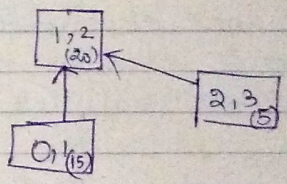


⇓ when 15 comes.



⇓ when 10 comes it need 0,1 which is busy it will go to -1,0 so it will get rejected

⇓ Now when 5 comes, it need 2,3 slot which is free so it will go there & it will attach to 1,2



⇓ When 1 comes, there is no free slot so it will ~~not~~ get rejected

So our final answer is:-

1	2	3
15	20	5

→ Ans

14/8/18

Huffman's code

→ Suppose we have 1Lk characters appearing in our code.
→ But there are 6 distinct characters that represent this 1Lk character.

	a	b	c	d	e	f
freq (thousands)	45	13	12	16	9	5
fixed length	000	001	010	011	100	101
variable length	0	101	100	111	1101	1100

→ If we store these characters as fixed length code then we need 3×100000 bits = 3lks bits.

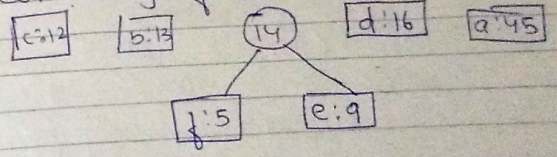
→ When we see variable length the no:- of bits = $45 \times 1 + 13 \times 3 + 12 \times 3 + 16 \times 3 + 9 \times 4 + 5 \times 4 = 224000$ bits.

→ We we arrange them in ascending order according to their frequency (greedy approach)

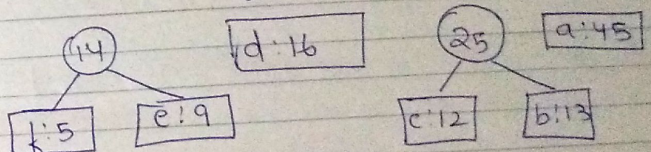
f:5 e:9 c:12 b:13 d:16 a:45

Now we will merge and again sort according to their new position in the arranged characters.

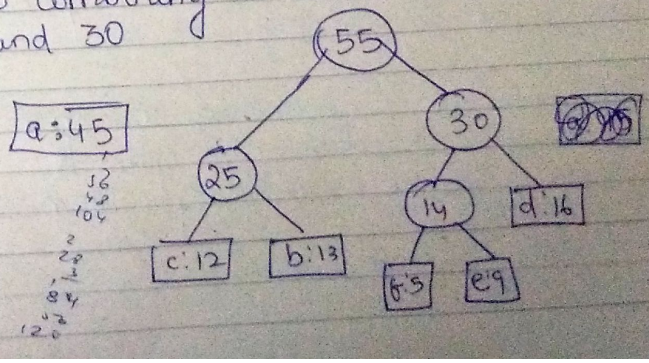
→ On combining f and e:



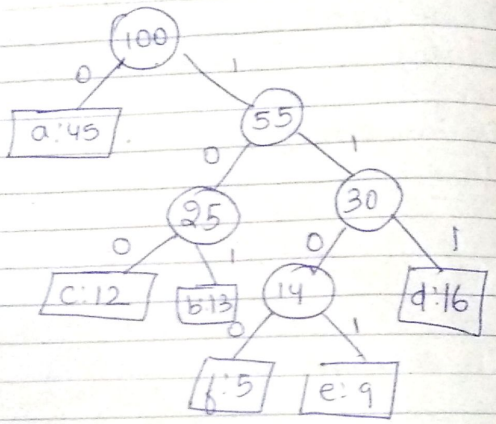
Now combining c and b



Now combining 16 and 14 and then 25 and 30



Now combining 45 and 55.



→ Assign 0 to all left nodes and 1 to all right nodes.

Neither of the code word is prefix of another code. That is why this is called prefix code.

→ If your tree is generating a prefix code then it will give optimal solution for compression problem.

a = 0	d = 111
c = 100	e = 1101
b = 101	f = 1100

→ If we want to encode the given compression code, then we can find character also.

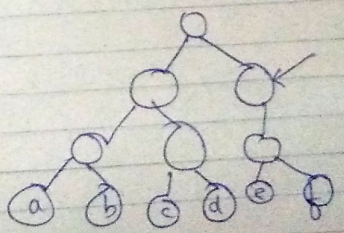
Suppose eg:- 0|0|101|1101|

aabe → Ans

→ It should always be a full binary tree (Full binary tree → either zero or two children). Only then we will get a optimal solution for compression.

→ If there are c characters in your file then there will be c number of leaf nodes in your tree and c-1 internal nodes.

→ If we see the tree of fixed length characters



(This is not our full binary tree so it doesn't give an optimal solu

$c \in C$ → (set of distinct characters in file).

c.freq - frequency of appearing of that character.

$d_T(c)$ = depth of that prefix code in a prefix tree

$d_T(e)$ of e is 4
$d_T(a)$ of a is 1
$d_T(c)$ of c is 3

} = length of the prefix code of that character

→ Number of bits required to stored the file is given as:-

$$B(T) = \sum_{c \in C} c.freq \times d_T(c)$$

Huffman(c)
n ← |c|
Q ← C ← (make min heap)
for i ← 1 to n-1
do allocate a new node z
left[z] ← x ← Extract ← min(Q)
right[z] ← y ← Extract ← min(Q)
f(z) ← f(x) + f(y)
Insert(Q, z)
}
return Extract-min(Q)

* Can be implemented through priority queue and min heap.
* complexity = n log n.

Assignment
Write function for extract min(Q) (min heap & delete root and insert n at root)

Optimal merge Pattern

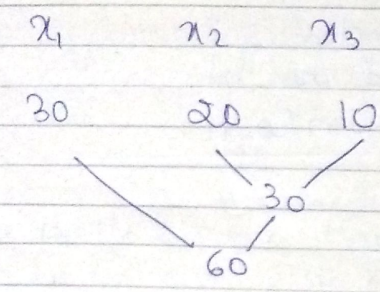
x_1	x_2	x_3	(We have 3 files)
30	20	10	(with these much no:- of records)

Prob:- We have to merge these files such that it get merge in minimum no:- of moves.

When we merge 2 files with m and n elements, then total moves required = $m+n$

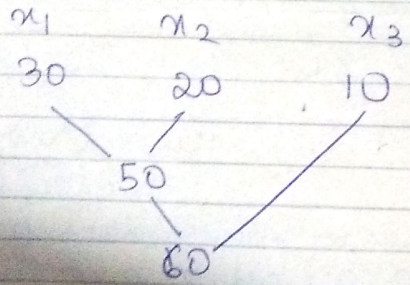
greedy approach:

We will start with minimum no:- of records (ie, in ascending order)



Total moves
$= 30 + 60$
$= 90$.

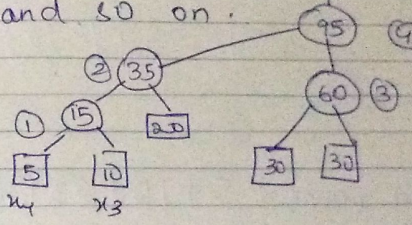
Without greedy approach



Total moves
$= 50 + 60$
$= 110$ (which is more

Q x_1 x_2 x_3 x_4 x_5 (20, 30, 10, 5, 30)

→ First find 2 min and merge them and then find the next two minimum and merge them and so on.



Total steps = $15 + 35 + 60 + 95$
$= 205$.

if d_i is the distance (depth) of x_i file from the root and q_i is the length of x_i then total no:- of moves required to merge =

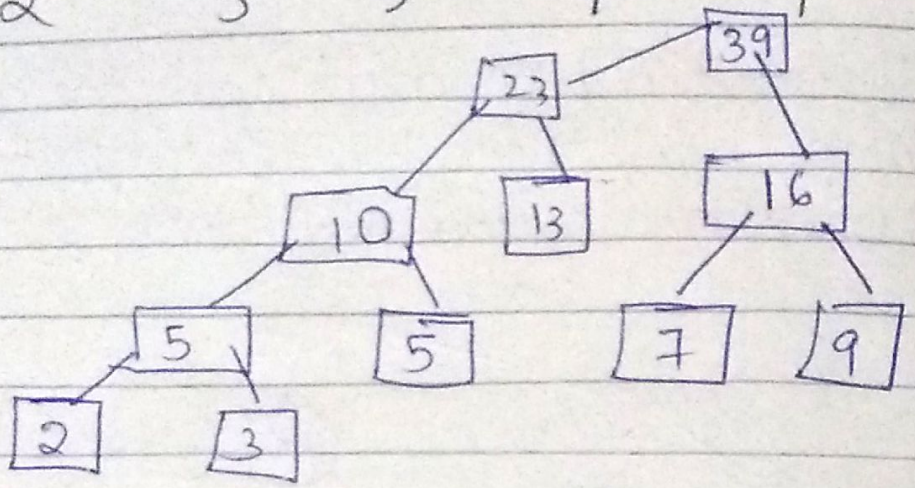
$$\sum_{i=1}^{n} d_i q_i$$

$= 3\times5 + 3\times10 + 2\times20 + 2\times30 +$
2×30 (only of records)

$= 15+30+40+60+60$
$= 205$.

Q. x_1 x_2 x_3 x_4 x_5 x_6

2 3 5 7 9 13



Total steps = 5 + 10 + 23 + 16 + 39

= 15 + 16 + 23 + 39

= 31 + 23 + 39

= 93.

With formula -

$4 \times 2 + 4 \times 3 + 3 \times 5 + 2 \times 13 + 2 \times 7 + 2 \times 9$

$= 8 + 12 + 15 + 26 + 14 + 18$

$= 93$