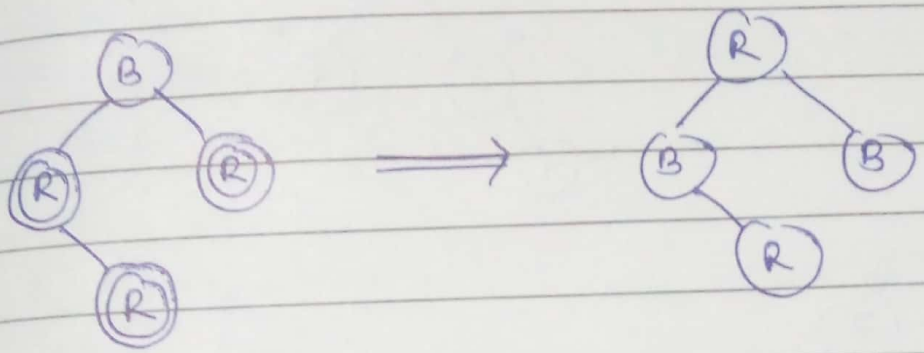
12	15	27	28	29

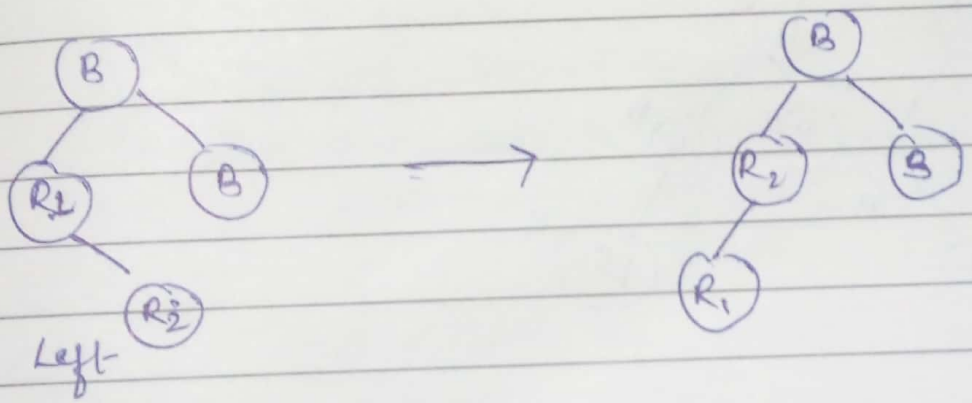
10	11	13	26	34

10	11	12	13	15	26	27	28	29

1.

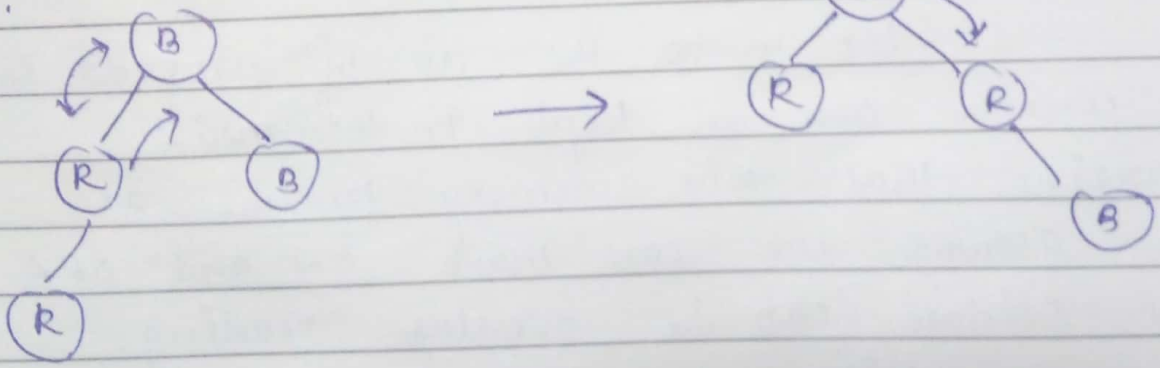


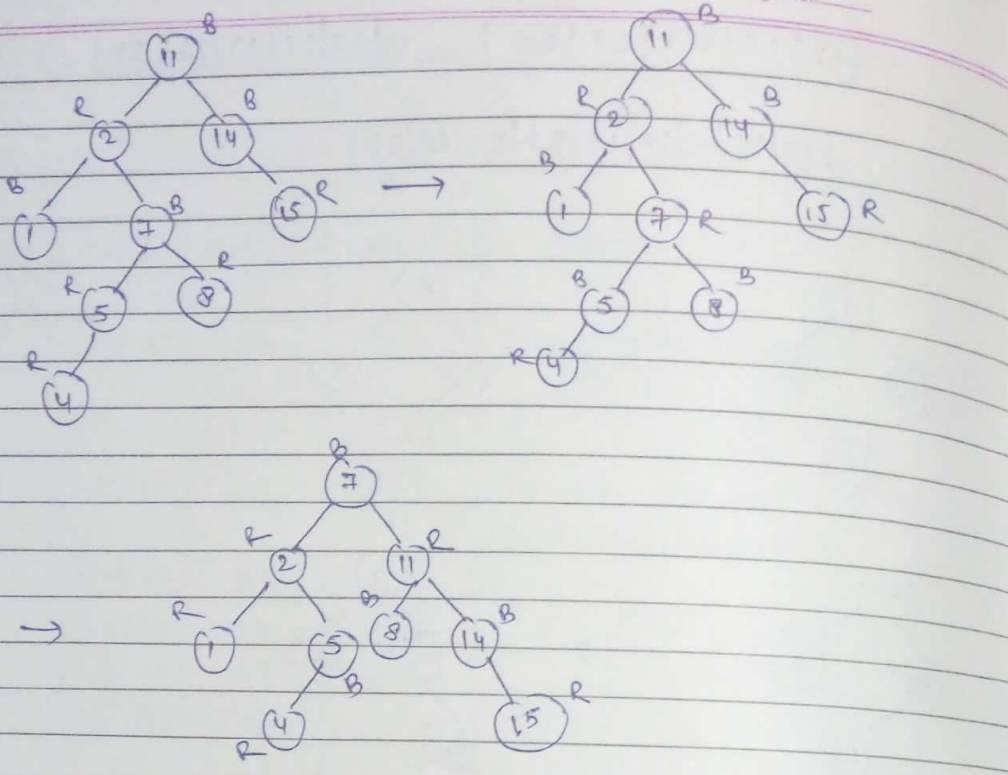
2.



Left

3.





Replacement selection Sort -

Let m be the no. of records to be sorted which can be kept in the main memory imagine that this memory location are registers an assume we can mark ON and OFF replacement section can be overlap reading sorting and writing now an algorithm

Step - 1. The m registers are filled with records from the input to be sorted.

Step - 2 All registers are put into the ON state.

Step - 3 Select the resistor which has the smallest of all ON state.

Step - 4 Transfer the content of the selected register to the output (Y)

Step - 5 Replace the content of the selected register by the next input record.

(i) if new record key > Y (Go to step 3)
(ii) if new record key = Y (go to step 5)
(iii) if new record key < Y (go to step 6)

Step - 6. Turn the selected by resistor off.

(i) if all resistors are new off we have completed the sorted sub string.

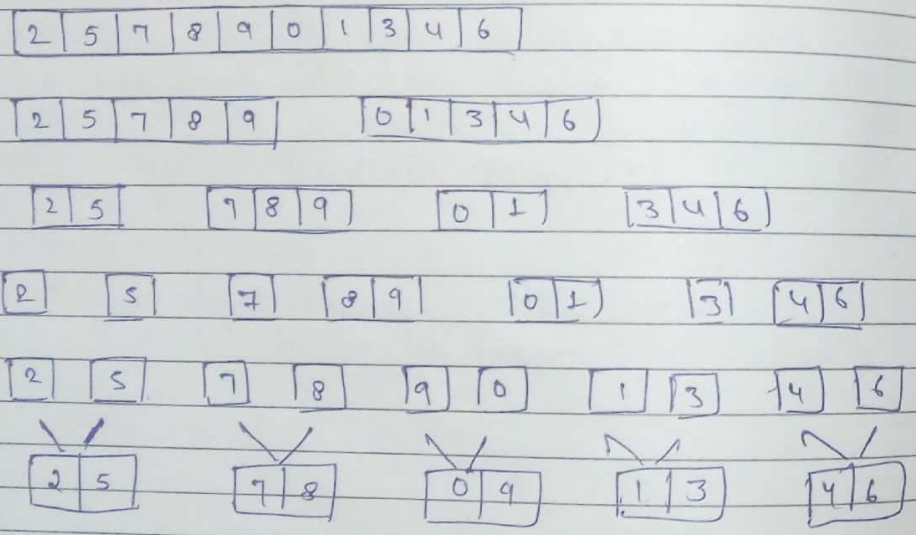
(ii) we start a new sub string group and go to step - 2

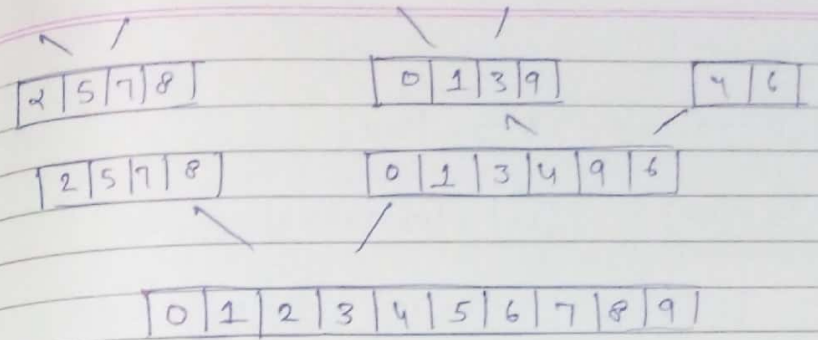
else go to step 3.

Q.1. 5, 2, 9, 7, 0, 8, 1, 6, 3, 4 for this input key suppose the number of resistor is 3.

Register	output (Y)
5, 2, 9	2
5, 7, 9	5
* 0, 7, 9	7
* 0, 8, 9	8
* 0, * 1, 9	9
* 0, * 1, 6 *	
0, 1, 6	0
3, 1, 6	1
3, 4, 6	3
-, 4, 6	4
-, -, 6	6
-, -, -	-

Now using merge sort:



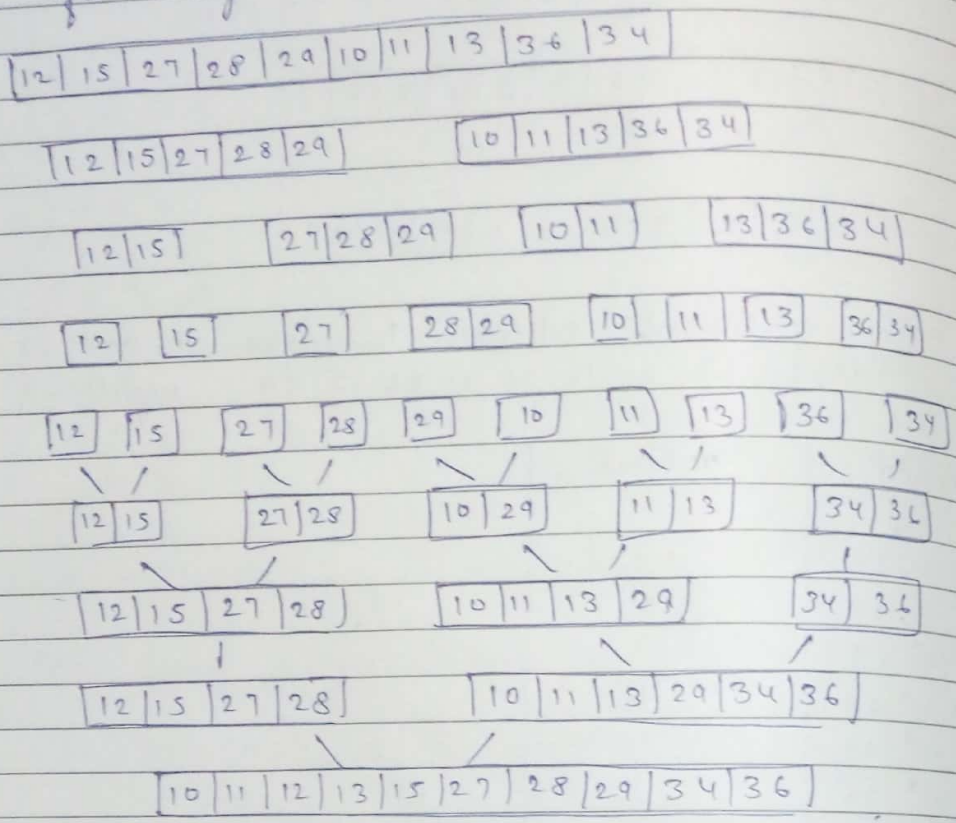


Q. Apply replacement selection technique to sort :→
15, 20, 29, 27, 10, 28, 11, 26, 13, 34 Register - 3

Register	output
15, 12, 29	12
27, 15, 29	15
27, 10*, 29	27
28, 10*, 29	28
* 11, 10*, 29	29
* 11, 10*, 26*	
11, 10, 26	10
11, 13, 36	11
34, 11, 36	13
34, -, 36	34
-, -, 36	36
-, -, -	-

Date: / / Page No:

After Merge sort

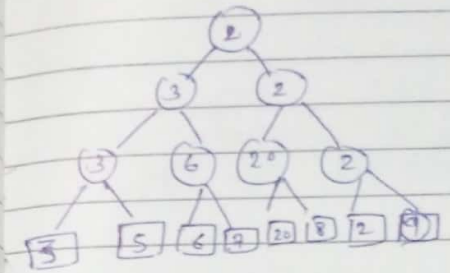


Tournament TREE :→

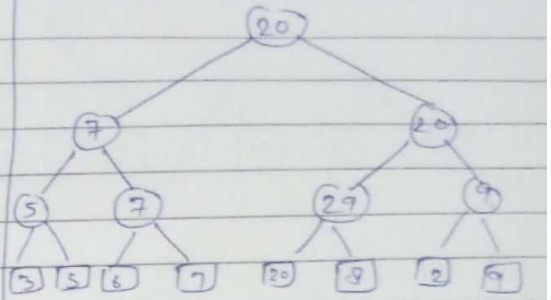
3, 6, 5, 6, 7, 20, 8, 2, 9

Date: / / Page No:

Minimum tournament tree



Maximum tournament Tree

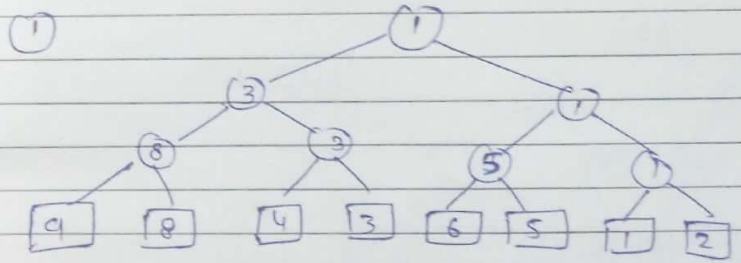


A tournament Tree is a complete binary tree with n external node n & n-1 internal nodes the external represent the winner of the match b/w two teams. A tournament - tree also called winner tree. A root of the tournament tree represent the overall winner of the tournament

Application of Tournament Tree :→

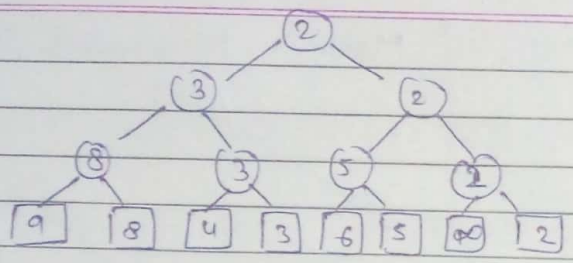
Sorting is an application of tournament - tree now the examples are -

9, 8, 4, 3, 6, 5, 1, 2



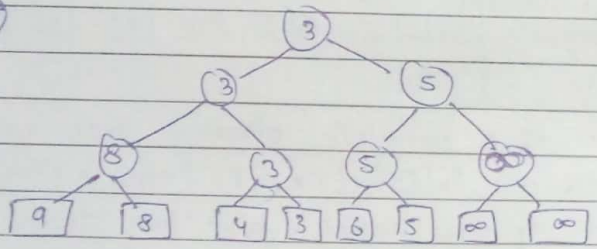
Date: / / Page No.:

2



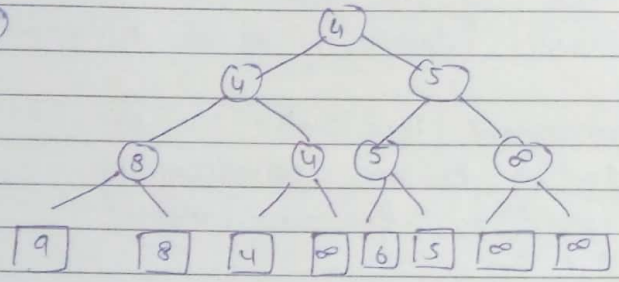
1

3



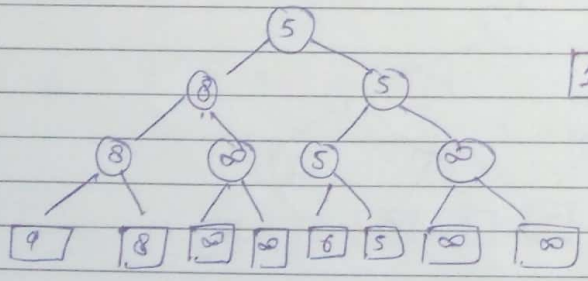
1	2

4



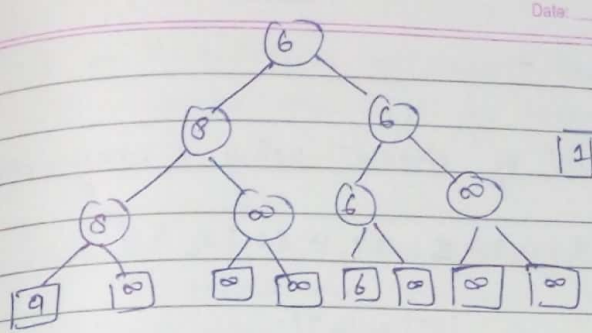
1	2	3

5



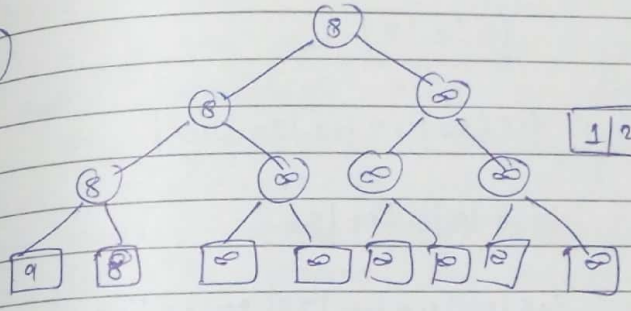
1	2	3	4

6



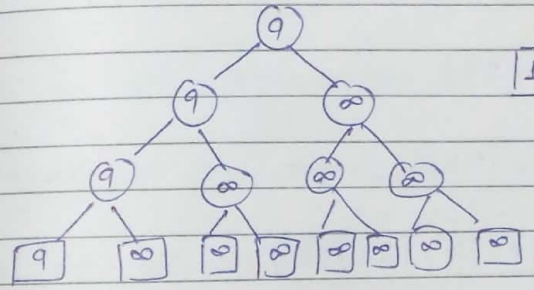
1	2	3	4	5

7



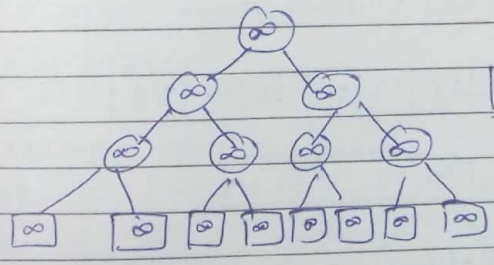
1	2	3	4	5	6

8



1	2	3	4	5	6	8

9



1	2	3	4	5	6	8	9

Date: / / Page No:

External merge sort -

Sort the following list of element using 2 way merge sort $m = 3$

20, 4, 15, 8, 9, 4, 40, 30, 12, 27, 11, 56, 28, 35

d_1 = 20, 47, 15 | 15 | 20 | 47 |

d_2 = 8, 9, 4 | 4 | 8 | 9 |

d_1 = 40, 30, 12 | 15 | 20 | 47 | 12 | 30 | 40 |

d_2 = 17, 11, 56 | 4 | 8 | 9 | 11 | 17 | 56 |

d_2 = 28, 35 | 15 | 20 | 47 | 12 | 30 | 40 | 28 | 35

D_A - | 4 | 8 | 9 | 15 | 20 | 27 |

D_B | 11 | 12 | 17 | 30 | 40 | 56 |

D_4 | 4 | 8 | 9 | 15 | 20 | 47 | 28 | 35 |

D_6 | 11 | 12 | 17 | 30 | 40 | 56 |

D_A | 4 | 8 | 9 | 11 | 12 | 15 | 17 | 20 | 28 | 30 | 35 |

| 45 | 47 | 56 |

Time complexity (N log N/n)

Date: / / Page No:

File :→ It is collection of record where the file size is limited the size by memory and storage medium.

File Organization :→ File organization ensure that record are available for passing. It is used to determine file organization, for each base relation.

ex - if we want to retrive records in file by alphabatical order of name sorting the file by employ name is good file organization. How ever if we want to retrive all employ whose marks are in a certain range a file is ordered by employ name would not be a good file organization.

Type of file organisation :→

1. Sequential file organization - Searching and sorting in continous blocks within files on top or disk it is called sequential file organization. records are sequentially in order. The decending order of field.

It starts from beginning of file and record can be added to the end of the file. It is not possible to add a record in middle of file without rewrite the file.

Date: / / Page No:

Advantage :- It is simple to program easy to design

Disadvantage - (1) It is time consuming process.
(2) It has high data redundancy Random search is not possible.

(2) Direct file Organisation - It is also known as random access or relative file organisation. In it all record are store in direct storage device such as hard disk. The records are randomly placed throughout the file. The record does not need to be in sequence. Because they updated directly an rewritten back in same location. It is also known as hashing.

(3)

Advantage - (1) It helps in online Transaction processing system like online reservation.

(2) In direct access file, sorting of records are not required, It updates several file quickley and have better control over record allocation.

Disadvantage - (1) It is expensive.
(2) less storage space as compare to sequential file organisation.

(4)

(3) Index sequential file organisation - It combines both sequence and directed file organisation. It records are store randomly on direct access using primary key. The data can be access sequentially ran-domly using index.

Advantage - In it sequential file or random file access is possible.
(2) It access the record by very fast if the index table is properly arranged.
(3) Records can be inserted in midde of file.

Disadvantage - It required keys and periodic reorganisation.

External Merge Sort - In external merge sort data store on secondary memory is part by part loaded into main memory, sorting can be done over them. The sorted data can be then sorted in intermidiate files. Finally this intermidiate file can be merge repatedly to get sorted data the use amount of data can be sorted using this technique.

Suppose there are thousands of records that can be sorted using external sorting method-

Multiway merge sort — Multiway merge sort is a technique of merging m sorted list into single sorted list. The two way merge sort is a special case of multiway merge sort.

Algorithm —

(1) Divide the element into the block of size m. Sort each block and write runs on disk.

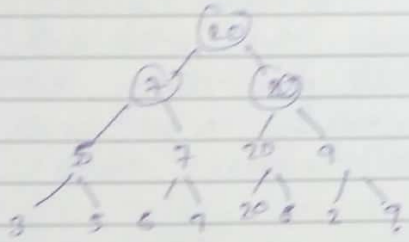
(2) merge 2 runs
- (a) Refirst value on each of 2 runs
- (b) Compare and sort
- (c) Write on output tape.

(3) Repeat the and get longer and longer runs and alternate tapes. Finally we get single sorted list.

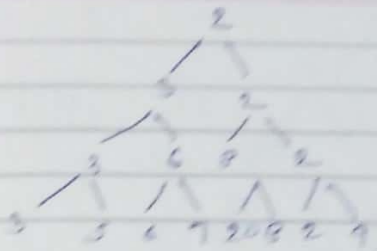
Tournament Tree —

Draw min & max winner Tree for 3, 5, 6, 7, 20

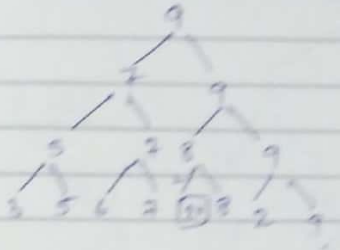
max winner tree



min winner tree

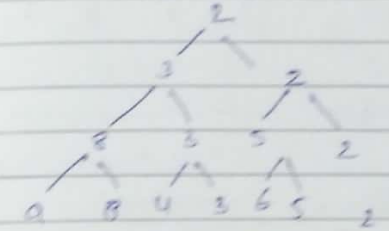


replace 1 max winner tree

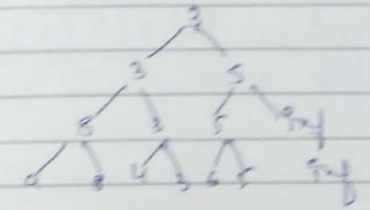


replace 1 min winner tree

(4) 9, 8, 4, 3, 6, 5, 2

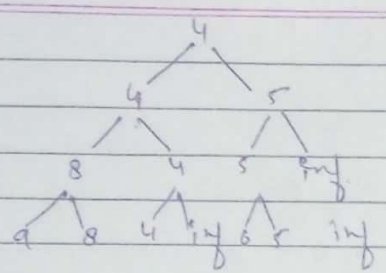


(1)



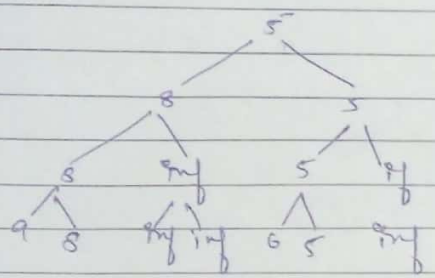
2

ii



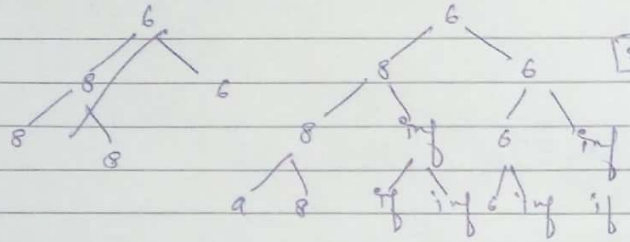
2	3

iii



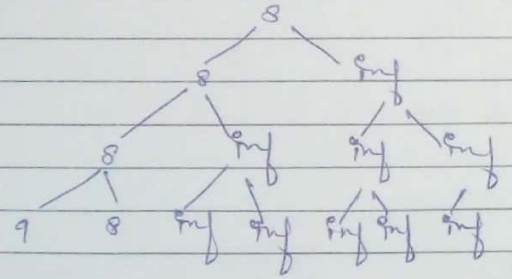
2	3	4

iv.



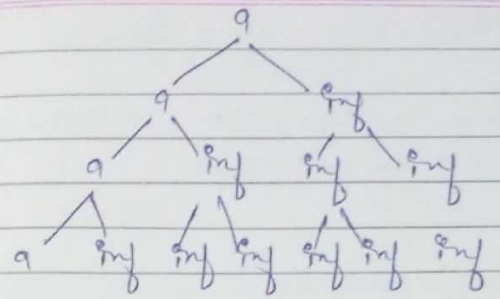
2	3	4	5

v.



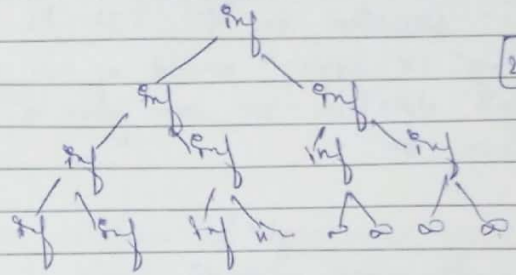
2	3	4	5	6

vi



2	3	4	5	6	8

vii



2	3	4	5	6	8	9

Tournament tree - It is complete binary tree with an external no nodes and n-1 internal node the external node represent players and internal node represent the winner of the match of two players. It is also called winner tree. selection tree. The root of tournament tree represent allover winner of tournament.

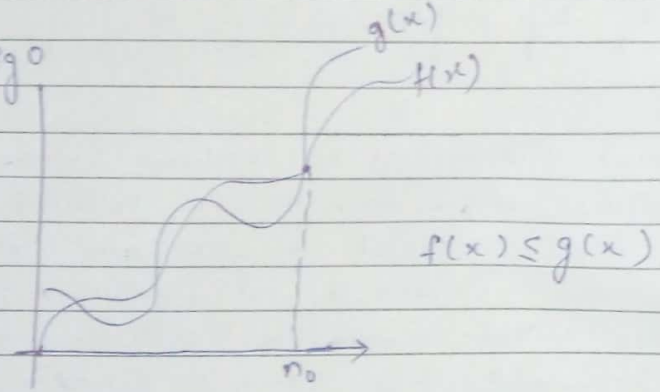
Various operation perform on winner tree are -.

1. Initialize - It takes bigo of O(n) time.

ii) Get winner - It takes bigo of 1 time

iii) Remove and replace winner - It takes bigo of log(n) time.

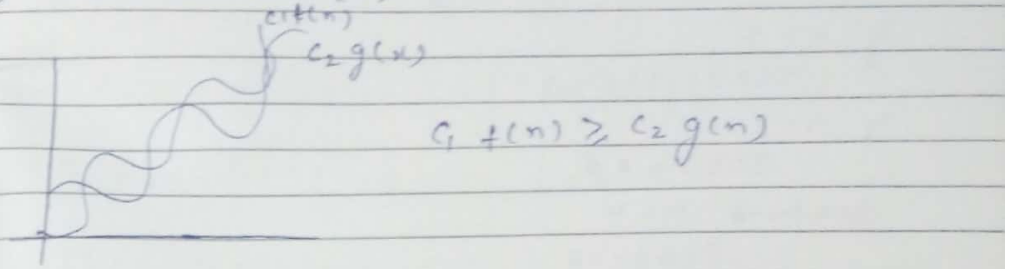
(i) Big O Notation :- Denoted by 'O'. It is a method of representing a upper bound of an algorithm run time. Using big O Notation we can give a longest amnt of time given an



$f(x) = 2x + 4$	$n = 2$	$n = 2$
$g(x) = x^2$	$f(n) = 6$	$f(n) = 9$
$n = 3$	$g(n) = 4$	$g(n) = 9$
$f(n) * n = 1$	$f(n) \geq g(n)$	$f(n) = g(n)$
$f(n) = 3$		
$g(n) = 1$		
$f(n) > g(n)$		

(ii) Omega Notation :→ (Ω) This notation is used to represent the lower ~~bond~~ bond of an algorithm running amnt. Using Omega Ω Notation we can explain or denote the shortest amnt. of time taken by an algorithm.

(iii) Theta Notation (θ) :→



The theta Notation is represented by θ. By this method the running time is b/w upper bond and lower bond

Recurrence Relation :→ The recurrence relation is an equation that define a sequence of recursively. It is normally in the following form -

$$T(n) = T(n-1) + n \quad \text{for } n > 0$$
$$T(0) = 0 ;$$

Here T(n) is called recurrence relation and T(0) is called ~~recurrences~~ initial condn. Solving

Solving Recurrence Relation - The recurrence relation can be solved by substitution method.

There are 2 types of substitution method. -

1) Forward

2) Backward

(1) Forward substitution -

~~T(0)~~ n = 1

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

$= T(0) + 1 = 1$

If n = 2

$T(2) = 1 + 2 = 3$

If n = 3

$T(3) = 6$

~~T(4) = 6~~ n = 4

$T(4) = 10$

n^{th} term $= \frac{n^2}{1} + \frac{n}{2}$

$= O(n^2)$

(2) Backword -

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

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

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

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

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

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

⋮

K ⋮

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

$K = n$

$T(n) = T(0) + 1 + 2 + \dots + n$.

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

$O(n^2)$

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

~~for K = n~~ $T(n/2) = 2T(n/4) + n/2$

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

$T(n) = 4T(n/4) + n$

$T(n/4) = 2T(n/8) + n/4$

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

$T(n) = 2^3 T(n/2^3) + 3n$

$T(K) = 2^K + \left(n/2K\right) + Kn$.

$T(n) = T(n-1) + 1$ — (1)

$T(0) = 0$

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

$= T(n-2) + 1$ — (2)

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

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

k^{th}

$T(K) = T(n-K) + K.$

Put $K = n$

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

$T(n) = T(0) + n$

$T(n) = 0 + n.$

$O(n)$

8 $T(n) = 3T\left(\frac{n}{3}\right) + n$ Ans $n \log_3 n$

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

$T\left(\frac{n}{3}\right) = 3T\left(\frac{n}{9}\right) + \frac{n}{3}$

=

$T\left(\frac{n}{9}\right) = 9\,T\left(\frac{n}{9}\right) + 2n.$

$T\left(\frac{n}{9}\right) = 9\,T\left(\frac{n}{27}\right) + \frac{n}{9}$

$T(n) = 27\,T\left(\frac{n}{27}\right) + 3n$

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

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

assume that $3^K = n$

$\log 3^K = n$

$K = \log_3 n$

$T(n) = n\,T(1) + n \log_3 n$

$= n + n \log_3 n$

Augmentation in Red Black Tree :-

Augmenting a data structure means adding some extra information to the existing data structure so that the data structure can be implemented efficiently. Typically any node in a binary tree contains some useful information such as data left pointer, right pointer, parent node and colour of node.

This information is useful for accessing particular node more efficiently from the given tree

Dynamic Order statistics :→

For augmenting a data structure we will simply store additional information of size The augmented tree is also called as order statistic tree. Basically the order statistics tree is a tree which supports order statics operations.

Hence every node in order statistics tree will we have following information -

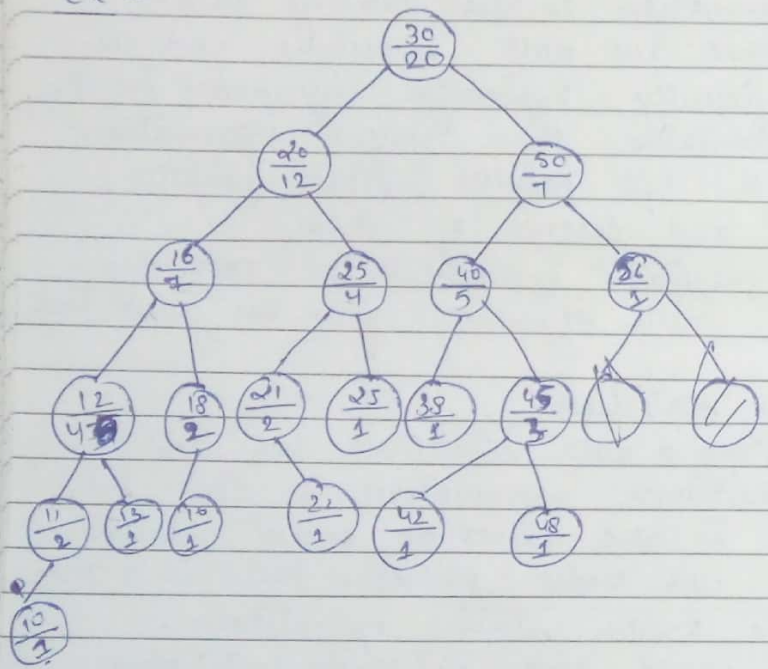
① Key Value
② Colour
③ Address of parent node
④ Adress of left child
⑤ Address of right child
⑥ Size of that node.

Size of Node :→

Size [K] = size [left (k)] + size [Right (k)] + 1

* Size of leaf node is always 1.

Ex –



Computing the Rank of Order statistics tree :→

In order to retrive the position of a node in order statistic tree. We need to compute the rank of each node.

Following algorithm illustrates how to retrive the desire node from the binary tree with the help of ~~rang~~ rank.

Algorithm select (K; i)
rank ← Size [left (K)] + 1
if (i = rank) then
return K
else if (i < rank) then
return select [left (K), i)
else
return select (right [K]), i->rank).
(i - rank)

Ques. Let us however obtain the 17th smallest element from the binary tree.

i = 17, K = 30
Rank = 12 + 1 = 13
i ≠ rank
i > Rank
17 > 13
K = 50,
i = 4
Rank = 5 + 1 = 6.
i < Rank
4 < 6

K = 40, i = 4
Rank = 1 + 1 = 2
i ≠ rank
K = 45, i = 2
Rank = 1 + 1
= 2
i = Rank
17th smallest element
is 45.

Determining the rank of the element :→
The rank of a particular node in a order statistic tree indicate the position of the node in inorder sequence.
following algorithm determine the position of a particular node -

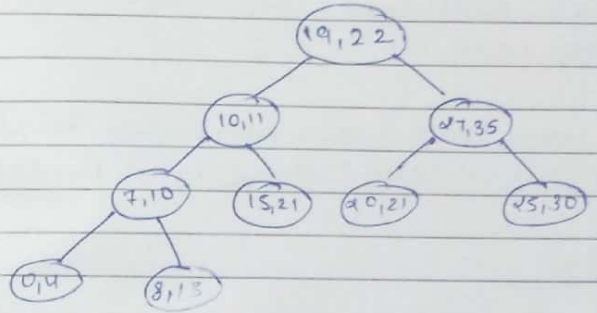
```
Algorithm Compute Rank (root, k)
{
rank ← size [left [k]] + 1
  temp ← k
  while (temp != root)
  {
  if (temp = right [ Parent (temp) ])
    temp ← parent [temp]  → then
    rank ← rank + size [left [parent (temp)]]
  }
  return rank.
}
```

Suppose we want to obtain rank of 50 then,

root 30, k = 50
rank ← 5 + 1 = 6
temp = 50.
while (50 != 30) true
if 50 = parent [30])
if (50 = 50)
rank ← 6 + 12 + 1 = 19
temp = 30

(30 != 30) false.
Rank = 19. //.

Interval Tree :→ Interval tree is a red black tree in which each node has interval represented by an ordered pair $[t_1, t_2]$ here $t_1 < t_2$.
Hence, t_1 is called lower end point and t_2 is called higher end point and if both the end points are mentioned in the nodes of the tree then it is known as close interval. If we omit both the end point then it is called open interval and if we omit one of the end point from the set then it is called half open interval. So we consider an example -



[a,b], [x,Y] will overlap if and only if
b > x & a < Y

Ex → [27, 35], [23, 28]
35 > 23
28 > 27
overlapping.

Now the augmentation of the interval tree data structure is as follows :→

Step 1 - choose the data structure which has to be augmented.

Step 2 - Find out the additional information that is need to support algorithm.

max [K] = max [high (int [K])]

max [Left [K]]
max [Right [K]]

Step 3 - Verification of additional information.

New operations for additional information :→

```
Algorithm search (root, i)
{ K ← root
while ((K != NULL) AND i does not overlap int[K])
if ((left[K] != NULL) AND (max[left[K] >= low[i])) then
{
K ← left[K]
else
K ← right[K]
}
return K
}
```

Suppose we want to find out the interval (23, 28). In a above interval tree.

Sol

Here root (19, 22) and i = 23.

a, b	x, y
19, 22	23, 28
22 > 23	

K = (19, 22)

while K != (19, 22) tree AND i does not overlap int[K]

if ((left[K] != NULL) and (max(left[K] >= low[i]) then

19 >= 23 (false)

K ← left[K] — K = 19.

K ← right[K] K = 27, 35

(ii) while ((K != NULL) AND i does not overlap int[K])

(True) (False)

a, b	x, y
27, 35	23, 28
27 > 23	27 < 28
	Overlapp.

(K = 27, 35) Ans.