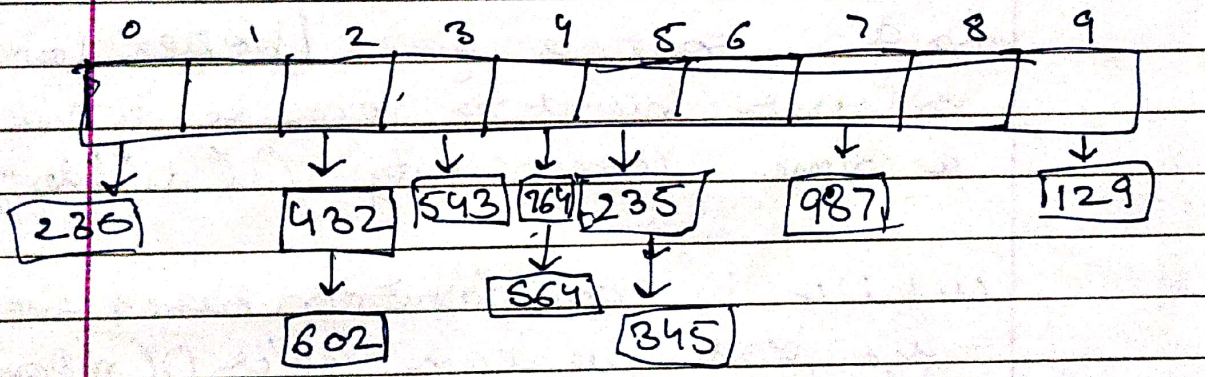
2024602
Vardaan

Date:
Page No.:

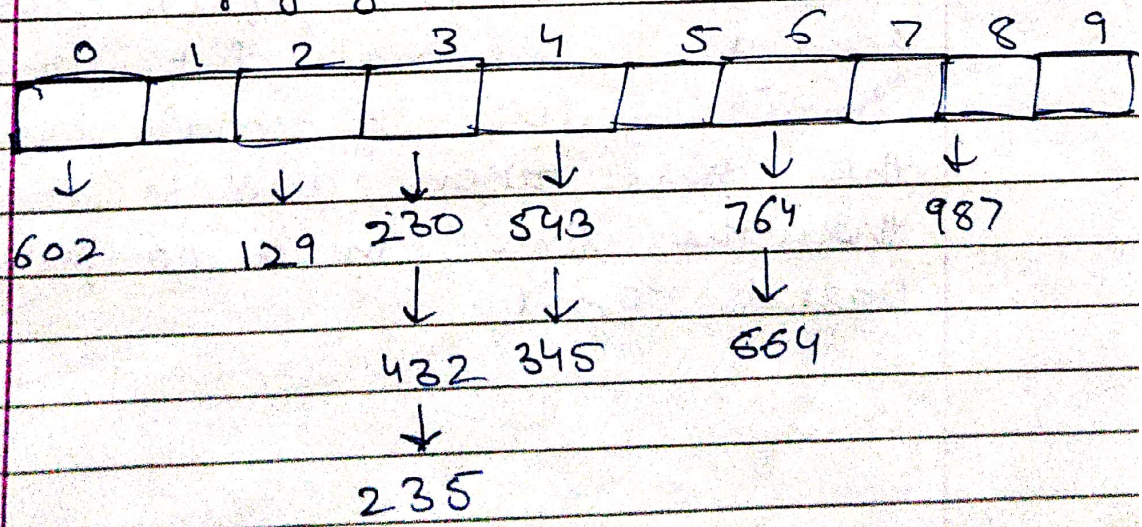
Assignment -2

Q1) 432, 235, 129, 543, 602
764, 230, 987, 564, 345

If we do in C / C++ then
~~sorting~~ arranging by 1's digit

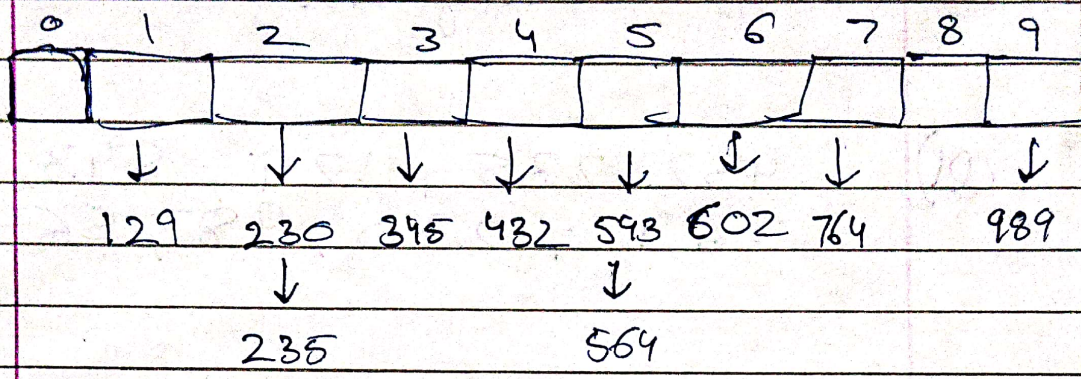


arranging by 10's digit



Date :
Page No. :

arranging by 100's digit.



Radix sort is good/better for smaller numbers because it has a time complexity $O(n * \log_{10} max)$

while in comparison based sort the best we can do is $O(n \log n)$

for smaller numbers radix sort is good as the max int present in it will be small and the time taken will be less than the time in a comparision based sort.

Date :
Page No. :

Q2. Priority_teller(old order, new order){

if(old order → weight > new order){
return -ve }
else if(old order ⇔ weigh <
new order → weight){
return +ve }
else { if (old order → perishibility)
new order → perishibility){
return -ve }
else if (old order → perish
< new order → perishibility){
return true }
else { if (int(old.order)
> int(new order))
{ return -ve }
else {return +ve}}}

old = root, par-old = null
while (old != nullptr) {
if (priority(old, new) == +ve) {
par_old = old
old = old → right }
else { par old = old
old = old → left }
}

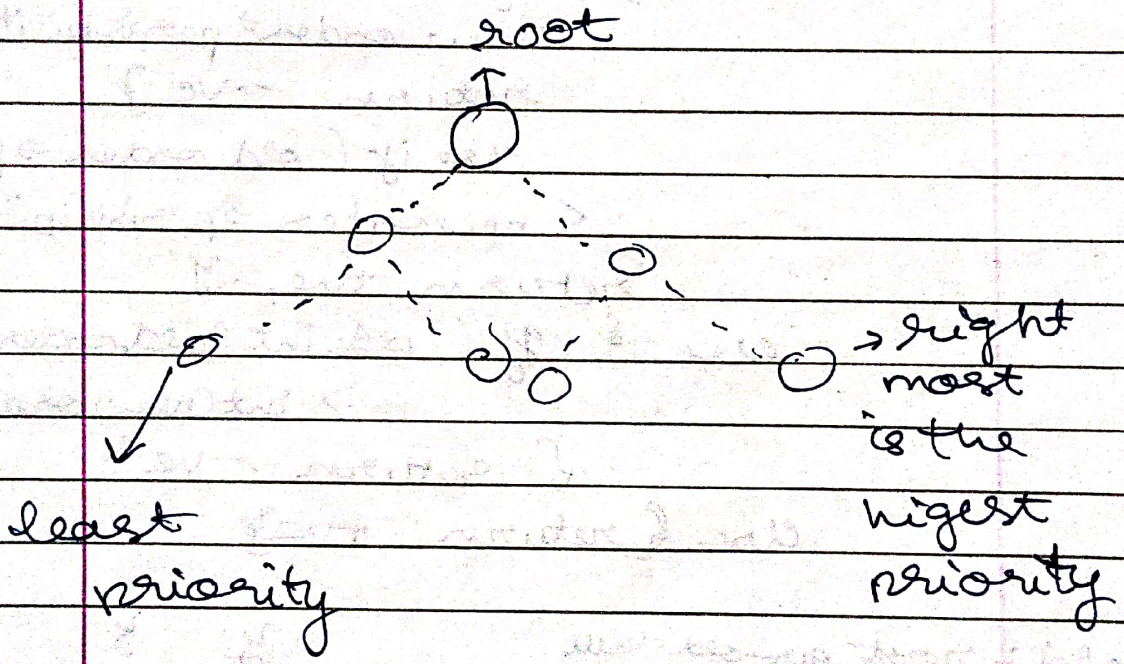
(4)

Date :
Page No. :

if (priority(par_old, new)) == true

{ par_old → right = new }

else { par_old → left = new }



& to get the order

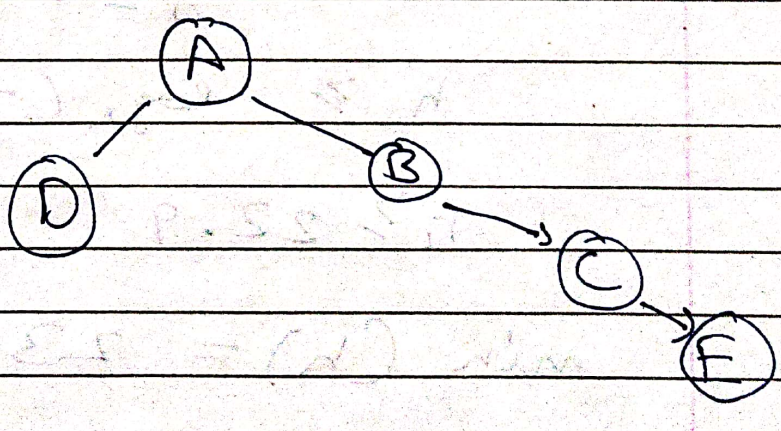
we can do the inorder traversal

(5)

Date :
Page No. :

Example →

	weight	Profitship
A	20	1
B	30	0
C	30	1
D	20	0
E	40	1



6.

Date :
Page No. :

12. The number of leaves that the tree can have is man of 10! which is all different ways an order can be processed.

$$10! \approx \sqrt{2\pi \cdot 10}\left(\frac{10}{e}\right)^{10}$$

$$= \sqrt{62} \cdot (3.68 * 10^4)$$

$$= 7.9 \times 10^6$$

minimum Height

$$h \geq \log_2(n!)$$

$$h \geq \log_2(7.9 \times 10^6)$$

$$h \geq 22.9$$

$$\min(h) = 23$$

∴ min height of tree = 23

(7)

Date / /
Page No.

Q3. node → stores the data of the parcel and the next ptr*

node_hash → stores the id & adress of the node.

hash table

~~Search~~

hashfunction()

insert()

delete_id()

exists()

Stack

add_parcel()

ht.insert(id, addr)

dispatch_parcel()

ht.delete_id(id)

Date / /
Page No.

View next ()

display all ()

count by priority ()

search ()

ht. exists ()

The serching is optimised as we use a hash table and it's node stores the id and address of the node which has the data about parcel.

other thing are done on stack.