Sorting

1. Selection Sort

($i = 0$ to $n-2$)

- outer loop → for different rounds
- select min^m element in array
- inner loop → for min^m element ($j = i$ to $n-1$)
- swap min^m w/ first element
- repeat for further rounds.

* no. of rounds = size − 1
* Time comp. = $O(n^2)$ → best, worst, average.

| 7 | 5 | 9 | 2 | 8 |

min = i

min ↑ i j

round 1:

| 7 | 5 | 9 | 2 | 8 |

i, min, j

min = j

| 2 | 5 | 9 | 7 | 8 |

swap (i, min)

round 2:

| 2 | 5 | 9 | 7 | 8 |

i j, min

* i^{th} round → i^{th} smallest no. at right place.

round 3:

| 2 | 5 | 9 | 7 | 8 | ⇒ | 2 | 5 | 7 | 9 | 8 |

i j, min

round 4:

| 2 | 5 | 7 | 9 | 8 | ⇒ | 2 | 5 | 7 | 8 | 9 |

i j, min

2. Bubble Sort

- outer loop → for diff. rounds ($i = 0$ to $n-2$)
- swap if next element is smaller than the current element → largest element at last. • inner loop ($j = 0$ to $n-1-i$)
  ↳ if not move ahead w/o swapping.

* (i^{th}) round → i^{th} largest element at right place.
* no. of rounds → $n-1$
  * T.C. = $O(n^2)$ * adjacent swapping.
  ↳ worst, avg.

| 13 | 46 | 24 | 52 | 20 |

round 1:

| 13 | 46 | 24 | 52 | 20 |

i, j

if (arr[j] > arr[j+1]) swap them;

| 13 | 24 | 46 | 52 | 20 |

i j

| 13 | 24 | 46 | 20 | 52 |

i=0 j

round 2:

| 13 | 24 | 46 | 20 | 52 |

j=0 i=1 j

| 13 | 24 | 20 | 46 | 52 |

i j

round 3:

| 13 | 24 | 20 | 46 | 52 |

j=0 j j=2

| 13 | 20 | 24 | 46 | 52 |

i j

round 4:

| 13 | 20 | 24 | 46 | 52 |

j i

→ * No swapping

$O(n)$ → best

∴ for optimisation

introduce a 'flag'

if no swapping → flag = false.

& if flag false → break through outer loop.

⇒ no further rounds.

3. Insertion Sort

- ~~select element from the array & place it at its correct position → outer loop i = 0 to n−1~~

3. Insertion sort

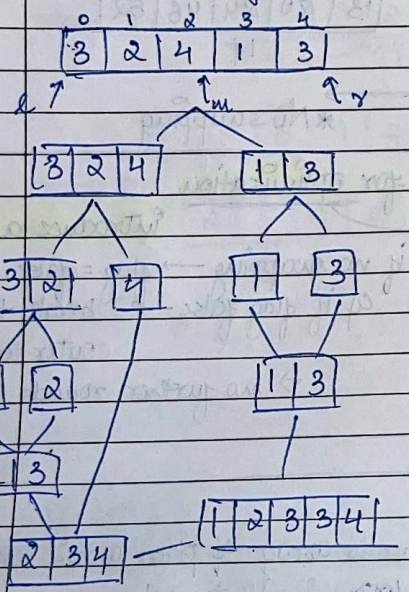
- select an element & place it at it's right position by keeping all the larger elements to right side of selected element

outer loop i=0 to n-1
j = i

inner while loop (j>0 & arr[j-1] > arr[j])
swap(arr[j], arr[j-1]); j--;

4. Merge sort

↳ O(NlogN)




```
merge (arr, l, m, r) {  temp = []
    i = low, j = mid+1
    while (i<=m && j<=r){
        if (arr[i] <= arr[j]){
            temp.push_back(arr[i])
            i++; }
        else {
            temp.push_back(arr[j])
            j++; }}

    while (i<=m) { temp.add(arr[i++]);
    while (j<=r){ temp.push_back(arr[j++]);
    while (i=l -> r)
        arr[i] = temp[i-l];
```

```
mergesort (arr, l, r) { if (l>=r) return;
    int mid = (l+r)/2
    mergesort (arr, l, mid);
    mergesort (arr, mid+1, r);
    merge (arr, l, mid, r); }
```

dividing into 2 arrays.

5. Quick sort

0	1	2	3	4	5	6	7
4	6	2	5	7	9	1	3

i, low i, pivot, high

- pick up the pivot & place it at it's correct place
- smaller on left & larger on right.

```
quicksort (arr, low, high)
{ p = partition (arr, low, high)
  quicksort (arr, low, p-1)
  quicksort (arr, p+1, high)
}
```

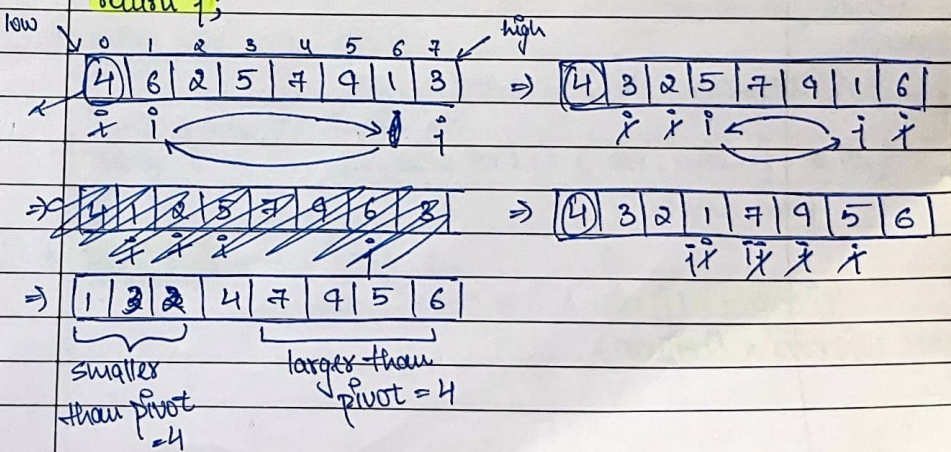
```
partition (arr, low, high) {
    pivot = arr[high]
    i = low - 1
    loop (j = low to high-1) {
        if (arr[j] <= pivot)
            i++; swap (j, i) }
    swap (i+1, high)
    return i+1; }
```

```
partition (arr, low, high)
    pivot = arr[low]
    i = low, j = high
    while (i<=j)
        { while (arr[i] <= pivot &
                i<=high-1) i++;
        { while (arr[j] > pivot &
                j>=low+1) j--;
        if (i<j)
            swap (i, j)
    swap (arr[low], arr[j])
    return j;
```

find first ele. larger than pivot

find first ele < pivot

swap the pointers



Arrays

⇒ **Easy problems**

1. **Second smallest / largest** (w/o sorting)

↳ small

- 2 variables → second-small & small → INT_MAX
- loop (i = 0 to n-1) → if (arr[i] < small)
  sec-small = small
  small = arr[i]

else if (arr[i] < sec-small & arr[i] != small)
  sec-small = arr[i]

1	2	3	4	5

s = ∞

i

1 < ∞ ⇒ ss = ∞ , s = 1

2 ≮ 1 ⇒ but 2 < ∞ ⇒ ss = 2, s = 1

3 ≮ 1 ⇒ & 3 ≮ 2 ⇒ ss = 2 , s = 1

4 ≮ 1 & 4 ≮ 2 ⇒ ss = 2, s = 1

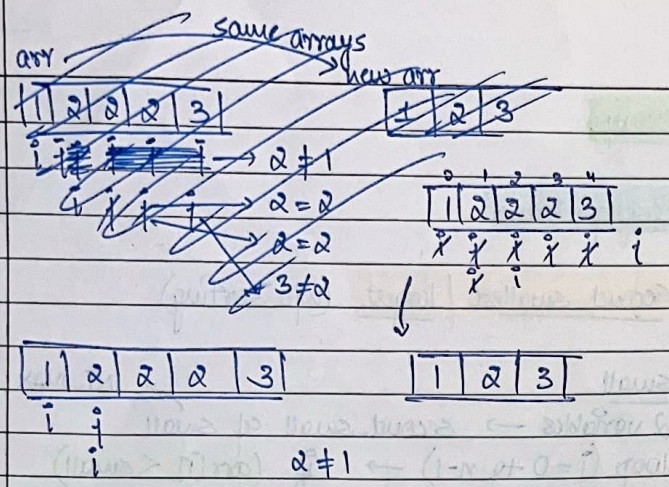
5 ≮ 1 & 5 ≮ 2 ⇒ ss = 2, s = 1

2. **Remove Duplicates**

* Using sets → set s; stores only unique elements
loop (i = 0 to n-1) { s.insert(arr[i]); }

index of new array ← index = 0
for (auto itr : s) { arr[index++] = itr }
↳ new array

* Using logic → i = 0
loop (j = i+1 to n-1) { if (arr[j] != arr[i])
{ arr[i+1] = arr[j]; i++; }



3. **Rotate array by 'k' places** (left)

$k \% n$ — size of array.

- outer loop (i=0 to k-1) → for 'k' rotations
- store first element in (temp) temp = arr[0]
- inner loop (j=0 to n-2)

  arr[j] = arr[j+1]

  4) out of this inner loop

  arr[n-1] = temp.

4. **Move zeroes to end**

1 2 0 0 2 3

i → counter for non zero

j → counter for zero

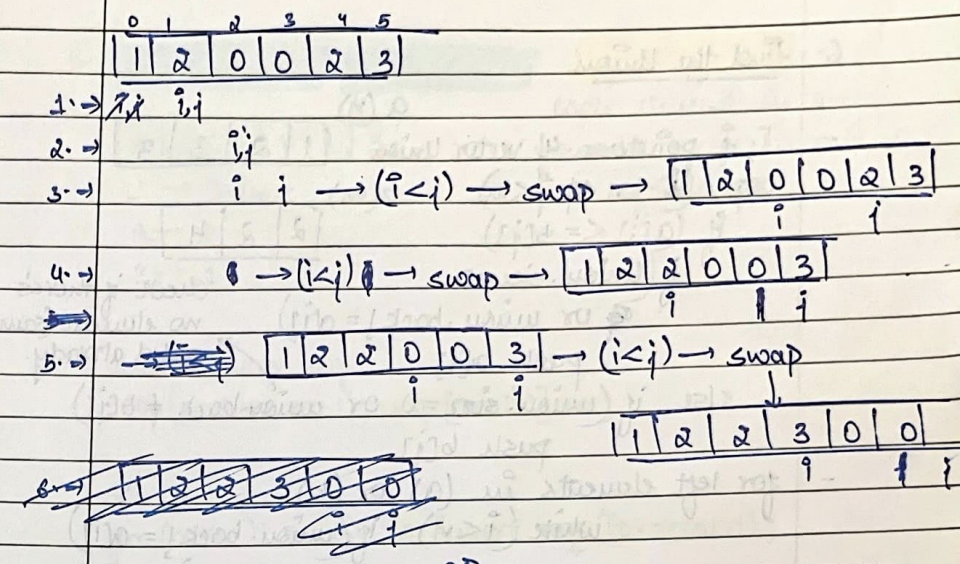
loop (until i & j < size) i++ if non zero

j++ if zero

if non zero ⇒ i > j → do nothing

but if i < j → swap (a[i], a[j]).

j++; → for next iteration.



OR

```
j = -1
loop (i=0 to n-1) { if (a[i] == 0)  j = i; break; }
loop (i = j+1 to n-1) { if (a[i] != 0)  swap; j++; }
```

** 5. **Max^m Consecutive 1's / Road Department.** (SDE)

size — max zeroes that can be changed to 1's.

N = 6, M = 3

0 1 0 0 1 0

```
- while loop (zerocut < M) {
  i=0  if (a[i] == 0) { a[i] = 1
                        zero cut ++ }
       i++ }
- for loop (i=0 to N-1) { if (a[i] == 1) { maxx = i+1 } }
```

```
- cut, maxx = 0
- loop (i=0 to N-1)  if (a[i] == 1) { cut ++ }
                     else  maxx = max(cut, maxx)
                           cut = 0
```

6. Find the Union

a(n)

1	2	3	3

b(m)

2	2	4

- i, j pointers of vector Union
- loop (i < n of j < m)
  - if (a[i] <= b[j])
    - if (Union.size = 0 or union.back != a[i])
      - push a[i]
    - else if (union.size = 0 or union.back ≠ b[j])
      - push b[j]
- for left elements in (a) or (b)
  - while (i < n) if (union.back != a[i]) push a[i]
  - while (j < m) if (union.back != b[j]) push b[j]

check if there's no element (same) added already.

SDE

7. Longest Subarray with sum k (+ve of -ve)

* Using Hashing

```
map<int, int> preSum;
loop(i=0 to n-1)
    sum += a[i];  if (sum = k) return (i+1, len)
                  maxlen = max(i+1, len)

rem = sum - k
if (preSum.find(rem) != preSum.end())
        len = i - preSum[rem]
        maxlen = max(maxlen, len)

if (preSum.find(sum) == preSum.end())
        preSum[sum] = i
```

8. Unique Number of Occurrences

1	2	2	1	1	3

1 → 3 times
2 → 2
3 → 1

each element in array should occur in unique no. of times.

```
unorder_map mp
loop(i=0 to n-1)  mp[arr[i]]++;

set s
for (auto it : mp)  s.insert (it.second);

return mp.size == s.size;
```

9. Find all Duplicates in Array

* using hashmap

```
vector ans;
map mp;
loop(i=0 to n-1)  mp[nums[i]]++;
loop (auto it : mp)  if (it.sec == 2)  ans.push(it.first);
```

*

```
vector ans
loop(i=0 to n-1)
x = abs(nums[i])
if (nums[x-1] < 0)  ans.push(x)
nums[x-1] *= -1
```

Important

Logic !!!

0	1	2	3	4	5	6	7
-4	+3	+2	-8	2	3	-1	-7

x = ~~4~~ ~~3~~ ~~2~~ ~~8~~ ~~2~~ ~~3~~ ~~1~~ 7

ans = 2, 3

⇒ Medium Problems

1. 2Sum Problem (SDE)

target = 5

arr

4	1	2	3	1

1	1	2	3	4

- sort the given array
- left = 0, right = size − 1
- loop(left < right) { if sum = target ⇒ return indices
  else if sum < target ⇒ left++
  else right-- }

2 pointer approach

OR

using hashing

- unordered map → T.C. = $O(N^* \log N)$
- loop(i = 0 to n−1) { num = arr[i]
  need = target − num
- if (mp.find(need) != mp.end) { return indices }
  {mp[need], i}
- mp[num] = i

2. Sort array of 0's, 1's & 2's (SDE)

0 to i → 0's
i+1 to k → 1's
k+1 to j → 2's

- i = 0, k = 0, j = n−1
- loop(k <= j) { if arr[k] = 0
  swap(k, i)
  k++, i++
  else if arr[k] = 1 k++
  else if arr[k] = 2 swap(k, j) j--

① 2 0 2 1 1 0 (i,k ... j)

② 0 0 2 1 1 2 (i,k i,k i,k ... j)

③ 0 0 1 1 2 2 (i,k i,k)

3. Rotate Matrix by 90 degrees (SDE)

- Transpose
  loop (i = 0 to n−2)
  loop (j = i+1 to n−1)
  swap ([i][j], [j][i])
- Reverse elements
  loop (i = 0 to n−1)
  reverse (mat[i].begin(), mat[i].end())

1	2
3	4

→

3	1
4	2

transpose ↓ = reverse →

1	3
2	4

→

3	1
4	2

4. Spiral Matrix

top = 0, right = m−1, (m−1), (n−1), bottom = n−1

1	3	5	7
10	11	16	17
23	30	34	60

o/p : 1, 3, 5, 7, 17, 60, 34, 30, 23, 10, 11, 16

loop (top <= bot & left <= right) left = 0

① loop (i = left to right)
  print [top] [i]
  top++

② loop (i = top to bot)
  print [i] [right]
  right--

③ loop (i = right to left)
  print [bot] [i]
  bottom--

④ loop bot to top)
  print [i] [left]
  left++

5. Next Permutation (SDE)

- find break point in array arr[i] < arr[i+1]
  ↳ break pt.
- find which is > break pt. but the smallest among them. & swap with the break pt.
- place the remaining in sorted manner

2	1	5	4	3	0	0

↓ i

~~(i) break pt.~~ ⇒

2	3	5	4	1	0	0

~~(ii) loop (i = int0 N-1)~~
~~if (arr[i] > arr[i]) {~~
~~num = arr[i]~~
~~mnu = min(num, mnu); }~~
~~swap (mnu, arr[i])~~

~~(iii) loop (i = i+1 to n-1)~~
~~sort arr~~

⇒

2	3	0	0	1	4	5

(i) break pt → loop (i = n-2 to 0)
if (arr[i] < arr[i+1]) ind = i;
break;

(ii) loop (i = ~~ind~~ n-1 to ind+1)
if (arr[i] > arr[ind]) swap(i, ind); break;

(iii) reverse (a.begin() + ind+1, a.end());

6. Set matrix zero (SDE)

(i) brute force T.C. = $O(n^3)$

loop (i = 0 to n-1)
loop (j = 0 to m-1)
if (arr[i][j] = 0)
markRow (i)
markCol (j)

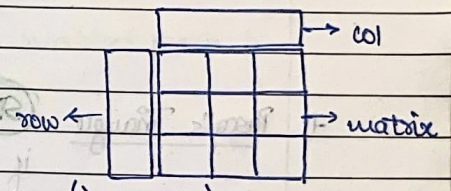
markRow (i) { loop (j = 0 to m-1)
if (arr[i][j] != 0)
arr[i][j] = -1

markCol (j) { loop (i = 0 to n-1)
if (arr[i][j] != 0)
arr[i][j] = -1

- now, in whole matrix, change all -1 to 0.

(ii) better

row[n] = {0}
col[m] = {0}

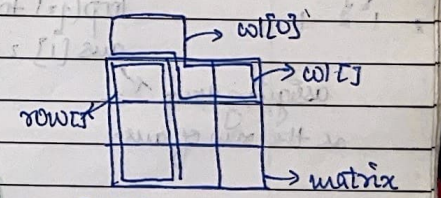


loop (i = 0 to n-1)
loop (j = 0 to m-1)
if (a[i][j] = 0)
row[i] = 1
col[j] = 1

loop (i = 0 to n-1)
loop (j = 0 to m-1)
if (row[i] || col[j])
a[i][j] = 0

(iii) optimal - int col0 = 1

- mark row[] & col[]



loop (i = 0 to n-1)
loop (j = 0 to m-1)
if (arr[i][j] = 0) → arr[i][0] = 0
if (j != 0) arr[0][j] = 0
else col0 = 0

- mark matrix from (1,1) to (n-1, m-1)

loop (i = 1 to n-1) loop (j = 1 to m-1) if (arr[i][0] OR arr[0][j] = 0)
arr[i][j] = 0

- work 1st col

  if (arr[0][0] = 0) { loop (j=0 to m-1)
  arr[0][j] = 0

- work 1st row

  if col0 = 0 { loop(i=0 to n-1)
  arr[i][0] = 0

7. Longest consecutive seq. in array

8. Count subarrays with sum

of Hard Problems.

7. Pascal's Triangle (SDE)

if uRow = 1 → return {1}
if uRow = 2 → return {{1}, {1,1}}

vector <vector <int>> ans (uRow); ans[0] = {1}
ans[1] = {1,1}

	i 0	1	2
0	1		
1	1	1	
2	1	2	1

loop(i=2; i< uRow; i++) (size of row)
vector <int> temp (i+1); temp[0] = 1; temp[i] = 1; (first, last of temp)
loop(j=1 to i-1) { temp[j] = ans[i-1][j] + ans[i-1][j-1] }
ans[i] = temp;

assigning temp as the row of ans.

8. Stock Buy & Sell (SDE)

n=6 => 7 1 5 3 6 4

sell on i^{th} day
buy on min price from 1st to $(i-1)^{th}$ day

int profit = 0, int mini = price[0]
loop(i=1 to n-1) { int cost = price[i] - mini
profit = max(profit, cost)
mini = min(mini, ~~profit~~ price[i]) }

return profit

mini → buying value
profit + mini → selling value.

Binary Search

⇒ BS on 1D Arrays

1. Peak Element

which is greater than its neigbours.

s = 0 ; e = n-1
if (n=1) return 0;
- for first element > second → return a[0]
- for last element > second last → return a[n-1]

loop (s<=e) mid = (s+e)/2
if a[mid] > both its neigbours → return mid
if a[mid] > left neigbour → s = mid+1
else → e = mid -1

2. Floor of Ceil Value

- floor / upper bound → loop (s<=e) mid = (s+e)/2
  if (a[mid] <= x) → floor = mid
  s = mid+1
  else → e = mid -1
- ceil / lower bound → loop (s<=e) mid = (s+e)/2
  if (a[mid] >= x) → ceil = mid
  e = mid-1
  else → s = mid+1.

SDE

3. Search in Rotated Sorted Array (unique element)

- binary search

  loop (low <= high) mid = (l+h)/2

  if (mid = target) return mid

- if left part is sorted

  ↳ if (arr[low] <= arr[mid])

  check if element is present

  if (arr[low] <= target && arr[mid] >= target)

  high = mid-1

  else low = mid+1

- if right part is sorted

  ↳ if (arr[~~high~~ low] >= arr[mid])

  if (arr[high] >= target && arr[mid] <= target)

  low = mid+1

  else high = mid-1

0	1	2	3	4	5	6
4	5	6	7	0	1	2
l		m				h

target = 0

$7 \neq 0$

$4 < 7 \rightarrow$ left part sorted

$4 \not< 0 \rightarrow$ low = mid+1 = 4

$\Rightarrow$

				4	5	6
4	5	6	7	0	1	2
l		m				h

$1 \neq 0$

$0 < 1 \rightarrow$ left part sorted

$0 <= 0$ && $1 > 0 \rightarrow$ high = mid-1 = 4

$\rightarrow$

				4		
4	5	6	7	0	1	2
				l, h, m		

$\boxed{0 = 0}$

* if duplicate elements

- add edge case

  if nums[mid] = [high] = [low] high--; low++; continue;

4. Number of Rotations

??

5. Pivot Index (rightsum = leftsum)

- rsum = lsum = 0
- store sum of every element of array in rsum
- loop (i=0 to n-1) rsum -= arr[i]

  check if rsum = lsum → return i

  if not lsum += arr[i]

return -1.

⇒ BS on 2D Arrays

1. Row with max Ones

0	1	1	1
0	0	1	1
1	1	1	1
0	0	0	0

↳ each row is sorted.

- function for first occuring 1

```
low = 0, high = n-1,
first = n
while (low <= high) mid;
    if [mid] = 1 → first = mid; high = mid-1
    else → low = mid+1
return first
```

- function for row with max 1's

```
maxx = INT_MIN
row = -1
loop (i = 0 to n-1)
    cut = m - first1 (arr, m)
    if (cut > maxx) { maxx = cut
                      row = i }

if (maxx = 0) return -1
return row
```

2. Search in 2D array

SDE

```
n = no. of row , m = no. of cols
row = 0, col = m-1
loop (row < n & col >= 0)
    if (arr [row][col] = target) return {row, col}
    else if (arr [row][col] < target)  row++;
    else col--;
return false
```

or (if last element of row is smaller than first element of next row)

```
n = (rows), m = (cols)
low = 0, high = n*m - 1
while (low <= high) mid; row = mid/m ; col = mid % m
if [row][col] == target → return true
    "  "  <  "  → low = mid+1
    else → high = mid - 1
return false.
```

3. Peak Element in 2D Array

- function to find maxElement in (mid) col. of ~~given~~ → max Row.

```
maxRow = -1 , maxx = 0
loop (i = 0 to n-1) if (arr [i] [mid] > maxx) maxRow = i
                                              maxx = arr[i][mid]
return maxRow.
```

- function to find peak Element → (greater than up, down, left, right element)

```
n = (rows) , m = (col)
low = 0, high = m-1
loop (low <= high) mid; maxRow;
    left = if (mid-1 >= 0) → arr[maxRow][mid-1]
           else -1
    right = if (mid+1 < m) → arr [maxRow][mid+1]
           else -1
if ([maxRow][mid] > left & right) return indices;
else if ( "  "  < left) → high = mid - 1
else → low = mid+1
```

4. Matrix Median

??

* Same logic, Same Code

Pointer's partition problems ↑
Split array largest sum

⟹ BS on Answers

1. Book Allocation Problem

if m > n → return -1

- contigous distribution.
- max^m pages allocated to a stu. is min^m.

arr (n) n = 4, m = 2

0	1	2	3
10	20	30	40

(i) 10, 20+30+40 = 10, 90 = 90
(ii) 10+20, 30+40 = 30, 70 = 70
(iii) 10+20+30, 40 = 60, 40 = 60 ans.

low = max (arr)
high = sum (arr)
loop (low <= high)
mid = (low + high) / 2.

low 40 —— 70 (mid) —— high 100

* countStudent (arr, mid)
sum = 0
stu = 1
loop (i = 0 to arr.size() - 1)
if (sum + arr[i] <= mid)
sum += arr[i]
else
stu ++
sum = arr[i].
return stu.

stu. 1 ⇒ 10 + 20 + 30 <= 70
stu 2 ⇒ 40 <= 70
↳ 2 students, possible sol^n.
but we want to find min^m. ⇒ high = mid -

low 40 —— mid 54 —— high 69

stu 1 ⇒ 10 + 20 <= 54
stu 2 ⇒ 30 <= 54
stu 3 ⇒ 40 <= 54
↳ 3 stu. > 2 No sol
∴ low = mid + 1

if (cntStud. <= m)
high = mid - 1
else
low = mid + 1

Binary search

return low.

Linear search
loop (i = low to high)
if (cntStud. == m)
return i;

2. K^{th} Missing Positive Number

arr	0	1	2	3	4	k = 5
	2	3	4	7	11	

(I)

```
loop (i=0 to n-1)
    if (arr[i] <= k) k++
        else break
return k
```

(II)

```
low = 0, high = n-1
loop (l<=h)
    mid
    missing = arr[mid] - (mid+1)
    if(missing < k)
        l = mid+1
    else h = mid - 1
return (low + k) OR (high+1+k)
```

1 2 3 4 5 6 7 8 9 10 11

(1) (2) (3) (4) (5) (6)

	0	1	2	3	4
	2	3	4	7	11
	~~l~~		~~m~~	~~l~~ h	~~h~~ l m
mid+1 ⇒	1	2	3	4	5

missing = arr[mid] - (mid+1)
= 4 - 3 = ~~1~~ ~~3~~ 6

tells that 'mid' index

tkk kitne missing nos. hain.

3. N^{th} Root of Number

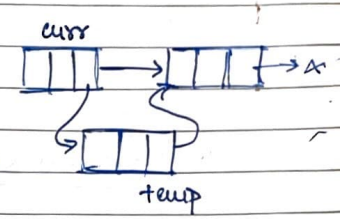
Linked List

→ Singly LL (Easy)

```
Node(int d) { this → data = d;
              this → next = NULL; }
~Node() { int val = this → data
          if (this → next ≠ NULL) { delete next;
                                    next = NULL; } }
```

1. Insertion of Node

* at Head → temp = new Node
temp → next = head
head = temp

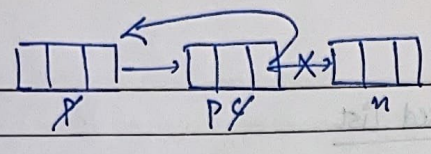


* at Tail → temp = new Node
~~temp~~ tail → next = temp
tail = temp

* at Position → if (p = 1) → insert at Head
temp = new Node
curr → jiske aage insert krna hai
if (curr → next = NULL) → insert at Tail
temp → next = curr → next
curr → next = temp

2. Deletion of Node → if only node to be deleted is given
→ node → val = node → next → val
node → next = node → next → next

curr = head, prev = NULL
if (p = 1) temp = head; head = head → next; temp → next = NULL
delete temp;
curr → jisko delete krna hai prev → usse pehle vali node.
if (curr → next = NULL) temp = tail; tail = prev
else [prev → next = ~~curr~~ temp → next
curr → next = NULL; delete temp



3. Reverse LL

SDE

curr = head ; prev = NULL ; next = NULL ;
loop (curr != NULL) next = curr → next
curr → next = prev.
prev = curr
curr = next

return prev.

⇒ Doubly LL (Easy) Node (int d) { this → data = d ;
this → prev, next = NULL ; }
~Node () { int val = this → data
if (this → next != NULL) {
delete next ; next = NULL ; }
}

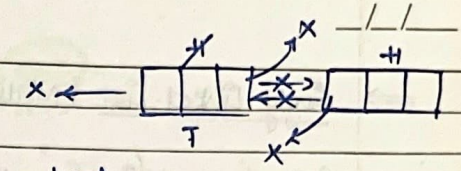
1. Insertion of Node

* at Head : temp = new Node
temp → next = head
head → prev = temp
head = temp

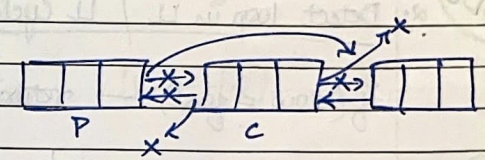
* at Tail : temp = new Node
curr = jiske aage insert krna hai / tail node
curr → next = temp
temp → prev = curr

* at Position : if (p = 1) → insertatHead
temp = new Node
curr = jiske aage insert krna hai
if (curr → next = NULL) → insert at Tail
temp → next = curr → next
curr → next → prev = temp
curr → next = temp
temp → prev = curr.

2. Delete Node

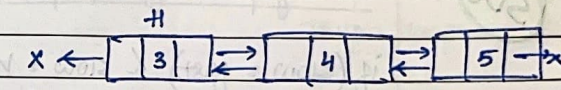


- if (pos = 1) → temp = head
temp → next → prev = NULL
head = temp → next
temp → next = NULL
- else curr = head → jisko delete krna
prev.
curr → prev = NULL
prev → next = curr → next
curr → next = NULL



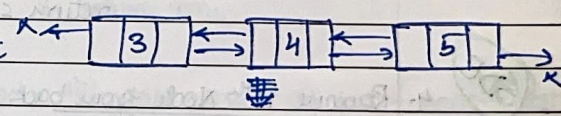
3. Reverse a DLL

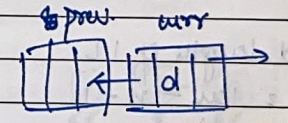
SDE



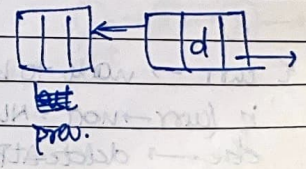
while (curr != NULL)
{
prev = curr → prev.
curr → prev =
curr → next
curr → next = prev
curr = curr → prev.
}

curr = head
prev = NULL





• store prev. node of curr



⟹ Singly Linked List (Medium)

SDE

1. Middle of LL (Tortoise - Hare Method)

- slow, fast = head
- 'slow' moves one step head & 'fast' moves 2 steps head at one time.
- loop (fast ≠ NULL && fast → next ≠ NULL) {
  slow = slow → next
  fast = fast → next → next } return slow;

SDE

2. Detect loop in LL / LL Cycle

if (slow = fast) → return true

SDE

3. Starting point of loop / LL cycle 2

if (slow = fast) & { slow = head
loop (slow != fast) {
slow = slow → next
fast = fast → next }
return slow }

SDE

4. Remove Nth Node from back

- find length of LL
- pos = len − n + 1
- if (pos = 1) → deleteAtHead; temp = head
  head = head → next
  temp → next = NULL.
- else { curr → node to be deleted ; prev
  if (curr → next = NULL) → deleteAtTail
  else → deleteAtPosition.

5. Intersection of 2 LL

SDE

- calc. length of each LL (len1 & len2)
- temp1 = head1 , temp2 = head2 , cut = 0
- if (len1 >= len2) cut = len1 − len2
  - now move temp1 until cut > 0

  else cut = len2 − len1
  - now move temp2 until cut > 0

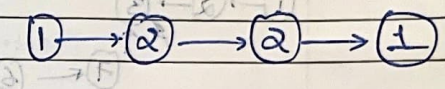
* Now temp1 & temp2 are on same line

- now move temp1 & temp2 until they are not equal.

SDE

6. Palindrome Linked List

(1) - push every element into stack

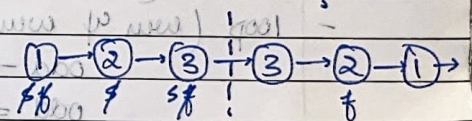


temp = head ; while(temp != NULL) st.push(temp → val)
temp = temp → next

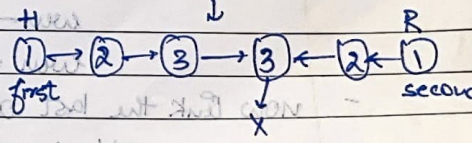
- temp = head
- now check temp → val is equal to st.top or not
  while (temp != NULL) { if (temp → val != st.top) → false
  else st.pop ; temp = temp → next
  } return true

(2)

- tortoise - hare method till fast → next & fast → next → next != NULL to find middle node
- now reverse the links of nodes after 'slow'

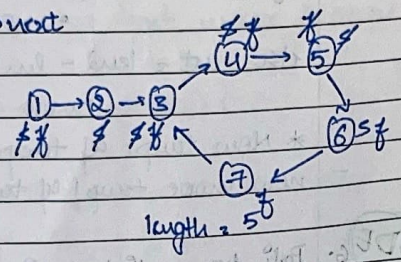


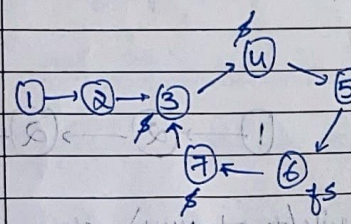
- now compare first & second & traverse till second != NULL & reverse the links again.



7. Length of Loop

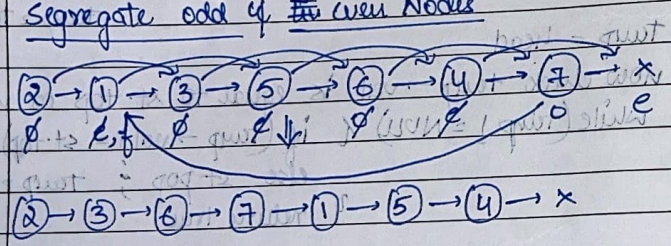
- tortoise - hare method till fast & fast → next != NULL
  if (slow = fast)
  int len = 1
  slow = slow → next
- move slow to the next till it's not equal to fast & increase the len





len = ~~1~~ ~~2~~ ~~3~~ ~~4~~ $\boxed{5}$

8. Segregate odd & even Nodes



2 → 3 → 6 → 7 → 1 → 5 → 4 → x

- odd → at first node (head); even & first even → (head → next)
- loop (even & even → next != NULL) {
  odd → next = even → next
  odd = even → next
  even → next = odd → next
  even = odd → next }
- now link the last odd node with first even node
  odd → next = firstEve

9. Add 2 Numbers as LL (SDE)

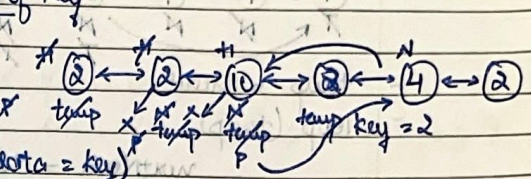
l1 = 2 → 4 → 3 → x ⇒ 342
l2 = 5 → 6 → 4 → x ⇒ 465
ans = 7 → 0 → 8 → x 807

- ans = new Node ; temp = ans ; carry = 0
- loop (l1 & l2 & carry)
  sum = 0
  store l1 → data & l2 → data in sum & move them to next node
- sum += carry
  carry = sum/10
- node = new Node (sum%10)
  temp → next = node
  temp = node.

10 Sort LL
11 Sort LL of 0, 1, 2
12 Add 1 to number by LL

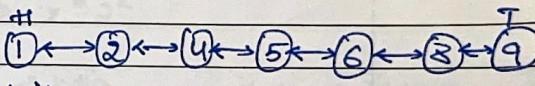
⇒ Doubly Linked List (Medium)

1. Delete all occurences of key



temp = head
loop (temp)
- if (temp → data = key)
  - if temp is at head
    - then just move the head
    - nextNode = temp → next
    - prevNode = temp → prev.
  - if (nextNode) → nextNode → prev = prevNode
  - if (prevNode) → prevNode → next = nextNode
  - temp = temp → next
- else temp = temp → next

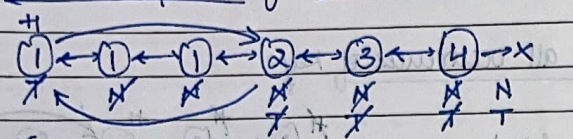
2. Find pair with given sum in DLL



- generate tail
- start = head , end = tail

while (start → data < end → data) target = 7
  sum = start + end (∵ sorted DLL)
  if (sum = target)
    push (start, end ka data)
    & move start & end
  else if (sum < target) move start
  else move end

3. Remove duplicate from sorted DLL



- temp = head
- loop (temp)
  nextNode = temp → next
- now the nextNode till its not NULL
  if till temp if nextNode ka data are equal
- now temp → next = nextNode
  if (nextNode) nextNode → prev = temp
  temp = temp → next

Bit Manipulation

* Convert 2 Binary (int n) {
  string res = "";
  loop (n != 1) { if (n = odd) res += "1";
  else res += "0"; n = n/2; }
  reverse (res); }

* Convert 2 Decimal (string x) {
  len = x.length; num = 0; p2 = 1
  for (i = len-1 to 0) {
  if (x[i] = 1) num += p2;
  p2 *= 2 }
  return num; }

* Right Shift $(x >> k) \Rightarrow \frac{x}{2^k}$
  Left Shift $(x << k) \Rightarrow x * 2^k$

* AND, OR, XOR, NOT → 1. flip
  2. check -ve → if yes → 2's comp
  → if no → stop
  odd no. of 1's = 1
  even no. of 1's = 0

1. Swap 2 Numbers

- using temp → temp = a; a = b; b = temp;
- using maths → a = a+b; b = a-b; a = a-b
- using XOR → a = a^b;
  b = a^b; = (a^b)^b = a
  a = a^b; = (a^b)^a = b

2. Check if i^{th} bit set or not

$N = 13 \quad i = 2$

$(1\boxed{1}01)_2$ → set

- using left shift

$1 << (i) \Rightarrow 100$

$\boxed{N \& (1 << i)} \Rightarrow$ 1101
0100
0100 → if this is $\neq 0$ then set else not

- using right shift

$(N >> i) \Rightarrow 1101 >> 2 \Rightarrow 0001$

$\boxed{1 \& (N >> i)} \Rightarrow$ 0001
0001
0001 → if this $= 0$ then not set else set

3. Set the ith Bit

$N = 9 \quad i = 2$

$\boxed{N \mid (1 << i)}$

1001
1
1101 = 13

4. Clear the ith bit

$N = 13 \quad i = 2$

$\boxed{N \& (\sim(1 << i))}$

1101
0
1001

5. Toggle the ith bit

$N = 13 \quad i = 2$

$\boxed{N \wedge (1 << i)}$

1101
XOR 0100 $(1 << i)$
1001

6. Remove last set bit (rightmost)

$\boxed{N \& (N-1)}$

7. Check if number is power of 2

if $(N \& (N-1) == 0) \rightarrow$ yes
else → no

8. Count number of set bits

- while $(i > 1)$ {
  ~~cnt = 0~~ → if (i) is odd
  cnt += $i \& 1$
  $i = i >> 1$ } → $i = i/2$
  if $(i = 1)$ cnt ++

- while $(N != 0)$ {
  $N = N \& (N-1)$
  cnt ++ }

cnt = 0

C++ STL
or __builtin_popcount(n)

$N = 13 \quad N-1 = 12$
1101
& 1100
1100

9. Get the ith Bit

$\boxed{(N >> i) \& 1}$

10. Divide 2 integers w/o using *, /, %

* Check sign

bool sign = 1

if (dividend or divisor has opp. sign)

sign = 0

* num = abs(dividend) = 20 → 3×7

den = abs(divisor) = 3

$3\times(2^2+2^1+2^0)$

$3\times2^2+3\times2^1+3\times2^0$

ans = 0

while (num >= den)

try removing bigger no. from num.

cut = 0

while (num >= (den << (cut+1))) {

$2^{1,2}$

cut++ }

num -= den << cut

ans += 1 << cut

}

⇒ Interview Problems

1. Min^m Bit flips to convert Number

start = 10 , goal = 7

1010 0111 ⟶ XOR ⇒ 1010

XOR 0111

1101 = ans

```
cnt = 0
while (ans) {
ans = ans ^ (ans - 1)
cnt++; }
```

no. of set bits will tell the no. of bits flipped.

2. Power Set (print all subsets)

$\boxed{\text{no. of subset (if n elements)} = 2^n}$ → $(1 << n)$

nums = [1 2 3] (0 1 2)

subsets = [[], [1], [2], [3], [1,2], [2,3], [1,3], [1,2,3]]

```
ans;
loop (num = 0 to 2^n - 1) {
    list;
    loop (i = 0 to n-1) {
        if (num & (1<<i))
            list.push(nums[i]) }
    ans.push(list)
}
```

0 → ✗ , 1 → ✓

	2	1	0
[] ←	0	0	0
[1] ←	0	0	1
[2] ←	0	1	0
[1, 2] ←	0	1	1
[3] ←	1	0	0
[1, 3] ←	1	0	1
[2, 3] ←	1	1	0
[1, 2, 3] ←	1	1	1

⟹ Advanced Maths

1. All divisors of a Number N = 36
   ans = {1, 2, 3, 4, 6, 9, 12, 18, 36}

* loop (i = 1 to N)
   if (N % i = 0) ans. push (i)

* loop (i = 1 to sqrt(N)) {
   if (N % i = 0) { ans. push (i)
      if (N/i != i) {
         ans. push (N/i) }}}

2. Check if prime or not.
   ↳ divisible by 1 & itself AND no. of divisors = 2.

* cnt = 0
   loop (i = 1 to N) if (N % i = 0) cnt ++
   if (cnt = 2) → "Yes"
   else → "No"

* cnt = 0
   loop (i = 1 to sqrt N) if (N % i = 0) cnt ++
      if (N/i != i) cnt ++
      if (cnt > 2) break → "No"
   if (cnt = 2) → "Yes"

3. Print prime factors of Number

* first finding all divisors (1.) & then checking if prime (2.)

* loop (i = 2 to N)
   if (N % i = 0) ans. push (i)
      while (N % i = 0)
         N = N/i

N = 780 → T.C. = O(13)
N = 37 → T.C. = O(37) = O(N)
↳ large no. which is itself a prime no.

* loop (i = 2 to sqrt(N))
   if (N % i = 0) ans. push (i)
      while (N % i = 0)
         N = N/i
   if (N != 1) ans. push (N)

4. Power Exponentiation

??

Sliding Window of Two Pointer Approach

1. Constant Window

eg arr = {-1 2 3 3 4 5 -1}, k = 4

return the max^m sum of a consecutive subarray of size k.

0	1	2	3	4	5	6
-1	2	3	3	4	5	-1
l	l	l	l r	r	r	r

sum = -1+2+3+3 = ~~7~~ ~~12~~ (2+3+3+4) ~~15~~ (3+3+4+5) 11 (3+4+5+(-1))

max_sum = max (max_sum, sum) = ~~7~~ ~~12~~ 15

∴ $\boxed{15}$

⇒ l = 0, r = k-1, max_sum = -∞, sum = sum from l→r.

⇒ loop (r < arr.size)

```
sum -= arr[l]; l++;
r++; sum += arr[r];
max_sum = max (max_sum, sum)
```

2. Longest subarray / substring with a <condition>

eg Longest subarray with sum <= k

arr = {2 5 1 7 10}, k = 14

⇒ Medium Questions

1. Max^m points obtained from cards

select (k) cards either from start or end of arr to get max^m pts.

arr = [6, 2, 3, 4, 7, 2, 1, 7, 1], k = 4

(2) (1) (2)

5 cases →
- 4 from front
- 3 front, 1 end
- 2 front, 2 end
- 1 front, 3 end
- 4 end

lsum = ~~0 15 14 8 6~~ 0
rsum = ~~0 1 8 9~~ 11

sum = 15, 12, 16, 15, 11
↳ max_sum = 16.

```
lsum = 0, rsum = 0, maxSum = 0
loop (0 to k-1) {  lsum += arr[i]
                   maxSum = lsum }
rindex = n-1
loop (i = k-1 to 0) {  lsum -= arr[i]
                       rsum += arr[rindex]
                       rindex --
                       maxSum = max (maxSum, lsum + rsum)
```

2. Number of Nice Subarrays

↳ subarray with 'k' odd nos.

ans = 0, odd_cnt = 0

nums

0	1	2	3	4
1	1	2	1	1

- traverse 'nums' & if element is odd → odd_cnt ++
- cnt [odd_cnt] ++
- if odd_cnt > K then check for another nice subarray

cnt (size+1, 0)

cnt[0] = 1

1	0	0	0	0	0

if (odd_cnt − k >= 0) ans += cnt [odd_cnt − k]

cnt

0	1	2	3	4	5
1	~~0~~ 1	~~0~~ 1 2	~~0~~ 1	~~0~~ 1	0

odd_cnt = ~~0 1 2 3~~ 4
k = 3
ans = ~~0 1~~ 2

3. Max. Consecutive Ones III

nums → array (binary)
k → integer (no. of flips allowed)

```
start = 0, zerocnt = 0
maxLen = 0
loop (end = 0 to n-1)
    if (nums[end] = 0) → zerocnt ++
    loop (zerocnt > k) → ~~zerocnt --~~
                    if (nums[start] = 0) zerocnt--
                    start ++
    maxLen = max (maxLen, end - start + 1)
```

4. Longest Repeating Character Replacement

Hard → longest substr, subarr with k, min win substr, min win subseq.

5. Binary Subarrays with Sum

0	1	2	3	4
1	0	1	0	1

nums = [1, 0, 1, 0, 1] goal = 2

4 subarrays with 2 1's.

l, r = 0

shrink → l
expand → r

sum = 0 1 1 2 2 3 2

store sum of element in window

cnt = 0 ~~3~~ 1 3 6 10 14

count no. of subarr with (<= goal) 1's

- we'll calculate no. of subarray with (<= goal) 1's
  & then
  ⇒ func(goal) − func(goal − 1)

```
loop (r<n) {
    sum += arr[r]
    loop (sum > goal) {
        sum -= arr[l]
        l++ }
    cnt += (r-l+1)
    r++
return cnt
```

exclude where no. of 1's are > goal

6. Fruits of Basket

types of fruits

arr = [3, 3, 3, 1, 2, 1, 1, 2, 3, 3, 4]

2 baskets ; 1 type of fruit in 1 basket ; 1 tree → 1 fruit ;
continous picking of fruits.
max no. of fruits ? → max. length of subarray with atmost 2 types of numbers.

Brute force

```
maxlen = 0
loop(i = 0 to n-1)
    set s;
    loop(j = i to n-1)  s.insert(arr[j])
                        if (s.size <= 2) maxlen = max(maxlen, j-i+1)
                        else break
return maxlen;
```

T.C → N^2 S.C → 3

Better → two pointer

0	1	2	3	4	5	6	7	8	9	10
3	3	3	1	2	1	1	2	3	3	4

l l l l l l l l l
r r r r r r r r r r r

	(4,1)
	(3,1) 2
×	(2,1) 2 1 0
×	(1,1) 2 3 2 1 0
×	(3,1) 2 3 2 1 0

map(num, freq)

T.C → ~~O(N)~~ O(2N)
S.C → O(3)

maxlen = 0 1 2 3 4 5

left, right, maxlen = 0
loop (right < n) {
    mp[arr[right]] ++
    loop (mp.size > 2) {
        ~~leftkey = arr[left]~~
        mp[arr[left]] --
        if (mp[arr[left]] == 0)
            mp.erase(arr[left])
        left++ }
    maxlen = max(maxlen, r-l+1)
    right++ }
return ans.

Optimised

```
l = 0, r = 0, ans = 0, mpp
loop (r<n) {
    mpp[arr[r]] ++
    if (mpp.size > 2) { mpp[arr[l]] --
                        if (mpp[arr[l]] == 0) erase arr[l]
                        l++ }
    if (mpp.size <= 2) { maxlen = max(maxlen, r-l+1)
                         } r++ };
return maxlen
```

7. Longest Substring Without Repeating characters

0 1 2 3 4 5 6 7 8

s = c a d b z a b c d

l l l

x x x x x x x x x

maxlen = 0 1 2 3 4 5

left, right = 0

left = mpp[s[right]] + 1

↳ if char already visited

len = r - l + 1

z, 4
b, 3 6
d, 2 8
a, 1 5
c, 0 7

↳ map(char, index)

↳ vector<int> mp(256, -1)

the index which you are checking (if its present or not), it should be after 'left' pointer.

if (mpp[s[r]] ≥ l)

8. No. of substrings containing all three chars

s = b b a c b a

Greedy Algorithms

⇒ Easy Problems

1. Assign Cookies

greed_factor = [1, 2, 3] (g)

size = [1, 1] (s)

- sort both arrs.
- s_ind, g_ind = 0
- loop (s_ind < s.size & g_index < g.size)
  if g[g] <= s[s] → g++;
  s++
  return g_ind.

2. Valid Parethesis

min = 0, max = 0 ⇒ Keep record of range

(I) if '(' → cnt++
'(' → cnt--
'*' → return check (cnt++) || check(cnt--) || check(cnt)
if cnt < 0 → false
return (cnt == 0) when traversed the whole string.

(II) maintain a range of cnt → [min, max] & neglect the case where cnt can be -1

min, max = 0
if '(' → min ++, max++
else if ')' → min --, max --
else → min --, max ++

if (min < 0) → min = 0 → neglect case of (-1)
if (max < 0) → return false → ')('
return (min == 0) → min = -1, max = -1

3. Lemonade Change

5 → 10 → 20

cost of lemonade = 5

change 5, change 10 = 0

- if (bill = 5) → change 5 ++
- else if (bill = 10) → ~~then~~ if (change5 > 0) change 5 --
  change 10 ++
  else return false
- else if (bill = 20) → if (change10 > 0 || change 5 > 0)
  change 10 - -
  change 5 --
  else if (change 5 > 2)
  change 5 -= 3
  else return false.

return true.

4. Min^m Coins

den = {1, 2, 5, 10, 20, 50, 100, 500, 1000}

loop (i = size-1, i >= 0, i++)

while (n >= den[i]) n -= den[i]

ans. push(den[i])

5. Fractional Knapsack

profit / weight

- sort ~~of~~ on the basis of this fraction
- ans = 0
- loop (i = 0 to n-1)

  if (weight_i < W) ans += profit_i
  W -= weight_i

  else ans += (profit_i / weight_i) * W

  break;

⇒ Lecture Problems / Medium Questions.

1. N Meetings in One Room / Activity Selection Problem.

- make a vector 'time' → stores start & end time of ith meeting.
- sort this vector on basis of end time. (lesser end time → first)
- cnt = 1 → no. of meeting ; end_time = first end time (time[0].sec)
  ↳ stores the end time of last counted meeting.
- loop (i = 1 to n-1)

  if (start_i > end_time)

  cnt ++

  end_time = end_i

time = [(1,2), (3,4), (0,6), (5,7), (8,9), (5,9)]

cnt = ~~1~~ ~~2~~ ~~3~~ 4 end_time = ~~2~~ ~~4~~ ~~6~~ ~~7~~ 9

∴ 4 meeting

(1,2), (3,2), (5,7), (8,9)

3 > 2
0 > 4
5 > 4
8 > 7
5 > 9

2. Shop in Candy Store

candies = [3, 2, 1, 4] N = 4

- min_amt → buy cheaper get costly
- max_amt → buy costly get cheaper

k = 2 → no. of free candies.

min. of max. amt. to buy all 'N' candies.

→ sort candies → [1, 2, 3, 4]

* for min_amt → buy from start & get free from end.

loop (b <= f) min_amt += [buy]

buy ++ ; free -= k

* for max_amt → buy from end & get free from start

loop (f <= b) max_amt += [buy]

buy -- ; free += k.

3. Reverse words in String

s = i.am.vt

- ans = ''; temp = ''
- loop (i = n-1 to 0)
  - → store every letter in 'temp' from last until there is a '.'
  - → if s[i] = '.' { reverse(temp)
    ans += temp; ans.push('.')
    temp = '' }
- for the first word of 's', there is no '.' & its still in 'temp'
  - ⇒ reverse(temp)
  - ⇒ ans += temp

ans = vt.am.i

temp = ~~vt~~ ~~am~~ i

4. Chocolate Distribution problem

arr = [3, 4, 1, 9, 56, 7, 9, 12], n=8, m=5

N packets, M stu.

- 1 packet → 1 stu.
- diff = max - min chocolates
  ↳ minimum

- sort arr
  → [1, 3, 4, 7, 9, 9, 12, 56] ⇒ select 'M' packets
- 9-1 = 8
  $\boxed{9-3=6}$
  12-4 = 8
  56-7 = 49

min_diff = ~~8~~ 6 ⇒ $\boxed{6}$

loop(m+i-1 < size)
diff = a[m-1+i] - a[i]
min_diff = min(min_diff, diff)
i++

return min_diff.

5. Min. Cost of Ropes

[4, 3, 2, 6]

[4, 5, 6]

[9, 6]

[15]

lengths = [4, 3, 2, 6]

cost → $length_1 + length_2$

cost = ~~0~~ 0 + 5 + 9 + 15 = $\boxed{29}$

- always add the least 2 lengths
- store the sum ⇒ min_heap.
- repeat

(i) push every element in min heap
(ii) cost = 0
(iii) until size of min heap > 1
first, second
cost += (first + second)
min_heap.push(first + second)

6. Job Sequencing Problem

- sort on the basis of profit. greater profit → first
- vector schedule (max_deadline + 1, -1)
- jobs = 0 → no. of jobs done ; maxprofit = 0 ;
- loop (k = deadline; to 1)
  if (not scheduled yet) jobs++;
  maxprofit += profit;
  schedule[k] = id;
  break;

return { jobs, maxprofit }.

Binary Trees

⟹ Traversals

* Full BT : either 0 or 2 children.

* Complete BT : • all levels completely filled except last level.
  • last level has all nodes on left as possible.

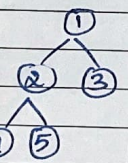
* Perfect BT : all leaf nodes at same level.

* Balanced BT : height (max) = log(N) ↳ no. of nodes.

* Degenerate trees ⟶ has only 1 child.

1. Inorder ⟹ if (!root) ⟶ return;
   (LNR)
   inorder (root → left)
   print (root → val)
   inorder (root → right)
2. Preorder ⇒ (NLR)
3. Postorder ⇒ (LRN)

4. Levelorder root = [1, 2, 3, 4, 5]

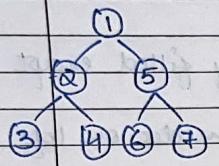


queue [1|x]

```
loop (!q.empty) { temp = q.front(), q.pop()
                  if (!temp) cout << endl;
                             if (!q.empty) q.push(x)
                  else print (root → val)
                       if (temp → left) q.push(temp → left)
                       if (  → right) q.push(  → right)
```

1. Iterative preorder
2. " inorder
3. postorder → 2 stack
4. " → 1 "
5. pre, in, post → in 1 traversal.

5. Preorder | Inorder | Postorder in 1 Traversal



if num = 1 & preorder (push in), ++ (num push if), left
if num = 2 & inorder, ++, right
if num = 3 & postorder

num = ~~1~~ 2

pre : 1, 2, 3, 4, 5, 6, 7
in : 3, 2, 4, 1, 6, 5, 7
post : 3, 4, 2, 6, 7, 5, 1

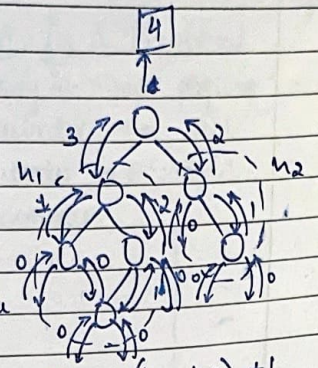
~~(3, 1) ~~2~~~~	~~3~~
~~(2, 1) ~~2~~~~	~~3~~
(1, 1) ~~2~~	3

(node, num) stack

⇒ Medium Questions

1. Height of BT

height (Node* node)
if (!root) → return
h_1 → height of left node
h_2 → height of right node
ans → max(h_1, h_2) + 1;
↓
1 root node.



⇒ max (h_1, h_2) +1

2. Balanced Tree

(I) if (!root) → true
(i) left = if left subtree is balanced
(ii) right = if right subtree is balanced
(iii) cond = abs (height (left) − height (right)) <= 1

2 functions are called → $O(n^2)$

return = (left && right && cond)

(II) make a func. which return the is Balanced & height together. & then these both can be used accordingly.

pair <bool, int> func (root) { if (!root) → return pair (true, 0)

left = func(root → left)
right = func(root → right)
(i) lAns = left.first
(ii) rAns = right.first
(iii) diff. = ~~abs l.first − r.fir~~ abs (l.second − r.second) <=1

ans.first = (lAns && rAns && diff)
ans.second = max (l.second, r.second) + 1
return ans.

3. Diameter of BT → longest path b/w 2 end nodes.

↓ leaf | root node.

left, right, through root.

(I) if (!root) → return 0
op1 = diameter (root → left)
op2 = diameter (root → right)
op3 = height (left) + height (right) + 1
ans = max {op1, op2, op3}

→ $O(n^2)$

(II) pair<int, int> func (root) { if (!root) → return {0, 0}
↓ diam. ↓ height

left = func (root → left)
right = func (root → right)

op1 = left.first
op2 = right.first
op3 = left.sec + right.sec + 1

⇒ ans.first = max {op1, op2, op3}
ans.sec = max (l.sec, r.sec) + 1

4. Identical Tree

if (!r1 && !r2) → true
if (!r1 && r2) → false
if (r1 && !r2) → false.

return (left && right && value)

left = isIdent. (r1 → left, r2 → left)
right = isIdent. (r1 → right, r2 → right)
value = r1 → data == r2 → data

5. Sum tree

check value

pair<bool, int> func (root) { if(!root) → return{true, 0}

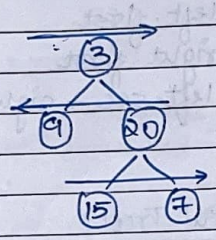
⇒ if(leaf node) → return (true, root→data)

left = func(root→left) leftans = l.first
right = func(root→right) rightans = r.first
cond = root→data == left.sec + right.sec.

⇒ ans.first = (lans && rans && cond)
ans.sec = 2 × root→data.

6. Zig Zag Traversal

o/p = {{3}, {20, 9}, {15, 7}}



q.push(root)

- level order traversal

ans = {{3},

- first push the elements of 1 level from the queue for a given dir^n in a temp vector

left to right = true
↳ toggle after every level

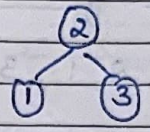
queue: 20, 9, 3

- vector temp (q.size)
- if left of node exist, push it in queue & same for right of node.

temp = ~~{3}~~, {20, 9}

index in which element will be,
= i (if left to right dir^n)
= n-1-i (if right to left dir^n)

7. Bottom Left Tree Value

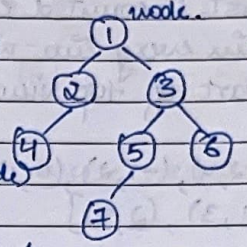


o/p = 1

queue: 4, 3, ~~2~~, ~~1~~

- make queue & push root.

loop (i = 0 to q.size - 1)
node = front()
pop()
if (i = 0 ⇒ root node)
ans = node
if left node exist push in queue & same with right node.



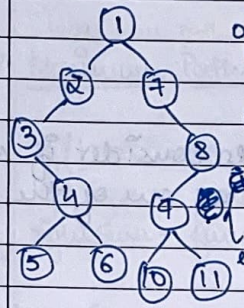
o/p = 7

because leftmost node will be stored first in queue.

ans = node = ~~1~~ 2

return ans → val.

8. Boundary Traversal



o/p ⇒ 1, 2, 3, 4, 5, 6, 10, 11, 9, 8, 7

(i) left boundary (exclude leaves)
(ii) leaf nodes
(iii) right boundary in reverse dir^n (exclude leaves)

(i)
- queue & push root
- if (left) → push left
- if (! left & right) → push right

else

(ii) if (! left & ! right) { inorder traversal }
⇓
leaf

(iii)
- stack
- if (right) → push right
- if (! right & left) → push left

else

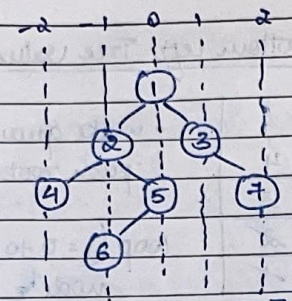
& then push one by one into the queue. (LIFO)

stack: 9, 8, 7

10	7
9	8
8	9
7	11
6	10
5	6
4	5
3	4
2	3
1	2
0	1

ans

9. Top View of Binary Tree



o/p = 4, 2, 1, 3, 7

- first encountered element in every line will be part of top view.

map ⇒ [(-2,4), (-1,2), (0,1), (1,3), (2,7)]

ans ⇒ 4, 2, 1, 3, 7
⇒ value of every element of map.

0 → 5 ↗ not considered.

	4 -1 → 6
+2 → 7	
-2 → 4	
+1 → 3	
-1 → 2	
0 → 1	

~~(6,-1)~~
~~(7,2)~~
~~(5,0)~~
~~(4,-2)~~
~~(3,+1)~~
~~(2,-1)~~
~~(1,0)~~

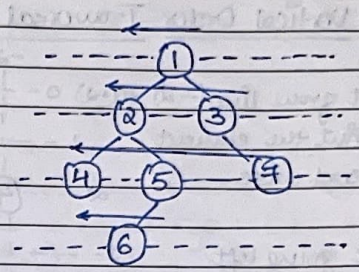
map (line, node)
↓
sorted

queue (node, line)

10. Bottom View of Binary Tree

same as top view but ~~don't~~ do consider if node comes out with a line same as one of the previous node.

11. Right Side view of BT



right view = 1, 3, 7, 6

- last node at every level is included in right view.
or first node if moving right to left dir^n.

reverse preorder : NRL

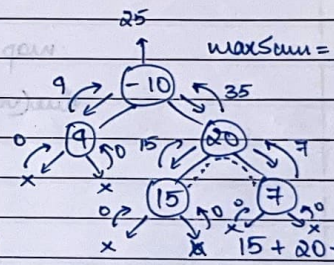
```
func (node, level, ans) {
    if (!node) return
    if (level = ans.size) ans.push(node)
    func (node → right, level+1, ans)
    func (node → left, level+1, ans)
}
```

for left view.
↓
(node → left, - -
(node → right - -

vertical order traversal

max-sum path.

12. Maximum Path Sum



maxSum = ~~-∞~~ ~~1~~ ~~5~~ 42

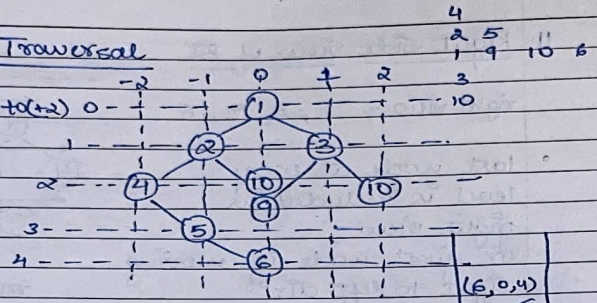
```
if (!root) return 0
leftSum = func (root → left)
rightSum = func (root → right)

maxSum = max (maxSum, lSum + rSum + root → val)
return (root → val) + max (lSum, rSum)
```

INT_MIN

ans. to the ques.

13. Vertical Order Traversal

- start from line (−2) to (+2) & print the element level wise.



- while going left → level ++, line --
- while going right → level ++, line ++

		⑤	0 → 2, → {10, 9}
		①	−2 → 2, → {4}
⑥	0 → 4 → {6}	④	1 → 1, → {3}
③	−1 → 3 → {5}	②	−1 → 1, → {2}
⑧	2, 2 →, {10}	⑦	0 → 0, → {1}

→ {9, 10, 5}

(6, 0, 4)
(5, −1, 3)
~~(10, 2, 2)~~
(9, 0, 2)
(10, 0, 2)
(4, −2, 2)
(3, −1, 1)
(2, 1, 1)
(1, 0, 0)

map <int, map <int, multiset<int>>>

line (vertical) → level

↳ nodes.

∵ same values

queue (node, line, level)

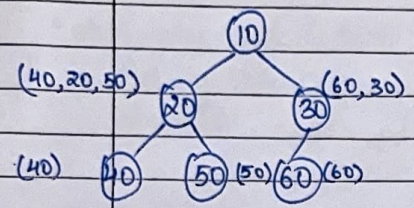
⟹ Hard Questions

1. Flatten a BT into LL

```
prev = NULL
flatten (node)
    if (!node) return
if (right) flatten (node → right)
if (left) flatten (node → left)

node → right = prev.
node → left = NULL
prev = node.
```

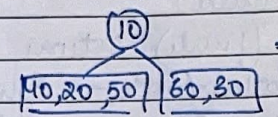
2. Min. Time To Burn a Tree from a Node.

3. BT from preorder & inorder



(LNR) in = 40, 20, 50, 10, 60, 30

(NLR) pre = 10, 20, 40, 50, 30, 60

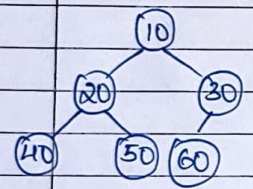


⇒ recursive approach.

4. BT from postorder & inorder

(LNR) in = 40, 20, 50, 10, 60, 30

(LRN) post = 40, 50, 20, 60, 30, 10



5. Count Complete Tree Nodes

4 Brute force ⇒ inorder (node, cnt) { if (!root) return

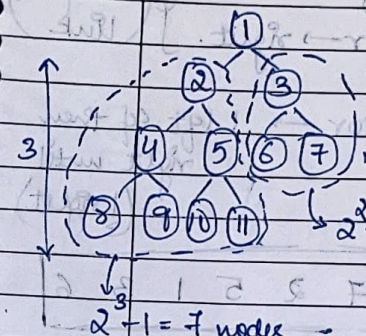
T.C. O(N) ← cnt++

S.C. O(h) → logN ∵ complete BT.

inorder (node → left)

inorder (node → right) }

4 Complete Tree property



Total Nodes = 1 (root) + 7 (left) + 3 (right)

• if subtree → full BT (either 0 or 2 child)

⇒ apply formula (2^H-1)

T.C.	$O(\log N)^2$
S.C.	$O(\log N)$

node 1: left Height = 4, right Height = 3 → not equal → Not full BT (no direct formula)

⇒ 1 + (left) + (right)

node 2: left H = 3, right H = 3 → equal → full BT ⇒ $2^3-1=7$

node 3: left H = 2, right H = 2 → equal → full BT → $2^2-1=3$

So 1 + (7) + (3) = $\boxed{11}$

Algo

left Height () while (node) h++; node = node → left

right Height ()

countNodes () if (lH = rH) return $(2^{lH}-1)$

return 1 + countNodes (left) + countN (right)

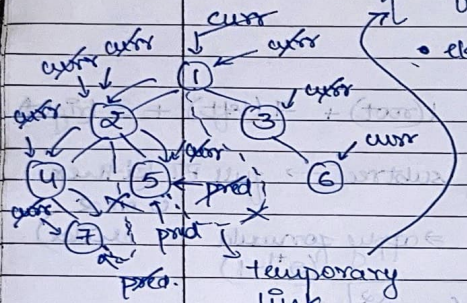
postorder → 7 4 5 2 6 3 1

6. Morris Traversal

S.C. O(1)
T.C. O(N)

Algo:
- curr = root
- while (curr)
  - if (!curr → left) visit (curr) → print curr
    curr = curr → right
  - else predecessor = find (curr)
    - if (!pred → right) pred → right = curr
      curr = curr → left
    - else pred → right = NULL
      visit (curr)
      curr = curr → right.

      (remove temp link)
    - * predecessor → 1 left & then right until (!right)



ans ⇒ 4 7 2 5 1 3 6

Inorder

* A node → atmost 3 times visited (create link, print, remove link)

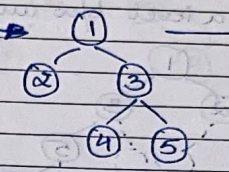
find (pred) → pred = curr → left
while (pred → right & pred → right != curr)
pred = pred → right

preorder → 1 2 4 7 5 3 6

↳ just visit (curr) in innermost 'if' instead of the innermost 'else'

if (!pred → right) pred → right = curr
visit (curr)
curr = curr → left

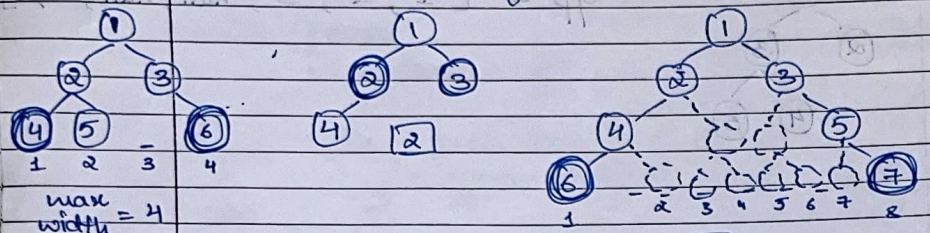
7. Serialize & Deserialize Binary Tree



→ O/P = ['1', '2', '3', null, null, '4', '5']

8. Maximum Width of Binary Tree

↳ no. of nodes in a level b/w any 2 nodes.

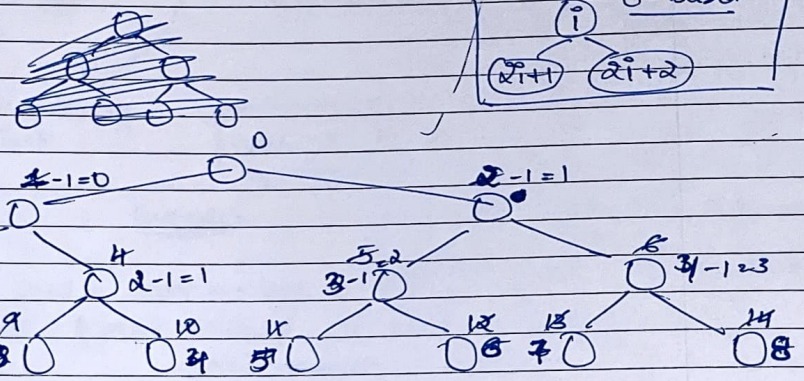


* level order traversal

(no. of nodes → 1st node of last node of that level)

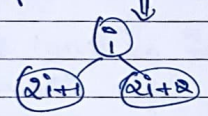
width = (lastNodeIndex) − (firstNodeIndex) + 1

max^m value

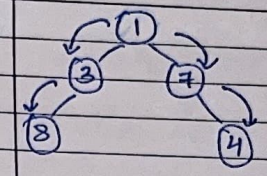


• On every level i = (i − min)

↳ left most index of level



eg



(4,4)
(8,1)
~~(7,2)~~
~~(3,1)~~
~~(1,0)~~

queue

width = ~~1~~ ~~2~~ $\boxed{4}$

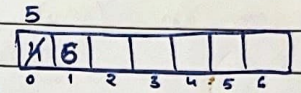
(last − first + 1)

9. Lowest Common Ancestor of BT

Stacks & Queues

⟹ learning

1. Stack using array



top = -1

int st[size]
top = -1
push(4), pop(), top(), push(5), push(6), size(), pop()

push (x) { if (~~top < size - 1~~) top++; st[top] = x else → overflow }

pop() { if (top >= 0) top-- ; else → underflow

top() { return st[top] → if top != -1
size() { return st.size() or top+1

2. Queue using array

5. Valid ~~Paren~~ Parenthesis

- if size(string) = 1 → false
- push the open brackets → '(', '[', '{' in the stack
- if it's a closing bracket
  - first check if there's any opening bracket → if (st.empty) → false
  - now check the top() element & compare it with it's corresponding closing bracket.
    ch = st.top → st.pop
    if (s[i] = ')' & ch = '(') or (s[i] = ']' & ch = '[') or (s[i] = '}' & ch = '{') → continue
    else → false
- at last, return true if stack is empty i.e all brackets pair exist.

- every open bracket → close bracket (same type) & vice versa
- in order → opening & closing of different brackets

⟹ Prefix, Infix, Postfix Conversion

* priority order : $(\wedge) > (*, /) > (+, -)$
* Infix expression : $(p+q)*(m-n)$
* Prefix expression : $*+pq-mn$
* Postfix expression : $pq+mn-*$ (stack based calculators)

1. Infix to Postfix

$a+b*(c\wedge d-e)$

i=0	stack	ans.
a		a
+	+	a
b	+	ab
*	+*	ab
(	+*(	ab
c	+*(	abc
^	+*(^	abc
d	+*(^	abcd
-	+*(-	abcd^
e	+*(-	abcd^e
)	+*	abcd^e-
		abcd^e-*+

- operand → ans.
- opening bracket → stack
- operator → if stack empty → stack
  - if stack top has less priority → stack
  - if stack top has more priority → pop st.top & push in ans + push opr. in st.
- closing bracket → pop everything until opening bracket. & push in ans.

```
intoPostfix (s) {
    i=0, st, ans = ""
    while(i<n)  if (operand) ans += s[i]
                else if ('(') st.push(s[i])
                else if (')') while (!st.empty &&
                                st.top != '(') {
                                ans += st.top
                                st.pop } st.pop
    operator ← else while (!st.empty &&
                        priority(s[i]) <= priority(st.top)
                        ans += st.top
                        st.pop
                    st.push(s[i])
                i++
    while (!st.empty)
                ans += st.top
                st.pop
    return ans → postfix exp. }

priority(ch) { if (ch = ^) return 3
               else if (ch = * or /) return 2
               else if (ch = + or -) return 1
               else return -1
```

2. Infix to Prefix
   - reverse infix expression
   - change open bracket to close bracket & vice versa
   - infix to postfix conversion
   - reverse postfix expression → (prefix)

```
if ('^') while (!st.empty && priority(s[i]) <= priority(top))
                ans += st.top   st.pop
else while (!st.empty && priority s[i] < priority(top))
                ans += st.top   st.pop
st.push(s[i]) }
```

else { → (operator)

3. Postfix to Infix

```
post-to-infix (s) {
  i=0, st
  while(i<n) { if (operand) (st.push s[i])
                 else  t1 = st.top ; st.pop
                       t2 = st.top ; st.pop
                       temp = '(' + t2 + s[i] + t1 + ')'
                       st.push(temp)
  i++ }
  return st.top().}
```

4. Prefix to Infix

```
pretoinfix (s) {
  i = n-1;
  while(i>=0) { if (operand) st.push(s[i])
                else  t1 = top, pop
                      t2 = top, pop
                      temp = '(' + t1 + s[i] + t2 + ')'
                st.push temp
  i++ }
  return st.top()
```

5. Postfix to Prefix i=0 ; temp = s[i] + t2 + t1

6. Prefix to Postfix i = n-1 ; temp = t1 + t2 + s[i]

⟹ Monotonic Stack & Queues Problems

1. Next Greater Element

~~Heaps~~ # Strings

~~Learning~~ ⇒ Basic & Easy

1. Remove outermost Parenthesis

valid parenthesis → "" (empty) ; "(" + A + ")" ;
A+B

A, B → valid parenthesis

primitive parenthesis → non empty & not able to split into valid paren.

$S = P_1 + P_2 + \dots + P_k$

P_i → primitive valid paren.

↓

remove outermost paren. of every P_i

⇒ ans (string), n (int) = 0
loop (i = 0 to size − 1)
if (opening bracket) { if (n > 0) → not outermost
{ put in ans, += S[i] }
n++ }
else (closing) { n--
if (n > 0) { put in ans } }
return ans.

2. Reverse words in string

"the sky is blue" → "blue is sky the"
"... hello world ..." → "world hello"
"a good ... example" → "example good a"

ans (string) ; temp (string) ; firstword (bool)

temp ⇓ to store curr. word

firstword ↓ true.

```
loop (i = size -1 to 0)
    if (blank space)
        if (temp not empty)
            if (not firstword) → add space to ans
            loop (temp -1 to 0) → ans += temp[i]
            initiate temp = " "
            firstword = false
    else
        temp += s[i]

if (! temp.empty)  if (! firstword) ans = ' '
                   loop (temp -1 to 0)
                       ans += temp[i]
return ans.
```

→ append the rev. word if temp not empty

3. Implement atoi (ascii to integer)

```
index = 0, sign = 1, ans = 0
(i) whitespaces → loop (i < size && s[i] = ' ') i++
(ii) sign → if (i < size && (s[i] == '+' || s[i] == '-'))
                sign = 1 (if '+')
                -1 (if '-')   i++
(iii) not a digit → if (i < size && !isdigit(s[i])) return 0
(iv) digit → while (i < size && isdigit(s[i])) {
                digit = s[i] - '0'
                ans = ans * 10 + digit
                i++ }
return sign * ans;
```

4. Roman to Integer

unordered map (m) → m['I'] = 1
m['V'] = 5 -- -- m['M'] = 1000

```
ans = 0
loop (i = 0 to size -1)
    if (m[s[i]] < m[s[i+1]])
        ans -= m[s[i]]
    else
        ans += m[s[i]]
return ans.
```

eg 'IV'
m['I'] < m['V']
1 5
ans = -1
then ans = -1 + 5
= $\boxed{4}$

5. Sort characters by frequency

'tree' → 'eert' or 'eetr'
'Aabb' → 'bbAa' or 'bbaA'
'A' & 'a' treated diff.

unordered map (freq)
<char, int>

(i) fill every char. with it freq.
(ii) vector to store pair of char & freq from map
vector <pair <char, int>> vector (freq.begin(), freq.end())
(iii) sort the vector acc. to freq.
sort (vector.begin(), vector.end(), [](auto &a, &b) { return a.second > b.second })
(iv) append in ans. string
loop (auto &p : vector) {
ans += string (p.second, p.first)
// append p.first , p.second times.

Arrays

⟹ Hard Questions

SDE

1. Pascals Triangle

```
                    if rows = 1 ⟶ return {1}
                    if rows = 2 ⟶ return {{1}, {1, 1}}
                                  i=
                    loop (2 to rows) i
                        temp (i+1)
                        temp [0] = temp [i] = 1
                        loop (j = 1 to i-1)
                            temp [j] = ans [i-1][j-1]
                                       + ans [i-1][j]
                        ans[i] = temp
                    return ans.
```

	(i)				
ans	[0]	1			
		[0]	[1]		
	[1]	1	1		
		[0]	[1]	[2]	
temp(3)	[2]	1	1+1	1	
		[0]	[1]	[2]	[3]
temp 4	[3]	1	3	3	1

SDE

2. Merge sorted array w/o extra space

	0	1	2	3	4	5
n1	1	2	~~3~~ 2	3	5	6
	i	i = m-1	k	k	k = m+n-1	

	0	1	2
n2	2	5	6
	j	j	j = n-1

```
loop (i ≥ 0 && j ≥ 0)
    if (n1[i] > n2[j])   n1[k] = n1[i]
                         i--
    else  n1[k] = n2[j]
          j--
    k--
loop (j ≥ 0)
    n1[k] = n2[j]
    j--
    k--
```

SDE

4. Repeat & Missing Number

arr(3) → n

1	3	3

(I)

S_n → sum of 1st (n) nos. X → missing no. (2)

S → sum of array element X → repeating no. (3)

⇒ $S - S_n = X - Y$ (val1)

S_{2n} → sum of sq. of 1st (n) nos.

S_2 → sum of sq. of array element

⇒ $S_2 - S_{2n} = X^2 - Y^2$ val2

$X = (val1 + val2)/2$

⇒ $X + Y = (X^2 - Y^2)/(X - Y)$ (val2)$_{new}$ $Y = X - val1$

$n(n+1)/2$ $S_n = 1+2+3 = 6$ $S = 1+3+3 = 7$

$n(2n+1)/6$ $S_{2n} = 1^2+2^2+3^2 = 14$ $S_2 = 1^2+3^2+3^2 = 19$

val1 = $S - S_n = 1(X-Y)$

val2 = $S_2 - S_{2n} = 5$ ⇒ val2 = val2/val1 ⇒ $5/1 = 5\ (X+Y)$

$X = (1+5)/2 = 3$

$Y = 3 - 1 = 2$

SDE

5. Find Duplicate in array of (N+1) integers

(I) Hashing → ~~freq(N+1) = {0}~~ freq(N+1, 0) → vector

loop(i to n-1)

if(freq[arr[i]] == 0)

freq[arr[i]]++

else return arr[i].

T.C. = O(n)

S.C = O(n)

(II) Linked List Cycle Method

arr[9]

arr[2] → 1 → 5 → 3 → 6 → 8 → 7 → 9 → 2

0	1	2	3	4	5	6	7	8	9
2	5	9	6	9	3	8	9	7	1

3. Largest Subarray with 0 Sum

SDE

0	1	2	3	4	5	6	7	8	9
1	-1	3	2	-2	-8	1	7	10	23

sum = 1 + (-1) = 0 + 3 = 3 + 2 = 5 + (-2) = 3 + (-8)
= -5 + 1
= -4 + 7
= 3 + 10
= 13 + 23
= 36

(36, 9)
(13, 8)
(-4, 6)
(-5, 5)
(5, 3)
(3, 2)
(1, 0)

maxi = 5

s=3 s=2

0 1 2 3 4 ⇒ length = 4 - 2 = 2

s=3

s=3 s=0

0 1 2 ... 7 ⇒ lm = 7 - 2 = 5

s=3

(k, v) → index.

prefix sum

T.C. = O(N log N)

S.C. = O(N)

loop(i=0 to n-1) { sum += arr[i]

if(sum == 0) maxi = i+1

else { if(sum present in map)

maxi = max(maxi, i - mp[sum])

else

mp[sum] = i

return maxi.

```
do { slow = num[slow]
     fast = num [num[fast]] }
while (slow != fast)
fast = num [0]
loop (slow != fast) { slow = num[slow]
                      fast = num[fast] } return slow;
```

6. Merge Overlapping Subintervals arr

SDE

i/p	[[1,3], [2,6], [8,10], [15,18]]
o/p	[[1,6], [8,10], [15,18]]

merge.

```
sort(arr.begin(), arr.end())
vector currInt = arr[0]

loop(i=1 to n-1)
    if (arr[i][0] <= currInt[1])
        currInt[1] = max (currInt[1], arr[i][1])
    else
        ans.push(currInt)
        currInt = arr[i]
ans.push(currInt)
```