*** 1<<n represents 2^n for a structure N of len(n). Nums from 0 to 2^n-1 represents the complete subset for N (i.e which which element to include or exclude to get an element of the subset) (0 → exclusion, 1 → inclusion)

✡ floor($\log_{10}$(num)) = no. of digits

CONVERSIONS

Binary to decimal

$\boxed{10010.01}$

$\Rightarrow (2^{-2}\times 1)+(2^{-1}\times 0)+(2^{0}\times 0+)\ldots$

✡ negative powers for decimal places ones.

decimal to base b

$\boxed{1056.24}$

$\Rightarrow$ b | 1056 | m
b | x | n
y

. 0.24 x b = p.q3

(contd termination)

ARRAY MEMORY

int arr[5]; // 4 bytes in integer ∴

8	10	12	4	7
x	x+4	x+8	x+12	x+16

✡ for diff. data types byte size changes ∴ eg ⇒

a	b	c
x	x+1	x+2

(as char ⇒ 1 byte)

BIT MAINUPULATION & MATHS

✡ 1 byte = 8 bits

✡ a<<1 ⇒ $2^1 \times a$

✡ a>>1 ⇒ a × ~~5~~ $2^{-(1)}$

✡ a & 1 = a

✡ a^1 = $\bar{a}$, a^0 = a, a^a = 0

✡ N = a1b, −N = $\bar{a}$1b, right most set bit isolation = N&−N

✡ for n bits, range = -2^{n-1} to $2^{n-1}-1$

✡ no. of digits in base b = (int)($\log_b n$)+1 (for num n)

✡ 1000 = 111+1

✡ sieve of eratosthenes ⇒ if a is prime the an is also prime

✡ newton raphson method ⇒ $\sqrt{N} = 0.5 \times (x + N/x)$; $|\sqrt{N} - x| = \varepsilon$

✡ ***

✡ euclidean algo ⇒ gcd(a,b) = gcd(b%a, a)

✡ lcm = $\frac{a \times b}{\gcd(a,b)}$

PAPERMINT

VECTORS

- {1, 2, 3} ⇒ size = 3, capacity = 3
  add 4 ⇒ {1, 2, 3, 4, _, _} ⇒ size = 4, capacity = 6
- O(1) time complexity for adding elements (total avg.)
- gets stored in heap memory (abundant) unlike that of array (stack memory (limited))

{1, 2, 3, 4}

v.begin() v.end() v.rend() v.rbegin()

{1, 2, 3, , , }

v.front() v.back()

2 POINTER

{0, 1, 2, 3, 4} ⇒ (start ++, end --)

start end

PREFFIX SUFFIX KADANES ALGO

[1, 2, 5, -2, 4]

preffix arr ⇒ [1, 3, 8, 6, 10]

suffix arr ⇒ [13, 12, 10, 2, 4]

kadane's algo ⇒ max sum = max (prefix, max sum)
if prefix < 0 ⇒ do prefix = 0
(start new subarray)
else continue extending current subarr.

PAPERMINT

2D ARRAYS

$$
\begin{array}{c|ccc}
 & 0 & 1 & 2 \\
\hline
0 & 00 & 01 & 02 \\
1 & 10 & 11 & 12 \\
2 & 20 & 21 & 22 \\
3 & 30 & 31 & 32
\end{array}
$$

(system memory)

0	1	2	3	4	5	6	7	8	9	10	11
00	01	02	10	11	12	20	21	22	30	31	32

(CONTINOUS MEMORY ALLOCATION)

- INDEX = row index × cols + col index
- row index = INDEX / cols
- col index = INDEX % cols
- address of arr[i][j] = base address + INDEX × size of data type
- eg ⇒ " " arr[2][1] = " + (2×3+1) × 4