Bloom Filter -

- Space efficient probabilistic data str. that's used to test whether an element is a member of a set.
- Eg: Checking availability of username is set membership problem, where set is list of all usernames.
- Price we pay for efficiency is that it is probabilistic in nature, that means there might be some False +ve results. False +ve means it might tell that username is already taken but actually its not.

Properties -

- Unlike std. hash table, Bloom filter of fixed size can rep. set with arbitrarily large no. of elements.
- Adding an element never fails. However, false +ve rate ↑ses steadily as elements are added until all bits in the filter are set to 1, at which pt. all queries yield a +ve result.
- Bloom filters never generate false -ve result i.e. telling you username doesn't exist when it actually exists.
- Deleting elements from filter not possible bcoz, if we delete a single element by clearing bits at indices generated by k hash fns, it might cause delet^n of few other elements.

Example -

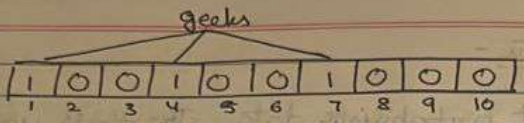
- We want to enter "geeks" in the filter we are using 3 hash fns & a bit array of len. 10 all set to 0 initially. 1st we'll calc. the hashes as follows:

  h1("geeks") % 10 = 1

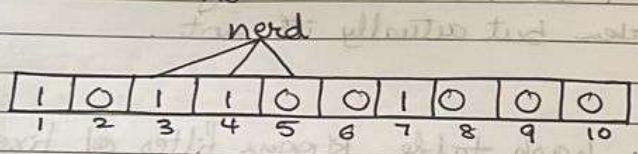
  h2("geeks") % 10 = 4

  h3("geeks") % 10 = 7

  Now we set bits at indices 1, 4 & 7 to 1



- Again we want to enter "nerd", we'll calc. hashes
  h1("nerd") = 3
  h2("nerd") = 5
  h3("nerd") = 4
  Set bits at indices 3, 5 & 4 to 1



- Now we want to check if "geeks" is present in filter or not. We'll do same process but in reverse order. We calc. resp. hashes using h1, h2 & h3 & check if all these indices are set to 1 in the bit array. If all the bits are set then we can say that "geeks" is probably present. If any of the bit at these indices are 0 then "geeks" is definitely not present.