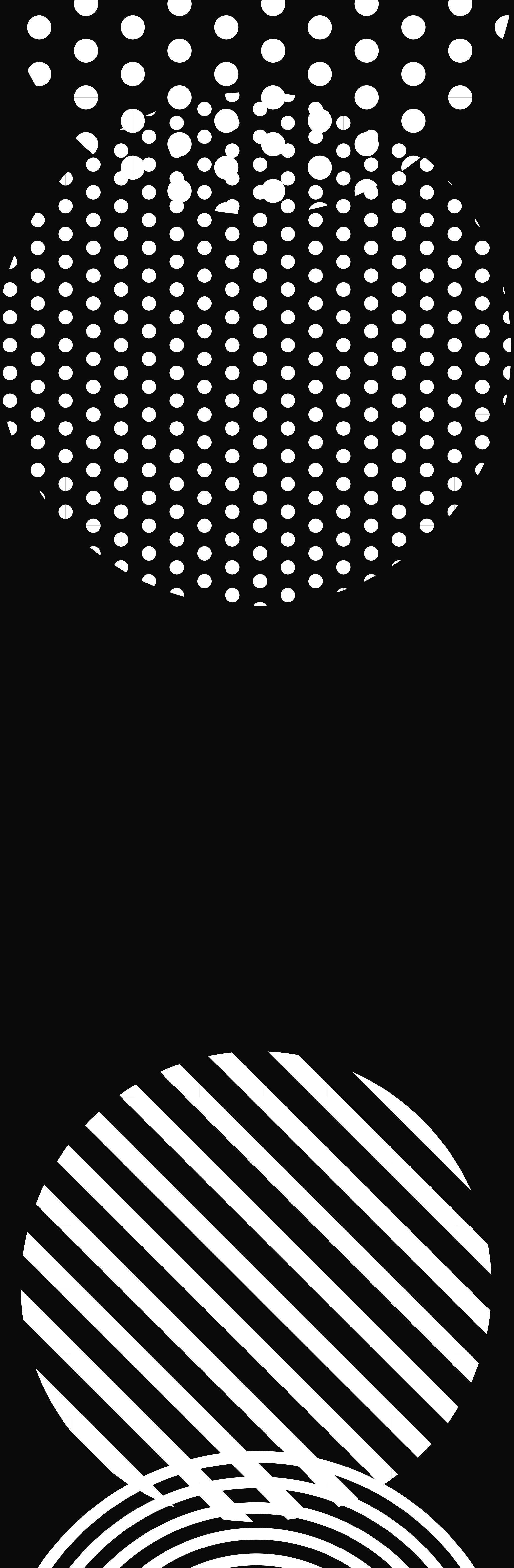


max array size $\equiv 10^5 \equiv$ 100100.

(i) vector <int> g[100100]; → basic adj. list for graphs.
int vis[100100];

(ii) To reverse a vector:; → this faster than using cmp bool of sort for descending order.

reverse (g.begin(), g.end())

(iii) Min / Max heap.

→ done by priority_queue q;

**for this whenever we are pushing into the queue,
we push (-x)

**and whenever we want the min element. int min = -q.front().

(iv) using ii = pair<int,int>;
using lli = long long;

(v) Vector <int> a(n, val);

{ initializing and stoting val in each element.

—— X ——

*** whenever there is a new number & a difference or something of that sort, we sort the array and use lower_bound & upper_bound.

1) int ind = lower_bound(a.begin(), a.end(), v) - a.begin()

↓

This gives the "first" index from left → right where the a[ind] "≥" v. greater than/equal to.

∴ ind-- is surely less than v.

2) ind = upper_bound(a.begin(), a.end(), v) - a.begin()

↓

This gives "first" index strictly greater than v. ⇒ $\overset{0}{1}\ \overset{1}{8}\ \overset{2}{8}\ \overset{3}{8}\ \overset{4}{16}$

upper(8) = 4

lower(8) = 1

ind-- may be $\underset{\leq v}{\text{equal}}$ to v or less than

Both are exactly the same when no's arent repeated in the array & query doesnt match any no.

(V) Now if u have any Seq. of Objects(int, pair<int,ints) and u have to search if they exist in the Container, Storing in a Vector would take O(n) but if u use map Or set, we can used the

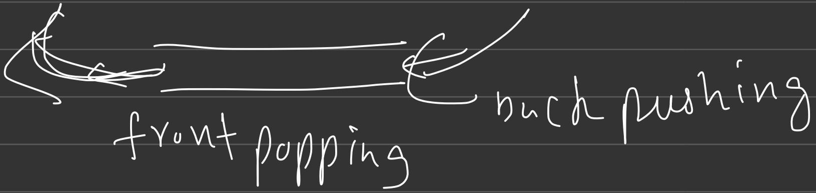
set<pair<int,int>> s;

s.find ({int1,int2}) fn which takes logn time to get.

if the query doesnt have the element, it returns s.end() so we can Check with an if cond^n whether an element exists!! in map,

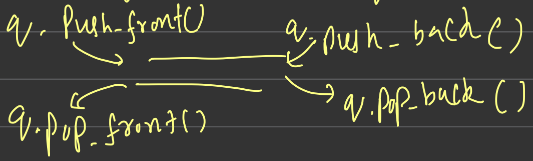
mp.find(K) where K is the key! not whole element.

iv) queue → only has q.push()



only 1 dirn! we cant push to the front of the queue.

Comes deque <int> q.



used in 0-1 bfs.

for 0 edged vertex, we push in the front, normal edges in the back.

Adj. List for weighted graphs:

```
vector <pair<int, int>> g[100100];
        g[a].push_back({b,c});
```

For Kruskal's algo: (edge list).

```
vector <pair<int, pair<int,int>>>  q;
              ↓             ↓  w  ↓
              wt            a ——→ b
  q.push_back{w,{a,b}};

 sort(q.begin(), q.end())
```

2D graph settings: