vector<int>prefixmin(n+1),suffixmin(n+1);

how u will represent this when compare to normal only n declaration

If you normally declare:

```cpp
vector<int> v(n);
```

Meaning:

- Size = `n`
- Valid indices = `0 … n-1`

But when you write:

```cpp
vector<int> prefixmin(n+1), suffixmin(n+1);
```

✔ Meaning

- Size = `n+1`
- Valid indices = `0 … n`

✔ Why n+1?

Because you want **1-based indexing**, so:

- `prefixmin[1]` corresponds to element `1`
- `prefixmin[n]` corresponds to element `n`
- `prefixmin[0]` is kept unused (or sometimes used as a default value)

✔ Representation difference

Declaration	Size	Index range	Use-case
`vector<int> v(n);`	`n`	`0 ... n-1`	normal C++ indexing
`vector<int> v(n+1);`	`n+1`	`0 ... n`	1-based indexing

✔ Diagram

Normal (`vector<int> v(n);`)

makefile Copy code

```
Index:   0   1   2   ...  n-1
Value:  [ ] [ ] [ ]     [   ]
```

1-based (`vector<int> prefixmin(n+1);`)

yaml Copy code

```
Index:   0   1   2   ...   n
Value:  [X] [ ] [ ]       [ ]
        |
        unused
```

✔ Conclusion

Using `n+1` gives you **clean 1-based indexing**, which matches many CP problems and avoids writing `i-1` everywhere.

If you want, I can also show how to fill `prefixmin` and `suffixmin`.