Arrays , Char Array/Strings, 2D array

Pointers - — Sorting

— Pointer Type casting

— char array

- 2D array

Programming Fundamentals

- Data — variables
- Instructions — different instructions
  - Control statements
    - Conditionals — if, else
    - Loops — for, while

$\rightarrow$ Solved Patterns

Arrays : collection of same data type.

Sorting an array: arrangement of elements in an order ⟹ by default ascending sort.

4 2 1 5 3

to sort this means ⟶ transform it to : 1 2 3 4 5.

✓ – Bubble Sort : we compare adjacent elements and swap if they are unordered.

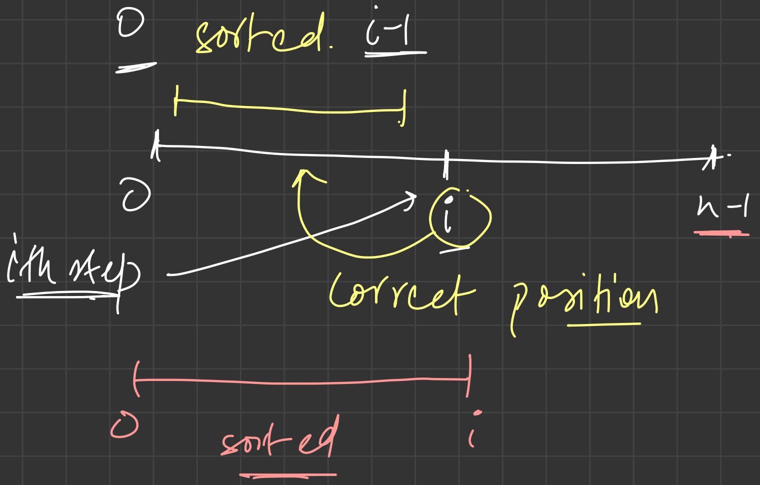
✓ – Selection Sort : we picked min. element and placed it at its correct index. And repeated this step (n-1) times.

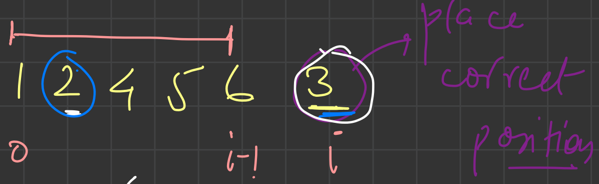
✓ – Insertion Sort

we considered one element at a time and made sure it is placed in sorted order wrt. all its prev elements.

(n-1) times ?? → if i sort n-1 elements, then last element would itself be sorted.

Optimized Bubble sort-





```
j = i-1        k = 3
while (a[j] > k) {
    a[j+1] = a[j];
}
a[j+1] = k;
```

friday : QPS

Functions
- how does it work
- anatomy
- call stack
- scope

Swapping two variable

Pointers

≠ just variables/data types
which addresses of another
variable.

{ initialization, execution, deletion }

- address of

dereference
declaration } → *

- type-casting : same as normal data types
pointers can be type-casted:

- all types of pointer store address in

Hexadecimal format. ⇒ typecasting has no effect on address.

— but pointer-type is imp. for dereferencing.

⇒ type-casting plays a role in dereferencing an address.

```
int x = 65;

int *xptr = &x;

cout << *xptr;   65

cout << *((char *)xptr);   A
```

→ void* ⇒ cannot be dereferenced.

↳ it is helpful to store address of any type, and is later used by dereferencing it.

⇒ Arrays and pointers are intricately linked.

→ arrays are passed as a pointer to function. (size is needed)

→ Pointer arithmetic

int * ptr = 100;

~~101~~

cout << ptr + 1;

↳ 104

ptr + 1 * (sizeof(type of ptr))

cout << ptr + 2; → ~~102~~

↳ 108

~~multiplication~~ & ~~division~~

subtraction : with a const.

cout << ptr - 2;

ptr - 2 × (4 bytes);

100 - 8 = 92

{ difference of two pointers gives the no. of buckets between them; }

int * ptr2 = 120;

cout << ptr2 - ptr; →

$$\frac{ptr2 - ptr}{sizeof(data\ type)}$$

$$\frac{120-100}{4} = \frac{20}{4} = 5$$

Character Arrays (strings)

→ char[] behaves differently with cin/cout than normal arrays.

→ it represents a string (sequence of chars). , and when cout it prints chars until \0 found.

→ it is important to place \0 correctly in char[] when needed.

→ there is strings (which is built upon char[]) and gives some additional features, and is easy-to-use.

→ null should always be placed to the next index of the last character. → represent : '\0' → ascii = 0

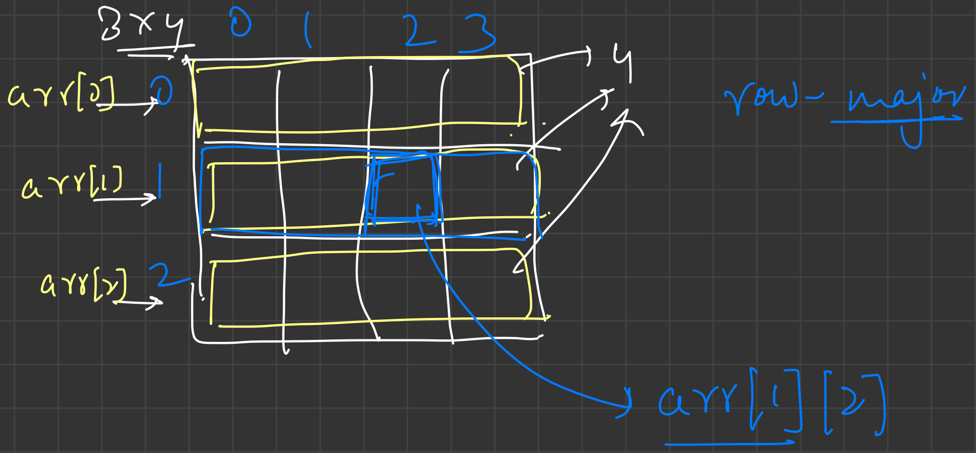
→ char str[]; → cout << str; ↙ address of the first bucket

↳ instead of printing the address, it goes byte-by-byte and prints the char, until an '\0' is found.

2D arrays

↳ array of arrays

int arr[3][4];

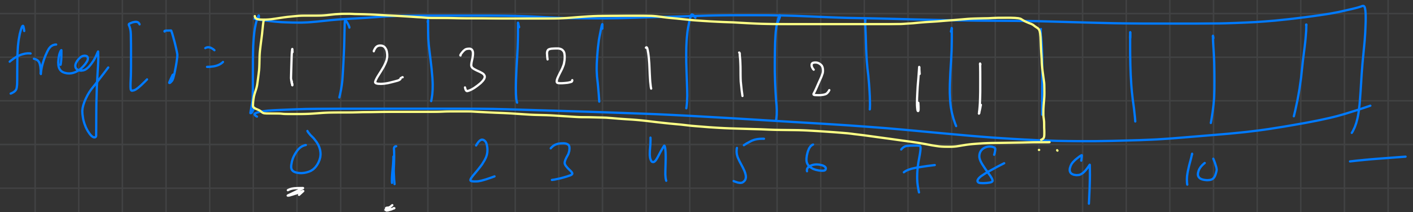


Frequency array :

→ is an array where the value at a certain index, i, represents the frequency of the value i.

freq[i] → the frequency of the value i.

arr[] = {1, 2, 2, 3, 1, 4, 5, 8, 0, 6, 6, 7, 2, 3, 4}.



freq[arr[i]]++;

find the frequency of 3 in the arr[].

freq[3]

→ negative values ✓ → shifting
→ very large range of values ✗ } demerits
→ values > 10^6 ✓ → shifting

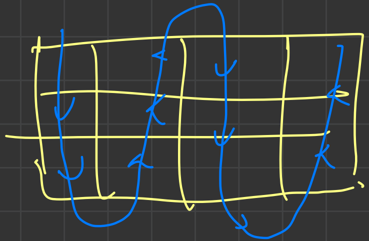
imp.

Pattern - C
Array & functions

Q. Given N strings, find the largest

Q. Find the max row/col in 2D array
largest sum

Q. Wave print



Q. Given list of N words, and word S, check whether S exists in the list.

Q. Reverse a number / Reverse an array / Rotate an array

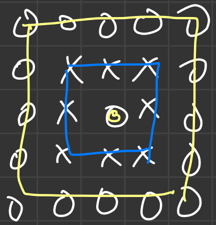
shift all elements one place right.

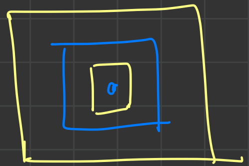
$\boxed{\text{Break till : 9:45}}$

Q. Read N strings, and sort them lexicographically.

Q. Given two sorted arrays, merge them into one array which remains sorted

Q. given n, m: n = 5, m = 5





alternate rectangles of O's and X's