Lecture 9

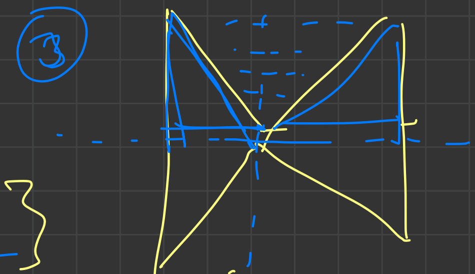
Instructions

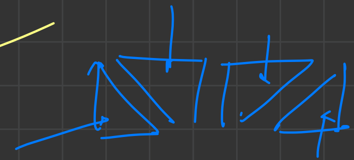
- input
- output
- ops. → calculative
- assign
- condition → { if-else }
- repetition → loop { for/while }

Data

Variables

Arrays





Q.

words ?? 10

chars ?!

lines ?! 3

$

MacbookBlocks → 26

Today is Saturday,

14th Jan 2023,

and% it also weekend $

words ?!

11

1 1 1 2 2 3 4 5 5 6 8

arr[11]

3

1 9 2

step ① take array as input.

② take M as input and use a loop to get the M values.

```
int m;
cin>>m

int n;
for(i=0 ; i<m; i++) {
    cin>> x;
    int freq=0;
    for(j=0 ; j<n ; j++) {
        if( arr[j]==x ) { freq++} }
    }
    cout<< x<<' '<< freq <<endl;
}
```

① array input.

② input M

③ loop M times, taking x as input

value whose frequency is to be found.

{
④ freq = 0.

⑤ loop over the array and increment if arr[j] equals x.
}

⑥ output freq.

{ int freq[11] = {0}; }

array which has name → freq
size → 11
type → int
indexes → 0 to 10
initialized with 0 at each index

assumptions
all values in the array are from 0 to 10.

freq =

0	3	2	1	1	1	1	0	1	0	0
0	1	2	3	4	5	6	7	8	9	10

index i stores freq of value i

N = 10

array →

1	1	1	2	2	3	4	5	6	8
0	1	2	3	4	5	6	7	8	9

execute

```
for(i=0; i<n; i++){
    freq[arr[i]]++;
}
```

dry run

i=0; arr[i] = 1 → freq[1] ++

i=1; arr[i] = 1 → freq[1] ++

i=2; arr[2] = 1 → freq[1]++

$i=3 \rightarrow arr[3] \rightarrow 2 : freq[2]++$

$i=4 \rightarrow freq[arr[4]]++ = freq[2]++ =$

$i=5 \rightarrow freq[arr[5]]++ = freq[3]++$

$i=6 \rightarrow freq[arr[6]]++ = freq[4]++$

$i=7 \rightarrow freq[arr[7]]++ = freq[5]++$

$i=8 \rightarrow freq[arr[8]]++ = freq[6]++$

$i=9 \rightarrow freq[arr[9]]++ = freq[8]++$

(0 — 10^5) freq array

Limitations

1. range must be known
2. range must be $<= 10^5$
3. -ve value, floating values, etc. are not supported.

Sorting

↳ to arrange the values in some order
⇓
increasing
non-decreasing (by default)

why sort??

↳ real-life application.

Contacts
search
phone book
sorted

300,000 words

Suppose dictionary didn't had words sorted (lexicographical)

Searching efficient ⟵ sorting

⇒ Sorting is important / relevant.

How ?!

6

arr[] = { 4 2 1 3 6 5 } ⇒ sort → increasing / non-decreasing order

transform

{ 1 2 3 4 5 6 }

sorted arr

① Bubble Sort

4 2 1 3 6 5

left - to - right ⇒ we fix the elements adjacently

arr	2	1	3	4	5	6
indx	0	1	2	3	4	5

i, i+1

i = 0 → arr[0] = 4 arr[0+1] = 2 ⇒ swap

i = 1 → arr[1] = 4 arr[1+1] = arr[2] = 1 → 4 > 1 → swap

i = 2 → arr[2] = 4 arr[3] = 3 → 4 > 3 → swap

i = 3 → arr[3] = 4 arr[4] = 6 → 4 < 6 → no swap

i = 4 → arr[4] = 6 arr[5] = 5 → 6 > 5 → swap

~~i = 5 → arr[5] = 6 arr[6] → Segmentation Fault~~

⇒ i = 0 → n − 2 adjacent elements compare

0 4

0 1 2 3 4 5

1 2 3 4 5 6 is sorted??

i = 0 → 2 > 1 → swap

i = 1 → 2 < 3 → no swap

i = 2 → 3 < 4 → no swap

i = 3 → 4 < 5 → no swap

i = 4 → 5 < 6 → no swap

~~i = 5~~

(2) times iterating the array sorts it

↓

swapping incorrect adjacent values

⇒ sorted

Is this process/algorithm correct?? → Yes

Do we know how many times to iterate the array??

					How many times??
1	2	3	4	→	0 ✓
2	1	3	4	→	1 ✓
3	2	1	4	→	2

4213 ①
2413
2143
2134
1234 ②

4 2 1 3 → 2 ✓
4 3 2 1 → 3
3 1 4 2 → 2

4321
3421
3241 ①
3214

2314
2134 ②

1234 ③

n-1 times is enough ?!

Swap

4 1 5 3 2

1 4 5 3 2

1 4 5 3 2

(1)

1 4 3 5 2

1 4 3 2 $\boxed{5}$

end → settle

disturb

(2)

1 4 3 2 5

1 3 4 2 5

1 3 2 $\boxed{4\ 5}$

(3)

1 3 2 4 5

1 3 2 4 5

1 2 $\boxed{3\ 4\ 5}$

In an array of size N, if N-1 elements are at their sorted position

(Nth)?! → sorted position ??

n-1 times

taking the max element to its correct position

4 2 1 3 6 5 max. elements traverse in a bubble to its last position

② Selection Sort

→ iterate the array, pick the smallest

→ swap it to its correct position

indexOfminValue

4	2	1	3	6	5
0	1	2	3	4	5

m = 0 →

arr[m] > arr[i]

m = 2

0	1	2	3	4	5
4	2	1	3	6	5

left to right

i = 0 : arr[0] arr[m]

4 == 4

i = 1 : arr[1] arr[m] = 4 ⇒ $\boxed{m = i}$

2 <

i = 2 : arr[2] arr[m] ⇒ m = i

1 < 2

Q. Find the index of smallest and largest value in the array ?!

min

0 4 2 1 3 6 5

Swap (1, 4)

1 1 2 4 3 6 5

Swap (2, 2) → no swap

2 1 2 4 3 6 5

Swap (4, 3)

3 1 2 3 4 6 5

Swap (4, 4)

4 $\boxed{1\ 2\ 3\ 4}$ 6 5

swap (6, 5)

5 $\boxed{1\ 2\ 3\ 4\ 5}$ 6

n-1 elements are sorted $\longrightarrow$ array is sorted

observation

n-1 times iterate

n=6

i → 0, 1, 2, 3, 4 $\boxed{\cancel{5}}$

place the minimum at index = i.

swap

Selection sort.

```
for( i=0 ; i<n-1 ; i++) {
    min_indx = i ;
    for( int j= i ; j<n ; j++) {
        if(arr[j] < arr[min_indx] ){
            minIndx = j;
        }
    }

    if( min_indx != i ){
        swap( arr[min_indx], arr[i] );
    }
}
```

Insertion Sort

4 1 2 3 6 5

1	2	4	5	3

array ← i

$k = 3$

place 3 at its correct position

1	2	4	5	3
0	1	2	3	4

sorted

$i : 3 \to 0$

$arr[i] > k \Rightarrow arr[i+1] = arr[i]$

1 2 3 4 5 → k = 3

5 > k →

4 > k →

2 < k → its correct place for k

k = 1

i → 0 1 2 3

2 2 3 4

k is smallest

put at

i = 2 : arr[i] = 4 > k ⇒ arr[i+1] = arr[i]

i = 1 : arr[1] = 3 > 1 ⇒ arr[2] = arr[1] index 0 | i = 0 : arr[0] = 2 > 1 ⇒ arr[1] = arr[0] }

i = -1

2 2 3 4

0 1 2 3

k = 1

2 3 4 1

1 2 3 4

arr[i] > k this will keep going left.

{ as soon as arr[i] <= k → stop and put k at arr[i+1] }

arr[i] < k

k = 3

1 2 4 4 5

1 2 4 5 3

1 2 3 4 5

1 2 4 5 3

Insertion Sort

4 5 1 3 6 2

n-1

{ size=1 → 4 → sorted !!! }

size=2 → 4 5 → sorted

size=3 → 1 4 5 3 6 2 → sorted

size=4 : 1 3 4 5 6 2

size=5 : 1 3 4 5 6 2

size=6 : 1 2 3 4 5 6