Lecture 52

Heap Hard

* **pair in STL :**

Two Element : int, char

①
```
class Node {
    int a;
    char b;
}
```

(int, char)
↑ ↑
first second

②
```
pair <int, char> p;
p.first = 10;
p.second = 'd';
```

②
```
pair <int, char> p;
p = make_pair (10, 'd');
```

③ for 3 Element :

```
pair <int, int, int> p;
```
↳ Wrong format.

we use nested pair :

```
pair <int, pair <int, int>> p;
```

(4) for 4 element :

```
pair <int, pair <int, pair <int, int>>> p;
```

(5) for 3 element :

```
pair < pair <int , int> , int > p;
```

↓

```
p.first.first = 10;
p.first.second = 20
```

```
p.second = 30;
```

```
p = make_pair (make_pair (10, 20), 30);
```

Code :

```
#include <iostream>
#include <vector>
using namespace std;

int main(){

    pair <int, int> p;
      p.first  = 10;
      p.second = 20;

//    p = make_pair (10, 20);
cout <<p.first << " " <<p.second <<" ";
```

```
pair<int, pair<int, int>> q;

    q.first = 1;
    q.second.first = 10;
    q.second.second = 20;

cout << q.first <<endl;
cout <<q.second.first <<endl;
cout << q.second.second <<endl;

return 0;
}
```

→

```
int main() {
    pair<int, int> p;
    p= make_pair (10, 20);

    pair<int, int> p2;
      p2 = p;                    // Copying.

  cout << p2.first << " " <<p2.second;
return 0;
}
```

Example

Sort on the bases of first : ascending

10 , 20	
8 , 7	sort according to the first :
4 , 9	
4 , 7	
9 , 15	

```
int main () {
  Vector <pair <int, int>> v;

    v.push_back (make_pair (10, 20));
    v.push_back (make_pair (8, 7));
    v.push_back (make_pair (4, 9));
    v.push_back (make_pair (4, 7));
    v.push_back (make_pair (9, 15));

  sort ( v.begin (), v.end ());

  for (int i=0;  i<5; i++)
    cout << v[i].first << " " << v[i].second << endl;

  return 0;
}
```

For descending :

```
sort (v.rbegin (),  v.rend ());
```

* Ascending order on the basis of second :

4, 8		11, 7	11, 7
11, 7	Ascending on	4, 8	3, 8
12, 13	⟶	3, 8	4, 8
3, 8	basis of second.	5, 9	5, 9
5, 9		12, 13	12, 13

↑ If second is same then check for first

```
sort(v.begin(), v.end(), sortbysec);
```

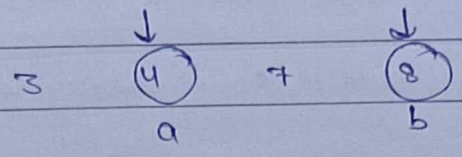
→
```
bool sortbysec(int a, int b){
    return a > b;
}
```

→
```
bool sortbysec(pair<int,int>a, pair<int,int>b){
    return a.second > b.second;   // ascending
                                  // descending
}
```

↴

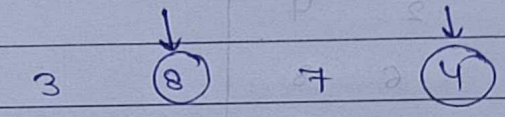
It works on the basis of second.

ascending order :



a < b (return 1)

No change.



8 < 4 (return 0)

so, print 4 first, then 8.

* If second is same then check in first :

10	7
11	8
9	7

a.second < b.second

(10, 7) (11, 8)

7 < 8 return 1

(10, 7) (9, 7)

7 < 7 return 1

(9, 7), (10, 7), (11, 8).

```
bool sortbysec (pair<int, int> a, pair<int, int> b) {
    return a.second < b.second || (a.second == b.second &&
                                   a.first < b.first);
}
```

All Concepts are defined in STL in sort function and vector.

* Merge K sorted Array (Interview Bit)

1	2	4
2	6	8
3	4	9
4	11	12

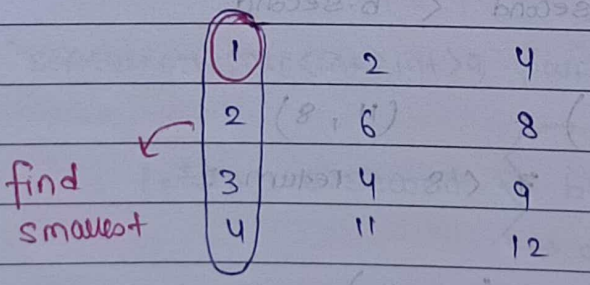
Method 1 :

1 2 4 2 6 8 3 4 9 4 11 12

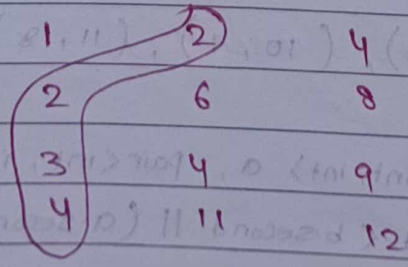
↳ sort these all

Time Complexity : $K * n \log(K * n)$.

Method 2 :



1



1, 2

for one element we do k comparison.

Total element = k * N

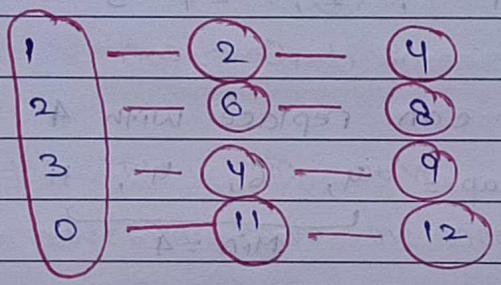
Time Complexity = k * k * N

= $k^2 * N$.

Method 3:

We want every time Min element.

So, we use Min Heap.



Heap : 1 2 3 0

Min = 0

Ans : 0,

0 can replace with 11

Heap : 1 2 3 11

Min = 1

Ans : 0 , 1

1 can replace with 2

Heap : 2 2 3 11

Min = 2

Ans = 0, 1, 2

2 can replace with 4

Heap = 4 2 3 11

Min = 2

Ans = 0, 1, 2, 2

2 can replace with 6

Heap = 4 6 3 11

Min = 3

Ans = 0, 1, 2, 2, 3

3 can replace with 4

Heap = 4, 6, 4, 11

Min = 4

By this :

Ans = 0, 1, 2, 2, 3, 4, 4, 6, 8, 9, 11, 12

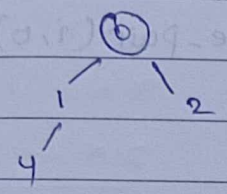
For 1 element : log k

Total element = n k

Time Complexity = n · k · log k.

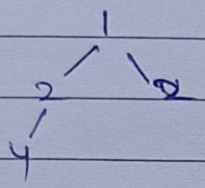
Example :

1 ⟶ 2 4
2 3 6
0 ⟶ 2 5
4 5 10



Heap : 1 2 (0) 4 (Min = 0)
2

Ans : 0



Heap = (1) 2 2 4 (Min = 1)
2

Ans : 0, 1

Heap = (2) 2 2 4 (Min = 2)

Ans = 0, 1, 2

By this we can solve,
we can give the next element also in the pair
(0, 2)
↑ on heap ← if 0 is go to ans, then replaced with 2.

Code :

```
vector<int> solution :: solve(vector<vector<int>> &A){
    vector<int> ans;
    int row = A.size();
    int col = A[0].size();

    // Min Heap
    /// pair<int, pair<int, int>> p;
priority_queue<pair<int, pair<int, int>>, vector<pair<int,
    pair<int,int>>>, greater<pair<int, pair<int ,int>>>>
                                        minheap;
```

```
// insert first column into minheap
for (int i=0; i<row; i++)
minheap.push(make_pair(A[i][0], make_pair(i,0)));

pair<int, pair<int,int>> p;

while(!minheap.empty()){
// Get minimum element, top
    p=minheap.top();
// Insert the value in answer vector
    ans.push_back(p.first);
    row = p.second.first;
    col = p.second.second;
    minheap.pop();
// Insert the next element of that row, first check
// Whether that row is exhausted or not.
    if(col<A[0].size()-1)
    minheap.push(make_pair(A[row][col+1],
                           make_pair(row, col+1)));
}

return ans;

}
```

* **Huffman coding:**

String S: A B B C D A B D - - - - -

50 character

A = 20

B = 4

C = 15

D = 19

E = 2

We send the ASCII Code of character.

A = 65

01000001

8 bit

50 character

↳ 50 × 8 bit = 400 bit.

We have to show only 5 character

हमलोग कुल 5 से नाम रख देंगे।

8 × 5 = 40 3 × 5 = 15 = 55

01000001	A	→	000
	B	→	001
	C	→	010
	D	→	011
	E	→	100

50 character

↳ 50 × 3 = 150 bits.

table : 15 + 40 = 55

Total = 150 + 55 = 205 bits.

Minimized.

A	= 20	0	→ 20 × 1 = 20
B	= 4	1	→ 4 × 1 = 4
C	= 15	00	→ 15 × 2 = 30
D	= 9	01	→ 9 × 2 = 18
E	= 2	10	→ 2 × 2 = 4
			76

We can't use this, this some gives error

Table = 40 + 8 = 48

76 + 48

= 124

↳ Minimized.

can we can minimize more ?

Yes

↳ How

of frequency is high, assign it with min-m no. of bit.

A	= 20	→ 0	=> 20
BC	= 15	→ 1	=> 15
D	= 9	→ 00	=> 18
B	= 4	→ 01	=> 8
E	= 2	→ 10	→ 4
			65

We can't use this it create some error

Table: 40 + 8 = 48

65 + 48

= 113

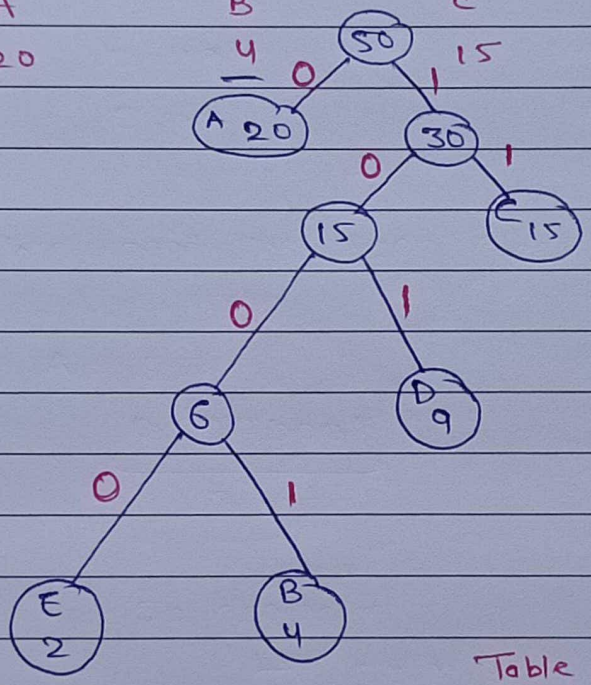
↳ minimum.

जिसका frequency सबसे ज्यादा उसको हम सबसे कम bit से represent करेंगे।

Freq ↑ = less bit

Freq ↓ = more bit.

A	B	C	D	E
20	4	15	9	2



A = 0 ⇒ 20

C = 11 ⇒ 30

B = 1001 ⇒ 16

D = 101 ⇒ 27

E = 1000 ⇒ 8

101

Table size = 40 + 14

= 54

Total = 155

Min Heap : Insert all element.

2 element pop