Duke

← D_1

D_2

D_3

D_4 → 12hrs

Netflix ← D_5 → 4hrs

D_2 ✓

D_3 ✓

U_1

U_2

UCSD

U_1

U_2

U_3 → Adv data structures

U_4 → Interview

U_5 → Capstone

```
class Car {
  pri int cc;
  pri Engine engine;
  pri String make;

     public int getCC(){}
     public Engine getEngine() {}
     public String getMake() {}
}

Car c = new Car(...);
c.getCC();
```

```
public class VigCipher {
      . . .   alp
      . . .   cc[ ]

      public String encrypt(..){ }
        '       '     decrypt(..){ }
}
```

Quake 1

EQ California

EarthQuake 1 California, USA

fbp ("any" , "1")

Jakarta

3 closest EQs

$$[\overset{10}{\cancel{EQ_1}}, \overset{17}{EQ_2}, \overset{3}{\cancel{EQ_3}}, \overset{9}{\cancel{EQ_4}}, \overset{15}{EQ_5}]$$

$$[EQ_3, EQ_4, EQ_1]$$

```
AL<QE> filterMag(AL<QE> data, double minMag){
    AL<QE> out = new AL<>();
    for (QE qe : data){
        if( qe.mag() > minMag ){
            out.add(qe);
        }
    }
    return out;
}
```

```
AL<QE> filter (AL<QE> data, Filter f ) {
    AE<QE> out = new AL<>();
    for (QE qe : data){
        if(f.satisfies(qe)){
            out.add(qe);
        }
    }
    return out;
}
```

```
public interface Filter {
    public boolean satisfies (QE qe);
}
```

```
public class MinMagFilter implement
                              Filter {
    private double minMag
    public MinMagFilter(double val ){
        minMag : val;
    }
    public boolean satisfies(QE qe){
        return qe.mag() > minMag;
    }
}
```

```
data = {
    EQ1(mag=3.7, depth= 150)
    EQ2(mag= 2.4 , depth= 550)
    EQ3(mag= 5.7, depth = 50)
    EQ4(mag: 9.0, depth=750)
    EQ5(mag=3.9, depth:250)
};
```

```
Filter  f1 = new MinMagFilter( 4 );

AL<QE> filtredData1 = filter(data, f1);
                 ↓
        {EQ3 , EQ4}

Filter f2= new MinMagFilter(6.0);
AL<QE> l2 = filter(data, f2);
           ↓
        {EQ4}
```

```
public class MinDepthFilter implements
                                  Filter{
           double
    private minDepth;
    public MDF(double val){
         minDepth = val;
    }
    public boolean satifies(QE qe){
         return qe.depth() >minDepth;
    }
}
```

```
Filter f3 = new MDF (200);

AL<QE> l3 = filter (data, f3);
          ↓
   { EQ2, EQ4, EQ5}

Filter f4 = ______________
AL<QE> l4= filter(data, f4); → >200
        ↓                     &<600
   { EQ2 , EQ4}                depth
```

```
public class MinMaxDF implements Filter {
    private double minDepth;
    private double maxDepth;

    // Constructor

    public boolean satisfies(QE qe){
        return qe.dep() > minDepth
              && qe.dep() < maxDepth;
    }
}
```

```
Filter f4 = new MinMaxDF(200,600);
AL<QE> l4 = filter(data, f4);

Filter f5 = new MinDF(200);
Filter f6 = new MaxDF(600);
AL<QE> l5 = filter(data, f5);
AL<QE> l6 = filter(l5, f6);
```

```
public class LocF implements F{
    pri double radius;
    pri Location loc;

    // Cons

    public boolean satisfies(QE qe){
        if(qe.loc().distTo(loc)
                     < radius){
            return true;
        } else {
            return false;
        }
    }

}
```

```
Filter f7 = new LocF(J, 1000);
AL<QE> l7 = filter(data, f7);
```

```
public interface Animal {
    public void makeSound();
}

public class Dog implements Animal {
    public void makeSound() {
        print("Woof");
    }
}

public class Cat implements Animal {
    public void makeSound() {
        print("Meow");
    }
}
```

```
1 Animal a1 = new Animal(); X
2 a1.makeSound(); X
3 Animal a2 = new Dog(); ✓
4 a2.mS(); // Woof
5 Dog a3 = new Dog(); ✓
6 a3.mS(); // Woof
```

```
7 Dog a4 = new Animal(); X
8 a4.makeSound(); X
9 Animal a5 = new Cat(); ✓
10 a5.makeSound(); // Meow
11 a3 = a5; X    Dog a3 = new Cat(). X
12 a3.mS();
```

```
Dog[] dogs = new Dog[3];
Animal[] animals
        = new Animal[3];
animals[0] = a2;
animals[1] = a3;
animals[2] = a5;
```

```
AL<QE> l1 = filter(data, f1);
AL<QE> l2 = filter(l1, f2);
AL<QE> l3 = filter(l2, f3);
```

f1 → > 5.0

f2 → > 3000 & < 6000

f3 → < 1000km from California

↓

(> 5.0
& 3000 < dep < 6000
& < 1000 km from California)

Hello: --- 0.8745

Hello: --- 12.5731

[X-DSpam-Confidence:]0.213

20

a = [1, 2, 3]

b = [4, 5]

a.append(b)	for x in b: a.append(x)
a ↓ $[\underset{0}{1}, \underset{1}{2}, \underset{2}{3}, \underbrace{[4,5]}_{3}]$	[1,2,3,4,5]

aa bb cc ad

lst → [ ]

line → "aa bb cc aa"

words → $\underbrace{[\text{"aa"}, \text{"bb"}, \text{"cc"}, \text{"aa"}]}$

mag>5
200< depth <300

```
public class MagDepthFilter
          implements Filter{
    minMag;
    minDep;
    maxDep;

    //Cons

    satisfies(QE qe) {
        ret qe.mag() >=minMag
            && qe.dep() >=minDep
            && qe.dep() <=maxDep;
    }
}
Filter f= new MagDepthFilter(5,200,300);
AL<QE> r = filter(list, f);
      ↓
```

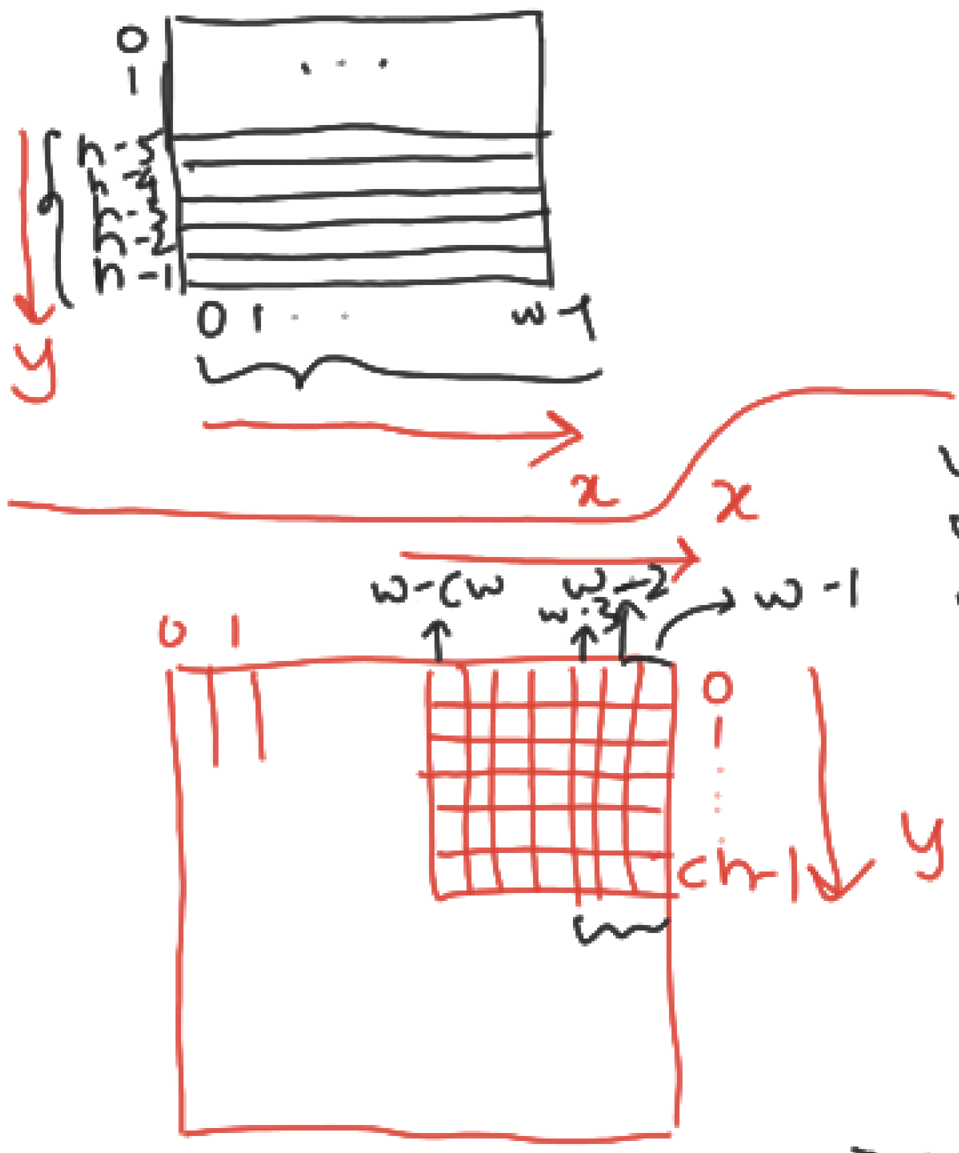
mag>4
150 < dep < 600
100km radius of LA

```
public MagDepLocFilter impF{
     minMag;
     minDep;
     maxDep;
     dist;
     loc;

     //Cons
   satisfies() {
       //dep, mag, loc
   }
}
```

```
}
```

```
list = [
    "2002 California". ✓
    "1997 Japan 200150"
    "2007 Jakarta" ✓
]
filter ("200", "start")
```

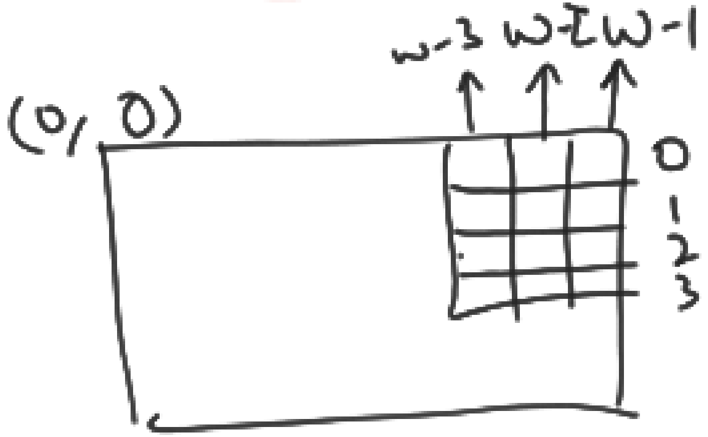
```
f1 = MagFil (3.0, 7.0)
f2 = LocFil (tokyo, 1000...);
f3 = PhrFil ("any", "hello");

r1 = filter (list, f1)    }
r2 = filter (r1, f2);     } →  Filter f = new MatchAllFilter();
r3 = filter (r2, f3);     }    f.addFilter(f1);
                               f.addFilter(f2);
                               f.addFilter(f3);

                               r = filter(list, f);
```



w - cw

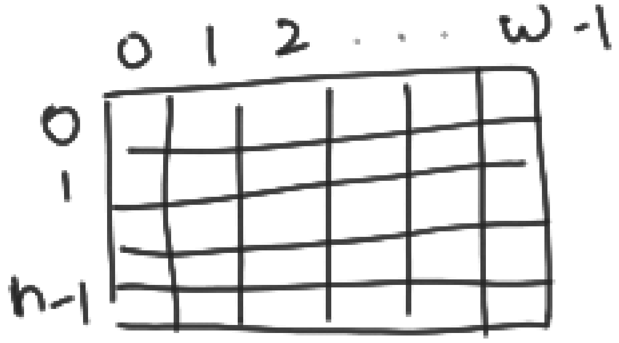
w - cw + 1

w - cw - 1



x → w-3 to w-1

x → w-cw to w-1



$\overbrace{0 \quad 1 \; 2 \quad 3 \; 4 \quad 5}$

```
arr= {
      {A, B, C},      → arr[0]
      {D, E, F}       → arr[1][2]
                      → arr[1][1]
     }
```

numRows = arr.length

numCols = arr[0].length

A
B
C
D
E
F

A
D
B
E
C
F

```
for(r=0; r<nR; r++){
    for(c=0; c<nC; c++){
        print(arr[r][c]);
    }
}
```

A
B
C
D
E
F

```
for(c=0; c<nC; c++){
    for(r=0; r<nR; r++){
        print(arr[r](c));
    }
}
```

3	9	11	17	22	31

sorted	not seen yet

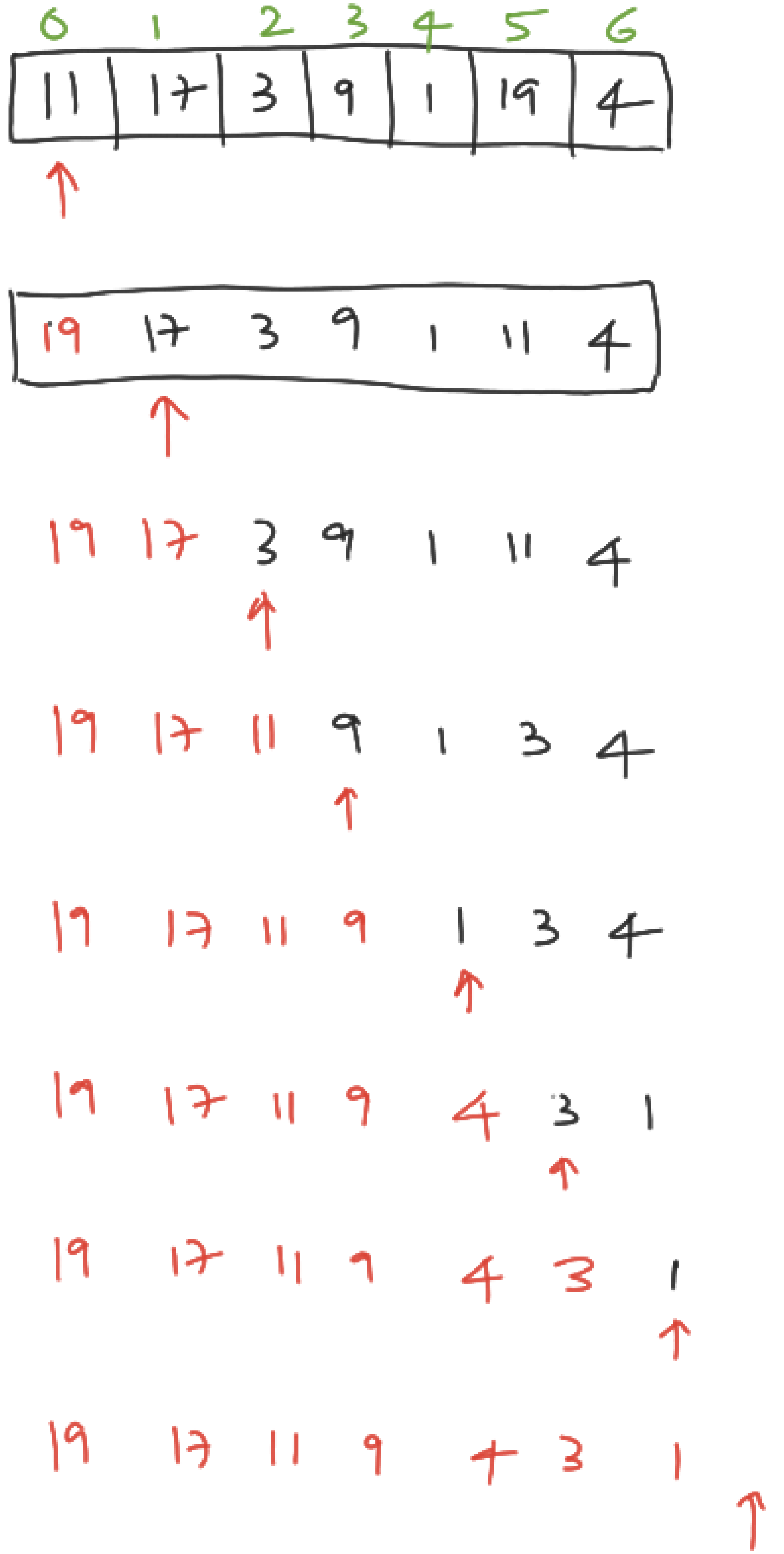
0	1	2	3	4	5
35	17	9	3	11	1

↑
i=2
max=4

in	- - -	y	- - -	x	- - -
		i		j	

```
copy=in.get(i)
✓ in.set(i, in.get(j))
  in.set(j, copy)

swap at i & j
```

```
x ↑ elemI = in.get(i)
y ↑ elemJ = in.get(j)
    in.set(i, elemJ)
    in.set(j, elemI)
```



0	1	2	3	4	5	6
11	17	3	9	1	19	4

19 17 3 9 1 11 4

19 17 3 9 1 11 4

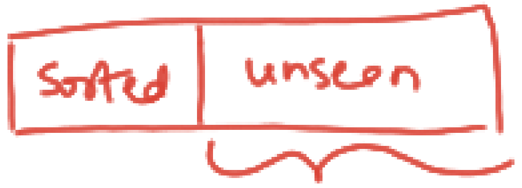
19 17 11 9 1 3 4

19 17 11 9 1 3 4

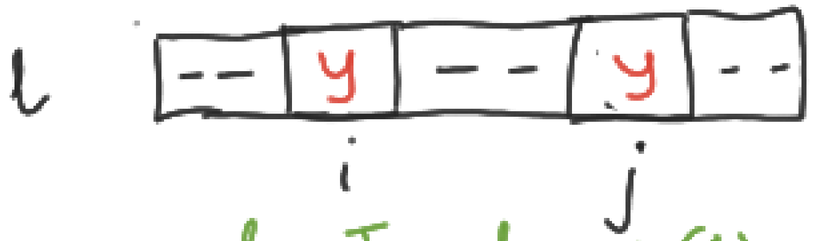
19 17 11 9 4 3 1

19 17 11 9 4 3 1

19 17 11 9 4 3 1



Sorted	unseen



```
elemI = l.get(i)
l.set(i, l.get(j))
l.set(j, elemI)
```

```
elemI = l.get(i)
elemJ = l.get(j)
l.set(i, elemJ)
l.set(j, elemI)
```

11 17 3 9 1 19 4

first pass

17 11 3 9 1 19 4

17 11 9 3 1 19 4

17 11 9 3 1 19 4

17 11 9 3 19 1 4

17 11 9 3 19 4 | 1

second pass

17 11 9 3 19 4 1

17 11 9 19 3 4 | 1

17 11 9 19 4 | 3 1

3rd

17 11 19 9 | 4 3 1

4th

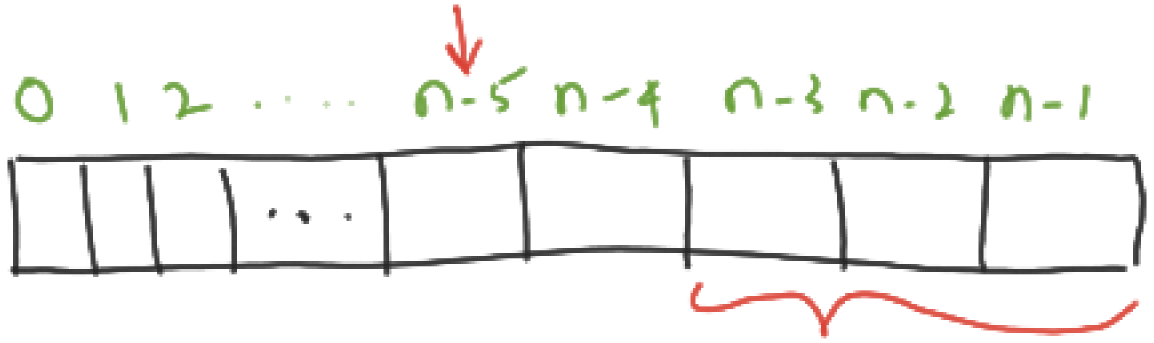
17 19 11 | 9 4 3 1

5th

19 17 | 11 9 4 3 1

6th

19 | 17 11 9 4 3 1



k elements sorted

$\Rightarrow$ last two elements compared are

i

↓

$n-k-2$, $n-k-1$

3 15 7 2 9 1 8

0 1 2 3 4 5 6

first pass

3 7 15 2 9 1 8

3 7 2 15 9 1 8

3 7 2 9 15 1 8

3 7 2 9 1 15 8

3 7 2 9 1 8 | 15

second pass

3 7 2 9 1 8 | 15

3 2 7 9 1 8 | 15

3 2 7 1 9 8 | 15

3 2 7 1 8 | 9 15

third pass

3 2 7 1 8 | 9 15

2 3 7 1 8 | 9 15

2 3 1 7 | 8 9 15

4th pass

2 3 1 7 | 8 9 15

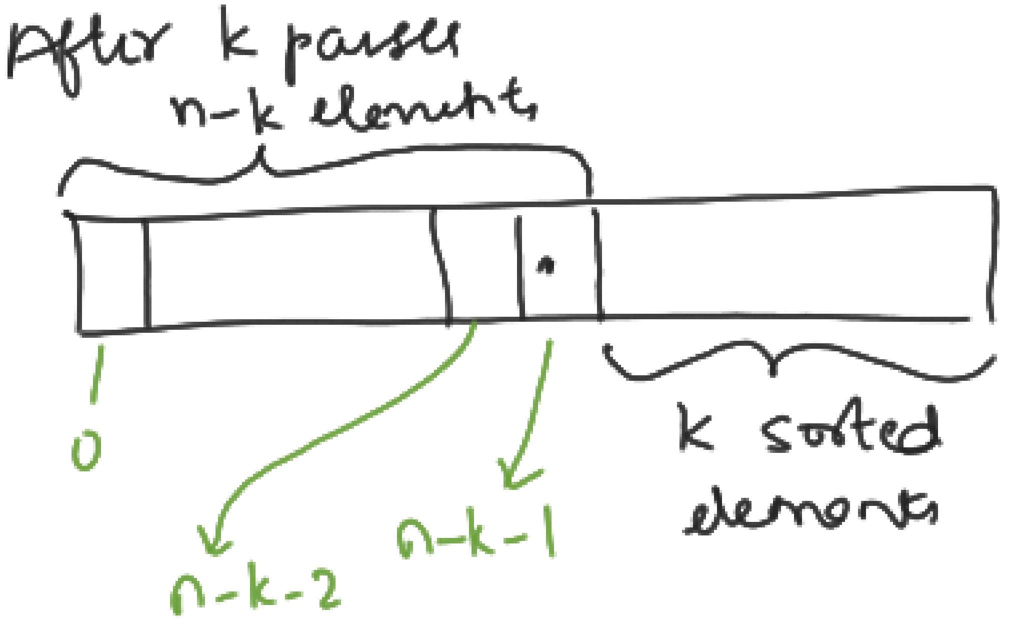
2 1 3 | 7 8 9 15

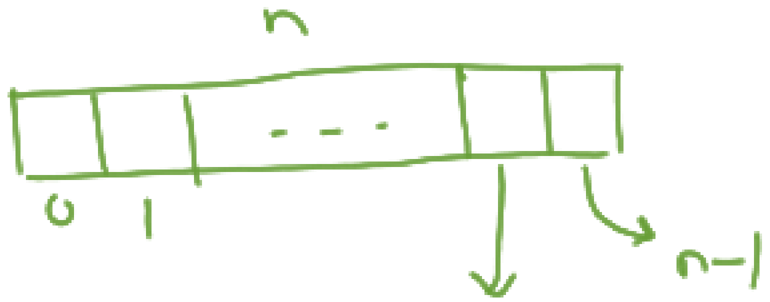
5th pass

1 2 | 3 7 8 9 15

6th pass

1 | 2 3 7 8 9 15

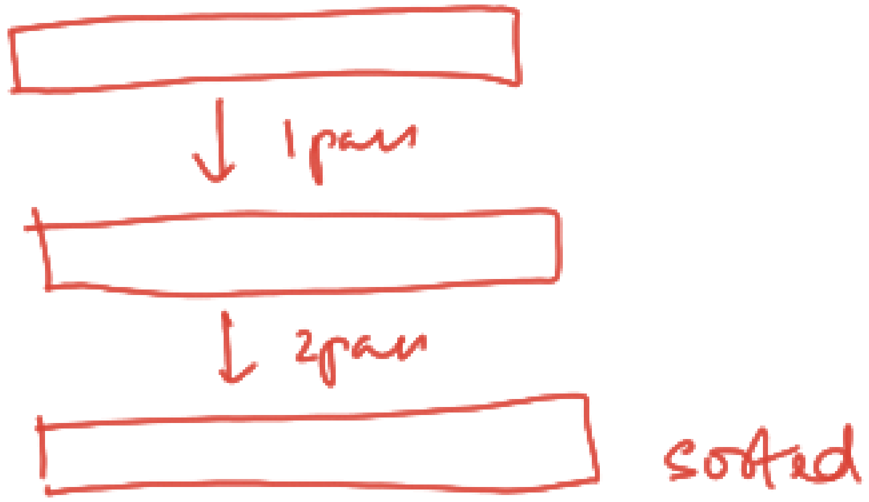




i goes from 0 to n-k-2

compare a[i] & a[i+1]

& swap if they are in wrong order



'n' times

```
for (i=0; i<n; i++)
for (i=1; i<=n; i++)
```

n-1 times

```
for (i=0; i<n-1; i++)
for (i=1; i<=n-1; i++)
```

n-3 times

```
for (i=0; i < n-3; i++)
for (i=1; i<=n-3; i++)
```

k times

```
for (i=0; i<k; i++)
for (i=1; i<=k; i++)
```