Lecture13

Sunday, 16 February 2020 2:38 PM

③ Inheritence ~ Reusability . ↪ (10.1), (10.2)
④ Polymorphism.

```
class P {
    int d=10;
    int d1=11;
    public void fun(){
        Syso("Hello Fun of P");
    }
    public void fun1(){
        Syso("  of P");
    }
}
```

```
class C extends P {
    int d=20;
    int d2=21;
    public void fun(){
        Syso("Hello Fun of C");
    }
    public void fun2(){
        Syso("Hello Fun2 of C");
    }
}
```


```
psvm(String[] args){
① P obj=new P();
  Syso(obj.d); //10
  Syso(obj.d1); //11
  Syso(obj.fun1();
  obj.fun();
}
```

Polymorphism
- Dynamic → Runtime
- Static → Compile.

Compiler: P obj=new C(); → JVM → Memory.

```
Syso(obj.d1);
Syso(obj.d2);
obj.fun2();
```

```
obj.fun1();
obj.fun();
Syso(obj.d); //10.
```

```
C obj=new C();
Syso(obj.d);
```

```
Syso(obj.fun());
obj.fun2();
Syso(obj.d1);
obj.fun1();
```

```
public void add(String a, String b){
    Syso(a+" "+b");
}
```

① 3

```
public void add(int a, int b){
    Syso(a+" "+b);
}
```

② 5

```
public int add(int a, int b){
    Syso(a+" "+b);
    return a+b;
}
```

③ ✗

add(2,3);
add("Hello","world");
add(3); ✗

add(2,3);

Abstract functions.

```
class Animal {
    can fly(){
        ==
    }
    public void eat(){
        Syso("Animal can be"
            1. Herbivores
            2. Carn—
            3. Om— ");
    }
}
```

Animal t=new Tiger();
t.eat();

Animal obj=new Animal(); ✗ (object)

```
abstract class Animal {
    public abstract void eat();
}
```

3

extends

```
Class Tiger extends Animal{
    Can fly(){
        ==No.
    }
    public void eat(){
        Syso("I am Carnivores");
    }
}
```

```
Class Cow {
    eat(){
        ==
    }
}
```

```
Bear{
    Can fly(){
        ==
    }
    eat(){
        ==
    }
}
```

```
Human {
}
```

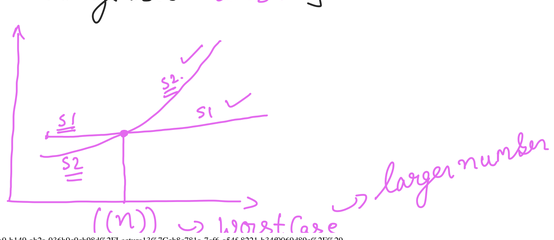
Time Complexity. ↪ to measure efficiency of algos.

algo → Same hardware → input-size.
5 sec 2.5 sec.

n=10; n=100, n=1000, n=10^9.

Sort (10^9) ↪ Bubble Sort → 7min. ↪ MergeSort.
↪ Merge Sort → 1ms.

(100) ↪ Bubblesort → 1ms
Merge Sort → 2ms.



Linear Search → n (10 arr) ⇒ ops → (10)
n = 100 ⇒ → 100
n = 1000 = 1000
n = 100000 = 100000
n = 10^9 ⇒ (10^9).
↳ O(n)

```
for (int i=0; i<n; i++){
    if (arr[i] == data){   -> O(1)
        return i;
    }
}
return -1
```

factorial → 5! → O(1)
↳ 5 x 4!
↳ 4 x 3!
↳ 3 x 2!
↳ 2 x !

```
ans = 1
for (int i=n; i>=1; i--){
    ans = ans * i;
}
```

↳ O(n)

Bubble Sort →

```
for (int i=1; i<=n-1; i++){
    for (int j=0; j<n-i; j++){
        if ( ) {
        }
    }
}
```

1, 2: $(n-1) + n-2 + n-3 + \dots 1$

$= \frac{(n)(n-1)}{2}$

$= \frac{n^2}{2} - \frac{n}{2}$

↳ $O(n^2)$

$n^2 + n$ ⇒ $O(n^2)$.
$2n^2$ ⇒ $O(n^2)$

B.S → 10, 20, 30, 40, 50 ↳ n^2
I.S → 10, 20, 30, 40, 50 ↳

```
for (int i=1; i<arr.length; i++){    => O(n) -> Best Case
    for (int J=i; J>0 && arr[J] < arr[J-1]; J--){
        swap(..).
    }
}
```

* → {3, 5, 1, 3, 7, 1} → n → n x n
↳ Duplicate → B.F ↳ n^2

{1, 2, 3, 4, 5, 6} → n^2
Better.
↳ Arrays.Sort(arr) → n logn. ↳ 10^9

{1, 1, 3, 3, 5, 7} n logn + n
↳ O(n logn)

↳ hashMap

0	1	2	3	4	5	6	7
	2		12		1		1

exp → $O(2^n)$
$O(n^3)$
n logn >> n^2 → $O(n^2)$
O(n logn)
O(n)
O(logn)
O(1)

$T(N-1) \simeq T(N-2)$

$T(N) = T(N-1) + T(N-2)$
$T(N) = 2T(N-1)$
$T(N-1) = 2T(N-2)$
$T(N-2) = 2T(N-3)$
⋮
$T(1) = 1$

$T(N) = 2 \times 2 \times \dots$ n times
$= 2^n$

arr = {3, 1, 15, 6, 4, 7} ↳ n^2 → x.
↳ O(n)

next Larger Element →
3 ⇒ 15
1 ⇒ 15
15 ⇒ -1
6 ⇒ 7
4 ⇒ 7
7 ⇒ -1.

3, 1, 2, 15
1 ⇒ 2
stack: 3, 2

Stack: 15, 6 7, 4

1 ⇒ 15
3 ⇒ 15
4 ⇒ 7
15 < 6
6 ⇒ 7
6 < 4
7 > 4
7 > 6

Q ⇒ First Negative in a window of Size K.
arr = {3, -1, -5, 6, +8, 7}
K=3

3, -1, -5 ⇒ -1

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

-1, -5, 6, -8 => -5 ✓

6, 8, 7 => 0

O(n)

int i = 0;

```
for (int i=0; i<K; i++){
    if (arr[i] < 0){
        queue.enque(i);
    }
}
```

3 3 3

3-3

front.

0 <= front

Queue => 1, 2

=> -1, -1, -5

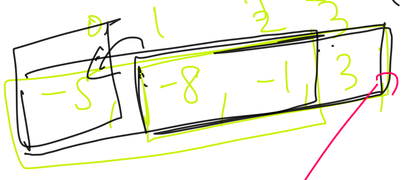
5-3

2 <= 2

0, 1, 2 3

```
for (; i< arr.length; i++){
    if (queue.isEmpty()){
        Syso("0");
    } else {
        Syso(arr[queue.front]);
    }
```

n-K+K



1, 2

1 <= 4-3

1 <= 0

0 <= 0

```
while(!queue.isEmpty && queue.front <= i-K){
    queue.deque();
}
if (arr[i] < 0){
    queue.enque(i);
}
```

if (Queue.isEmpty) { No negative };
else {
Syso(arr[queue.front]);
}