Collections

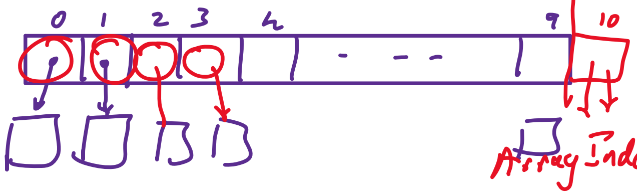
Arrays

Student[] s = new Student[10];

0 <-> 9

s[0]=new Student();

s[1]=new Student()



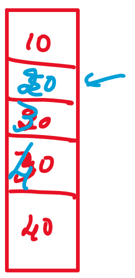
ArrayIndexOutOfBoundException

1) Max limit

2) No dynamic inc-feature

3) Insert & Deletion takes time

4)



N=4

Data Structure implemented classes

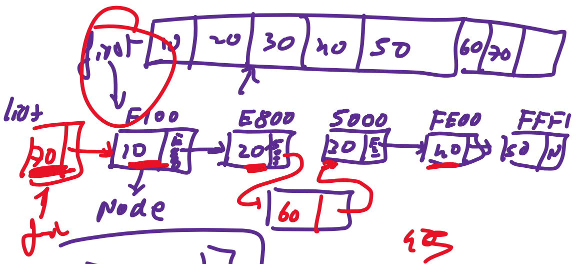
1) ArrayList -> Array

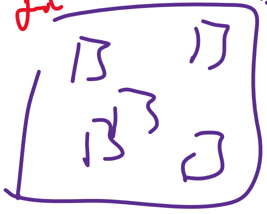
2) LinkedList -> Linked list

3) HashSet => Hashing

4) Binary Search Tree

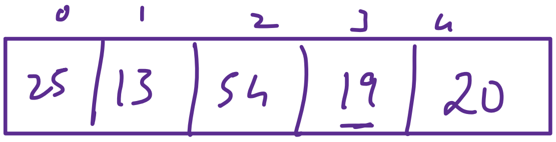
TreeSet





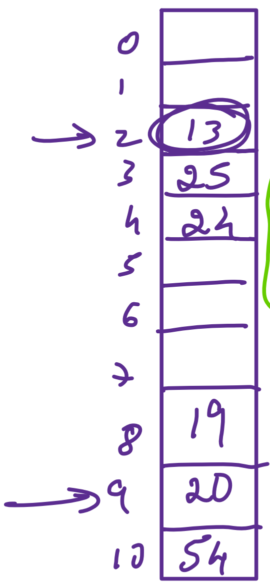
N -> O(N)

Hash table -> O(1)



25	13	54	19	20

11 % 11 = 0



0	
1	
2	13
3	25
4	24
5	
6	
7	
8	19
9	20
10	54

13
25
24
19
20
54

20 % 11 = 9 Hash Function

data % HashTable Size

Collections.sort

Hash Value

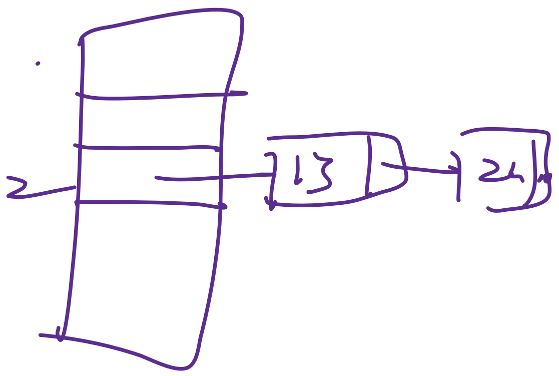
25 % 11 = 3

13 % 11 = 2

54 % 11

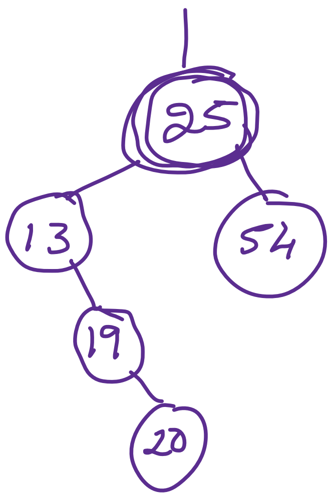
44
10

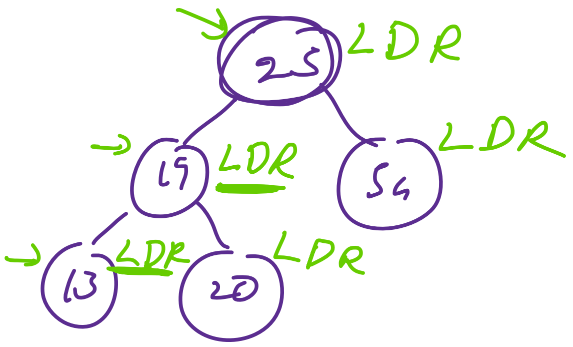
24 % 11 = 2



Binary Search Tree → Balanced → AVL
Fib
Red-Black

25 13 54 19 20





n/2
n/2/2
n/4/2
n/8/2

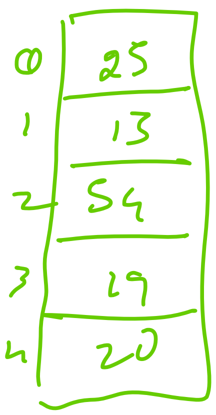
$O(\log n)$

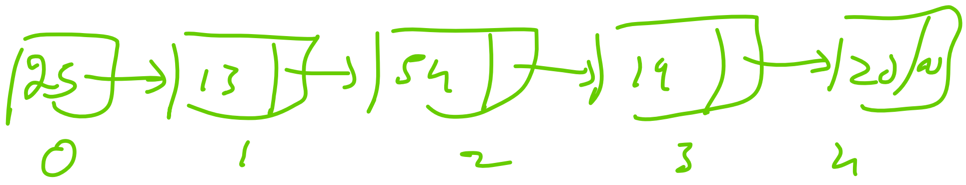
↓
LDR

Order

In-order traversal

13 19 20 25 54



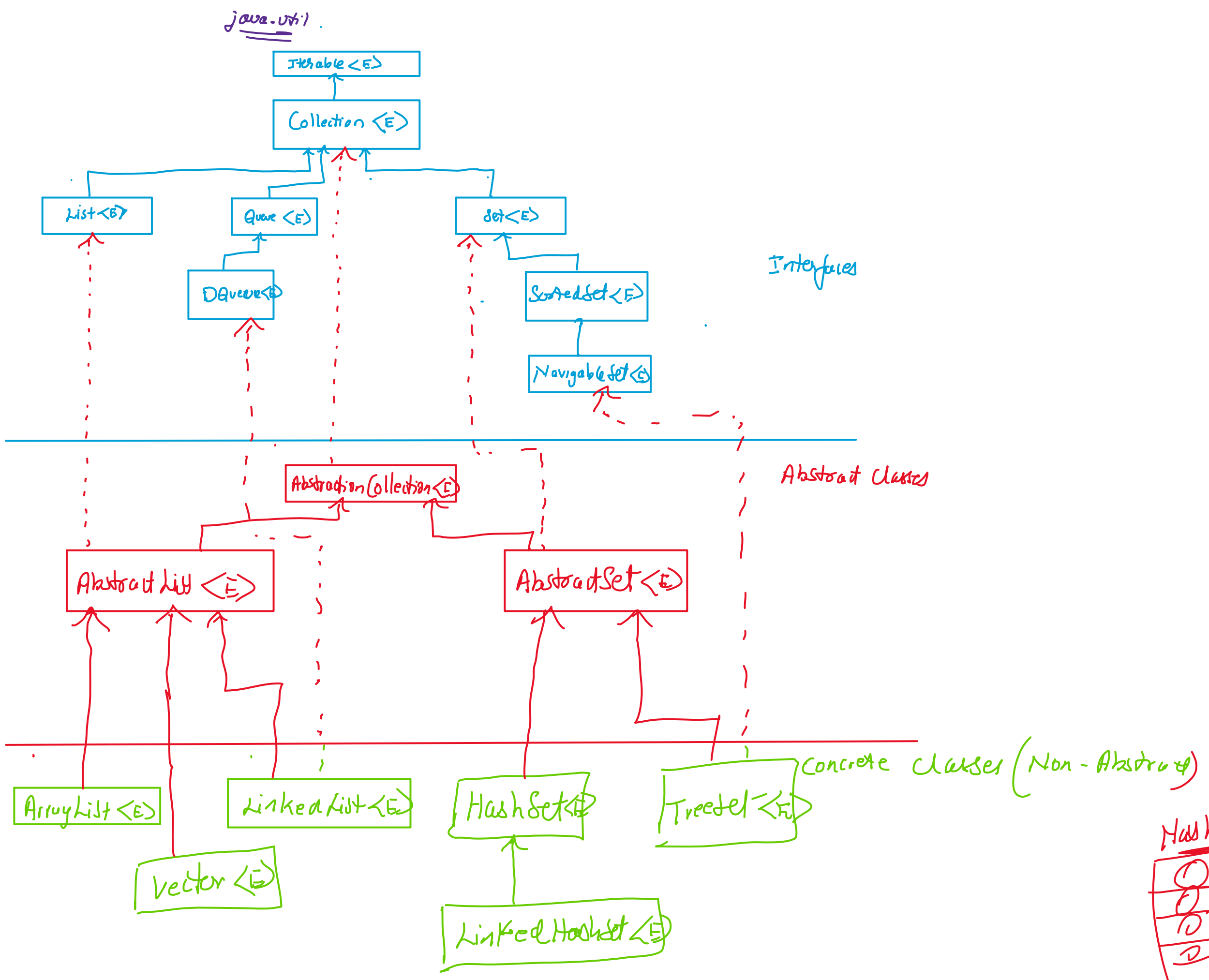


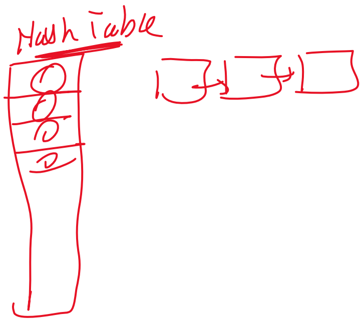
Order List → ArrayList
→ LinkedList

un-order List ← HashSet
← TreeSet

list.get(3)

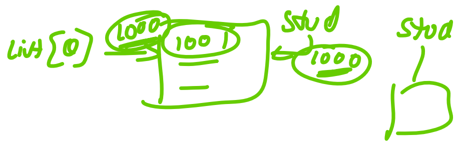
- → Stack
- → Queue
- → Array
- → Linked List
- → Hashing
- → Binary Tree





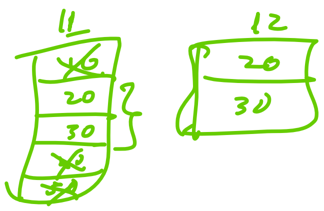
Collection<E> Interface

1) boolean add(E obj)
2) boolean addAll(Collection c)
3) boolean contains(Object obj)
4) boolean remove(Object
5) boolean containsAll(Collection<E>)
6) boolean removeAll(Collection<E>)
7) boolean retainAll(collection<E>)
8) int size
9) Object[] toArray()
10) Iterator<E> iterator()

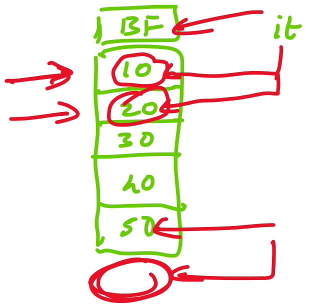


Class Object
{

boolean equals(Object obj)
{



l1.retainAll(l2)

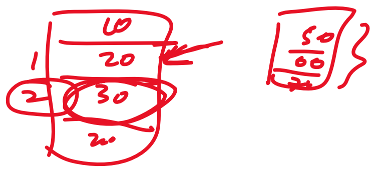


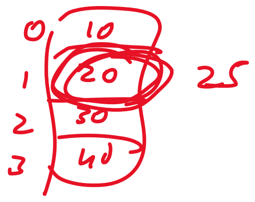
E next()
boolean hasNext()

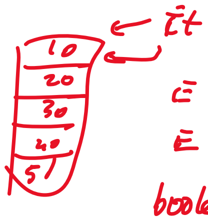
```
Collection<Student>list=newArrayList<Student>();
list.add(newStudent(1001,"Amit",55.55f));
list.add(newStudent(1002,"Kiran",66.66f));
list.add(newStudent(1003,"Ravi",77.77f));
list.add(newStudent(1004,"Rakesh",88.88f));

Iterator<Student>it=list.iterator();
while(it.hasNext()){
System.out.println(it.next());
}


for(Students:list){
System.out.println(s);
}
```

List<E>

1) void add(int index, E obj)
2) void add(int index, Collection<E>)
3) E get(int index)
4) int indexOf(Object obj)
5) E remove(int index)
6) E set(int index, E obj)
7) ListIterator<E> listIterator()

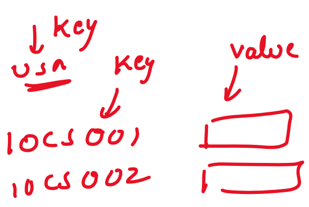




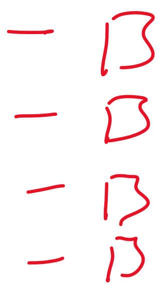


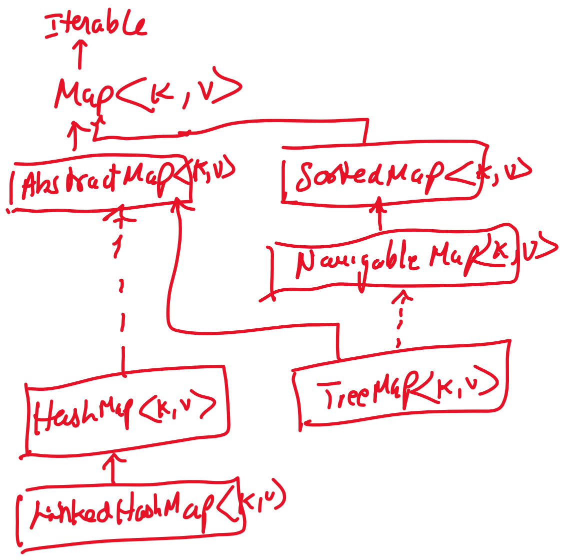
E next()
E previous()
boolean hasNext()
boolean hasPrevious()

```
List<Student> list = new ArrayList<Student>();

list.add(new Student(1001,"Amit", 55.55f));
                ↓ value    Student.class
  class Student
  {
        private Integer rollno;
        private String name;
        private float percentage;

        ≡
  }
```



Map





```
Map<String, Student>   map = HashMap<String, Student>
```

java.util.stream.Stream<E>
↳ Collection of D

System.out::println

void accept(E obj)

stream.forEach(Consumer<E> consumer)

```
for(Student s : list)
{
    System.out.println(s);
}
```

```
list.stream().forEach((s)->{
                        s.o.p(s)
                     });

list.stream().forEach((s)-> s.o.p(s));

list.stream().forEach(System.out::println);
```

```
for(i=0; i< list.size(); i++)
{
    if(list.get(i).getPercentage()>60)
        s.o.p(list.get(i));
}
```

```
list.stream().forEach((s)->{ if(s.getPercentage()>60)
                                 s.o.p(s);
                        });
```

```
list.stream().filter((s) -> s.getPercentage()>60)
             .forEach(System.out::println);
```

```
forEach(Consumer<E> consume)
{

}
```

```
@FunctionalInterface
interface Consumer<E>
{
    void accept(E obj);
}
```

Stream API

1)
```
Stream<String> empty = Stream.empty()
```

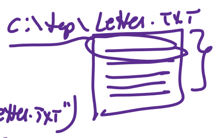
2)
```
Collection<String> collection = Arrays.asList("Anil","Kiran","Ravi","Raju");
Stream<String> stream = collection.stream();
```

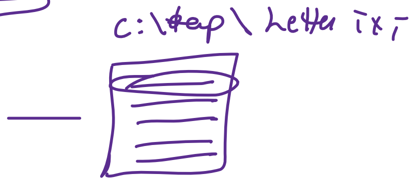
3) Stream of array
```
String[] st = new String[]{"Ravi","Amit","Anil"};
Stream<String> stream = Arrays.stream(st);
```

4) stream.builder()
```
Stream<String> stream = Stream.<String>builder().add("Ravi")
                                                .add("Kiran")
                                                .add("Archit");
```

6) Stream iterator   40, 42, 44, 46, 48
```
Stream<Integer> stream = Stream.iterate(40, (n) -> n+2)
                               .limit(20);
```

7) Stream of file contents
```
Path path = Paths.get("C:\\temp\\Letter.TXT")
Stream<String> fileStream = Files.lines(path);
fileStream.forEach(System.out::println);
```





---

filter

```
Stream<E>.filter(Predicate<E> predicate)
                    ↳ interface Predicate<E>
                      {
                          boolean test(E obj);
                      }
```

```
Stream<Integer> stream = Stream.iterate(1, n -> n+1).limit(20)

stream.filter((x) -> x%5==0)
      .forEach(System.out::println);
```