

Function PosNums(n):

w^or^st c^as^e $O(n)$

$a \leftarrow \text{new ArrayList}$

For i in 1 to n :

amortized w^or^st c^as^e $O(n)$.

$a \cdot \underline{\text{insert At Tail}}(i)$

Complete binary trees

↓
Array hist

insert

$O(n)$ add

$O(\log n)$ bubble

$\frac{O(n)}{O(n)}$

amort $O(1)$

$O(\log n)$

amort

$O(\log n)$

w^or^st c^as^e $O(n)$

amortized w^or^st c^as^e $O(1)$

avg complexity over
a sequence

Hash Table (is a kind of Dictionary)

	unbalanced BST	AVL	HashTable (our goal)
get	$O(n)$	$O(\log n)$	$O(1)$
put	{ insert update remove	$O(n)$ $O(n)$ $O(n)$	$O(1)$ $O(1)$ $O(1)$
		$O(\log n)$ $O(\log n)$ $O(\log n)$	$O(1)$
			average

Dictionary

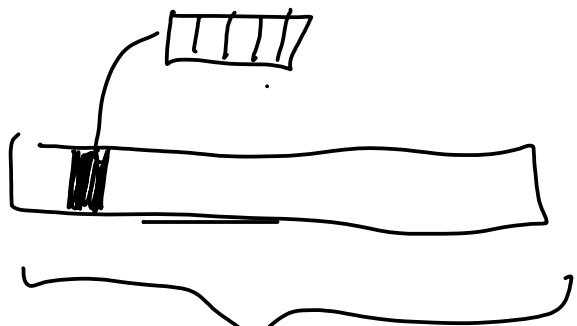
constant time for all ops

1. all keys are int

2. all keys are positive
3. all keys < 1,000,000

4. no keys will overlap

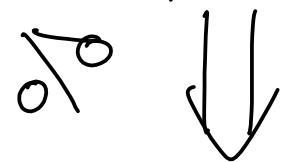
pair < pair<K, V>
, bool > *



1 000 000

{ insert(2 000 005, "hi")
insert(5, "hi again")
→ "Collision"

1. Any int key $\xrightarrow{\text{hash}}$ C. Any key



2. Key in int range

Write $\xrightarrow{\text{hashing}}$ func

$K \rightarrow \text{int}$

\uparrow

hash

"hello how are"

\downarrow hash

5023

\downarrow

$\% \text{arraySize} \text{ (pretend 10)}$

3

Hashing Functions: $K \rightarrow \text{int}$

when K is int:

```
int hash(int key) {  
    return key;  
}
```

Purpose of hashing function: turn key into an int

when K is string

```
int hash(string s) {
```

```
int acc = 0;  
for (int i = 0; i < 4 &&  
     i < s.length(); i++) {  
    acc *= 256;  
    acc += s[i];  
}  
return acc;
```

"hi"
"happy"
"bears"
"happiness"

stoi
int acc = 0;
for each c in str:
 acc *= 10;
 acc += c as num;
 (c - '0')

```
int hash(string s) {  
    return s.length();  
}
```

✓

"hello" ↗ 5
"bears" ↗ 5

Collisions:

Linear Probing

Chaining

Hash Tables:

1. Turn key into int
2. Mod int into array index
3. Use array to store data
4. Deal w/ collisions

Linear Probing : on collision, just use next

3 things per bucket

K key
V value

bool inUse

Diagram illustrating Linear Probing:

A horizontal array of 4 slots, labeled 1, 6, and 11 above them. Below the array, a bracket indicates its capacity.

1	"hi"	6	"hello"	11	"test"
X	✓	X	✓	X	✓

The array is shown with three rows: the top row contains the indices 1, 6, and 11; the middle row contains the string values "hi", "hello", and "test"; and the bottom row contains the boolean values X, ✓, X, ✓, X, and ✓. Arrows point from the labels K, V, and bool inUse to the corresponding columns in the array. A bracket below the array is labeled "capacity".

```

insert(1, "hi")
insert(6, "hello")
insert(3, "bye")
insert(11, "test")
remove(3)
    
```

Linear probing Hash Table

array of triples K, V, inUse-bool

insert (K key, V value) :

h ← hash(key)

i ← h $\%$ capacity

while array[i].inUse :

i++

if ($i \geq \text{capacity}$)

i ← 0

array[i].key ← key

array[i].value ← value

array[i].inUse ← true

get (K key)

h ← hash(key)

i ← h $\%$ capacity

while array[i].inUse :

if array[i].key == key
return array[i].value

i++

if ($i \geq \text{capacity}$)
i ← 0

throw exception

remove(K key)

↑ figure out start index i

look for matching $l \in$

remove matching key

as long as next box has key does not hash to its idx
move that box left