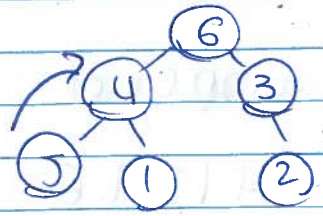
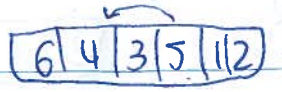
Exercise 2 :





indicates removal point (taken off array)

start at 3 ✓ its ok

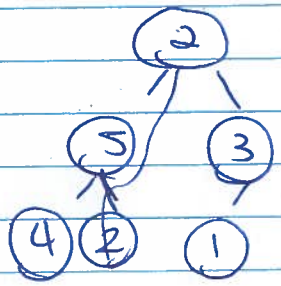
fix 1: swap 5 with w/ 4 because larger child

(removed) 6

6	5	3	4	1	2

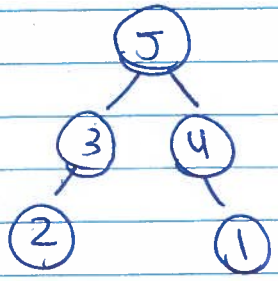
Remove 6 from the original array. The value of 6 is no longer

5	3	4	1	2	6



swap 2 to the front of heap

2	5	3	4	1



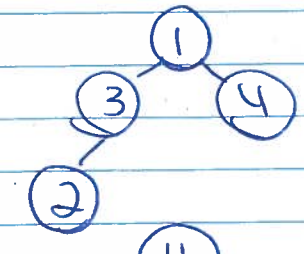
Percolate to where it belongs

5	3	4	2	1

6 5

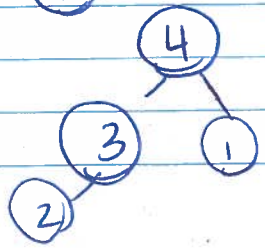
Now repeat w/ new node

1	3	4	2	5 | 6

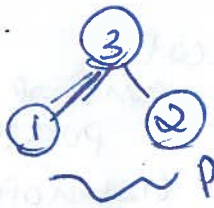


percolate to where it belongs

4	3	1	2



remove node



percolate across

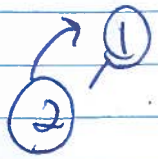
removed

3	1	2

4 | 5 | 6

s

remove root, swap end



percolate down

2	1

3 | 4 | 5 | 6

2	1

sorted remove

sorted! whoop!

1 | 2 | 3 | 4 | 5 | 6

Exercise 3:

Given hash table with m slots that stores n elements

Load factor: α for $T = n/m$

(average number of elements stored in a chain) → $\alpha < 1$, $\alpha > 1$, $\alpha = 1$

In our analysis, we will look at the expected number of clashes also in terms of big OH.

WC: We will look first at the worst case.

- all n keys hash to the same slot m n.
- the expected number of clashes is n
- the worst-case for operations is $O(n)$ + time to compute the function
- load ~~factor~~

$$\left(\frac{n \cdot n}{m}\right)$$

AC: We will now look at the average case. The AC performance depends on ~~the~~ how well the h func. distributes n over m on average.

If we assume simple uniform hashing (equally likely to hash into any m slots)

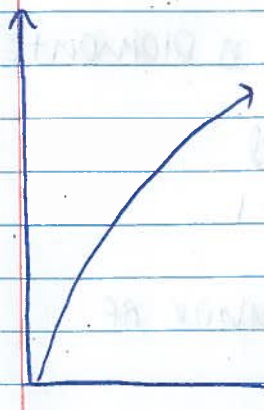
For $j = 0, 1 \ldots m-1$ → length of $T[j]$ by n_j

where $n = n_0 + n_1 + n_{-3}$

so the expected value of $n_j = E[n_j]$ meaning our load factor is $\alpha = n/m$

$(n_j/m-1)$

- In this analysis, we can look at the expected number of clashes like the birthday problem.



As the number of keys increases, number of clashes goes up. The load factor stays ~~at~~ within bounds assuming the hash table & function perform u

← In terms of big oh, this AC is O(n).

↑ as set of keys increases, number of hash colli does as well.

B - binary → comment + spaced ✓

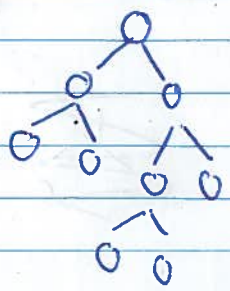
4.

```
RotateRight (B, x)
    y ← left[x]
          //set y
    left ← right[x]
          //turn y's subtree into
          //x's left subtree
    if right[y] ≠ NIL
          then p[right[y]] ← x
    //link x's parent to y
    p[y] ← p[x]
    if p[x] = NIL                    //Checking here for the roots
          then root[B] ← y
          else if x = right[p[x]]
                then right[p[x]] ← y
                else left[p[x]] ← y
//Put x on y's right here
    right[y] ← x
    left[x] ← y
```

Exercise 5:

BST w/ n keys 1, 2, 3, n

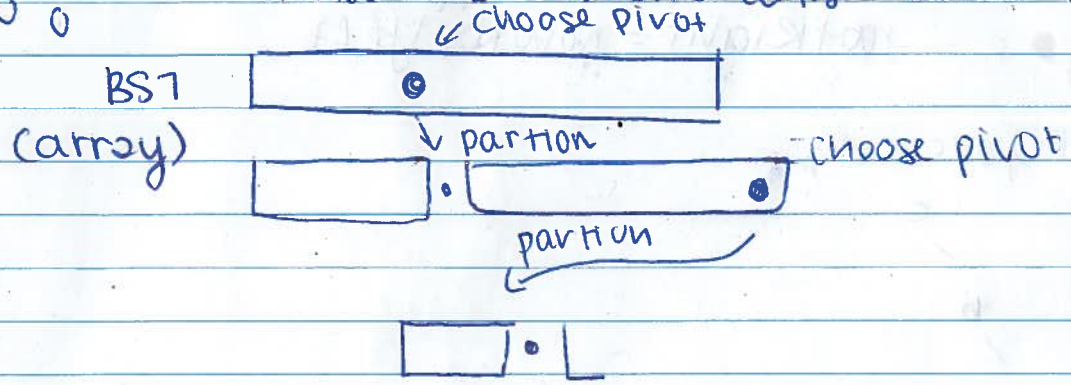
suppose BST has n elements 1, 2, 3



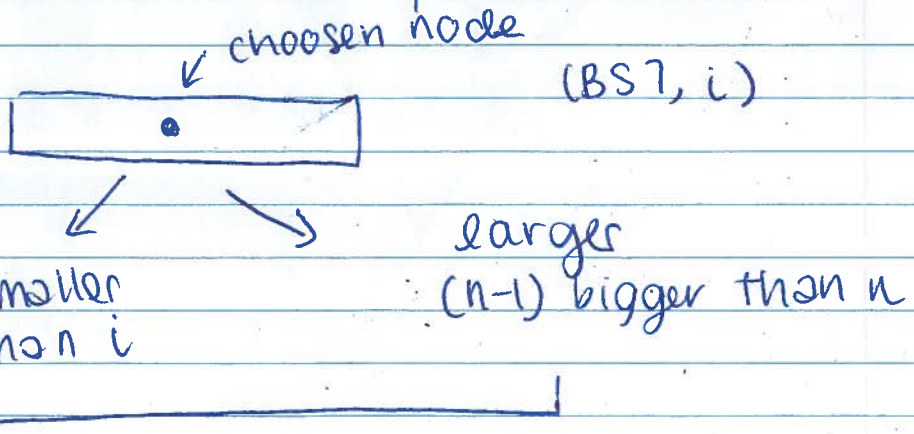
BST.size

0

Consider the recursive calls select (BST, i)



Consider all possible BST with each element at the ROOT. If there are n nodes, then for each choice of root node, there are n-1 non-root nodes. These non-root nodes must be portioned into those that are less than a choosen root & those that are greator.



For each of these two node sets, there is a certain number of possible subtrees.

~ Let $t(n)$ be the total number of BST w/ n nodes.
The total number of BSTs w/ i at root is $t(i-1)\,t(n-i)$

($t(i-1)$: less; $t(n-i)$: bigger)

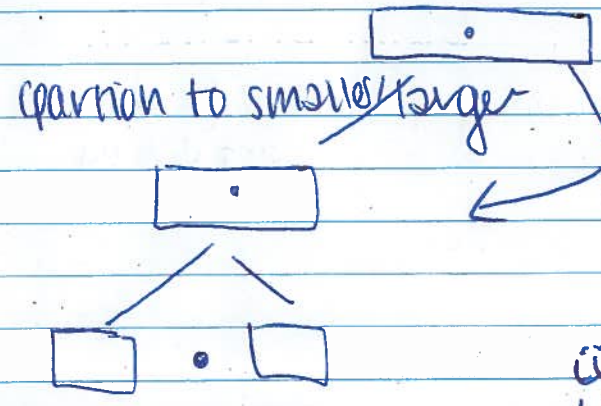
For each arrangement in the left tree and for each arrangement in the left and right trees are independent. That is, for each arrangement in the left and for each arrangement in the right, we arrive at one BST with i at the root.

Summing over i gives us the total BST w/ n nodes.

Base case: $t(0) = 1$ and $t(1) = 1$

There is one empty BST and there is one BST with a node.

Visually:



i) $t(2) = t(0)\cdot t(1) + t(1)\cdot t(0)$

ii) $t(3) = t(0)\cdot t(2) + t(1)\cdot t(1) + t(2)\,t(0) = 2 + 1 + 2 = 5$

iii) $t(4) = t(0)\,t(3) + t(1)\,t(2) + t(2)\,t(1) + t(3)\cdot t(0) = 5 + 2 + 2 + 5 = 14$

iv) $t(5) = t(0)\cdot t(4) + t(1)\,t(3) + t(2)\cdot t(2) + t(3)\cdot t(1) + t(4)\,t(0) = 14 + 5 + 4 + 5 + 14 = 42$

induction Proof:

The total number of trees w/ "k" at the root will be $t(k-1)\cdot t(n-k)$.