[redacted]

1) a)

key	value	
E	0	E0 — no compare
A	1	A1 → E0 — 1 compare
S	2	S2 → A1 → E0 — 2 compare
Y	3	Y3 → S2 → A1 → E0 — 3 compare
Q	4	Q4 → Y3 → S2 → A1 → E0 — 4 compare
U	5	U5 → Q4 → Y3 → S2 → A1 → E0 — 5 compare
! E	6	U5 → Q4 → Y3 → S2 → A1 → E6 — 1 compare
! S	7	U5 → Q4 → Y3 → S7 → A1 → E6 — 3 compare
T	8	T8 → U5 → Q4 → Y3 → S7 → A1 → E6 — 6 compare
I	9	I9 → T8 → U5 → Q4 → Y3 → S7 → A1 → E6 — 7 compare
O	10	O10 → I9 → T8 → U5 → Q4 → Y3 → S7 → A1 → E6 — 8 compare
N	11	N11 → O10 → I9 → T8 → U5 → Q4 → Y3 → S7 → A1 → E6 — 9 compare

unsorted linked-list

accesses all nodes until it finds a duplicate. Then it updates the value.

49 compares

This uses $\frac{N^2}{2}$ compares to search, insert, and for search hit.

1) b) using sorted array:

	0	1	2	3	4	5	6	7	8	9	10	11
keys[]	E	A	S	Y	Q	U	E	S	T	I	O	N
vals[]	0	1	2	3	4	5	6	7	8	9	10	11

key	value	keys[] 0 1 2 3 4 5 6 7 8 9 10 11	N
E	0	(E)	1 0
A	1	(A) E	2 1 0
S	2	A E (S)	3 1 0 2
Y	3	A E S (Y)	4 1 0 2 3
Q	4	A E (Q) S Y	5 1 0 4 2 3
U	5	A E Q S (U) Y	6 1 0 4 2 5 3
! E	6	A (E) Q S U Y — replace val	7 1 (6) 4 2 5 3
! S	7	A E Q (S) U Y — replace val	8 1 6 4 (7) 5 3
T	8	A E Q S (T) U Y	9 1 6 4 7 8 5 3
I	9	A E (I) Q S T U Y	10 1 6 9 4 7 8 5 3
O	10	A E I (O) Q S T U Y	11 1 6 9 10 4 7 8 5 3
N	11	A E I (N) O Q S T U Y	12 1 6 9 10 11 4 7 8 5 3

also, it can be visualized as:

1 

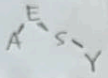
0 compares

2 

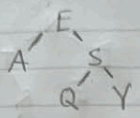
1 compare

3 

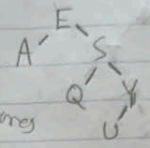
1 compare

4 

2 compare

5 

2 compares

6 

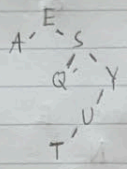
3 compares

7 replace value of E with 6

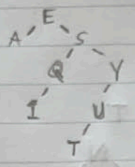
(1 compare)

8 replace value of S with 7

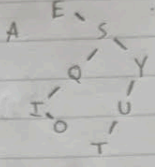
(2 compares)

9 

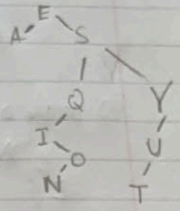
4 compares

10 

3 compares

11 

4 compares

12 

5 compares

28 comparisons

takes $\frac{N^2}{4}$

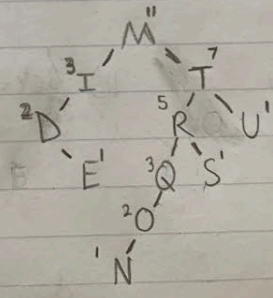
2) a)

key	value	
M	0	M
I	1	IM
D	2	DIM
T	3	DIMT
E	4	DEIMT
R	5	DEIMRT
M	6	DEI(M)RT
Q	7	DEIMQRT
U	8	DEIMQRTU
E	9	D(E)IMQRTU
S	10	DEIMQRSTU
T	11	DEIMQRS(T)U
I	12	DE(I)MQRSTU
O	13	DEIMOQRSTU
N	14	DEIMNOQRSTU

key value

S 15 DEIMNOQRSTU

37 comparisons



comparisons: found by counting nodes at each level!

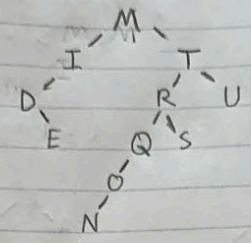
1	1
2	2
3	3
4	3
5	1
6	1

so 1 + 4 + 9 + 12 + 5 + 6

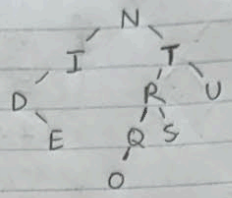
= 37

2 b) Deleting "MIDTERMQUESTIONS" in that order:

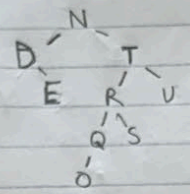
0) original



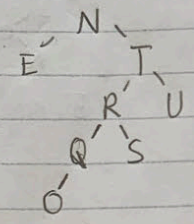
1) delete M



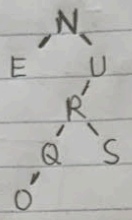
2) Delete I



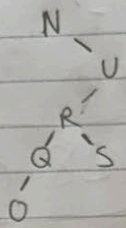
3) Delete D



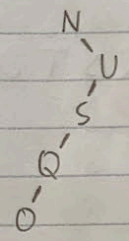
4) Delete T



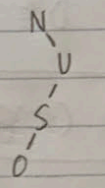
5) Delete E



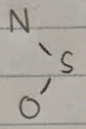
6) Delete R



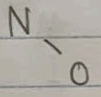
7) Delete M ✓
Delete Q



8) Delete U



9) Delete E ✓
Delete S



10) ~~delete~~ T ✓
delete I ✓
delete O

N

11) delete N

done!

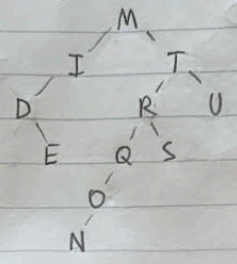
2 C)

excess keys cut out

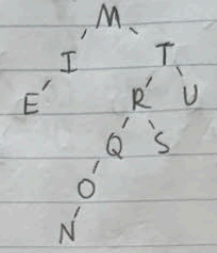
1 2 3 4 5 6 7 8 9 10 11

Deleting "DEIMNOQRSTU" In that order

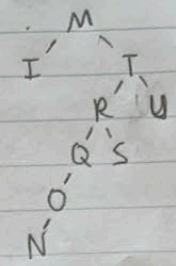
0) original



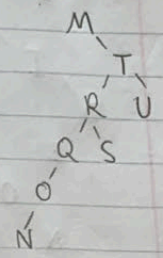
1) - D



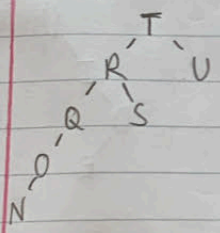
2) - E



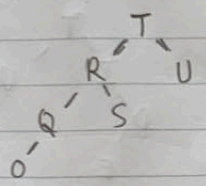
3) - I



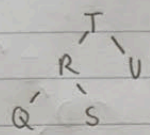
4) - M



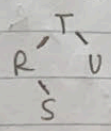
5) - N



6) - O



7) - Q



8) - R

9) - S

10) - T

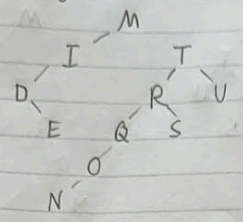
U

11) - U

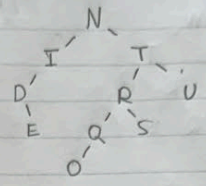
done

2 d) deleting the node at the root each time:

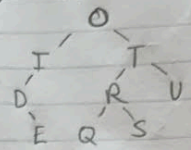
0) original



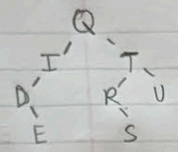
1) - M



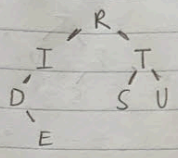
2) - N



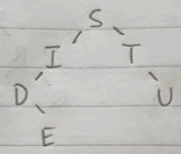
3) - O



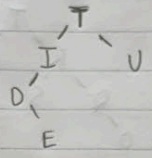
4) - Q



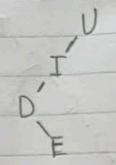
5) - R



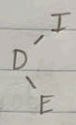
6) - S



7) - T



8) - U



9) - I

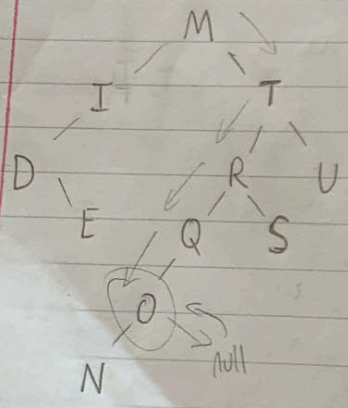
10) - D

E

11) - E

done

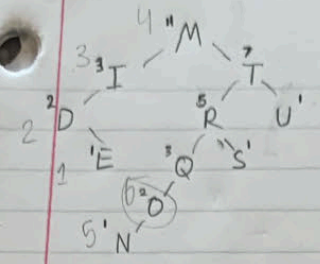
2 e) a. Floor (P) - largest val ≤ P



call

1) P is more than M, so Floor(P) may be on the right. Return M
2) P is less than T, so Floor(P) could be on the left. Return M
3) P is less than R, so floor(P) could be on the left. Return M
4) P is less than Q, so floor(P) could be on the left. Return M
5) P is more than O, so floor(P) may be on the right. Return O
6) the right of O is null. Return null
7) Floor(P) is "O".

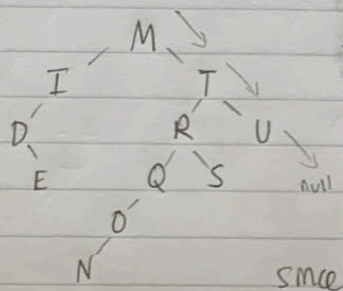
2 e) b. select(5) – asking us what is the key of rank 5



works by traversing from minimum value in the BST to greatest and then finding the 6th least value (as it will be greater than 5 keys)

So 1) traverse to min() by going left until null is reached. Count = 1
2) traverse right, find E, count = 2, check for children
3) traverse to I as no children for E. count = 3
4) traverse to M as no children for I. count = 4
5) traverse to T → U → U.left. return to T
6) traverse T.left until null. Return to N and count = 5
7) traverse N. Right which is null, then return to O which makes count = 6. So **select(5) = 'O'**

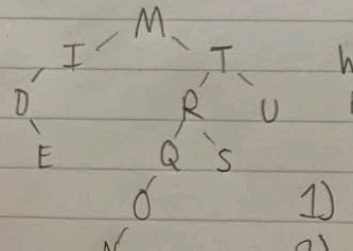
c. Ceiling(V) – smallest val ≥ V



call
1) V > M, so go right, return null
2) V > T, so go right, return null
3) V > U, so go right, return null
4) U.right == null, meaning that max() is U.

since no value is greater than V in the tree, **Ceiling("V") is null**

d. rank("S") – asking us how many keys S is greater than



works like select except we traverse and count until S is reached, then return the count.

1) traverse to min() until null. D makes count = 1
2) count the children, E makes count = 2
3) traverse to I and count its children in that order. I → count = 3
4) count = 4 with M, go to T → U → U.right to find null, check U for children which is null, go to T.min which is N. N → count = 5
5) O makes count = 6. → Q makes count = 7 → R makes count = 8
6) check right children of R and find S, return count. **rank("S") = 8**