1a)

i	j	0	1	2	3	4	5	6	7	8	9
0	10	11	12	4	13	2	5	11	5	16	14
1	7	11	12	4	13	2	5	11	5	16	14
1	7	11	5	4	13	2	5	11	12	16	14
3	6	11	5	4	13	2	5	11	12	16	14
3	6	11	5	4	11	2	5	13	12	16	14
6	5	11	5	4	11	2	5 !	13	12	16	14
Final	5	5	5	4	11	2	11	13	12	16	14

1b)

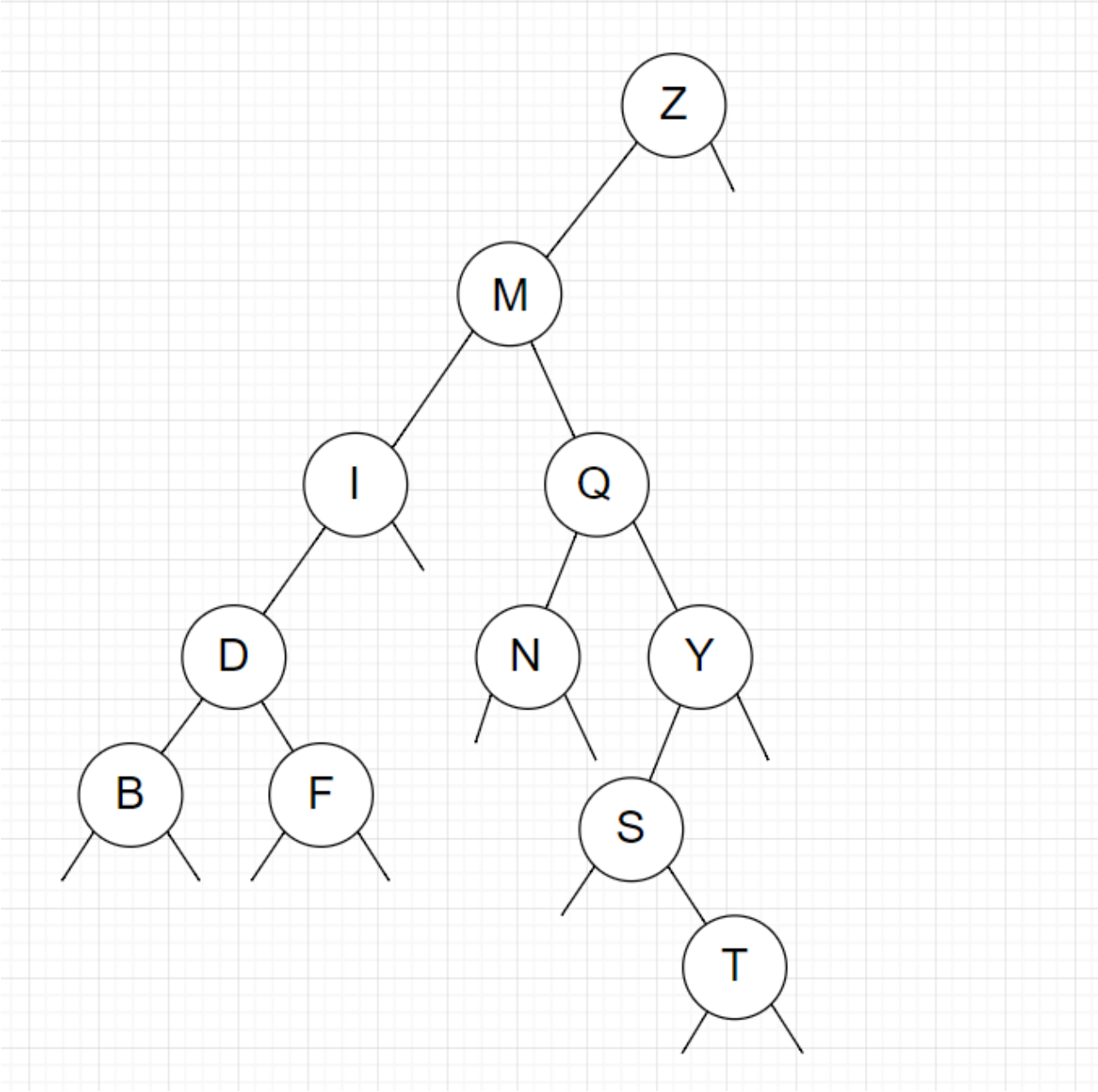
lb	pivPos	ub	0	1	2	3	4	5	6	7	8	9
0	5	10	5	5	4	11	2	11	13	12	16	14
0	2	5	4	2	5	11	5	11	13	12	16	14
0	1	2	2	4	5	11	5	11	13	12	16	14
3	4	5	2	4	5	5	11	11	13	12	16	14
6	7	10	2	4	5	5	11	11	12	13	16	14
8	9	10	2	4	5	5	11	11	12	13	14	16

1c)
The problem arises when there are long sequences of the same number, resulting in a StackOverflowError. With the less/more than or equal to requirement for i and j, the maximum number of function calls on the stack would be something like log(n) with n being the length of the same-number-sequence. While for the strictly less or more requirement, it would be n. The former requirement ultimately returns a pivotPosition somewhere in the middle of the sequence, while the latter returns lb. The problem then arises when `quickSort(arr, pivotPosition + 1, ub)` is called repeatedly as it goes through the same-number-sequence one by one position.

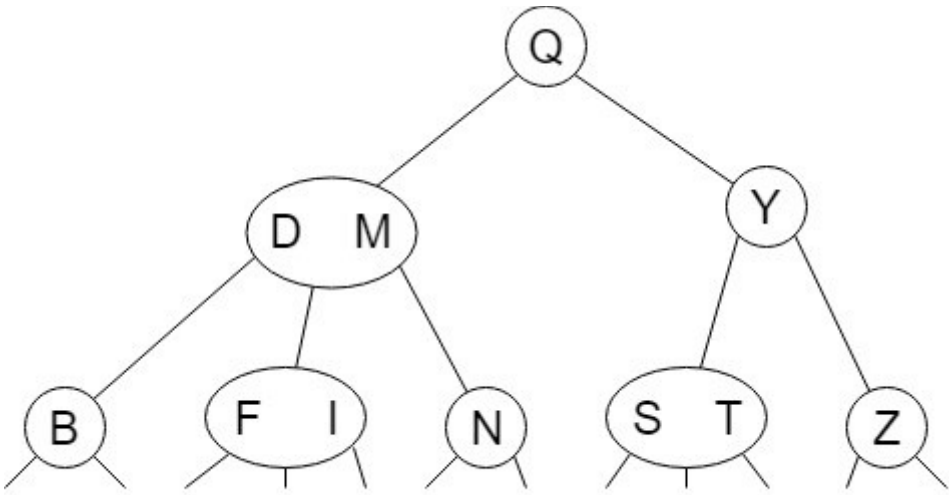
2a)
[null, T, S, R, N, P, O, A, E, I, G, H]

2b)
If only one maximum/minimum value is tracked, this value would of course change when the maximum/minimum is removed. Then one would have to find a new maximum/minimum, which in the worst case would take linear time for each remove. It would then be better to keep the data structure sorted in the first place.

3a)



4a)

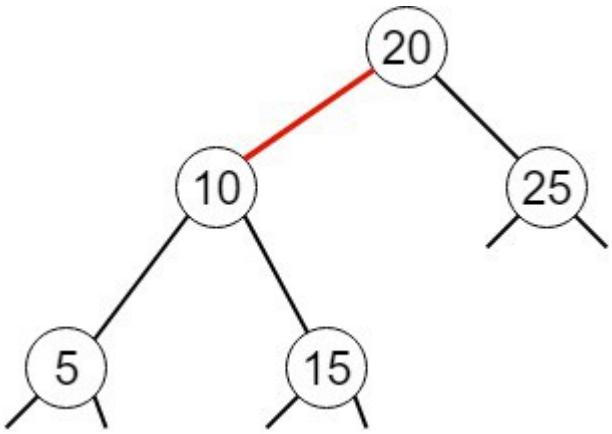


4b)
B F Y N S T I D

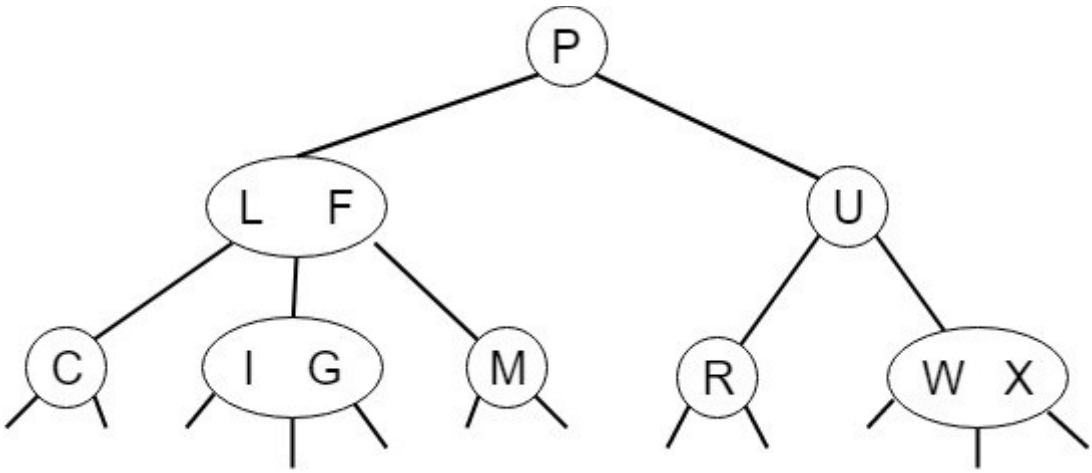
4c)
(iii) and (iv)

4d)

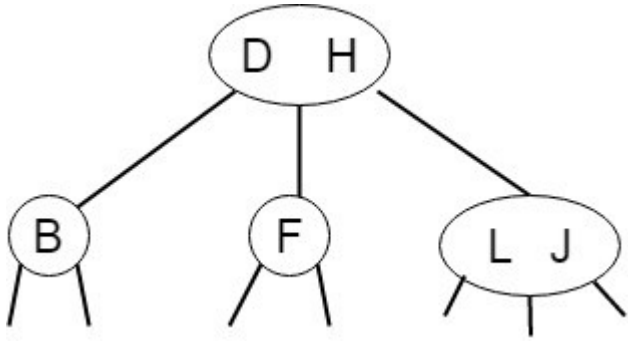
(ii)



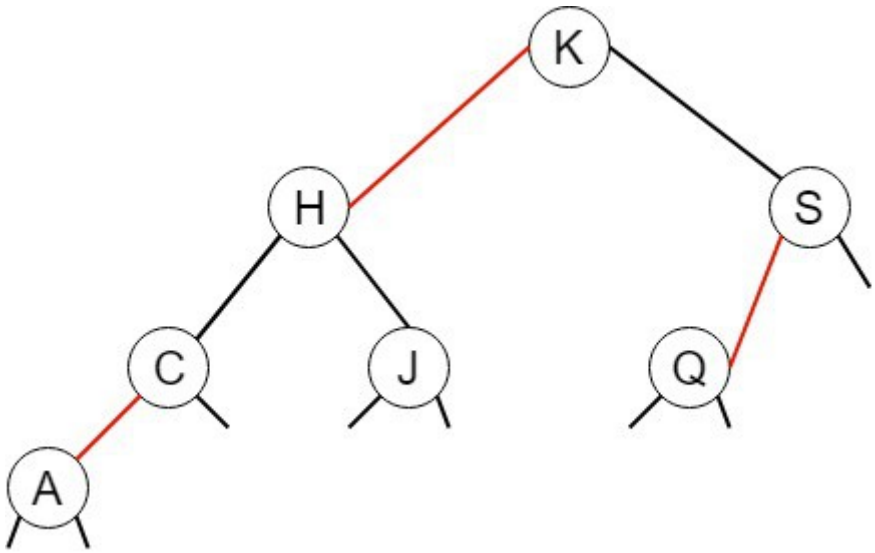
(iii)



(iv)



(v)



4e)
L R F F R F R F R F L

4f)
The maximum depth increments by 2 for every increment in height, starting at maximum depth 2 for height 0. Red-black trees are perfectly balanced. This means that the minimum number of nodes n for an increment in height k is given by 2^{k+1} -1. Therefore, height is a function of n by the expression $lg_2(n + 1) – 1$. The maximum number of nodes from root to null in a red-black tree of size n is then:
$2 + 2 * height = 2 + 2 (lg_2(n + 1) – 1) = 2 lg_2(n + 1)$.