

Question 1

a). Why can't you make an object of type `LinkedList<int>`? What should you do instead?

Ans `LinkedList` is Generic type and use object class to generic not use Primitive type. Can use `LinkedList<Integer>` because `Integer` is object class of `int` primitive

b). What is an iterator, and why are iterators necessary for generic programming?

Ans is an object that allow programmer to traverse all element of collection

c) Implement the method `counter()` by using an iterator to scan a `LinkedList` collection.

Return an integer that gives the number of occurrences of the item object in the list.

```
public static int counter(LinkedList<?> aList, Object item) {
    int result = 0;
    Iterator it = aList.iterator();
    while (it.hasNext()) {
        Object o = it.next();
        if (o.equals(item)) {
            result++;
        }
    }
    return result;
}
```

d) Re-implement `counter()` without using an iterator

```
public static int reCounter(LinkedList<?> aList, Object item) {
    int result = 0;
    for (int i = 0; i < aList.size(); i++) {
        Object o = aList.get(i);
        if (o.equals(item)) {
            result++;
        }
    }
    return result;
}
```

e) In what situations is a `ListIterator` especially useful?



Question 2

A 15-element array is to be sorted using radix sort, utilizing 10 bins(queues) for the digits 0-9. The initial array is

363, 251, 670, 84, 175, 45, 123, 389, 90, 8, 122, 7, 491, 593, 528

a). Draw th contents of the 10 bins after pass 0 of the radix sort. Also, show the new order of ther array after pass 0.

Ans

			593						
90	491		123		45			528	
670	251	122	363	84	175		7	8	389
0	1	2	3	4	5	6	7	8	9

After pass 0 : 670 90 251 491 122 363 123 593 84 175 45 7 8 528

b). Draw th contents of the 10 bins after pass 1 of the radix sort. Also, show the new order of ther array after pass 1.

Ans

		528							593
8		123					175		491
7		122		45	251	363	670	84	90
0	1	2	3	4	5	6	7	8	9

After pass 1 : 7 8 122 123 528 45 251 363 175 670 84 90 491 593

c). Draw th contents of the 10 bins after pass 2 of the radix sort. Also, show the new order of ther array after pass 2.

Ans

90									
84									
45	175								
8	123				593				
7	122	251	363	491	528	670			
0	1	2	3	4	5	6	7	8	9

After pass 1 : 7 8 45 84 90 122 123 175 251 363 491 593 528 670



d) Discuss the implementation and efficiency of radix sort.

Ans radix sort use $O(d*n)$ d = number of digit of biggest element and n = number of element but radix sort use a lot of memory

Question 3

a). What are main differences between a TreeMap and a HashMap?

Ans TreeMap implement by binary tree is sorted order and HashMap implement by hash and not order

b) Finish the implementation of the MyInteger class given below

```
public class MyInteger implements Comparable<MyInteger>{
    private int value;

    public MyInteger(int n) {
        this.value = n;
    }
    public void setValue(int val){
        this.value = val;
    }
    public int getValue(){
        return this.value;
    }
    public void increment(){
        value++;
    }
    public String toString(){
        return "Value = "+value;
    }
    public int compareTo(MyInteger m) {
        return value - m.getValue();
    }
}
```

b) Write code that employs a TreeMap whose entries use MyInteger keys and MyInteger values

The code generates 5000 random integers in the range 0 to 9. Each integer is checked against the map. If the integer is the key of an existing entry, then that entry's value is incremented by 1. Otherwise a new entry is created with the integer as its key, and a value of 1.

After all the randomly generated integers have been checked against the map, use the map to print out their occurrence frequencies.



```
public static void main(String[] args) {
    TreeMap<MyInteger, MyInteger> maps = new TreeMap<MyInteger,
MyInteger>();
    for (int i = 0; i < 5000; i++) {
        int random = (new Double(Math.random() * 10).intValue())%10;
        MyInteger key = new MyInteger(random);
        MyInteger value = maps.get(key);
        if (value == null) {
            value = new MyInteger(1);
        } else {
            value.increment();
        }
        maps.put(key, value);
    }
    Set<Map.Entry<MyInteger, MyInteger>> entries = maps.entrySet();
    Iterator<Map.Entry<MyInteger, MyInteger>> iter = entries.iterator();
    while(iter.hasNext()){
        Map.Entry<MyInteger, MyInteger> entry = iter.next();
        System.out.println(String.format("Key %d Value
%d",entry.getKey().getValue(), entry.getValue().getValue()));
    }
}
```



Question 4

The following hash table has space for 101 entries, but only the first 4 cells contain elements; the other cells are empty.

The has function is $\text{hash}(\text{element}) = \text{element} \% 101$

0	202
1	304
2	508
3	707
	.
	.
	.
100	

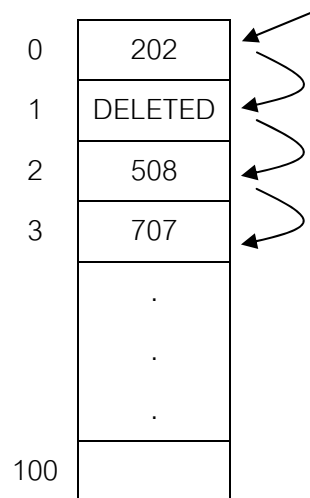
a) Delete element 304 in cell 1 by clearing the cell so it is empty. What happens when you then search for 707? Explain in word why emptying a cell is not a hood way to implement deletion in hash tables.

0	202	↖
1		↖
2	508	
3	707	
	.	
	.	
	.	
100		

Ans $\text{hash}(707) = 0$ when it probe until 1 found empty so it return can't find 707



b) Explain, without the use of code, a better way of representing deletion in hash tables, Illustrate your approach by repeating the 304 deletion and 707 search of pat(a)



c) Use pseudo code to explain how the following three methods would use your deletion approach

- a delete() method that deletes a specified table element;

Ans - start at cell $h(k)$

- probe until

- A cell with key k is found put DELETED , return true
- found empty cell , return false
- n cell have been probed, return false

- a locate() method that tries to locate a given element in the table;

Ans - start at cell $h(k)$

- probe until

- A cell with key k is found put DELETED , return true
- found empty cell , return false
- n cell have been probed, return false

- a insert() method that tries to locate a given element in the table;

Ans - start at cell $h(k)$

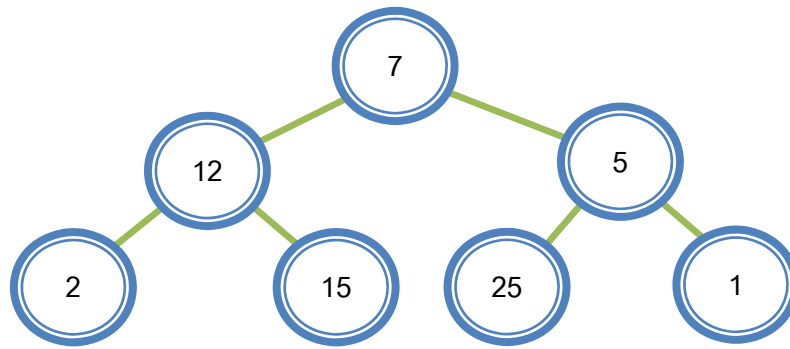
- probe until

- a cell has empty or DELETED put value in cell, return true
- n cell have been probed, return false

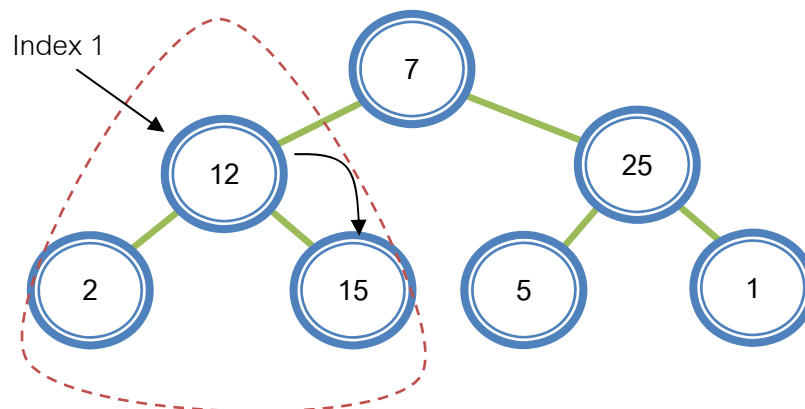
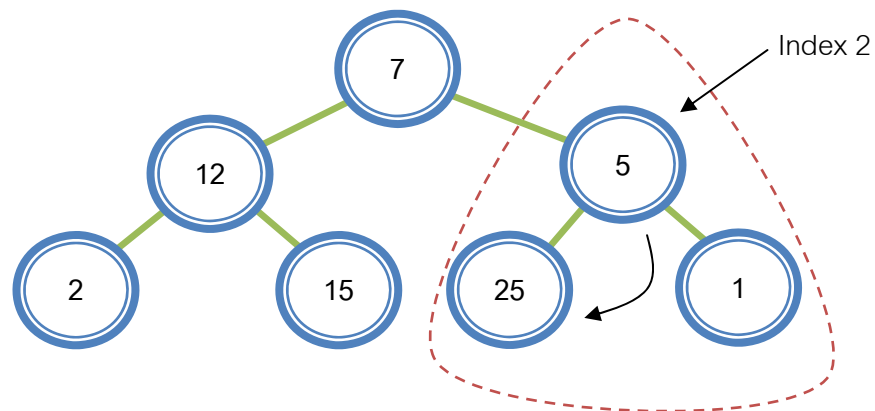
Question 5

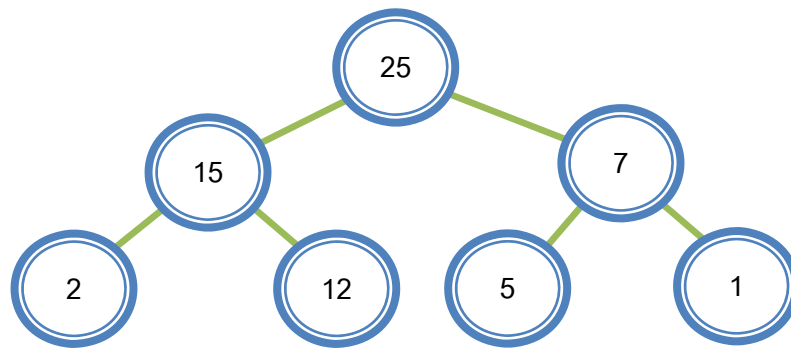
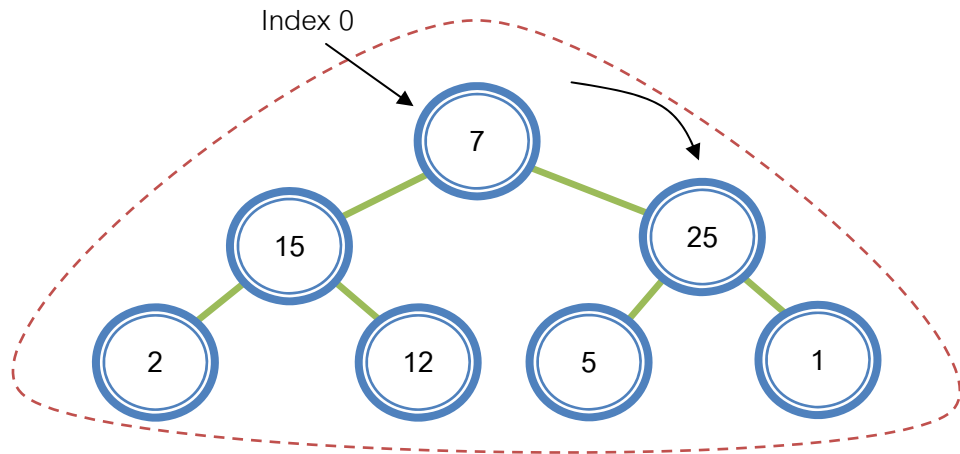
a) Heapify the following array: {7, 12, 5, 2, 15, 25, 1}, creating a max heap. Draw all the stages in the heapification, and summarize what is happening. Also show the resulting array.

Ans convert to BST



adjustHeap() start at array index = $(n-2)/2 = (7-2)/2 = 2$

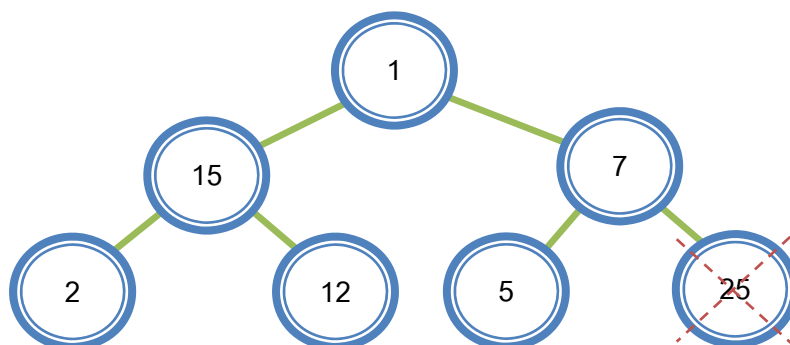


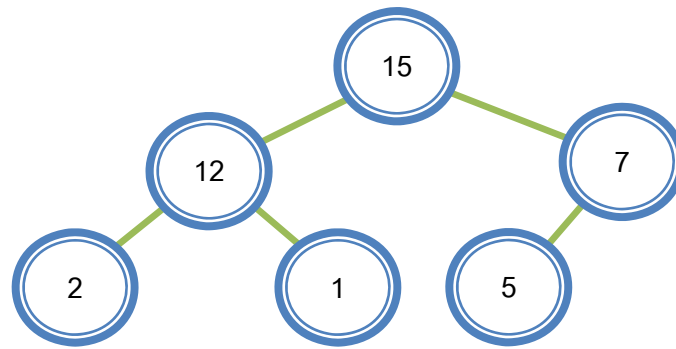


result 25, 15, 7, 2, 12, 5, 1

b) Perform a Heap sort on the array result from part(a). Only draw the first three stages in the sort, where the largest three elements of the array are put into ascending order position. Explain in words what is happening.

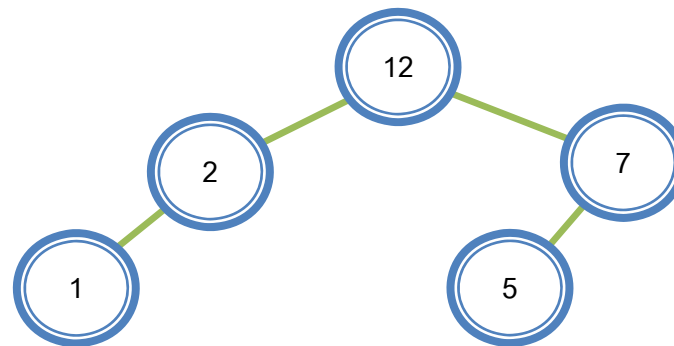
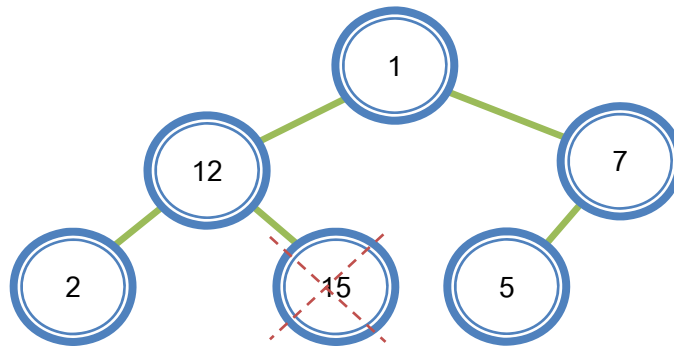
Ans swap 1 and 16 and remove last and adjustHeap()





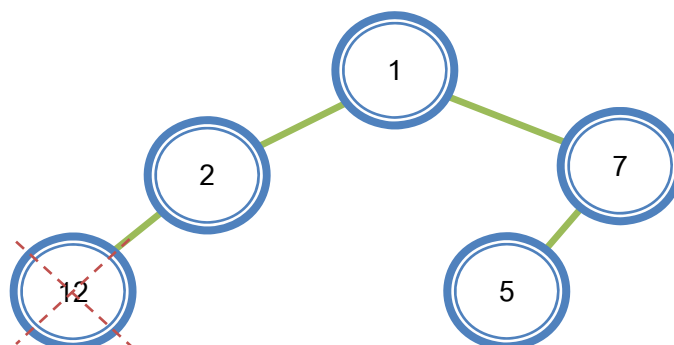
result 25

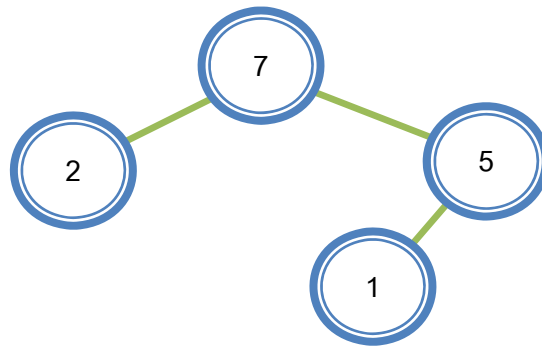
swap 1 and 15 and remove last and adjustHeap()



result 25, 15

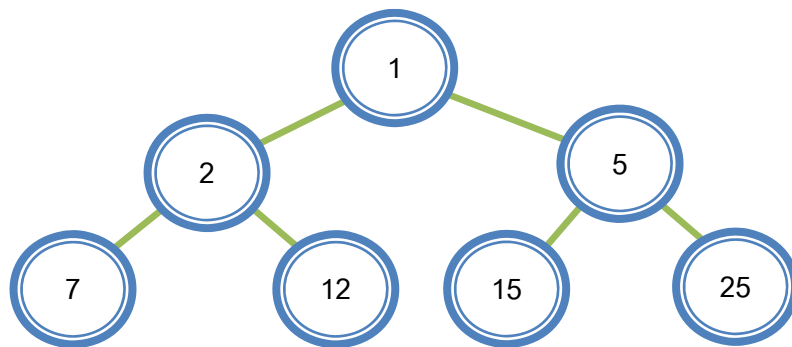
swap 1 and 15 and remove last and adjustHeap()





result 25, 15, 12

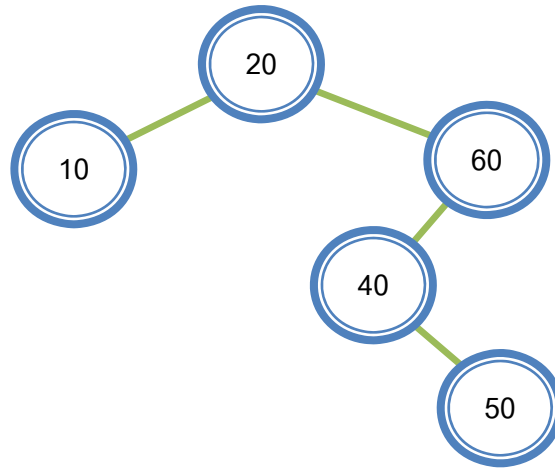
c) Draw the final sorted heap, and the corresponding sorted array. Briefly explain in words the number and types of operations that led to this result



sorted array 1, 2, 5, 7, 12, 15, 25

Question 6

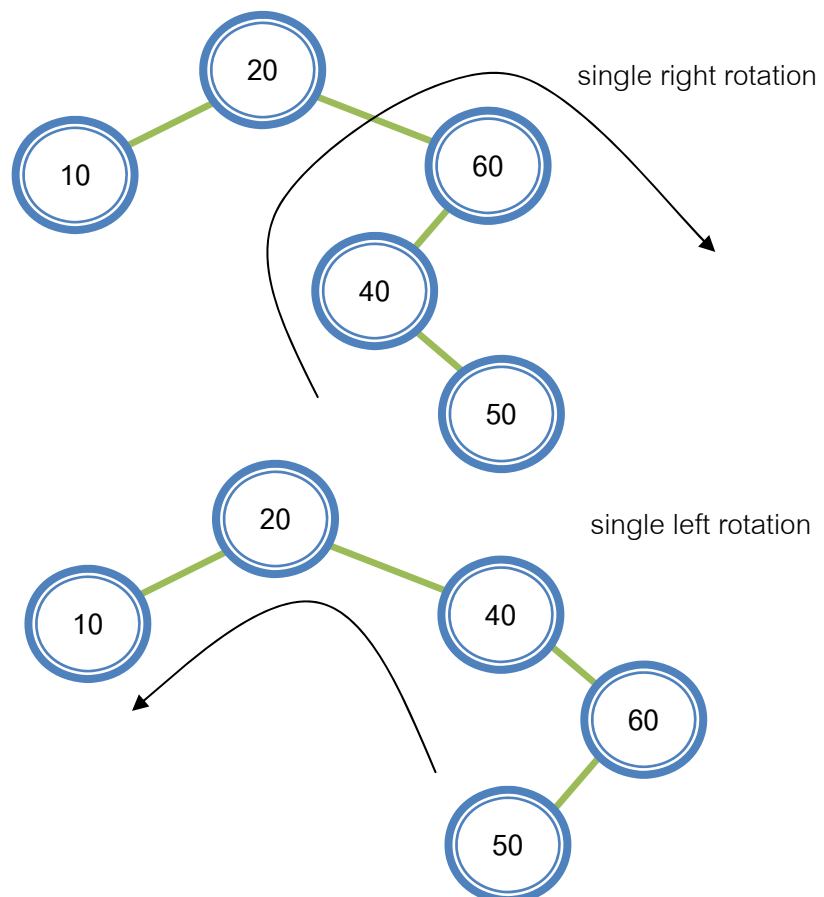
a) Carry out rotations on the following tree to create an AVL tree.

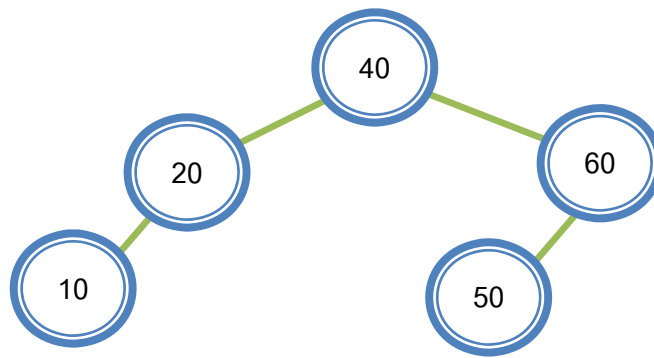


Draw a diagram for each rotation of the tree, and explain each rotation in words

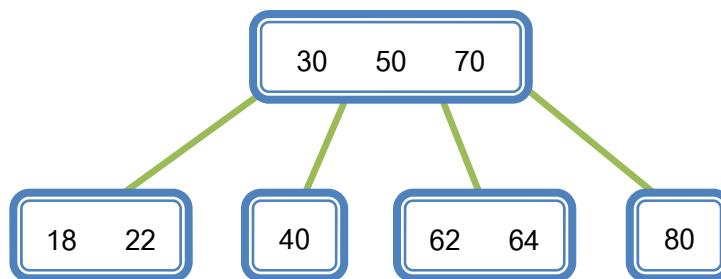
Ans To heavy on the right inside grandchild must double left rotation to make AVL tree

balance

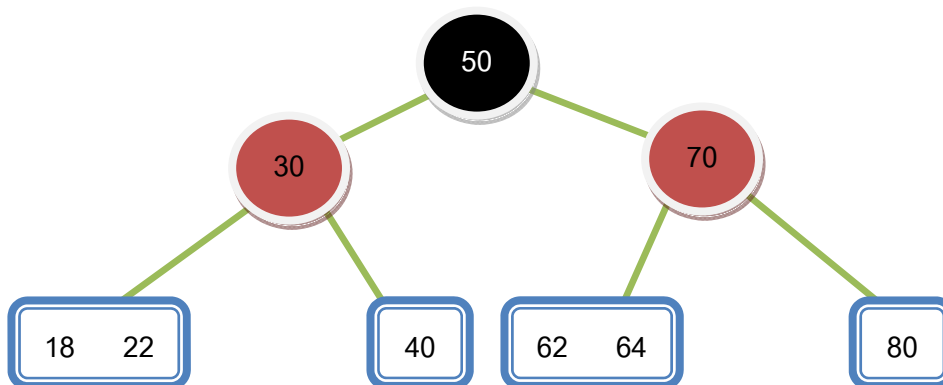




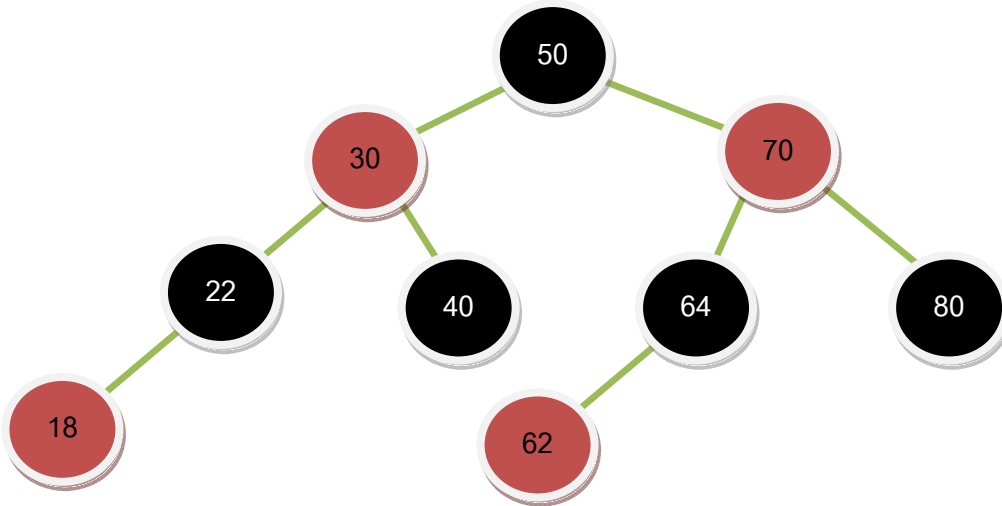
b) Create the red-black tree for the following 2-3-4 tree.



First change root node to RED-BLACK tree by change 50 to root node and 30,70 to children node and set root node to black and children not red



Second change 3-node (8,12) and 3-node(62,64) to RED-BLACK tree by change node 22 to parent and color black , change 18 to left children node and color red, change node 64 to parent and color black, 62 to left children node and color red



c) Explain the necessary of a red-black tree.

- Root is BLACK
- RED parent never has RED child
- Every path from root to empty must have same number of BLACK node

