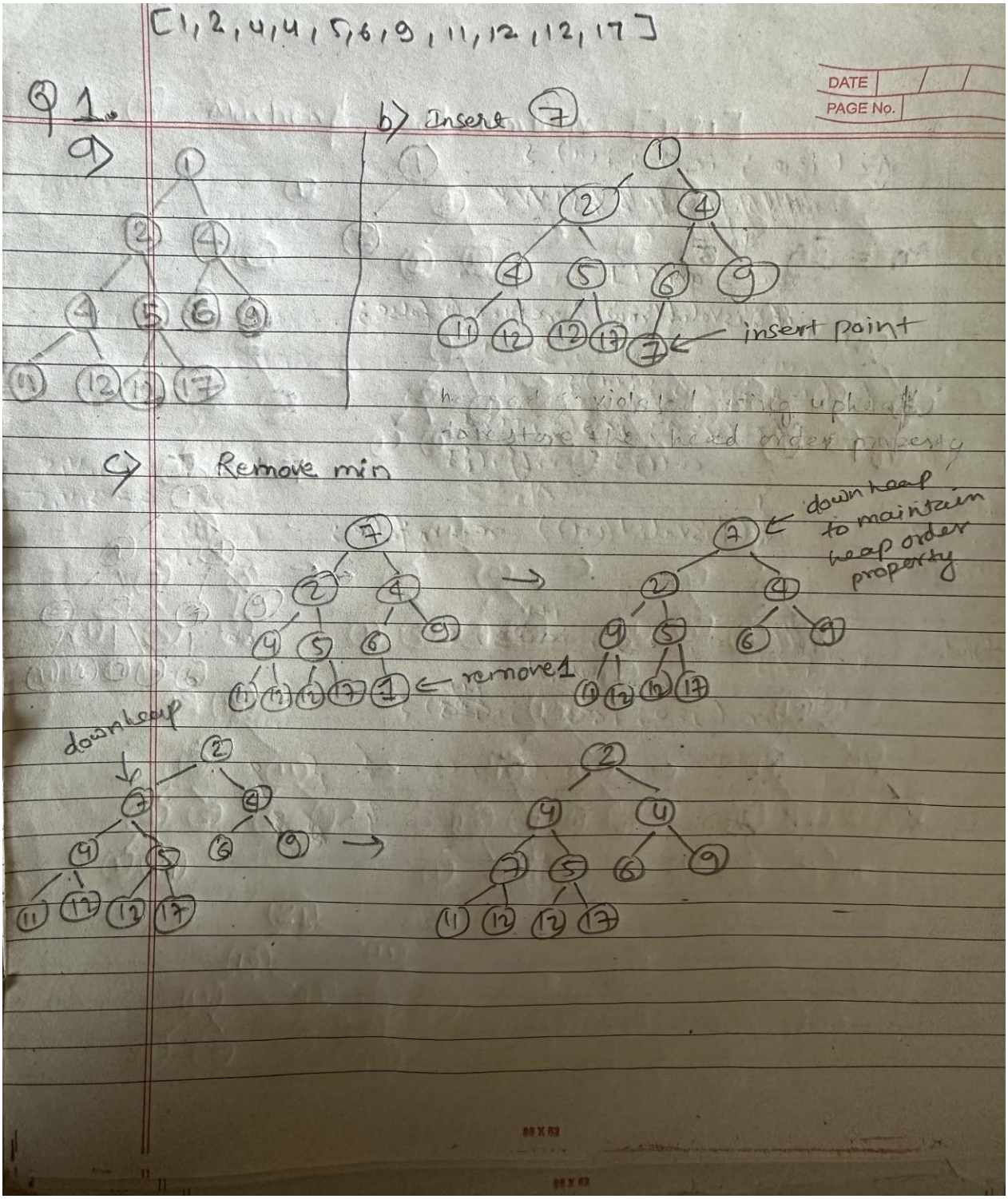
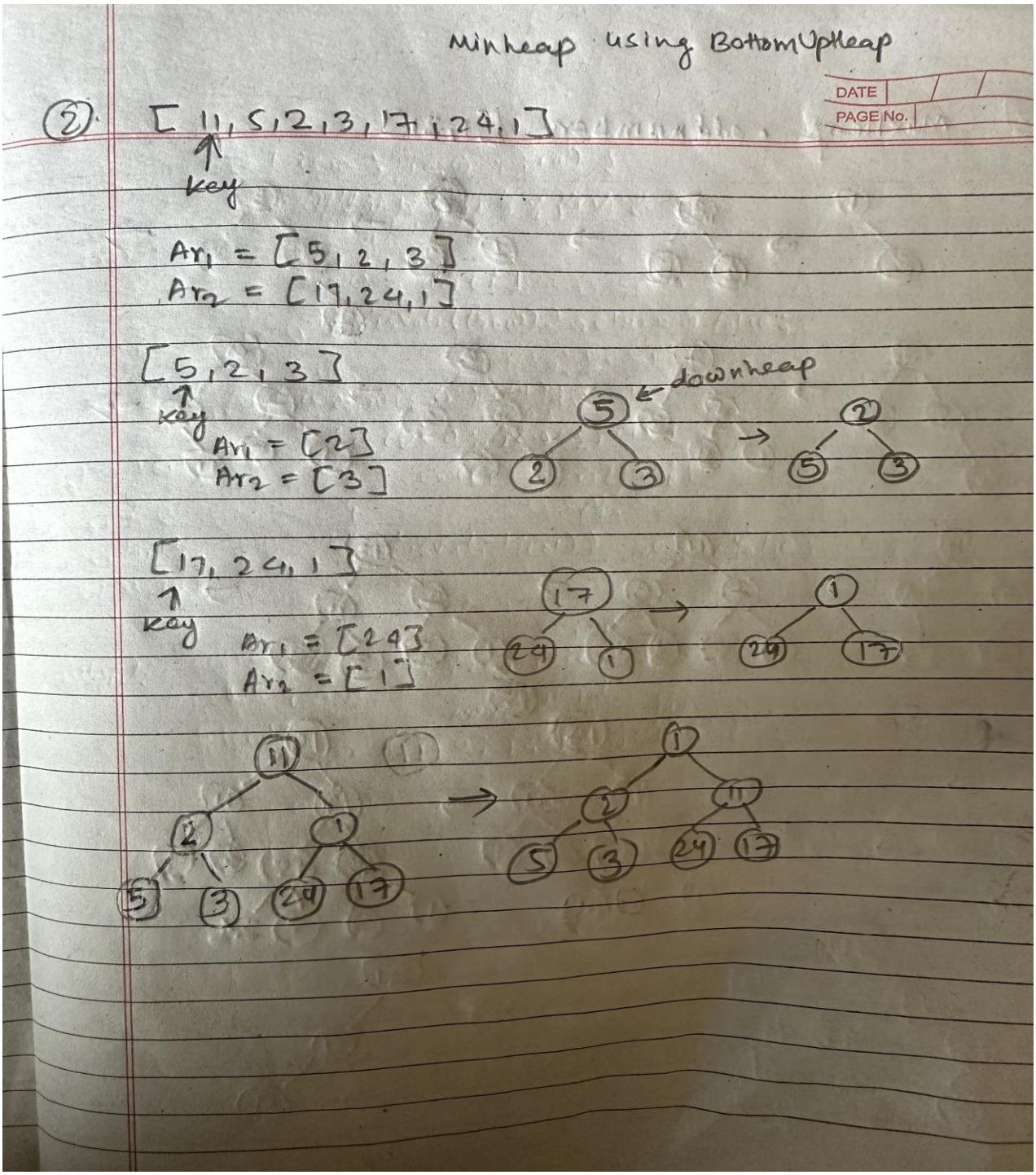
Q1.

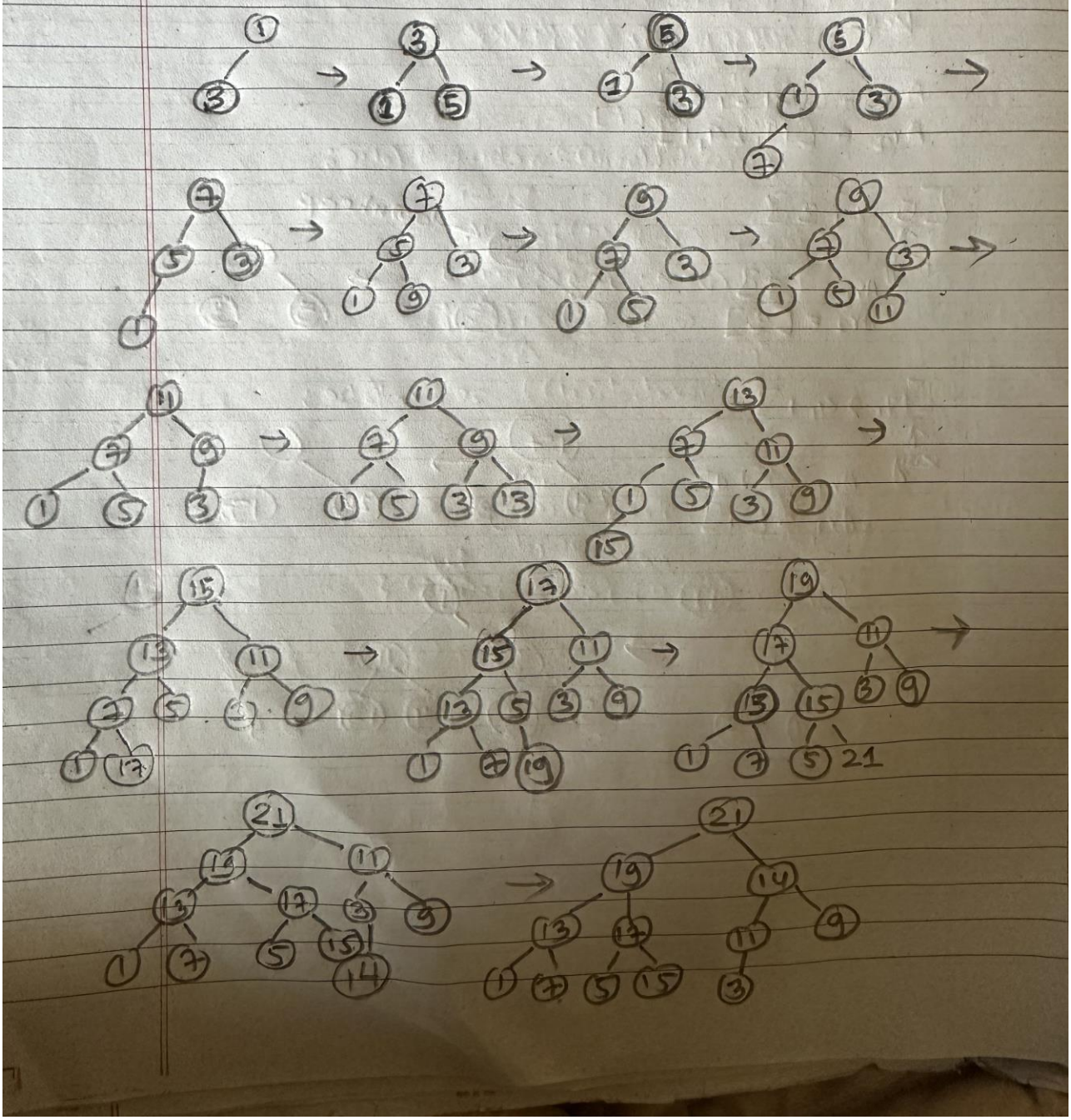

[1, 2, 4, 4, 5, 6, 9, 11, 12, 12, 17]
Q 1.
a)
b) Insert 7
insert point
c) Remove min
remove 1
down heap to maintain heap order property
downheap

Q2.


Minheap using BottomUpheap
DATE
PAGE No.
(2) [11, 5, 2, 3, 17, 24, 1]
key
Ar1 = [5, 2, 3]
Ar2 = [17, 24, 1]
[5, 2, 3]
key
Ar1 = [2]
Ar2 = [3]
downheap
[17, 24, 1]
key
Ar1 = [24]
Ar2 = [1]

Q3.

③ Maxheap . odd numbers [1,21]



Q4.

```
class Solution {
static class Node implements Comparable<Node> {
int[] point;
double distance;
Node(int[] point){
this.point = point;
this.distance = (Math.pow(point[0],2) + Math.pow(point[1],2));
}

public int compareTo(Node obj){
return Double.compare(this.distance, obj.distance);
}
}

PriorityQueue<Node> maxHeap = new PriorityQueue<Node>();
public int[][] kClosest(int[][] points, int k) {
for(int[] arr: points){
maxHeap.add(new Node(arr));
}

int[][] result = new int[k][2];
int length = points.length;
int pos = 0;
for(int i=0;i<k;i++){
result[i] = maxHeap.poll().point;
}
return result;
}
}
```

Q5.

```
class Solution {

public String frequencySort(String s) {
HashMap<Character,Integer> dict = new HashMap<>();
for(char c: s.toCharArray()){
dict.put(c,dict.getOrDefault(c,0)+1);
}

PriorityQueue<Map.Entry<Character,Integer>> queue = new PriorityQueue<>((a,b)->
b.getValue()-a.getValue());
queue.addAll(dict.entrySet());

StringBuilder sb = new StringBuilder();
while(!queue.isEmpty()){
Map.Entry<Character,Integer> item = queue.poll();
for(int i=0;i<item.getValue();i++){
sb.append(item.getKey());
}
}
return sb.toString();
}
}
```