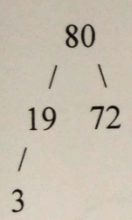
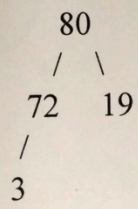
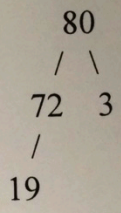
1. From Charn Suppapanya.

1. For set {3, 80, 19, 72}, there are **3 possible max heaps**.

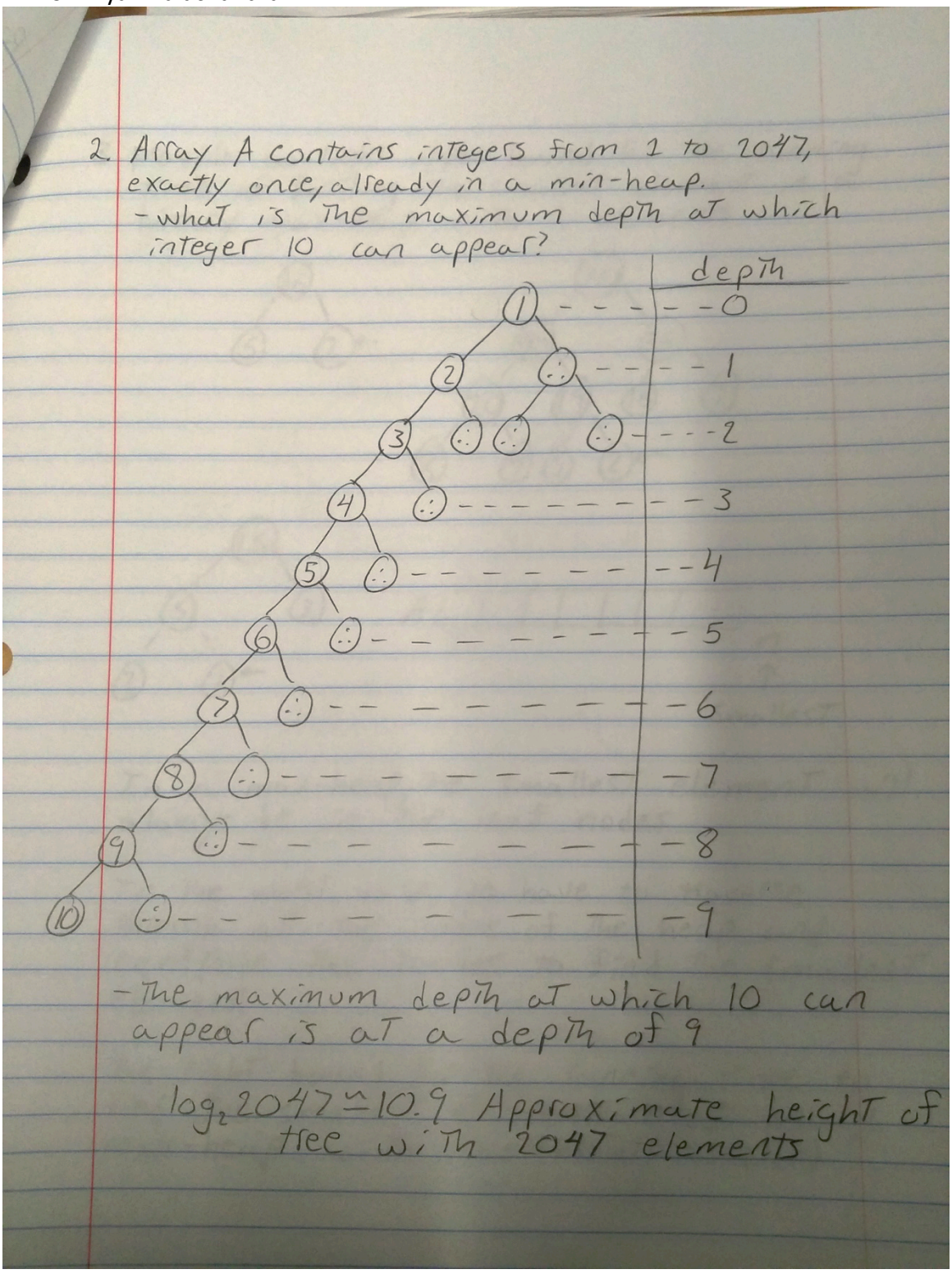




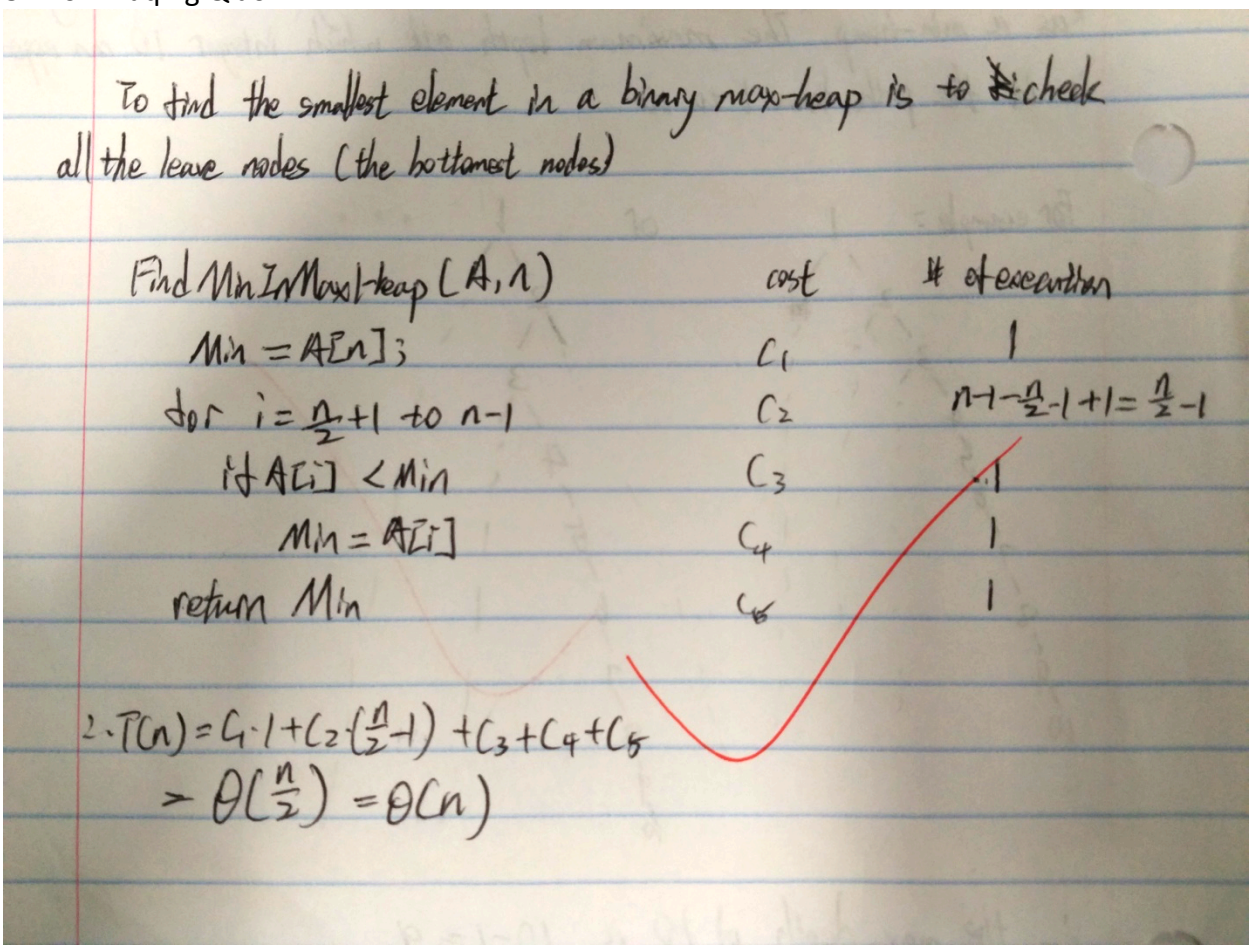


80 is highest, so it must always be the root.
The 3 remaining elements can result in the three heaps above. Any combination of 2 elements (provided that the smaller element is the leaf) can be chosen for the 2 elements in the left sub-tree since they are all smaller than the root, and same goes for the 1 element in the right sub-tree.

2. From Ryan Balachandran.



3. From Yuqing Qiao.



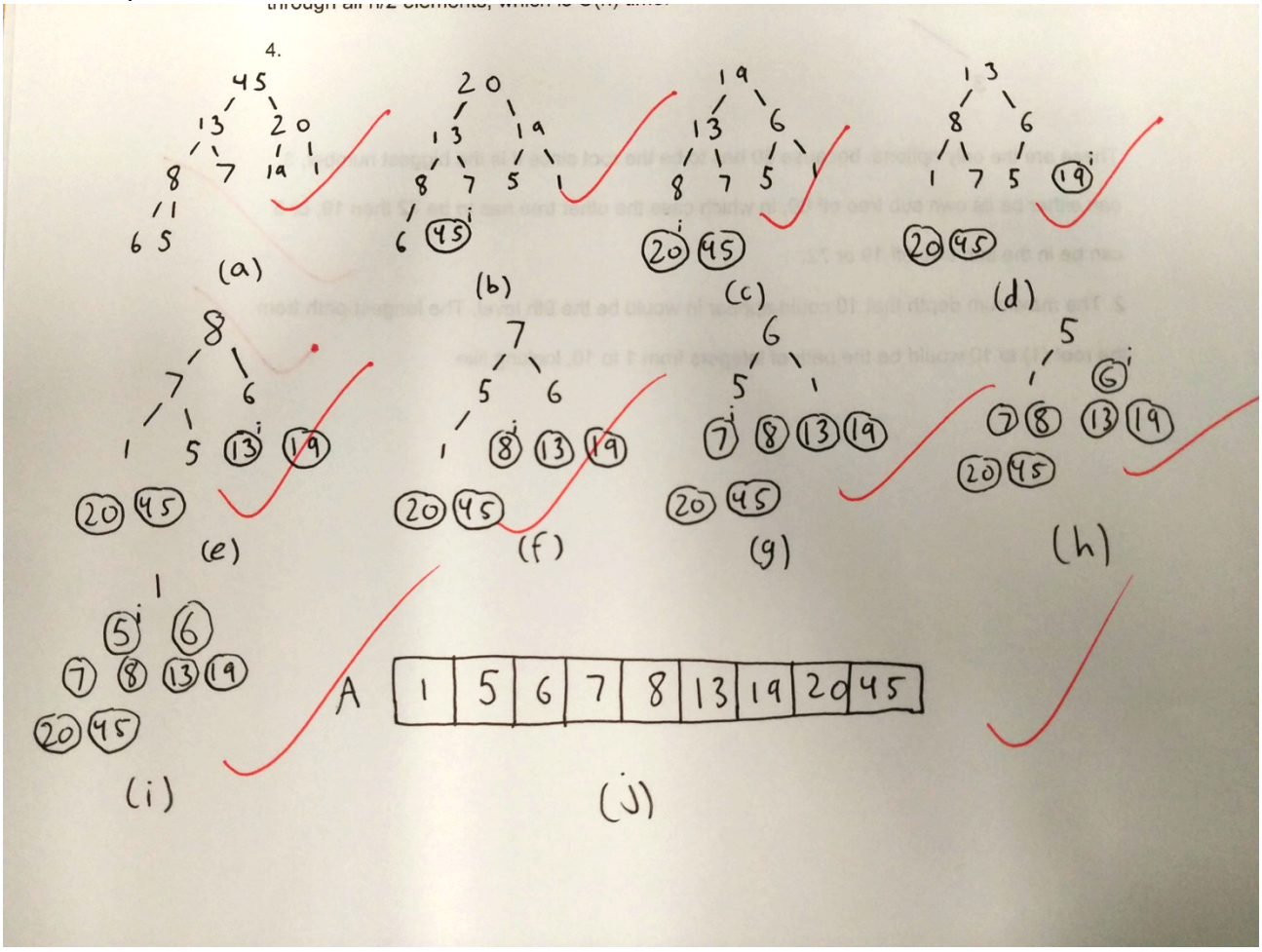
To find the smallest element in a binary max-heap is to check all the leave nodes (the bottomest nodes)

```
Find MinInMaxHeap(A, n)        cost      # of execution
    Min = A[n];                C1        1
    for i = n/2+1 to n-1       C2        n-1-n/2-1+1 = n/2-1
        if A[i] < Min          C3        n/2-1
            Min = A[i]         C4        1
    return Min                 C5        1
```

2. $T(n) = C_1 \cdot 1 + C_2 \cdot (\frac{n}{2} - 1) + C_3 + C_4 + C_5$

$\rightarrow \theta(\frac{n}{2}) = \theta(n)$

4. From Kyle Dokus.



5(a). From Etienne Buhrle.

5 Analysis of d-ary heaps (ex6-2 p167)

(a) The d-ary heap could be represented in memory just like the binary heap, i.e. sequentially, enumerating the nodes in top-down, left-to-right manner.

To get the n-th child ($1 \leq n \leq d$) of the node with index i, we call

$$\text{CHILD}(i, d, n) = di + (1 + n - d)$$

The second term arises because the product di won't correspond to the first child anymore (e.g. in the case of a 3-ary or 4-ary heap), but to the (d-1)-th child.

To get the parent of the node with index i, we call

$$\text{PARENT}(i, d) = \left\lfloor \frac{i + (d - 2)}{d} \right\rfloor$$

For every $1 \leq n \leq d$, $\text{PARENT}(\text{CHILD}(i, d, n), d) = \lfloor (di + n - 1)/d \rfloor = i$.

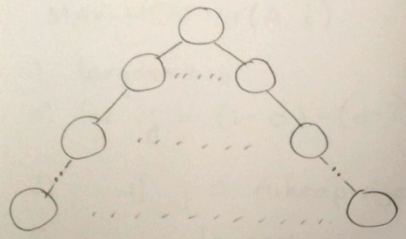
(b) For a d-ary heap with n nodes and height h, we have

$$d^0 + d^1 + \ldots + d^{h-1} = \frac{d^h - 1}{d - 1} < n$$

$$d^0 + d^1 \ldots \quad d^{h+1} - 1$$

5(b). From Saara Luna.

b) Find the height of a d-ary heap in terms of n and d.



nodes:

1

d

d^2

d^i

Total # nodes = $n = 1 + d + d^2 + \ldots + d^i$

$$n = \sum_{k=0}^{i} d^k = \frac{d^{i+1} - 1}{d - 1}$$

Solve for i:

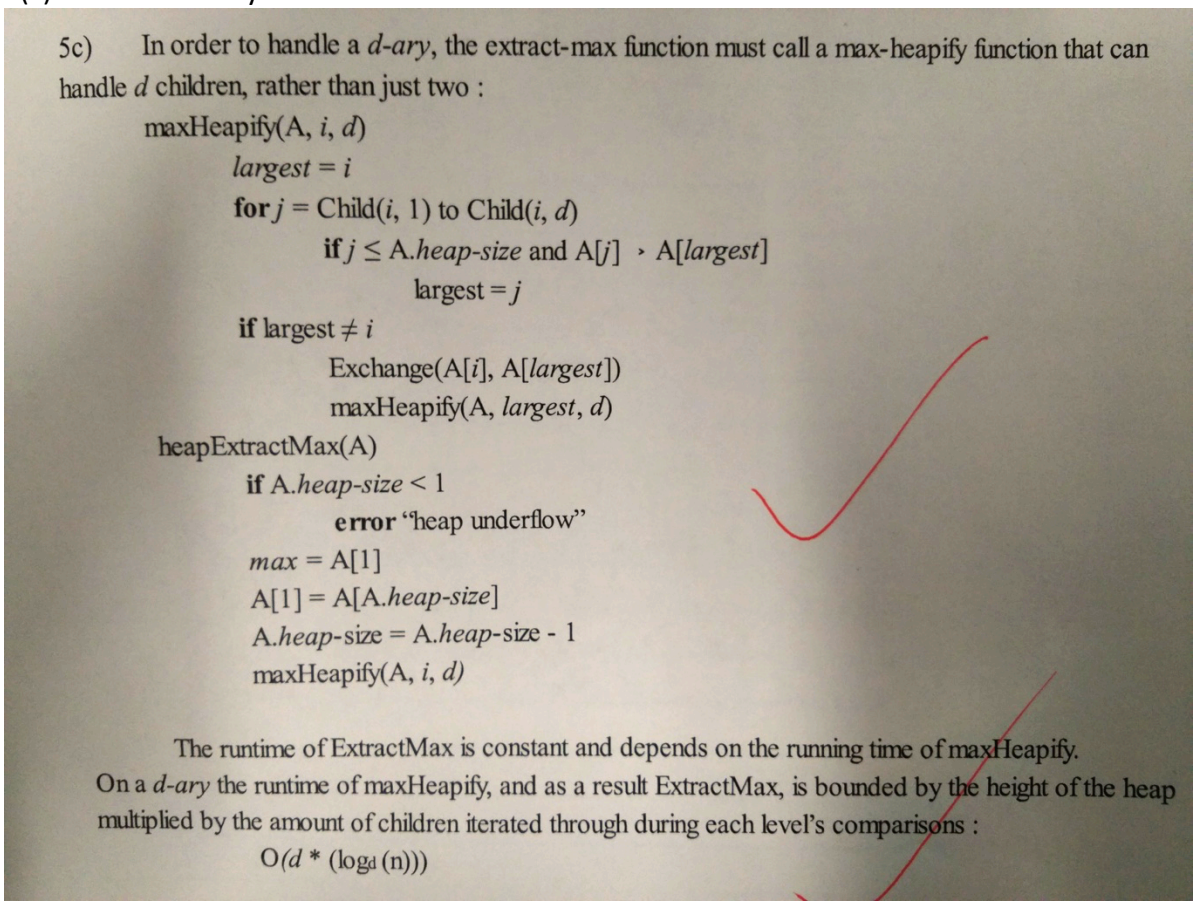
$$n(d-1) = d^{i+1} - 1 = d(d^i) - 1 \rightarrow \frac{n(d-1)+1}{d} = d^i$$

$$\log_d\left(\frac{n(d-1)+1}{d}\right) = \log_d d^i = i = \log_d(n(d-1)+1) - \log_d d$$

$$= \log_d[n(d-1)+1] - 1$$

Therefore, the height is $\boxed{\log_d[n(d-1)+1] - 1} \approx \log_d n$

5(c). From Liam Reynolds.



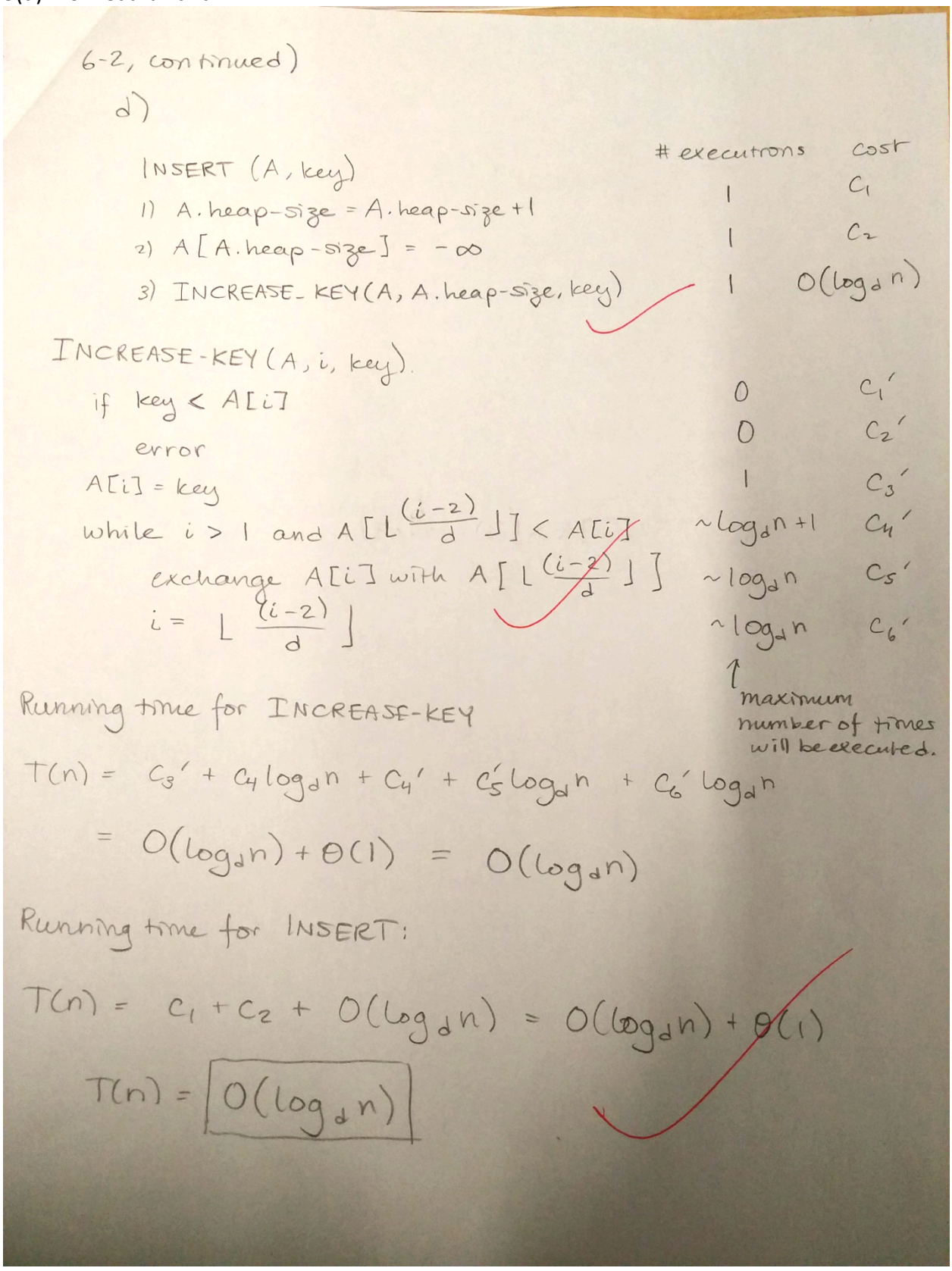
5c) In order to handle a *d-ary*, the extract-max function must call a max-heapify function that can handle *d* children, rather than just two :

```
maxHeapify(A, i, d)
        largest = i
        for j = Child(i, 1) to Child(i, d)
                if j ≤ A.heap-size and A[j] › A[largest]
                        largest = j
        if largest ≠ i
                Exchange(A[i], A[largest])
                maxHeapify(A, largest, d)
heapExtractMax(A)
        if A.heap-size < 1
                error "heap underflow"
        max = A[1]
        A[1] = A[A.heap-size]
        A.heap-size = A.heap-size - 1
        maxHeapify(A, i, d)
```

The runtime of ExtractMax is constant and depends on the running time of maxHeapify. On a *d-ary* the runtime of maxHeapify, and as a result ExtractMax, is bounded by the height of the heap multiplied by the amount of children iterated through during each level's comparisons :

$O(d * (\log_d (n)))$

5(d) From Saara Luna.



6-2, continued)

d)

	# executions	cost
INSERT (A, key)		
1) A.heap-size = A.heap-size + 1	1	c_1
2) A[A.heap-size] = $-\infty$	1	c_2
3) INCREASE_KEY(A, A.heap-size, key)	1	$O(\log_d n)$

INCREASE-KEY(A, i, key)		
if key < A[i]	0	c_1'
error	0	c_2'
A[i] = key	1	c_3'
while i > 1 and $A[\lfloor \frac{(i-2)}{d} \rfloor] < A[i]$	$\sim \log_d n + 1$	c_4'
exchange A[i] with $A[\lfloor \frac{(i-2)}{d} \rfloor]$	$\sim \log_d n$	c_5'
$i = \lfloor \frac{(i-2)}{d} \rfloor$	$\sim \log_d n$	c_6'

↑ maximum number of times will be executed.

Running time for INCREASE-KEY

$$T(n) = c_3' + c_4' \log_d n + c_4' + c_5' \log_d n + c_6' \log_d n$$

$$= O(\log_d n) + \Theta(1) = O(\log_d n)$$

Running time for INSERT:

$$T(n) = c_1 + c_2 + O(\log_d n) = O(\log_d n) + \Theta(1)$$

$$T(n) = \boxed{O(\log_d n)}$$