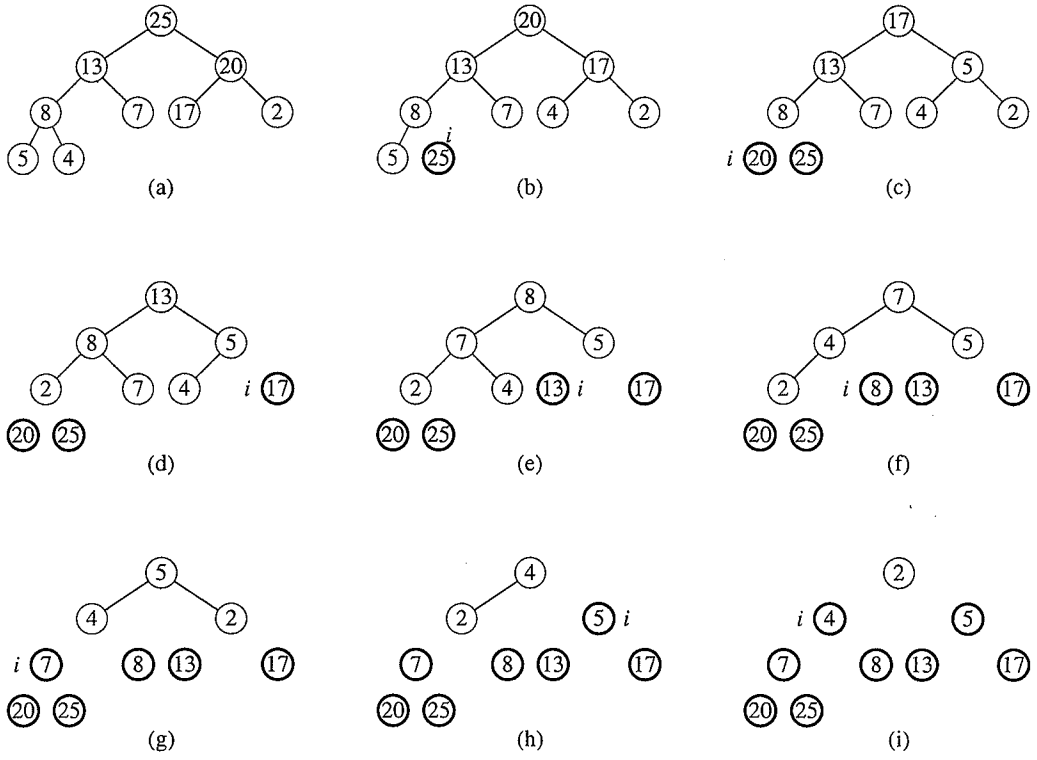
Solution to Exercise 6.4-1



A	2	4	5	7	8	13	17	20	25

Solution to Exercise 6.1-2

Given an n-element heap of height h, we know from Exercise 6.1-1 that
$2^h \le n \le 2^{h+1} - 1 < 2^{h+1}$.
Thus, $h \le \lg n < h + 1$. Since h is an integer, $h = \lfloor \lg n \rfloor$ (by definition of $\lfloor\ \rfloor$).

d. The procedure MAX-HEAP-INSERT given in the text for binary heaps works fine for d-ary heaps too. The worst-case running time is still $\Theta(h)$, where h is the height of the heap. (Since only parent pointers are followed, the number of children a node has is irrelevant.) For a d-ary heap, this is $\Theta(\log_d n) = \Theta(\lg n / \lg d)$.

e. D-ARY-HEAP-INCREASE-KEY can be implemented as a slight modification of MAX-HEAP-INSERT (only the first couple lines are different). Increasing an element may make it larger than its parent, in which case it must be moved higher up in the tree. This can be done just as for insertion, traversing a path from the increased node toward the root. In the worst case, the entire height of the tree must be traversed, so the worst-case running time is $\Theta(h) = \Theta(\log_d n) = \Theta(\lg n / \lg d)$.

```
D-ARY-HEAP-INCREASE-KEY(A, i, k)
A[i] ← max(A[i], k)
while i > 1 and A[PARENT(i)] < A[i]
    do exchange A[i] ↔ A[PARENT(i)]
       i ← PARENT(i)
```

Solution to Problem 6-2

a. A d-ary heap can be represented in a 1-dimensional array as follows. The root is kept in $A[1]$, its d children are kept in order in $A[2]$ through $A[d+1]$, their children are kept in order in $A[d+2]$ through $A[d^2+d+1]$, and so on. The following two procedures map a node with index i to its parent and to its jth child (for $1 \le j \le d$), respectively.

```
D-ARY-PARENT(i)
return ⌊(i − 2)/d + 1⌋
```

```
D-ARY-CHILD(i, j)
return d(i − 1) + j + 1
```

To convince yourself that these procedures really work, verify that

D-ARY-PARENT(D-ARY-CHILD$(i, j)) = i$,

for any $1 \le j \le d$. Notice that the binary heap procedures are a special case of the above procedures when $d = 2$.

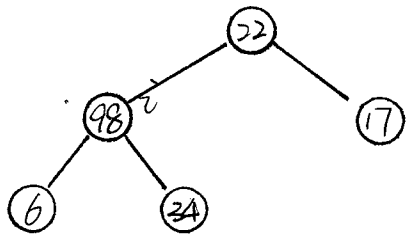
b. Since each node has d children, the height of a d-ary heap with n nodes is $\Theta(\log_d n) = \Theta(\lg n / \lg d)$.

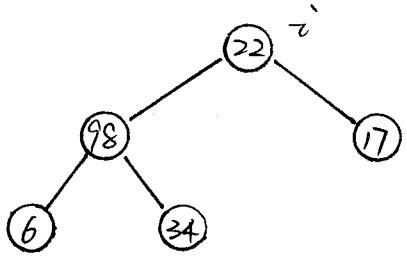
c. The procedure HEAP-EXTRACT-MAX given in the text for binary heaps works fine for d-ary heaps too. The change needed to support d-ary heaps is in MAX-HEAPIFY, which must compare the argument node to all d children instead of just 2 children. The running time of HEAP-EXTRACT-MAX is still the running time for MAX-HEAPIFY, but that now takes worst-case time proportional to the product of the height of the heap by the number of children examined at each node (at most d), namely $\Theta(d \log_d n) = \Theta(d \lg n / \lg d)$.

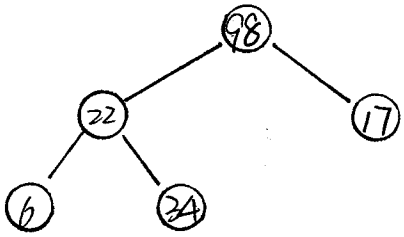
HW4.(4) Solution: No, it is not a MAX-HEAP.
Because it is not a binary tree.

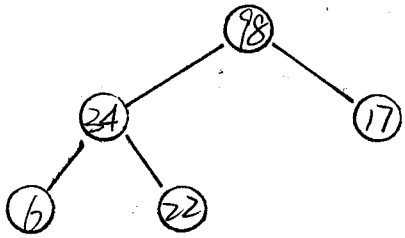
The time for insert key is also $O(\lg n)$ the algorithm of insert key is only add a constant time to increase key algorithm.

Problem 4
Use Figure 6.3 Build-Max-Heap





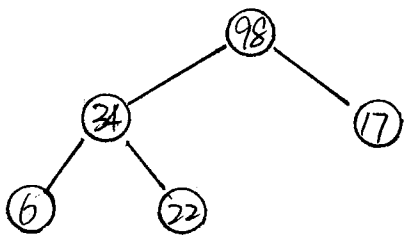


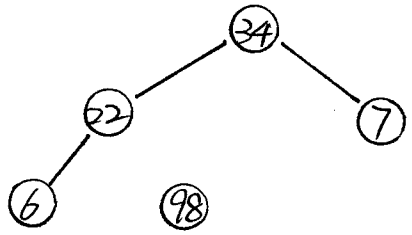


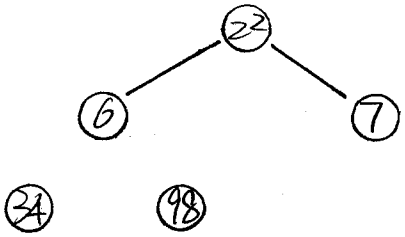
swap : $98 \leftrightarrow 22$ $(A[2] \leftrightarrow A[1])$

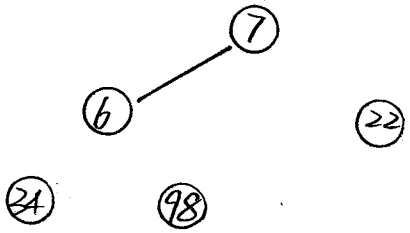
$22 \leftrightarrow 34$ $(A[2] \leftrightarrow A[5])$

Use Figure 6.4 Heapsort



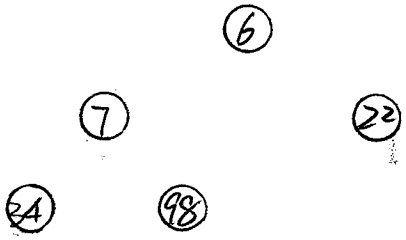






A:

6	7	22	34	98



swap:

98 ↔ 22 (A[1] ↔ A[5])

22 ↔ 34 (A[1] ↔ A[2])

34 ↔ 6 (A[1] ↔ A[4])

6 ↔ 22 (A[1] ↔ A[2])

22 ↔ 7 (A[1] ↔ A[3])

7 ↔ 6 (A[1] ↔ A[2])

Total: 8 swaps