1 Chapter 6

Question 6.1 Can both `insert` and `findMin` be implemented in constant time?
Solution.
Yes. Consider the following data structure.

```
class data_structure_1
{
  public:
    data_structure_1();
    insert(int X);
    findMin();

  private:
    int min;                       \\stores the minimum element
    list<int> L_1;                 \\linked list of integers
};

data_structure_1::data_structure_1()
   {
     min = ∞ ;
     L_1.makeEmpty();
   }

data_structure_1::insert(int X)
  {
    if ( X < min)
      min = X;

    L_1.insert(X);
  }

data_structure_1::findMin()
  {
    return min;
  }
```

Question 6.2 a. Show the result of inserting 10, 12, 1, 14, 6, 5, 8, 15, 3, 9, 7, 4, 11, 13, and 2 one at a time into an initially empty heap.
Solution.

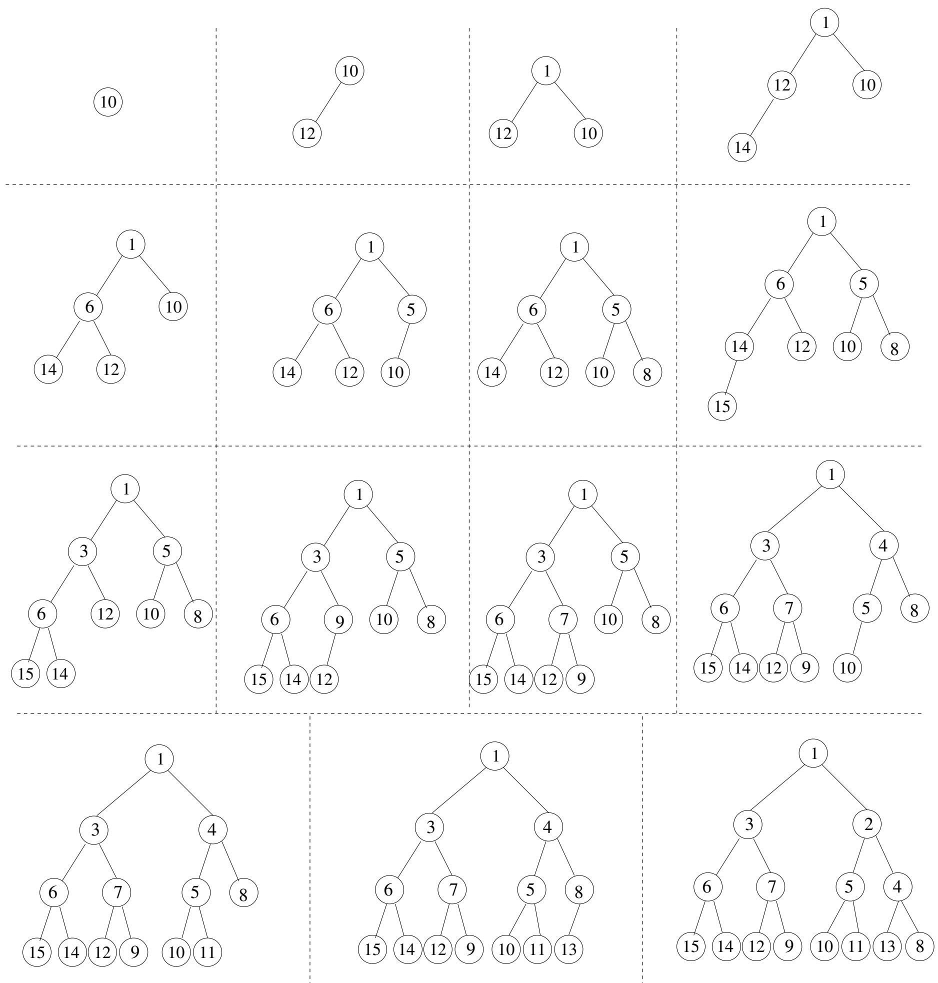

Fig. 1.

Question 6.2 b. Show the result of using the linear-time algorithm to build a binary heap using the same input.
Solution.

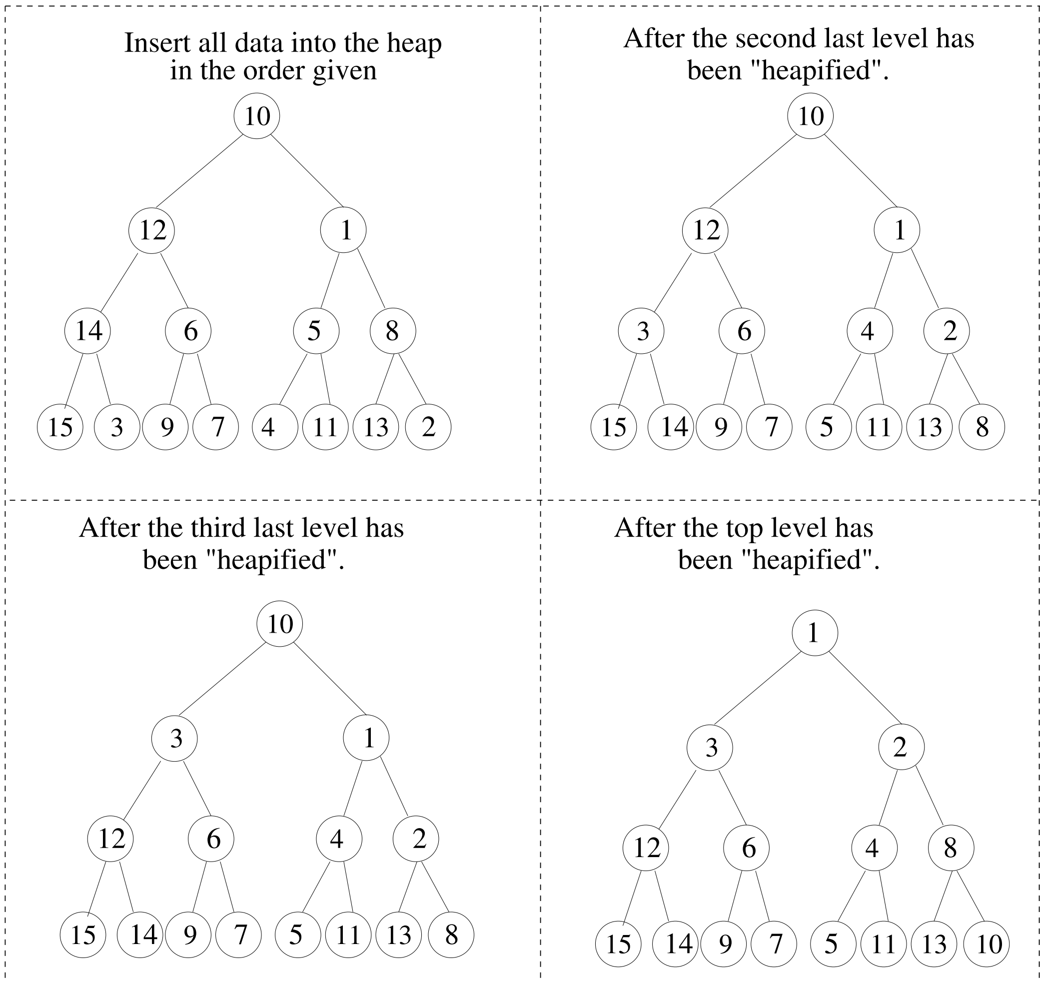

Fig. 2.

Question 6.3 Show the result of performing three `deleteMin` operations in the heap of the previous.

Solution. We provide the solution for part a. The solution to part b is similar.

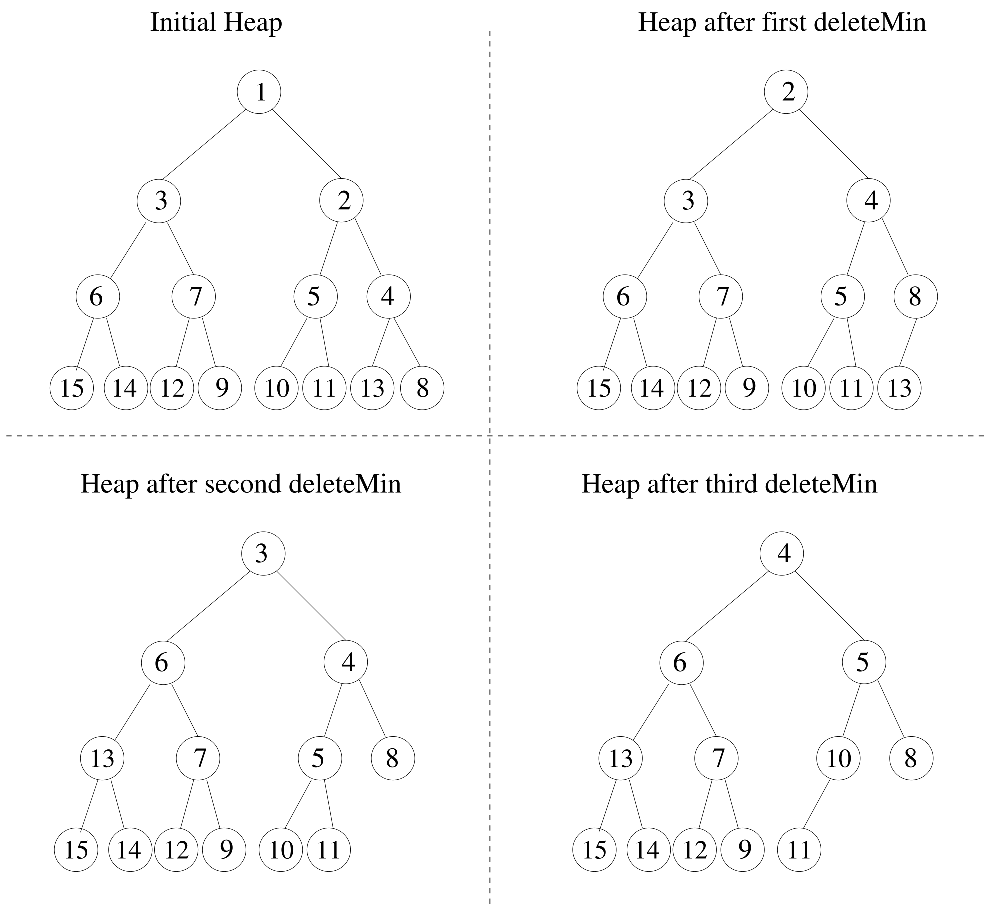

Fig. 3.

Question 6.6 How many nodes are in the large heap in Figure 6.13?

Solution. We use the fact that if a node is in position i then its children are in position $2i$ and $2i + 1$. Follow a path from the root (which is at position 1) to the node in the last position, doubling every time you follow a left child and doubling and adding one every time you follow a right child.

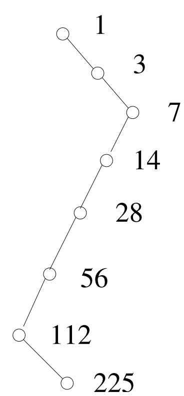

Fig. 4. The path from the root to the node in the last position.

Q 6.14 If a d-heap is stored as an array, for an entry located in position i

a) Where are the children of i?

b) Where are the parents of i?

Solution. a) Assume that position i corresponds to the Xth node of level l. Therefore

$$i = \sum_{j=0}^{l-1} d^j + X \tag{1}$$

$\sum_{j=0}^{l-1} d^j$ is a geometric series whose first term equals 1, whose common ratio is d, and that contains l terms in total. Using the formula for summing a geometric series that we learned at the start of the module we get.

$$\sum_{j=0}^{l-1} d^j = \frac{d^l - 1}{d - 1}.$$

Substituting this value into Equation 1 gives.

$$i = \frac{d^l - 1}{d - 1} + X$$

We now calculate the position of i's second last child in terms of d, l, and X. This equals i, plus the number of nodes after i on level l, plus d times the number of nodes before i on level l, plus $d - 1$.

$$\begin{aligned}
&= \frac{d^l - 1}{d - 1} + X + d^l - X + (X - 1)d + d - 1 \\
&= \frac{d^l - 1}{d - 1} + X + d^l - X + dX - d + d - 1 \\
&= \frac{d^l - 1}{d - 1} + X + d^l - X + dX - 1 \\
&= \frac{d^l - 1}{d - 1} + d^l - 1 + dX \\
&= \frac{d(d^l - 1)}{d - 1} + dX \\
&= d(\frac{(d^l - 1)}{d - 1} + X) \\
&= di
\end{aligned}$$

Therefore the second last child of i is in position id. It follows that the children of i are in positions $id - (d - 2), \ldots, id + 1$

b) A node is a child of i if and only if it is in one of the positions $id-(d-2),\ldots,id+1$. So what you want here is a function that will map each of these to i, but will not map any other value to i. Let j be any of these values. Clearly,

$$\lfloor\frac{j+(d-2)}{d}\rfloor = i$$

But if j is greater than $id+1$ or less than $id-(d-2)$ then

$$\lfloor\frac{j+(d-2)}{d}\rfloor \neq i$$

Thus we have our function which can now be used to work out the position of the parent of i.

$$\lfloor\frac{i+(d-2)}{d}\rfloor$$

Question 6.15 Suppose that we need to perform M `percolateUps` and N `deleteMins` on a d-heap that initially has N elements.

a. What is the total running time of all operations in terms of M,N, and d?
b. If $d = 2$, what is the running time of all heap operations?
c. If $d = \theta(N)$, what is the running time of all heap operations?

Solution.

a. A `percolateUp` operation on a d-heap with N elements takes $O(\log_d N)$ steps. A `deleteMin` operation on a d-heap with N elements takes $O(d \log_d N)$ steps. Thus in total this will take $O(M \log_d N + N d \log_d N)$ steps.

b. Substituting 2 into the formula calculated in part a gives $O((M+N)\log_2 N)$.

c. If $d = \theta(N)$ then $d = cN$, where c is a constant value independent of N. Substituting cN into the formula calculated in part a gives:

$$M \log_{cN} N + NcN \log_{cN} N = O(M + N^2)$$