Heaps and Balanced Trees

Dr. Marek A. Suchenek ©

April 24, 2011

0.1 Binary representations of positive natural numbers

How many bits are needed to represent a number $M > 0$ in binary? Let's say it's n. We have:

$$M \leq \underbrace{11\ldots1}_{n}$$

$$\underbrace{11\ldots1}_{n}+1 = 1\underbrace{00\ldots0}_{n} = 2^n$$

So,

$$\underbrace{11\ldots1}_{n} = 2^n - 1$$

or

$$M \leq 2^n - 1$$

or

$$M + 1 \leq 2^n$$

or

$$\log_2(M+1) \leq n$$

or

$$\lceil \log_2(M+1) \rceil \leq n$$

or

$$\lfloor \log_2 M \rfloor + 1 \leq n.$$

So, the smallest n that is large enough is $\lfloor \log_2 M \rfloor + 1$; that is how bits are needed to represent number $M > 0$ in binary.

Exercise. Do you see a set of binary sequences on Figure 1? Do you see a complete binary tree there?

Figure 1: Do you see a set of binary sequences and a complete binary tree here?

0.2 Heaps

A heap may be defined as a contiguous, partially ordered binary tree. *Contiguous* means that all levels of the tree in question, except , perhaps, for the last level, contain the maximum number of nodes, and if the last level of the tree contains a lesser number of nodes then they all are flushed as much to the left as possible.

Figure 2 visualizes an example of heap with 17 nodes. (A heap so large that one could not see the individual nodes and edges on a picture that fits on one page may look like one presented on Figure 3.) It shows nodes' ordinal numbers in decimal. Their binary representations are of the form: 1 followed by a sequence of edges' labels along the path from the root to the node in

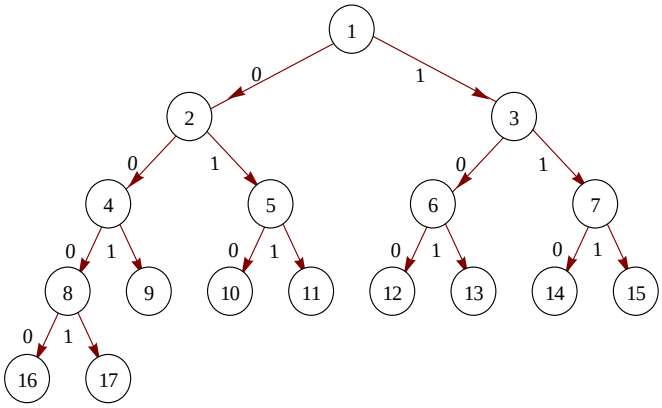

Figure 2: A heap with 17 nodes showing nodes' ordinal numbers in decimal.

question. For instance, the sequence of labels along the path from the root to node number 17 is 0001 and the binary representation of 17 is 10001. The depth of the node number 17, defined as the length of path from the root to that node, is 4 and may be computed as one less than the length of the binary representation of 17, that is, $\lfloor \log_2 17 \rfloor$. Since it is the last node of the heap, it is also the depth of the heap. (We will comment more on this later.)

Figure 3: A really large heap on a small picture.

In addition to providing navigation information, labels of the edges indicate orientation of children: an edge labeled with 0 points to the left child and one labeled with 1 points to the right child. It so happens that the children of node i are $2i$ and $2i + 1$, as it has been visualized on Figure 4.

This important fact may be easily established by looking at binary representations of nodes' ordinal numbers. Since each such representation is a

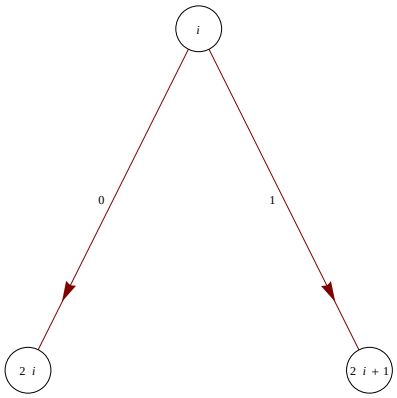

Figure 4: Ancestral information translated onto ordinal numbers.

sequence of bits that determine the path from the root to the node in question, the binary representation of a parent node is a result of truncating the last bit from the binary representation of any of its children. Truncating the last bit yields the same result as the `shiftright` operation, that performs the integer division by 2. So, if j is the ordinal number of a child then the ordinal number i of its parent is

$$i = \lfloor \frac{j}{2} \rfloor,$$

or, in other words,

$$j = \begin{cases} 2i \text{ if } j \text{ is the left child of } i \\ \\ 2i + 1 \text{ if } j \text{ is the right child of } i. \end{cases} \tag{1}$$

For example, the path from the root to node 13 in the heap on Figure 2, is 101 which can be obtained from the binary representation of 13, that is, from 1101, by dropping its first digit 1. So the path to the parent of 13 is 10 and the ordinal number of the node at the end of that path (the parent of 13) is 110, or in 6 in decimal. So, 6 is the parent of 13. Of course, $6 = \lfloor \frac{13}{2} \rfloor$.

Also, the children of 6, if it has any, must have ordinal numbers that in binary read 1100 and 1101 since these are the only numbers that when divided by 2 will yield 110 or 6. These are 12 and 13.

In a similar fashion one can determine if a node i has a child of children by comparing $2i$ to n. If $2i \leq n$ then i has a child or children and if $2i > n$ then it has not (is a leaf, that is). If $2i = n$ then $2i + 1 > n$ and so node i does not have the right child. If $2i < n$ then $2i + 1 \leq n$ and so node i does have the right child.

Partially ordered means that every sequence of nodes along a path from the root to a leaf in the tree is ordered in a non-increasing order. Or, in other words, that children, if any, of a node are not larger than their parent.

Figure 5 visualizes an example of a heap with 10 nodes with the values of the nodes shown instead of their ordinal numbers.

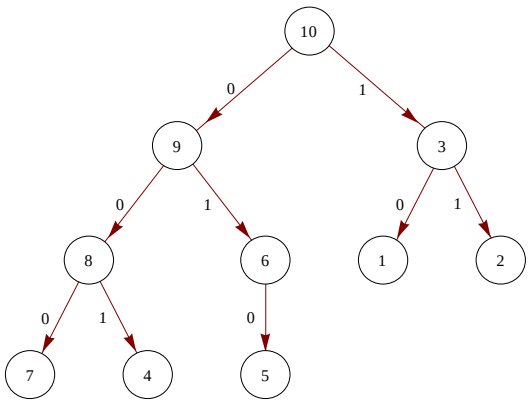

Figure 5: A heap with 10 nodes showing their values and not the ordinal numbers.

Contiguous trees are easy to represent with one-dimensional arrays that store the nodes of the tree according to their level-by-level order. Naturally, the root of the tree is stored at index 1, and the children, if any, of a node stored at index i are stored at indicies $2i$ (the left child) and $2i+1$ (the right child).

The table in Figure 6 shows an array that represents the heap of Figure 2 with the indicies of the array shown in the top row of the table.

1	2	3	4	5	6	7	8	9	10
10	9	3	8	6	1	2	7	4	5

Figure 6: Array representation of the heap of Figure 5.

0.3 The height (or depth) of a heap

Each node of of a heap with n nodes is represented by a binary sequence (a path from the root of the heap to that particular node). The depth of the heap is equal to the maximal (over all nodes of the heap) length of such a path. Since the last node in the heap belongs to the last level of the heap, the length of the path from the root to that last node is maximal.

Let p be the path from the root to the last node of the heap. As we noticed before, the binary representation of that node's ordinal number (n, that is) is 1 followed by p.

In other words, the depth D_n of the heap with n nodes, which is equal to the length of p, is one less than the number of bits needed to represent n. So

$$D_n = \lfloor \log_2 n \rfloor.$$

Exercise Show that a heap with l leaves (not nodes) has a depth D that satisfies

$$\lceil \log_2 l \rceil \leq D \leq \lfloor \log_2 l \rfloor + 1.$$

Note Since the depth of a heap with n nodes is also the level of node n, it follows that the level of node i is:

$$level(i) = \lfloor \log_2 i \rfloor.$$

To see why, remove from the heap all the nodes after i. The resulting heap will have i nodes so its height is $\lfloor \log_2 i \rfloor$, which (by the definition of the height of a heap) happens to be the same as the level of i.

0.4 The running time of H.remove() and H.insert(x)

Let H be a heap with n nodes. The number of comparisons $C_{remove}(n)$ done in the worst case by H.remove() is less equal to 0 if $n = 1$ or equal to $2 \times D_{n-1}$

or $2 \times D_{n-1} - 1$ otherwise, where D_{n-1} is the depth of the heap with $n-1$ nodes (after removal, that is), and the number of comparisons $C_{insert}(n)$ done in the worst case by H.insert(x) is less than or equal to D_{n+1}, where D_{n+1} is the depth of the heap with $n+1$ nodes (after inserting of x, that is).

So,

$$2 \times \lfloor \log_2(n-1) \rfloor - 1 \leq C_{remove}(n) \leq 2 \times \lfloor \log_2(n-1) \rfloor \text{ for } n > 1,$$

and

$$C_{insert}(n) = \lfloor \log_2(n+1) \rfloor.$$

Therefore,

$$C_{remove}(n) \in \Theta(\log n) \text{ and } C_{insert}(n) \in \Theta(\log n).$$

0.5 The running time of PriorityQueueSort

Assume that the array to be sorted has n elements. The number of comparisons in the first for-loop is

$$\Sigma_{i=0}^{n-1} C_{insert}(i) = \Sigma_{i=0}^{n-1} \lfloor \log_2(i+1) \rfloor$$

and the number of comparisons in the second for-loop is

$$\Sigma_{i=2}^{n}(2 \times \lfloor \log_2(i-1) \rfloor - 1) \leq \Sigma_{i=2}^{n} C_{remove}(i) \leq \Sigma_{i=2}^{n} 2 \times \lfloor \log_2(i-1) \rfloor$$

or

$$\Sigma_{i=2}^{n} 2 \times \lfloor \log_2(i-1) \rfloor - (n-1) \leq \Sigma_{i=2}^{n} C_{remove}(i) \leq \Sigma_{i=2}^{n} 2 \times \lfloor \log_2(i-1) \rfloor$$

so that the total number of comparison in both loops is

$$\Sigma_{i=0}^{n-1} \lfloor \log_2(i+1) \rfloor + \Sigma_{i=2}^{n} 2 \times \lfloor \log_2(i-1) \rfloor - (n-1) \leq C_{sort}(n) \leq$$

$$\leq \Sigma_{i=0}^{n-1} \lfloor \log_2(i+1) \rfloor + \Sigma_{i=2}^{n} 2 \times \lfloor \log_2(i-1) \rfloor$$

that is,

$$\Sigma_{i=1}^{n}\lfloor\log_2 i\rfloor + 2\Sigma_{i=1}^{n-1}\lfloor\log_2 i\rfloor - (n-1) \leq C_{sort}(n) \leq$$
$$\leq \Sigma_{i=1}^{n}\lfloor\log_2 i\rfloor + 2\Sigma_{i=1}^{n-1}\lfloor\log_2 i\rfloor$$

or

$$3\Sigma_{i=1}^{n-1}\lfloor\log_2 i\rfloor - n + \lfloor\log_2 n\rfloor + 1 \leq C_{sort}(n) \leq 3\Sigma_{i=1}^{n-1}\lfloor\log_2 i\rfloor + \lfloor\log_2 n\rfloor. \quad (2)$$

Let's compute first the sum $S_M = \Sigma_{i=1}^{M}\lfloor\log_2 i\rfloor = \Sigma_{i=1}^{M} level(i)$. This sum is adding the levels of all nodes of the heap with M nodes together, so it can be split on the sum of all levels of the nodes that are in the first D_M levels (ranging from 0 to $\lfloor\log_2 M\rfloor - 1$) plus the sum of the levels of the nodes that are in the last level $D_M = \lfloor\log_2 M\rfloor$).

For the example of heap on Figure 2 ($M = 17$ in this case), Figure 7 shows how to split $\Sigma_{i=1}^{17}\lfloor\log_2(i)\rfloor$ on $\Sigma_{i=1}^{\lfloor\log_2 17\rfloor-1} i\times 2^i$ plus $(17-2^{\lfloor\log_2 17\rfloor}+1)\lfloor\log_2(17)\rfloor$.

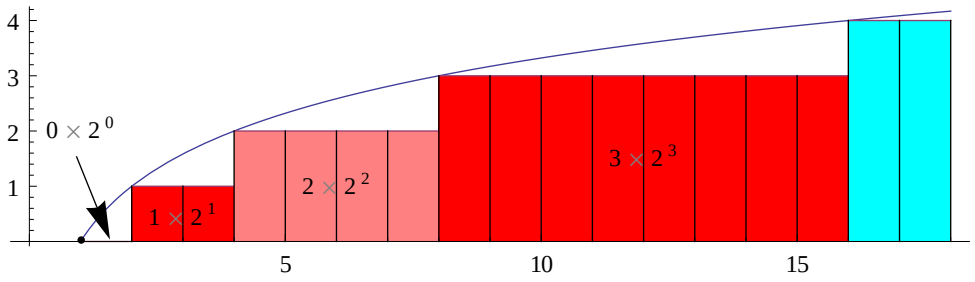

Figure 7: Computation of $\Sigma_{i=1}^{17}\lfloor\log_2(i)\rfloor$ (the colored area) as $\Sigma_{i=1}^{\lfloor\log_2 17\rfloor-1} i\times 2^i$ (the reddish area) + $(17 - 2^{\lfloor\log_2 17\rfloor} + 1)\lfloor\log_2(17)\rfloor$ (the cyan area).

Clearly, there are

$$\Sigma_{j=0}^{\lfloor\log_2 M\rfloor-1} 2^j = 2^{\lfloor\log_2 M\rfloor} - 1$$

nodes in the first $\lfloor\log_2 M\rfloor - 1$ levels of the heap. (Note that $2^{\lfloor\log_2 M\rfloor}$ is the largest power of 2 that is not greater than M.) So, the last level must contain $M - (2^{\lfloor\log_2 M\rfloor} - 1) = M - 2^{\lfloor\log_2 M\rfloor} + 1$ nodes. Therefore:

$$S_M = \Sigma_{i=1}^{M} level(i) = \Sigma_{i=1}^{2^{\lfloor \log_2 M \rfloor}-1} level(i) + \Sigma_{i=2^{\lfloor \log_2 M \rfloor}}^{M} level(M) =$$

$$\Sigma_{i=1}^{2^{\lfloor \log_2 M \rfloor}-1} \lfloor \log_2 i \rfloor + \Sigma_{i=2^{\lfloor \log_2 M \rfloor}}^{M} \lfloor \log_2 M \rfloor =$$

$$\Sigma_{i=1}^{\lfloor \log_2 M \rfloor - 1} i \times 2^i + \lfloor \log_2 M \rfloor \times (M - 2^{\lfloor \log_2 M \rfloor} + 1) =$$

$$(\lfloor \log_2 M \rfloor - 2) 2^{\lfloor \log_2 M \rfloor} + 2 + \lfloor \log_2 M \rfloor \times (M - 2^{\lfloor \log_2 M \rfloor} + 1) =$$

$$(\lfloor \log_2 M \rfloor - 2) \times 2^{\lfloor \log_2 M \rfloor} + \lfloor \log_2 M \rfloor \times M - \lfloor \log_2 M \rfloor \times 2^{\lfloor \log_2 M \rfloor} + \lfloor \log_2 M \rfloor + 2 =$$

$$\lfloor \log_2 M \rfloor \times 2^{\lfloor \log_2 M \rfloor} - 2 \times 2^{\lfloor \log_2 M \rfloor} + \lfloor \log_2 M \rfloor \times M - \lfloor \log_2 M \rfloor \times 2^{\lfloor \log_2 M \rfloor} + \lfloor \log_2 M \rfloor + 2 =$$

$$-2 \times 2^{\lfloor \log_2 M \rfloor} + \lfloor \log_2 M \rfloor \times M + \lfloor \log_2 M \rfloor + 2 =$$

$$M \lfloor \log_2 M \rfloor - 2^{\lfloor \log_2 M \rfloor + 1} + \lfloor \log_2 M \rfloor + 2 =$$

$$(M+1) \lfloor \log_2 M \rfloor - 2^{\lfloor \log_2 M \rfloor + 1} + 2.$$

Hence,

$$S_M = \Sigma_{i=1}^{M} \lfloor \log_2(i) \rfloor = (M+1) \lfloor \log_2 M \rfloor - 2^{\lfloor \log_2 M \rfloor + 1} + 2$$

.

In particular,

$$S_{n-1} = S_n - \lfloor \log_2 n \rfloor = n \lfloor \log_2 n \rfloor - 2^{\lfloor \log_2 n \rfloor + 1} + 2,$$

that is,

$$\Sigma_{i=1}^{n-1} \lfloor \log_2 i \rfloor = n \lfloor \log_2 n \rfloor - 2^{\lfloor \log_2 n \rfloor + 1} + 2. \tag{3}$$

Combining (2) and (3), we conclude that the total number $C_{sort}(n)$ of comparisons is:

$$C_{sort}(n) \leq 3 \times S_{n-1} + \lfloor \log_2 n \rfloor = 3(n\lfloor \log_2 n \rfloor - 2^{\lfloor \log_2 n \rfloor + 1}) + \lfloor \log_2 n \rfloor + 6. \quad (4)$$

Let $x = \log_2 n - \lfloor \log_2 n \rfloor$, that is,

$$\lfloor \log_2 n \rfloor = \log_2 n - x. \quad (5)$$

Applying (5) to (4) we obtain:

$$C_{sort}(n) \leq 3(n(\log_2 n - x) - 2^{\log_2 n + 1 - x}) + \log_2 n - x + 6 =$$

$$3(n(\log_2 n - x) - n2^{1-x}) + \log_2 n - x + 6 = 3n(\log_2 n - (x + 2^{1-x})) + \log_2 n - x + 6 =$$

$$(3n+1)\log_2 n - 3(x + 2^{1-x})n - x + 6 = (3n+1)\log_2 n - 3\alpha n + \beta,$$

where $\alpha = x + 2^{1-x}$ and $\beta = 6 - x$, that is, $1.91 < \alpha \leq 2$ and $5 < \beta \leq 6$.

The same way we compute that

$$(3n+1)\log_2 n - 3\alpha n + \beta - (n-1) \leq C_{sort}(n),$$

or

$$(3n+1)\log_2 n - (3\alpha + 1)n + \beta + 1 \leq C_{sort}(n).$$

Hence,

$$(3n+1)\log_2 n - 7n + 5 < C_{sort}(n) < (3n+1)\log_2 n - 5.73n + 6. \quad (6)$$

Of course,

$$C_{sort}(n) \in \Theta(n \log n) \quad (7)$$

as both

$$(3n+1)\log_2 n - 7n + 6 \in \Theta(n \log n)$$

and

$$(3n+1)\log_2 n - 5.73n + 6 \in \Theta(n \log n).$$

Figure 8 visualizes $C_{sort}(n)$ and its upper and lower bounds $(3n+1)\log_2 n - 7n + 6$ and $(3n+1)\log_2 n - 5.73n + 6$.

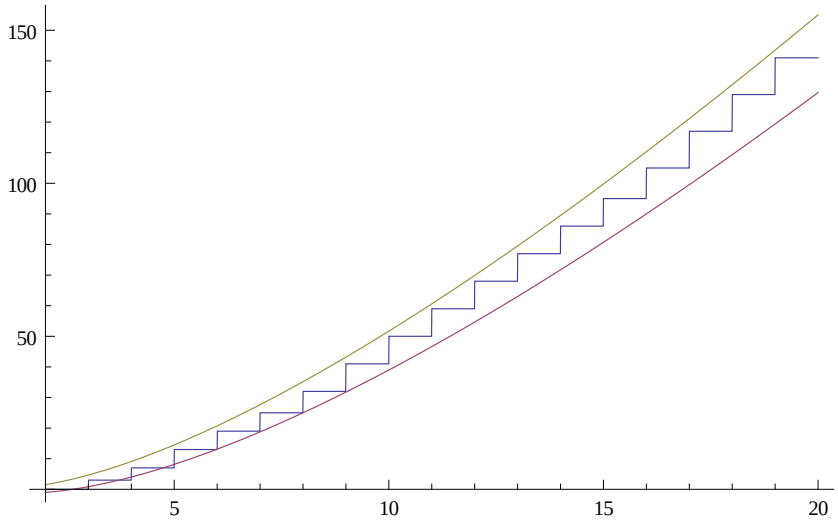

Figure 8: Graph of $C_{sort}(n)$ and its upper and lower bounds.