A Tight Bound on the Worst-Case Number of Comparisons for Floyd's Heap Construction Algorithm

Ioannis Paparrizos

Abstract In this paper a tight bound on the worst-case number of comparisons for Floyd's well-known heap construction algorithm is derived.[1] It is shown that at most $2n - 2\mu(n) - \sigma(n)$ comparisons are executed in the worst case, where $\mu(n)$ is the number of ones and $\sigma(n)$ is the number of zeros after the last one in the binary representation of the number of keys n.

Key words Algorithm analysis • Worst case complexity • Data structures • Heaps

1 Introduction

Floyd's heap construction algorithm [3] proposed in 1964 as an improvement of the construction phase of the classical heapsort algorithm introduced earlier that year by Williams [9] in order to develop an efficient in-place general sorting algorithm. The importance of heaps in representing priority queues and speeding up an amazing variety of algorithms is well documented in the literature. Moreover, the classical heapsort algorithm and, hence, Floyd's heap construction algorithm as part of it is contained and analyzed in each textbook discussing algorithm analysis, see [1] and [2] for example.

---

[1]This paper was also presented at Student Research Forum of SOFSEM'11 [Paparrizos, A tight bound on the worst-case number of comparisons for Floyd's heap construction algorithm (2011)].

I. Paparrizos (✉)
Computer Science Department, Columbia University, New York, NY, USA
e-mail: jopa@cs.columbia.edu

Floyd's algorithm is optimal as long as complexity is expressed in terms of sets of functions described via the asymptotic symbols O, Θ, and Ω. Indeed, its linear complexity $\Theta(n)$, both in the worst and best case, cannot be improved as each object must be examined at least once. However, it is an established tradition to analyze algorithms solving comparison-based problem by counting mainly comparisons, see, for example, Knuth [5] who states that the theoretical study of comparison counting gives us a good deal of useful insight into the nature of sorting processes.

Despite the overwhelming attention received by the computer community in the more than 45 years of its life, a tight bound on the worst-case number of comparisons holding for all values of n, is, to our knowledge, still unknown. Kruskal et al. [6] showed that $2n - 2\lceil \log(n+1) \rceil$ is a tight bound on the worst-case number of comparisons, if $n = 2^k - 1$, where k is a positive integer, and, to our knowledge, this is the only value of n for which a tight upper bound has been reported in the literature.

Schaffer [8] showed that $n - \lceil \log(n+1) \rceil + \lambda(n)$, where $\lambda(n)$ is the number of zeros in the binary representation of n, is the sum of heights of sub-trees rooted at internal nodes of a complete binary tree, see also [4] for an interesting geometric approach to the same problem. Using this result we show that $2n - 2\mu(n) - \sigma(n)$ is a tight bound on the worst-case number of comparisons for Floyd's heap construction algorithm. Here, $\mu(n)$ is the number of ones and $\sigma(n)$ is the number of zeros after the last (right most) one in the binary representation of the number of keys n.

2 Floyd's Heap Construction Algorithm

A *maximum heap* is an array H the elements of which satisfy the property:

$$H(\lfloor i/2 \rfloor) \geq H(i), i = 2, 3, \ldots, n. \tag{1}$$

Relation (1) will be referred to as the heap property. A *minimum heap* is similarly defined; just reverse the inequality sign in (1) from $\geq$ to $\leq$. When we simply say a heap we will always mean a maximum heap. A nice property of heaps is that they can be represented by a *complete binary tree*. Recall that a complete binary tree is a binary tree in which the root lies in level zero and all the levels except the last one contain the maximum possible number of nodes. In addition, the nodes at the last level are positioned as far to the left as possible. If $n = 2^k - 1$, the last level $\lfloor \log n \rfloor = k - 1$ contains the maximum possible number $2^k - 1$ of nodes. In this case the complete binary tree is called *perfect*. The *distinguished path*, introduced in [5], of a complete binary tree that connects the root node 1 with the last leaf node n, will play an important role in deriving our results. It is well known, see for example [5], that the nodes of the distinguished path correspond to the digits of the binary expression of n. Figure 1 illustrates a complete binary tree, its distinguished path
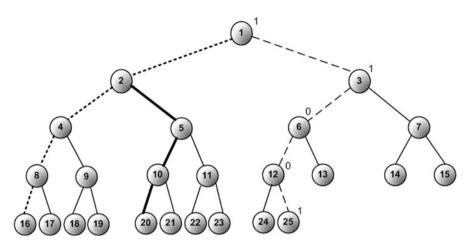
Fig. 1 A complete binary tree, its distinguished path (*dashed edges*), a special path (*thick edges*) and the leftmost path (*dotted edges*). The numbers by the nodes of the distinguished path are the digits of the binary expression (11001) of $n = 25$

and the corresponding binary expression of n. In terms of binary trees the heap property is stated as follows:

The value of a child is smaller than or equal to the value of its parent.

It is easily verified that the value of the root is the largest value. Also, each sub-tree T_j of the complete binary tree representing a heap is also a heap and, hence, the value $H(j)$ is the largest value among those that correspond to the nodes of T_j. A sub-array $H(i : n)$ for which the heap property is satisfied by each node j the parent of which is an element of $H(i : n)$, is also called a heap. Here the expression $j : n$ denotes the sequence of indices $j, j+1, j+2, \ldots, n$.

An *almost heap* is a sub-array $H(i : n)$ all nodes of which satisfy the heap property except possibly node i. If key $H(i)$ violates the heap property, then $H(i) < \max\{H(2i), H(2i+1)\}$.

The main procedure of Floyd's heap construction algorithm, called in this paper heapdown, works as follows. It is applied to an almost heap $H(j : n)$ and converts it into a heap. In particular, if $m = H(j)$ satisfies the heap property $H(j) \geq \max\{H(2j), H(2j+1)\}$ and, hence, $H(j : n)$ is a heap, the algorithm does nothing. Otherwise, it swaps key $m = H(j)$ with the maximum child key $H(j_{\max})$. Then, it considers the child $j_{\max}$ which currently contains key m, and repeats the procedure until the heap property is restored. Algorithm 1 shows a formal description of the algorithm.

Algorithm 1: HEAPDOWN(H(i...n))

---

while $2i+1 \leq n$ **do**
  $k = 2i$
  if $H(k) < H(k+1)$ **then**
    $k = k+1$
  end if
  if $H(i) < H(k)$ **then**
    $swap(H(i), H(k))$
    $i = k$
  else
    return $H(i...n)$
  end if
end while
if $2i = n$ and $H(i) < H(n)$ **then**
  $swap(H(i), H(n))$
  return $H(i...n)$
end if

Algorithm 2: FLOYD-BUILDHEAP(H)

---

for $i = \lfloor n/2 \rfloor$ to 1 step -1 **do**
  heapdown(H(i...n))
end for
return H

Floyd's heap construction algorithm, called Floyd—buildheap procedure in this paper, applies procedure heapdown to the sequence of almost heaps

$$H(\lfloor n/2 \rfloor : n), H(\lfloor n/2 \rfloor - 1 : n), \ldots, H(1 : n). \qquad (2)$$

As the sub-array $H(\lfloor n/2 \rfloor + 1 : n)$ consists of leafs and, therefore, it is a heap and procedure heapdown converts an almost heap to a heap, the correctness of procedure Floyd—buildheap is easily shown.

When procedure heapdown is applied to the almost heap $H(j : n)$ key $m = H(j)$ moves down one level per iteration. In general, two comparisons are executed per level, one comparison to find the maximum child and one to determine whether key m should be interchanged with the maximum child key. However, there is a case in which just one comparison is executed. This happens when key m is positioned at node $\lfloor n/2 \rfloor$ and n is even. Then, internal node $n/2$ has just one child, the last node n, and therefore no comparison is needed to find the maximum child. We will see in the next section, when we will investigate the worst case of procedure Floyd-buildheap, that this situation happens quite often, if n is even. Procedure FLOYD-BUILDHEAP describes formally Floyd's algorithm.

3 A Tight Bound on the Worst-Case Number of Comparisons

It is well known that the number of interchanges performed by Floyd's heap construction algorithm is bounded above by the sum $t(n)$ of heights of sub-trees rooted at the internal nodes of a complete binary tree. Schaffer [8] showed that:

$$t(n) = n - \lceil \log(n+1) \rceil + \lambda(n), \tag{3}$$

where $\lambda(n)$ is the number of zeros in the binary representation of n. For the sake of completeness of the presentation we provide a short proof based on the geometric idea described in [4]. We associate a *special path* with each internal node of the binary tree. The special path connects a node, say j, with a leaf of the subtree T_j rooted at node j. The first edge of the special path is a *right edge* and all the remaining edges are *left edges*, see Fig. 1. In particular, the nodes of the special path are $j, 2j+1, 2^2j+2, 2^3j+2^2, \ldots, 2^k j + 2^{k-1}$. Observe now that the edges of all special paths cover all the edges of the binary tree exactly ones except the $\lfloor \log n \rfloor$ edges of the *leftmost path*, see Fig. 1. As no two special paths contain a common edge, the number of edges of all special paths is $n - 1 - \lfloor \log n \rfloor$.

The lengths of special paths are closely related to the heights of the sub-trees. Recall that the *length* of a path is the number of edges it contains. Denote by $sp(j)$ the special path corresponding to node j. If internal node j does not belong in the distinguished path, then $length(sp(j)) = h(T_j)$. If internal node j belongs in the distinguished path and the right edge $(j, 2j+1)$ is an edge of the distinguished path, then $length(sp(j)) = h(T_j)$. In that case the first edge of $sp(j)$ belongs in the distinguished path and the digit of the binary expression of n corresponding to node j is 1. If internal node j belongs in the distinguished path and the left edge $(j, 2j)$ is an edge of the distinguished path, then $h(T_j) = length(sp(j)) + 1$. In that case the first edge of $sp(j)$ does not belong to the distinguished path and the digit of the binary expression of n corresponding to node j is 0. Summing up all heights of internal nodes we get

$$t(n) = n - 1 - \lfloor \log n \rfloor + \lambda(n) = n - \lceil \log(n+1) \rceil + \lambda(n). \tag{4}$$

In computing, our tight bound on the worst-case number of comparisons, two cases must be considered, n even and n odd. We first take care of the case n is odd.

Lemma 1. *Let n be odd. Then the maximum number of comparisons executed by Floyd's heap construction algorithm is*

$$2t(n) = 2(n - \lceil \log(n+1) \rceil + \lambda(n)). \tag{5}$$

Proof. If n is odd, each internal node has exactly two children and, hence, each key swap corresponds to two key comparisons. Therefore $2t(n)$ is an upper bound on the number of comparisons. $\square$

We show now that this bound is tight. To this end we construct a special worst-case array H. In particular H satisfies the following properties

1. The elements of H are the n distinct keys $1, 2, \ldots, n$.
2. The nodes in the distinguished path are assigned the $\lceil \log(n+1) \rceil$ largest keys. In particular, the nodes in levels $0, 1, 2, \ldots, \log(n)$ are assigned the keys $n - \lceil \log (n+1) \rceil + 1, n - \lceil \log(n+1) \rceil + 2, \ldots, n$, respectively.
3. If j is a node not belonging in the distinguished path, sub-tree T_j is a minimum heap.

Apply now procedure Floyd-buildheap to the array H described previously. When procedure heapdown is called on the almost heap $H(j:n)$ and j is not a node of the distinguished path, key $m = H(j)$ will move all the way down to the bottom level of sub-tree T_j. This is so because key m is the smallest among the keys corresponding to nodes of the sub-tree rooted at node j, see property 3. Also, two comparisons are executed per level. When procedure heapdown is applied to an almost heap $H(j:n)$, where j is a node of the distinguished path, key $m = h(j)$ will follow the distinguished path all the way down to the bottom level taking the position of leaf node n, see property 2. Again, two comparisons are executed per level and, hence, the number $2n - 2\lceil \log(n+1) \rceil + 2\lambda(n)$ is a tight bound on the worst-case number of comparisons.   □

Next lemma takes care of the case n even.

Lemma 2. *If n is even the exact worst case number of comparisons for Floyd's heap construction algorithm is*

$$2(n - \lceil \log(n+1) \rceil + \lambda(n)) - \sigma(n), \tag{6}$$

where $\sigma(n)$ is the number of zeros after the last one in the binary representation of n.

Proof. Let $(b_m b_{m-1} \ldots b_2 b_1 b_0)$ be the binary representation of n. Let also $b_k b_{k-1} \ldots b_2 b_1 b_0$ be the last $k+1$ digits of the binary representation of n such that $b_k = 1$ and $b_{k-1} = b_{k-2} = \cdots = b_1 = b_0 = 0$.   □

As n is even $b_0 = 0$ and, hence, $k \geq 1$. Consider now an internal node of height $j \leq k$ lying at the distinguished path. It is easily verified, using inductively the well-known property $\lfloor \lfloor n/2 \rfloor / a \rfloor = \lfloor n/a^2 \rfloor$ of the floor function, that the index at that node is $\lfloor n/2^j \rfloor$. When procedure Floyd-buildheap calls procedure heapdown on the almost heap $H(\lfloor n/2^j \rfloor : n)$ key $m = H(\lfloor n/2^j \rfloor)$ will move down the levels either following the distinguished path or moving to the right of it at some point. This is so because all the edges $(n, \lfloor n/2 \rfloor), (\lfloor n/2 \rfloor, \lfloor n/2^2 \rfloor), \ldots, (\lfloor n/2^{j-1} \rfloor, \lfloor n/2^j \rfloor)$ of the distinguished path are left edges. In the former case at most $2j - 1$ comparisons are executed and this happens when key $H(\lfloor n/2^j \rfloor)$ is placed either at the bottom level or at the level next to bottom. In the latter case at most $2(j-1)$ comparisons are executed. Hence for each node of the distinguished path at height $j = 1, 2, \ldots, k$ the maximum number of comparisons is one less than 2 times the height of the sub-tree rooted at that node. For all the remaining internal nodes i the maximum
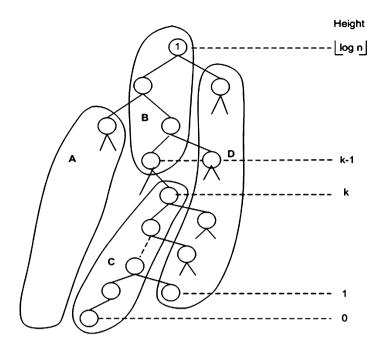
Fig. 2 Partition of the nodes of a complete binary tree into sets A, B, C, D

number of comparisons is $2h(T_i)$, where $h(T_i)$ is the height of the sub-tree rooted at node i. As the number of internal nodes of the distinguished path at heights $1, 2, \ldots, k$ is $\sigma(n)$, the previous arguments show that the number

$$2(n - \lceil \log(n+1) \rceil + \lambda(n)) - \sigma(n) \qquad (7)$$

is an upper bound on the number of comparisons for procedure Floyd-buildheap.

We describe now an array H on which procedure Floyd-buildheap executes exactly $2(n - \lceil \log(n+1) \rceil + \lambda(n)) - \sigma(n)$ comparisons, thus showing that this number is a tight upper bound for n even. In order to describe the structure of the worst-case example H we partition the nodes of the complete binary tree into 4 sets A, B, C, D. Set A contains all the nodes on the left side of the distinguished path. Set D contains all nodes lying on the right side of the distinguished path. Set C contains the nodes of the distinguished path of height $j = 0, 1, 2, \ldots, k$ and set B contains all the remaining nodes of the distinguished path. Figure 2 illustrates a complete binary tree and the sets of nodes A, B, C, D.

The structure of array H is described in the following properties:

1. The elements of H are the n distinct keys $1, 2, \ldots, n$.
2. If i, j, k, l are nodes belonging to the sets A, B, C, D, respectively, then
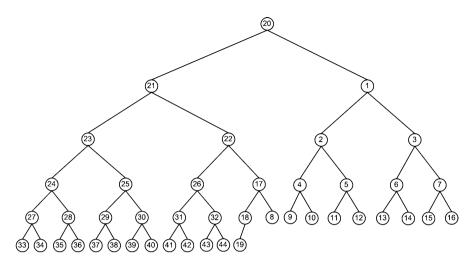
$$H(i) > H(j) > H(k) > H(l). \qquad (8)$$

Fig. 3 A worst-case complete binary tree for Lemma 2. It is $n = 44$, $k = 2$, $|A| = 23, |B| = 3$, $|C| = 3, |D| = 15$. The number inside node j is the key $H(j)$

3. The keys in the distinguished path that belong to the set B are in increasing order from the top to the bottom. The keys in the distinguished path that belong to the set C are in increasing order from the top to the bottom.
4. If j is a node not belonging to the distinguished path, the sub-tree T_j is a minimum heap.

Although there are more than one way to assign the keys $1, 2, \ldots, n$ to the elements of H so that properties (2)–(4) are satisfied, an easy way to do that is as follows. Place the $|A|$ largest keys to the sub-trees on the left of the distinguished path so that each sub-tree is a minimum heap. The symbol $|A|$ denotes the number of elements of set A. Obviously $|A|$ is the number of nodes on the left of the distinguished path. Place in increasing order from top to bottom levels the next $|B|$ largest elements at the nodes of the distinguished path that belong to the set B. Also, place in increasing order from top to down levels the next $|C|$ largest elements at the nodes of the distinguished path that belong to the set C. Obviously $|B| + |C| = 1 + \lfloor \log(n) \rfloor = \lceil \log(n+1) \rceil$. Finally, place the remaining $|D|$ smallest keys, i.e, the keys $1, 2, \ldots, |B|$ at the sub-trees right to the distinguished path so that each sub-tree is a minimum heap. Figure 3 illustrates such a worst-case example for $n = 44$.

Apply now procedure Floyd-buildheap on the array H described previously. Let $H(j:n), j = \lfloor n/2 \rfloor, \lfloor n/2 \rfloor - 1, \ldots, 1$ be the almost heap on which procedure heapdown is applied to after it is called by procedure Floyd-buildheap. If j is not a node at the distinguished path, key $H(j)$, because of property (4), will move all the way down to the bottom level of sub-tree T_j and $2h(T_j)$ comparisons will be executed. If j is a node of the distinguished path belonging to set C, key $H(j)$, because of properties (2)–(4), will follow the distinguished path never making a right turn. In this case, key $H(j)$ will be placed at node n executing

$2h(T_j) - 1$ comparisons. Finally, if node j belongs in set B, key $H(j)$ will definitely make a left turn before reaching the node $\lfloor n/2^k \rfloor$ of the distinguished path, see properties (2) and (3). Then it will move all the way down to bottom level executing 2 comparisons per level. Again $2h(T_j)$ comparisons are executed.

Summing up the comparisons for all the calls of procedure heapdown we see that the total number of comparisons is as stated in the Lemma. □

Observe that the array H described in the previous Lemma is not a minimum heap. In particular the sub-trees rooted at nodes of the distinguished path are not minimum heaps.

Theorem 1. *The number $2n - 2\mu(n) - \sigma(n)$, where $\mu(n)$ is the number of ones and $\sigma(n)$ is the number of zeros after the last one in the binary representation of n, is a tight bound on the worst-case number of comparisons for Floyd's heap construction algorithm.*

Proof. If n is odd, then $b_0 = 1$ and, hence, $\sigma(n) = 0$. Combining Lemmas 1 and 2 we see that a tight bound on the worst-case number of comparisons is the number $2[n - \lceil \log(n+1) \rceil + \lambda(n)] - \sigma(n) = 2[n - (\lambda(n) + \mu(n)) + \lambda(n)] - \sigma(n) = 2n - 2\mu(n) - \sigma(n)$. □

4   Conclusion

Deriving worst case tight upper bound examples for an algorithm implies that the worst-case complexity of the algorithm cannot be improved. We derived our worst-case examples by the use of simple geometric ideas. As the binary trees and heaps are involved in many other algorithms for which worst-case tight examples are not known, we hope that our results will contribute in solving those problems.

References

1. Aho, A.V., Hopcroft, J.E., Ullman, J.D.: The Design and Analysis of Computer Algorithms. Addison-Wesley Series in Computer Science and Information Processing, Addison-Wesley Publishing Company (1974)
2. Cormen, T., Leiserson, C., Rivest, R., Stein, C.: Introduction to Algorithms, 2nd edn. MIT, Cambridge (2001)
3. Floyd, R.: Algorithm 245: treesort 3. Comm. ACM **7**, 701 (1964)
4. Goodrich, M.T., Tamassia, R.: Algorithm Design, Foundations, Analysis, and Internet Examples. Wiley, New York (2002)
5. Knuth, D.: The art of computer programming, vol. 3, 2nd edn. Searching and Sorting. Addison Wesley, Redwood city (1998)
6. Kruskal, C.P., Weixelbaum, D.: The worst case analysis of heapsort. Technical Report no 018a, Department of Computer Science, New York University (1979)

7. Paparrizos, I.: A tight bound on the worst-case number of comparisons for Floyd's heap construction algorithm. In: Proceedings of the 37th International Conference on Current Trends in Theory and Practice of Computer Science (2011)
8. Schaffer, R.: Analysis of heapsort. Ph.D. Thesis, Department of computer science, Princeton University (1992)
9. Williams, J.W.J.: Algorithm 232: heapsort. Comm. ACM **6**, 347–348 (1964)