nodes in group $g$ is at most
\[
\frac{n}{F(g)} (F(g) - F(g - 1)) \leq n.
\]
Since there are at most $\log^* n$ groups $(0, 1, \ldots, \log^* n - 1)$, it follows that the number of node charges assigned to all nodes is $O(n \log^* n)$. Combining this with the $O(m \log^* n)$ charges to the find instructions yields the following theorem:

**Theorem 4.3** Let $T(m)$ denote the running time required to process an interspersed sequence $\sigma$ of $m$ union and find operations using union by rank and path compression. Then $T(m) = O(m \log^* n)$ in the worst case.

Note that for almost all practical purposes, $\log^* n \leq 5$. This means that the running time is $O(m)$ for virtually all practical applications.

### 4.4 Exercises

4.1. What are the merits and demerits of implementing a priority queue using an ordered list?

4.2. What are the costs of insert and delete-max operations of a priority queue that is implemented as a regular queue.

4.3. Which of the following arrays are heaps?
   - (a) $8 \ 6 \ 4 \ 3 \ 2$
   - (b) $7$
   - (c) $9 \ 7 \ 5 \ 6 \ 3$
   - (d) $9 \ 4 \ 8 \ 3 \ 2 \ 5 \ 7$
   - (e) $9 \ 4 \ 7 \ 2 \ 1 \ 6 \ 5 \ 3$

4.4. Where do the following element keys reside in a heap?
   - (a) Second largest key.
   - (b) Third largest key.
   - (c) Minimum key.

4.5. Give an efficient algorithm to test whether a given array $A[1..n]$ is a heap. What is the time complexity of your algorithm?

4.6. Which heap operation is more costly: insertion or deletion? Justify your answer. Recall that both operations have the same time complexity, that is, $O(\log n)$.

4.7. Let $H$ be the heap shown in Fig. 4.1. Show the heap that results from
   - (a) deleting the element with key 17.
   - (b) inserting an element with key 19.

4.8. Show the heap (in both tree and array representation) that results from deleting the maximum key in the heap shown in Fig. 4.4(e).
4.9. How fast is it possible to find the minimum key in a max-heap of \( n \) elements?

4.10. Prove or disprove the following claim. Let \( x \) and \( y \) be two elements in a heap whose keys are positive integers, and let \( T \) be the tree representing that heap. Let \( h_x \) and \( h_y \) be the heights of \( x \) and \( y \) in \( T \). Then, if \( x \) is greater than \( y \), \( h_x \) cannot be less than \( h_y \). (See Sec. 3.5 for the definition of node height).

4.11. Illustrate the operation of Algorithm MAKEHEAP on the array

\[
\begin{array}{cccccccc}
3 & 7 & 2 & 1 & 9 & 8 & 6 & 4 \\
\end{array}
\]

4.12. Show the steps of transforming the following array into a heap

\[
\begin{array}{cccccccc}
1 & 4 & 3 & 2 & 5 & 7 & 6 & 8 \\
\end{array}
\]

4.13. Let \( A[1..19] \) be an array of 19 integers, and suppose we apply Algorithm MAKEHEAP on this array.

(a) How many calls to Procedure SIFT-DOWN will there be? Explain.
(b) What is the maximum number of element interchanges in this case? Explain.
(c) Give an array of 19 elements that requires the above maximum number of element interchanges.

4.14. Show how to use Algorithm HEAPSORT to arrange in increasing order the integers in the array

\[
\begin{array}{cccccccc}
4 & 5 & 2 & 9 & 8 & 7 & 1 & 3 \\
\end{array}
\]

4.15. Given an array \( A[1..n] \) of integers, we can create a heap \( B[1..n] \) from \( A \) as follows. Starting from the empty heap, repeatedly insert the elements of \( A \) into \( B \), each time adjusting the current heap, until \( B \) contains all the elements in \( A \). Show that the running time of this algorithm is \( \Theta(n \log n) \) in the worst case.

4.16. Illustrate the operation of the algorithm in Exercise 4.15 on the array

\[
\begin{array}{cccccccc}
6 & 9 & 2 & 7 & 1 & 8 & 4 & 3 \\
\end{array}
\]

4.17. Explain the behavior of Algorithm HEAPSORT when the input array is already sorted in

(a) increasing order.
(b) decreasing order.

4.18. Give an example of a binary search tree with the heap property.
4.19. Give an algorithm to merge two heaps of the same size into one heap. What is the time complexity of your algorithm?

4.20. Compute the minimum and maximum number of element comparisons performed by Algorithm HEAPSORT.

4.21. A d-heap is a generalization of the binary heap discussed in this chapter. It is represented by an almost-complete d-ary rooted tree for some \( d \geq 2 \). Rewrite Procedure SIFT-UP for the case of d-heaps. What is its time complexity?

4.22. Rewrite Procedure SIFT-DOWN for the case of d-heaps (see Exercise 4.21). What is its time complexity measured in terms of \( d \) and \( n \)?

4.23. Give a sequence of \( n \) union and find operations that results in a tree of height \( \Theta(\log n) \) using only the heuristic of union by rank. Assume the set of elements is \( \{1, 2, \ldots, n\} \).

4.24. Give a sequence of \( n \) union and find operations that requires \( \Theta(n \log n) \) time using only the heuristic of union by rank. Assume the set of elements is \( \{1, 2, \ldots, n\} \).

4.25. What are the ranks of nodes 3, 4 and 8 in Fig. 4.8(f)?

4.26. Let \( \{1\}, \{2\}, \{3\}, \ldots, \{8\} \) be \( n \) singleton sets, each represented by a tree with exactly one node. Use the union-find algorithms with union by rank and path compression to find the tree representation of the set resulting from each of the following unions and finds: \text{union}(1, 2), \text{union}(3, 4), \text{union}(5, 6), \text{union}(7, 8), \text{union}(1, 3), \text{union}(5, 7), \text{find}(1), \text{union}(1, 5), \text{find}(1).

4.27. Let \( T \) be a tree resulting from a sequence of unions and finds using both the heuristics of union by rank and path compression, and let \( x \) be a node in \( T \). Prove that \( \text{rank}(x) \) is an upper bound on the height of \( x \).

4.28. Let \( \sigma \) be a sequence of union and find instructions in which all the unions occur before the finds. Show that the running time is linear if both the heuristics of union by rank and path compression are used.

4.29. Another heuristic that is similar to union by rank is the weight-balancing rule. In this heuristic, the action of the operation \( \text{union}(x, y) \) is to let the root of the tree with fewer nodes point to the root of the tree with a larger number of nodes. If both trees have the same number of nodes, then let \( y \) be the parent of \( x \). Compare this heuristic with the union by rank heuristic.

4.30. Solve Exercise 4.26 using the weight-balancing rule and path compression (see Exercise 4.29).

4.31. Prove that the weight-balancing rule described in Exercise 4.29 guarantees that the resulting tree is of height \( O(\log n) \).

4.32. Let \( T \) be a tree resulting from a sequence of unions and finds using the
heuristics of union by rank and path compression. Let \( x \) be the root of \( T \) and \( y \) a leaf node in \( T \). Prove that the ranks of the nodes on the path from \( y \) to \( x \) form a strictly increasing sequence.

4.33. Prove the observation that if node \( v \) is in rank group \( g > 0 \), then \( v \) can be moved and charged at most \( F(g) - F(g - 1) \) times before it acquires a parent in a higher group.

4.34. Another possibility for the representation of disjoint sets is by using linked lists. Each set is represented by a linked list, where the set representative is the first element in the list. Each element in the list has a pointer to the set representative. Initially, one list is created for each element. The union of two sets is implemented by merging the two sets. Suppose two sets \( S_1 \) represented by list \( L_1 \) and \( S_2 \) represented by list \( L_2 \) are to be merged. If the first element in \( L_1 \) is to be used as the name of the resulting set, then the pointer to the set name at each element in \( L_2 \) must be changed so that it points to the first element in \( L_1 \).

(a) Explain how to improve this representation so that each find operation takes \( O(1) \) time.

(b) Show that the total cost of performing \( n - 1 \) unions is \( \Theta(n^2) \) in the worst case.

4.35. (Refer to Exercise 4.34). Show that if when performing the union of two sets, the first element in the list with a larger number of elements is always chosen as the name of the new set, then the total cost of performing \( n - 1 \) unions becomes \( O(n \log n) \).

4.5 Bibliographic notes

Heaps and the data structures for disjoint sets appear in several books on algorithms and data structures (see the bibliographic notes of chapters 1 and 3). They are covered in greater depth in Tarjan (1983). Heaps were first introduced as part of heapsort by Williams (1964). The linear time algorithm for building a heap is due to Floyd (1964). A number of variants of heaps can be found in Cormen et al. (1992), e.g. binomial heaps, Fibonacci heaps. A comparative study of many data structures for priority queues can be found in Jones (1986). The disjoint sets data structure was first studied by Galler and Fischer (1964) and Fischer (1972). A more detailed analysis was carried out by Hopcroft and Ullman (1973) and then a more exact analysis by Tarjan (1975). In this paper, a lower bound that is not linear was established when both union by rank and path compression are used.