Washington State University
School of Electrical Engineering and Computer Science
Spring 2022

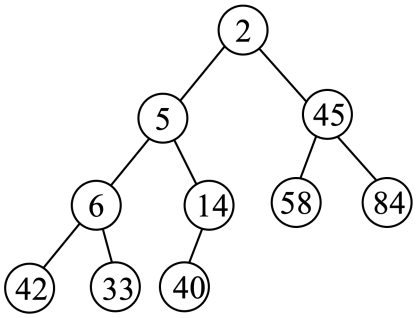
CptS 223 Advanced Data Structures in C++
Homework 9 - Solution
Due: March 30, 2022 (11:59pm pacific time)

General Instructions: Put your answers to the following problems into a PDF document and upload the document as your submission for Homework 9 for the course CptS 223 Pullman on the Canvas system by the above deadline.

Note: Your solutions should match the result from using the algorithms in the textbook, which may be different from the results obtained by web apps or other implementations. See the accompanying `src.zip` *file for the textbook code related to the questions below. You are welcome to use the code to help with answering the questions.*

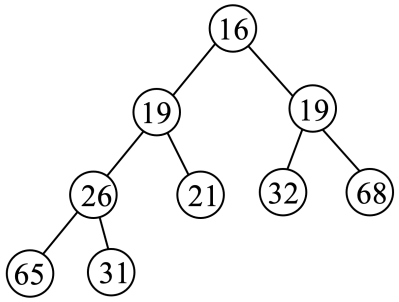
1. Show the final binary heap after performing the linear-time `buildHeap` algorithm on the sequence 42, 33, 45, 5, 14, 58, 84, 6, 2, 40. Your final binary heap should be drawn as a tree.

   Solution:



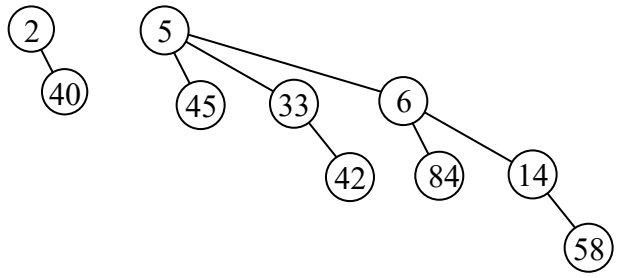
2. Show the binary heap after performing a `deleteMin` on the binary heap on the right side of Figure 6.11. Your binary heap should be draw as a tree.

   Solution:



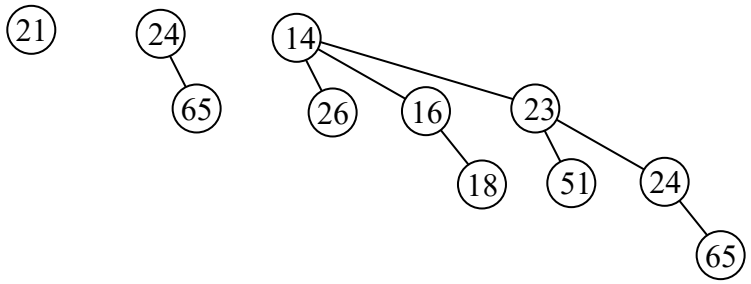
3. Show the final binomial queue after inserting the keys 42, 33, 45, 5, 14, 58, 84, 6, 2, 40 (in this order) into an initially empty binomial queue. Your final binomial queue should be drawn as a forest of trees.

   Solution:



4. Show the binomial queue after performing a `deleteMin` on the binomial queue in Figure 6.49. Your binomial queue should be drawn as a forest of trees.

   Solution:



5. Show the steps of QuickSort to sort the array <5,3,5,6,2,9,5,1,4,1,3>. You should use the code from Figures 7.15-7.17 in the textbook, except in line 11 of Figure 7.17, replace the 10 with a 3. Specifically, for each call (initial and recursive) to `quicksort` in Figure 7.17, show the array upon entry to the procedure, the chosen pivot, and the array after partitioning or calling `insertionSort`.

   Solution: See accompanying Sort.h and testsort.cc files. In particular, see "HW9" commented changes to quicksort function to output the requested information.

```
quicksort(A,0,10):
  Initial A = 5 3 5 6 2 9 5 1 4 1 3
  pivot = 5
  Partitioned A = 3 3 4 1 2 1 5 6 5 5 9

quicksort(A,0,5):
  Initial A = 3 3 4 1 2 1 5 6 5 5 9
  pivot = 3
  Partitioned A = 1 1 2 3 3 4 5 6 5 5 9
```

```
quicksort(A,0,2):
  Initial A = 1 1 2 3 3 4 5 6 5 5 9
  After insertionSort(A,0,2) A = 1 1 2 3 3 4 5 6 5 5 9

quicksort(A,4,5):
  Initial A = 1 1 2 3 3 4 5 6 5 5 9
  After insertionSort(A,4,5) A = 1 1 2 3 3 4 5 6 5 5 9

quicksort(A,7,10):
  Initial A = 1 1 2 3 3 4 5 6 5 5 9
  pivot = 6
  Partitioned A = 1 1 2 3 3 4 5 5 5 6 9

quicksort(A,7,8):
  Initial A = 1 1 2 3 3 4 5 5 5 6 9
  After insertionSort(A,7,8) A = 1 1 2 3 3 4 5 5 5 6 9

quicksort(A,10,10):
  Initial A = 1 1 2 3 3 4 5 5 5 6 9
  After insertionSort(A,10,10) A = 1 1 2 3 3 4 5 5 5 6 9
```

6. For the initial array in question 5, determine the number of comparisons performed by the following algorithms to sort the array.
   a. InsertionSort. Specifically, how many times is the statement "`tmp < a[j-1]`" executed in line 12 of Figure 7.2? Show your work.
   b. MergeSort. Specifically, how many times is the statement "`a[leftPos] <= a[rightPos]`" executed in line 19 of Figure 7.12? Show your work.

   Solution: See accompanying Sort.h and testsort.cc files. In particular, see "HW9" commented changes to insertionSort and mergeSort functions to collect the total number of comparisons for each sort.
   a. insertionSort: #comparisons = 40
   b. mergeSort: #comparisons = 26