Data Structures

Fall 2020, Final Exam. (Solutions) Jan. 11, 2021

1. (10 pts) True or False? Write O for true and $\times$ for false. Score = max $\{0, Right\ -\ \frac{1}{2}\ Wrong\}$.

   (1). ... $\times$ ... Inserting n keys into an initially empty binomial heap takes $\Theta(n \log n)$ time.
   Sol: $O(n)$ time.

   (2). ...O... Given a top-down red-black tree with n elements, it is possible to sort the n elements using the tree in $O(n)$ time.
   Sol: Use inorder traversal.

   (3). ...O... Consider a bottom-up red-black tree, it is true that in the worst case, an insertion requires $O(1)$ rotations.
   Sol: 2 rotations suffice.

   (4). ... $\times$ ... Quicksort is a stable sorting algorithm whose worst case running time is $O(n^2)$.
   Sol: Not stable.

   (5). ...O... Given an undirected graph $G = (V, E)$, it can be tested to determine whether or not G is a tree in $O(|V| + |E|)$ time. A tree is a connected graph without any cycles.
   Sol: Using either DFS or BFS

   (6). ...O... The Bellman-Ford algorithm can be used to detect the existence of a negative-weight directed cycle, if there is one, in a directed weighted graph.
   Sol: G has a negative cycle if the algorithm fails to stabilize after $|V|$ iterations.

   (7). ...O... There are 3 binomial trees in a binomial heap with 13 elements.
   Sol: 13=1101 in binary representation

   (8). ... $\times$ ... In Union-Find, if only union-by-rank is applied (i.e., no path compression), then the worst-case time of a find operation becomes $\Theta(n)$
   Sol: $O(\log n)$

   (9). ... $\times$ ... The height of an n-node leftist tree is $O(\log n)$.
   Sol: Could be n.

   (10). ... $\times$ ... Applying quicksort (using the leftmost key as the pivot) to a reverse sorted (such as 9, 8, 7, ..., 2, 1) array of n elements takes $O(n \log n)$ time.
   Sol: $O(n^2)$

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

2. (6 pts) If a data structure supports an operation *foo* such that a sequence of n *foo*'s takes $O(n \log n)$ time in the worst case, then the amortized time of a *foo* operation is $\Theta(T_1(n))$ while the actual time of a single *foo* operation could be as low as $\Theta(T_2(n))$ and as high as $\Theta(T_3(n))$. What are $T_1(n), T_2(n), T_3(n)$? No explanations needed.
Sol: $T_1(n) = \log n, T_2(n) = 1, T_3(n) = n \log n$

3. (14 pts) Consider Prim's algorithm for finding the minimum spanning tree of a weighted graph $G = (V, E)$ with n nodes and m edges (i.e., $|V| = n, |E| = m$, $m \geq n - 1$). It is known that the priority queue ADT plays a key role in Prim's algorithm.

   (a) (2 pts) What is the number of *delete-min* operations involved in Prim's algorithm?
   Sol: $|V| = n$

   (b) (2 pts) What is the number of *decrease-key* operations involved in Prim's algorithm?
   Sol: $|E| = m$

   (c) For each of the following priority queue implementations, write down the running time of Prim's algorithm. Your solution should be a function in terms of variables m and n (such as $O(n^2 + m \log n + mn)$, $O(mn^2)$ ...). No explanations needed.
   (1) Binary heap **(2) Binomial heap** **(3) Fibonacci heap** **(4) Leftist heap** **(5) Skew heap**
   Sol:

      i. Binary heap: $O(m \log n)$
      ii. Binormial heap: $O(m \log n)$
      iii. Fibonacci heap: $O(m + n \log n)$
      iv. Leftist heap: $O(m \log n)$
      v. Skew heap: $O(m \log n)$

4. (10 pts) Many balanced binary tree data structures (such as AVL trees, 2-3-4 trees, etc) maintain a collection of elements, subject to *insertion* and *deletion* operations that both take worst-case time proportional to the logarithm of n (i.e., $\log_2 n$).

(a) Find a potential function ϕ for which (without changing the data structure, only its analysis) the amortized time per *deletion* is $O(1)$, while the amortized time per *insertion* is still only $O(\log n)$.

(b) Show how this choice of ϕ gives these amortized time bounds for insertion and deletion.

Recall that the amortized time of an operation O is $\hat{c}_O = c_O + (\phi(D') - \phi(D))$, where applying O to data structure D yields D' and c_O is the actual time of O.

Sol:

(a) Define $\phi(T) = n \log n$, where n is the number of nodes in the tree.

(b) Deletion: $c_{del} = \log n + (\phi(T') - \phi(T)) = \log n + ((n-1)\log(n-1) - n\log n) \leq \log n + ((n-1)\log n - n \log n) = O(1)$.

(c) Insertion: $c_{in} = \log n + (\phi(T') - \phi(T)) = \log n + ((n+1)\log(n+1) - n\log n) \leq \log n + ((n+1)\log(n+1) - n\log n) = O(\log n)$.

5. (8 pts) Consider the use of the double hashing probing technique for collision resolution, and let the primary and secondary hash functions be

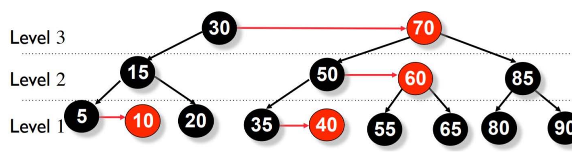
$$h_1(x) = x \ mod \ m \qquad h_2(x) = 1 + (x \ mod \ (m-1)).$$

Insert the keys **28, 59, 47, 13, 39, 69, 12, 6** into the hash table of size $m = 11$. Show the content of the table.

Sol:

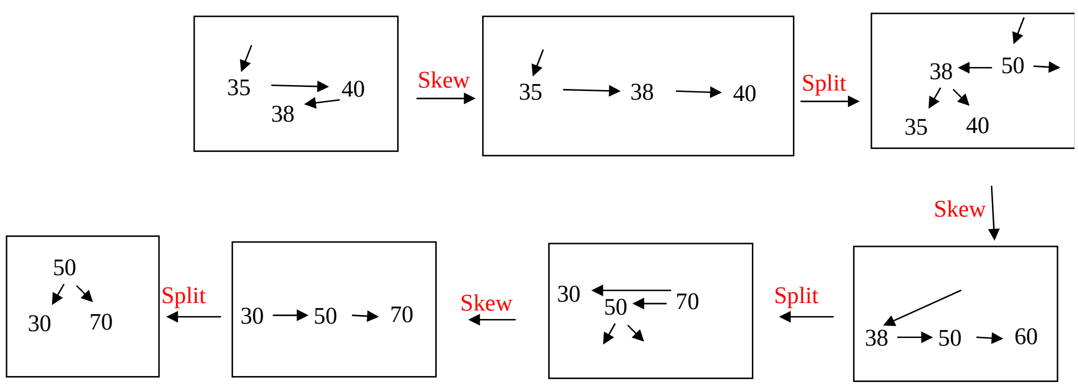
0	1	2	3	4	5	6	7	8	9	10
	69	13	47	59	39	28	12		6	

6. (12 pts) Consider the following AA-tree. Suppose that you apply an *insertion* and a *deletion* below into the AA-tree. For each operation, give the number of *skews*, the number of *splits*, and the key that appears in the root node when the operation is completed. Recall that a *split* is to remove two consecutive right horizontal links, while a *skew* removes a left horizontal link. The operations are NOT cumulative - you are applying each operation into the AA-tree below.

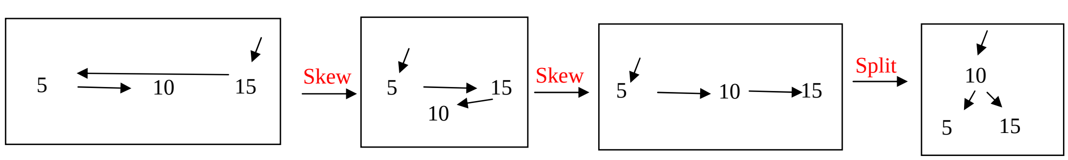


Operation	# of skews	# of splits	Root
Insert 38	3	3	50
delete 20	2	1	30

Sol:



Insert 38



Delete 20

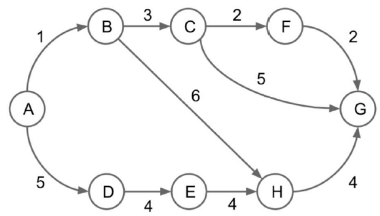
7. (10 pts) For the following example, suppose we run Dijkstra's algorithm starting from node A. Answer the following questions:

(i) (6 pts) Give the order of nodes in which Dijkstra's Algorithm would visit. Fill in the following blanks $\boxed{1}$-$\boxed{7}$ with letters B - H.

Sol:

0	1	2	3	4	5	6	7
A	B	C	D	F	H	G	E

(ii) (2 pts) What is the best data structure (i.e., most efficient) data structure for implementing Dijkstra's algorithm?
Sol: Fibonacci Heap



(iii) (2 pts) What is the total weight of the shortest path from A to G?
Sol: 8

8. (10 pts) The leftmost column (I) contains an array of 24 integers to be sorted; the rightmost column (O) contains the integers in sorted order. Each of the remaining columns (A)-(E) gives the contents of the array during some intermediate step of one of the algorithms listed below: (1) Merge; (2) Quick; (3) Heap; (4) Bubble; (5) Selection; (6) Insertion; (7) Straight Radix; (8)Radix Exchange; (9) Shell. Match each column (A)-(E) with its corresponding algorithm. No penalty for wrong answers.

I	A	B	C	D	E	O
37	11	18	11	11	60	11
79	18	30	18	18	58	18
11	22	11	28	22	56	22
50	28	22	37	28	57	28
39	30	28	39	30	30	30
66	37	37	50	37	37	37
78	39	78	56	39	39	39
18	50	66	66	50	50	50
80	56	80	78	56	11	56
28	57	39	79	64	22	57
56	58	56	80	66	28	58
98	60	98	98	78	18	60
92	92	92	22	79	61	61
22	66	50	30	80	63	63
30	78	79	57	92	64	64
64	64	64	58	98	66	66
86	86	86	60	86	78	78
57	79	57	61	57	79	79
83	83	83	63	83	80	80
63	63	63	64	63	81	81
60	98	60	81	60	83	83
61	61	61	83	61	86	86
58	80	58	86	58	92	92
81	81	81	92	81	98	98

Sol:

A	B	C	D	E
Selection sort	Quick sort	Merge sort	Insertion sort	Heapsort

(A) selection sort after 12 iterations

(B) quicksort after first partitioning step

(C) mergesort just before the last call to merge()

(D) insertion sort after 16 iterations

(E) heapsort after heap construction phase and putting 12 keys into place

9. (10 pts)

(1) (5 pts) What is the size of the largest min-binary-heap (i.e., min-binary-heap with the largest number of nodes) in which the 5^{th} largest element is a child of the root? Assume the heap has no duplicate elements. Given (draw) an example of the binary heap assuming all keys are integers.
Sol: 13

```
              [1]
          ____/ \____
       [2]           [9*]
       / \           / \
    [3]    [4]    [10] [11]
    / \    / \    / \
 [5] [6][7] [8][12][13]
```

(2) (5 pts) What is the size of the largest max-binary-heap that is also a valid binary search tree? Draw an example assuming all keys are integers. Assume the heap has no duplicate elements.
Sol: 2; in a max-heap, each node is than each of its children for the heap-order property. Because of this, we are able to have one element in the left subtree, and this does not violate the completeness property as before.

10. (10 pts) Insert 4 into the following *bottom-up* red-black tree. Show your derivations in detail. Be sure to mark red nodes clearly.
Sol: