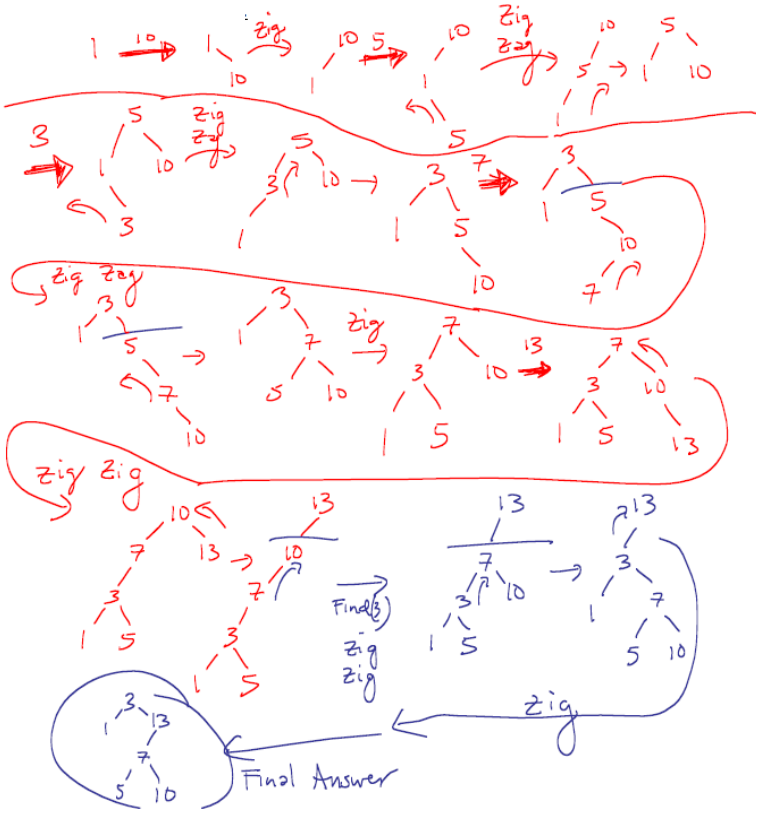
Data Structures and Programming

Spring 2015, Final Exam. Solutions

Date: June 23, 2015

⟨ 1 ⟩ (10 pts) Imagine that the following operations are performed on an initially empty (two-pass) splay tree: *Insert(1), Insert(10), Insert (5), Insert (3), Insert (7), Insert (13), Find (3).*
Show the state of the splay tree after performing each of the above operations. Be sure to label each of your trees with what operations you have just completed.
Solution



⟨ 2 ⟩ (10 pts) Suppose that r is a root of some tree in a Fibonacci heap. Recall that the *rank* of a node is the number of children of the node. Assume that just before a *deletemin* operation, r has no children (i.e., rank(r)=0) and that after the deletemin r has 10 children (i.e., rank(r)=10). Let C be the set of children of r after the deletemin and let $ranks(C)$ be the set of rank values for the nodes in C (since this is a set, if several nodes in C have the same rank, their rank value appears just once in $ranks(C)$). (For example, if r has 5 children of ranks 1, 2, 2, 5, 5, then $|ranks(C)| = 3$, max $ranks(C)$ is 5.)

(a) What is the largest value in $ranks(C)$ (i.e., max $ranks(C)$) right after the deletemin?
Sol. 9 Remark: The tree with root r is the result of merging two trees of rank 9.

(b) What is the value of $|ranks(C)|$ (i.e., the number of elements in set $ranks(C)$) right after the deletemin?
Sol. 10 Remark: Before deletemin, the rank of r is 0. Hence, the only way for r to have 10 children is that r merges with a tree of rank 0 to yield a tree of rank 1, then merges with a tree of rank 1 to yield a tree of rank 2, and so on. r remains the root during the above process.

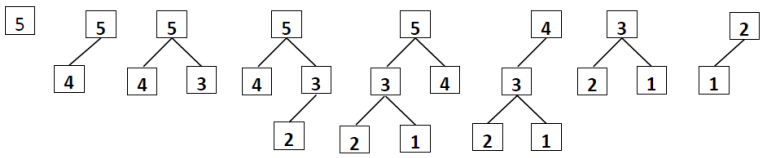
(c) Assume that some time later r is still a tree root and has the same set of children. What is the smallest possible value for max $ranks(C)$ at this point?
Sol. 8 Remark: It is possible for a child of the subtree of rank 9 to be cut, due to a decrease key.

(d) Continue from (c) above. What is the smallest possible value for $|ranks(C)|$ at this point?
Sol. 5 Remark: $\{9 \to 8, 8, 7 \to 6, 6, 5 \to 4, 4, 3 \to 2, 2, 1 \to 0, 0\}$

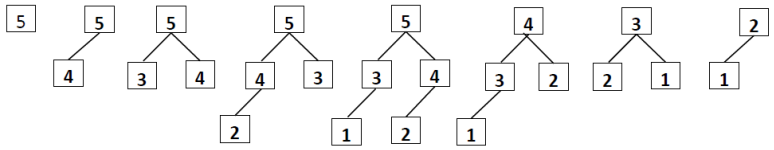
(e) Suppose some time later, r is still a tree root but no longer has the same set of children? Let C' be its current set of children. What is the smallest possible value for $|C'|$?
Sol. 0 Remark: If a cut is performed on each of r's children, C' becomes empty.

⟨ 3 ⟩ (21 pts) Consider a **max**-ordered leftist heap, a **max**-ordered skew heap, and a **max**-ordered binomial heap. Initially suppose each heap contains only a root node with key 5. First insert these keys in the given order: 4, 3, 2, 1. Next call *deleteMax* three times. Redraw the heap after each operation. For each heap, you must draw the following intermediate heaps in detail:
Solution

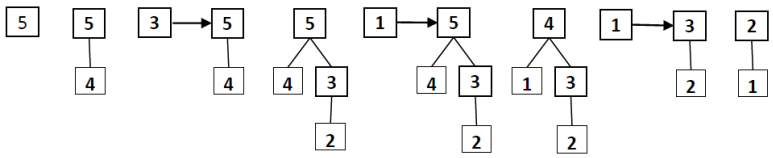
Leftist



Skew



Binomial



⟨ 4 ⟩ (10 pts) The hash table shown below has capacity $N = 11$ and hash function $H(x) = x \ mod \ 11$. Note that the keys 16 and 20 have already been inserted. Now insert these additional keys into the hash table in the specified order: **60, 38, 71, 75, 35** using the following two methods.

0	1	2	3	4	5	6	7	8	9	10
					16				20	

(a) Using *Quadratic probing*
Solution

0	1	2	3	4	5	6	7	8	9	10
35		75	38		16	60			20	71

(b) Using *Double hashing* with $H'(x) = 7 - (x \ mod \ 7)$
Solution

0	1	2	3	4	5	6	7	8	9	10
71	35	38		75	16			60	20	

⟨ 5 ⟩ (17 pts) Answer each of the following questions briefly yet precisely.

(a) (4 pts) Define the *skew* and the *split* operations in AA-trees. Draw pictures to show how they work.

(b) (4 pts) Explain how *randomization* is used in *Skip lists* and *Treaps*.

(c) (4 pts) When we say a sorting algorithm is *stable*, what does that mean? Is *heapsort* stable? Why?

(d) (2 pts) Define the function $\log^* n$.

(e) (3 pts) Define *union by rank* and *path compression* in disjoint set union-find.

Solution See textbook and classnotes.

⟨ 6 ⟩ (18 pts) Which of the following operations are supported by
(A): (classical) Binary Heaps. (B): Binomial Heaps. (C) Fibonacci Heaps.

(1) heapify$(a_1, ..., a_n)$ = create a heap on a given set of n elements in $O(n)$ steps.

(2) insert(a, H) = insert element a to heap H in $O(1)$ amortized steps.

(3) insert(a, H) = insert element a to heap H in $O(\log n)$ steps.

(4) deletemin(H) = delete the minimum element of heap H in $O(1)$ steps.

(5) deletemin(H) = delete the minimum element of heap H in $O(1)$ amortized steps.

(6) deletemin(H) = delete the minimum element of heap H in $O(\log n)$ amortized steps.

(7) merge(H_1, H_2) = combine heaps H_1 and H_2 into one heap in $O(\log n)$ amortized steps

(8) delete(a, H) = remove element a from heap H in $O(\log n)$ amortized steps.

(9) decrement(a, x, H) = decrease the value of the element a in H by amount x in amortized $O(1)$ steps.

Complete the following table. If an entry is filled with AB, then the operation is supported by binary heaps and binomial heaps, but not by Fibonacci heaps. Write *None* if supported by none of the three heaps.
Solution

1	2	3	4	5	6	7	8	9
ABC	BC	ABC	NONE	NONE	ABC	BC	ABC	C

⟨ 7 ⟩ (14 pts) In the following figure, the leftmost column (i.e., Column 1) is the original input of strings to be sorted, and the rightmost column (i.e., Column 9) is the sorted result. The other columns are the contents at some intermediate step during one of the 7 sorting algorithms listed below. Match up each algorithm by writing its letter under the corresponding column. Use each letter exactly once.
(A). Bottom-up mergesort (B). Shellsort (C). Insertion sort (D). Quicksort (E). Selection sort (F). Top-down mergesort (G). Heapsort
Solution

1	2	3	4	5	6	7	8	9
input	F	B	E	D	C	G	A	sorted

Recall that a *top-down mergesort* algorithm recursively splits the list into sublists until sublist size is 1, then merges those sublists to produce a sorted list. On the other hand, a *bottom-up mergesort* algorithm treats the list as an array of n sublists of size 1, and iteratively merges sub-lists back and forth between two buffers.

1	2	3	4	5	6	7	8	9
that	been	also	also	into	been	year	been	also
even	even	down	back	even	even	with	even	back
than	ever	come	been	than	from	will	than	been
been	fell	been	come	been	more	more	that	come
from	from	back	down	from	next	were	from	down
next	loss	even	even	next	over	plea	next	even
show	more	ever	ever	show	plea	well	show	ever
with	next	into	fell	jobs	show	lost	with	fell
more	over	fell	from	more	than	even	more	from
were	plea	from	have	much	that	some	over	have
over	show	jobs	into	over	were	very	plea	into
plea	than	next	jobs	plea	with	next	were	jobs
fell	that	have	with	fell	fell	lead	ever	lead
time	time	lead	time	back	time	time	fell	loss
loss	were	loss	loss	loss	loss	that	loss	lost
ever	with	over	show	ever	ever	jobs	time	more
lost	also	lost	lost	lost	lost	been	also	much
also	come	more	that	also	also	also	down	next
down	down	much	more	down	down	down	lost	over
said	have	show	said	said	said	said	said	plea
some	lost	plea	some	some	some	from	come	said
have	said	that	were	have	have	have	have	show
very	some	said	very	lead	very	over	some	some
come	very	some	than	come	come	come	very	than
into	back	will	over	that	into	into	back	that
lead	into	very	lead	very	lead	fell	into	time
back	jobs	time	next	time	back	back	lead	very
year	lead	than	year	year	year	than	year	well
will	much	with	will	will	will	show	jobs	were
well	well	well	well	well	well	loss	much	will
much	will	were	much	were	much	much	well	with
jobs	year	year	plea	with	jobs	ever	will	year