CS 477/677 Analysis of Algorithms

Spring 2024

Homework 5 – Solutions

Due date: March 19, 2024

1. (U&G-required) [20 points] Answer the following questions:

(a) [10 points] Illustrate the operation of RADIX_SORT on the following array `A = [29134, 20134, 9134, 134, 34, 4]`. Show the order of the elements after sorting for each digit. **Note:** missing digits should be considered as 0s.

Initial	**Digit 0**	**Digit 1**	**Digit 2**	**Digit 3**	**Digit 4**
29134	2913**4**	000**0**4	00**0**04	0**0**004	**0**0004
20134	2013**4**	291**3**4	00**0**34	0**0**034	**0**0034
09134	0913**4**	201**3**4	29**1**34	2**0**134	**0**0134
00134	0013**4**	091**3**4	20**1**34	0**0**134	**0**9134
00034	0003**4**	001**3**4	09**1**34	2**9**134	**2**0134
00004	0000**4**	000**3**4	00**1**34	0**9**134	**2**9134

(b) [10 points] Illustrate the operation of COUNTING_SORT on the following array `A = [15, 3, 17, 2, 9, 10, 8]`. Show the counting and output arrays after each iteration.

After the first for loop, the counting array C is:

Idx	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Val	0	0	1	1	0	0	0	0	1	1	1	0	0	0	0	1	0	1

After the second for loop, C becomes:

Idx	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Val	0	0	1	2	2	2	2	2	3	4	5	5	5	5	5	6	6	7

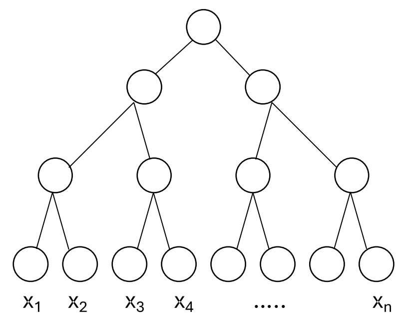
During the last for loop the array B changes as follows:

A = [15, 3, 17, 2, 9, 10, 8].

Idx	1	2	3	4	5	6	7	C array prior	C array after
Val	0	0	**8**	0	0	0	0	C[8] = 3	C[8] = 2
	0	0	8	0	**10**	0	0	C[10] = 5	C[10] = 4
	0	0	8	**9**	10	0	0	C[9] = 4	C[9] = 3
	2	0	8	9	10	0	0	C[2] = 1	C[2] = 0
	2	0	8	9	10	0	**17**	C[17] = 7	C[17] = 6
	2	**3**	8	9	10	0	17	C[3] = 2	C[3] = 1
	2	3	8	9	10	15	17	C[15] = 6	C[15] = 5

2. (U&G-required) [20 points]

Write pseudocode for a procedure BUILD_NEW_MAX_HEAP that takes as input an array A = [x_1, …, x_n] of *n* integer numbers and builds a **max-heap** in which the values at the bottom level are, from left to right, the integers x_1, … x_n (as in the figure on right). Indicate the size of the heap that is created. Assume that *n* is a power of 2.



A heap with n = 2^k nodes (power of two) is a full heap. Since there are n nodes as leaves, there are exactly n - 1 more nodes in the heap above the leaves (all nodes from [heapsize/2]+1 are leaves). Therefore, we need to allocate an array for n (the leaves) + n – 1 nodes to store the new heap.

```
BUILD-NEW-MAX-HEAP(A)
  n = length[A]
  allocate new array B of size heapsize = 2n-1
  copy elements of A into the last n elements of array B
    // the leaves
  for i ← ⌊heapsize/2⌋ downto 1
    do B[i] = B[2i] + B[2i+1]
    // any other operation that combines B[2i] and B[2i+1]
    into something larger than max (B[2i], B[2i+1]) is ok
```

3. (U&G-required) [20 points]

Give a justification to show that the longest simple path from a node x in a red-black tree to a descendant leaf has length at most twice that of the shortest simple path from node x to a descendant leaf.

From property 5 of RBTs, all paths from the root to the descendent leaves have the same number of black nodes. Thus, a shortest path would consist of black nodes only. A longest path would have red nodes alternating with black nodes (from property 4 – no two red nodes in a row). From this, we can infer that the longest path can be no longer than twice the length of the shortest path (for each black node, there is a red node as a child).

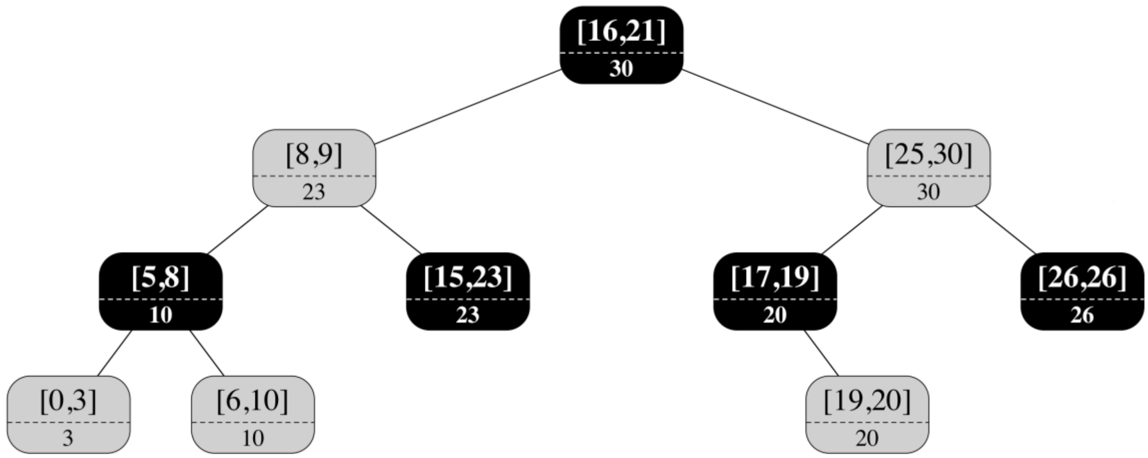
4. (U & G-required) [20 points]

(a) [10 points] Show how INTERVAL-SEARCH(T, i) operates on the tree T shown in the figure below, with `i = [22, 24]`.

1) Check overlap between i = [22, 24] and root [16, 21] – no overlap
2) Compare low[i] = 22 with max[8, 9] = 23: 22 < 23, potential for overlap in left subtree, move search left to [8, 9]
3) Check overlap between i = [22, 24] with [8, 9] – no overlap
4) Compare low[i] = 22 with max[5, 8] = 10: 22 > 10, no possible overlap in left subtree, move search right to [15, 23]
5) Check overlap between i = [22, 24] with [15, 23] – overlap found, return node [15, 23]

(b) [10 points] Show the tree that results after inserting interval `i = [11, 40]` into the tree T shown in the figure below (black nodes have dark background). Make sure to restore any red-black tree properties that may be affected during the insert.

The new node is inserted to the left of [15, 23] as a red node. No RBT properties are affected. However, the additional information (the max fields) for all the nodes on the insert path: [16, 21], [8, 9], [15, 23] needs to be updated to 40.



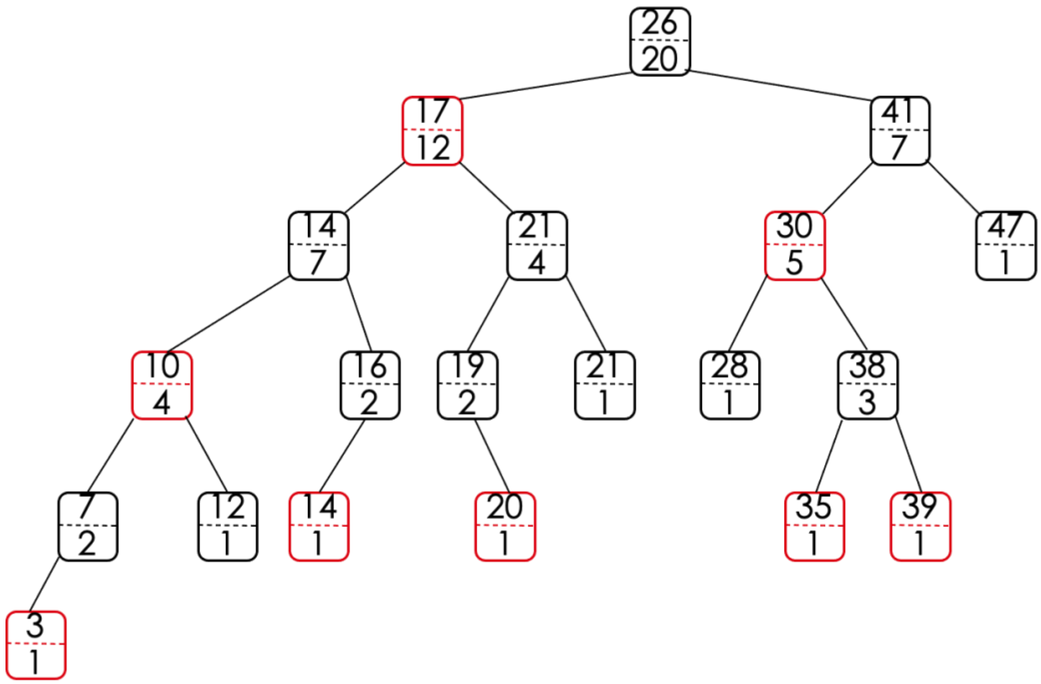
5. (U & G-required) [20 points]

(a) [10 points] Show how OS-SELECT (*T.root*, 16) operates on the red-black tree shown in the figure below.

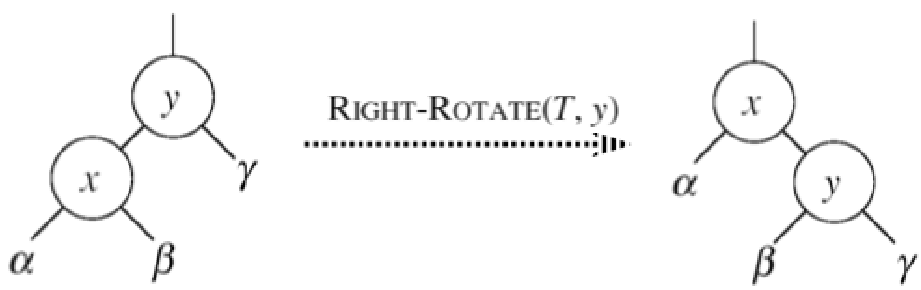
1) Compute rank'[26] = 12+1 = 13
2) 16 > 13, thus continue search to the right subtree for the 16-13 = 3rd order statistic
3) Compute rank'[41] = 5+1 = 6
4) 3 < 6, thus continue search to the left subtree
5) Compute rank'[30] = 1+1 = 2
6) 3 > 2, thus continue search to the right subtree for the 3-2 = 1st order statistic
7) Compute rank'[38] = 1+1 = 2
8) 1 < 2, thus continue search to the left subtree
9) Compute rank'[35] = 1
10) 1 == 1, found, return node 35

(b) [10 points] Show how OS-RANK(*T*, *x*) operates on the red-black tree shown in the figure below and the node x with *x.key* = 20.

1) rank[20] = 1
2) go up to parent 19: rank[20] = rank[20] + 1= 2
3) go up to parent 21, rank remains unchanged
4) go up to parent 17: rank[20] = rank[20] + 7 + 1= 2 + 7 + 1 = 10
5) go up to parent 26, rank remains unchanged
6) Return 10



6. (G-required) [20 points] Let *a*, *b* and *c*, be arbitrary nodes in subtrees α, β, and γ in the left figure below. Indicate how the **depths** of *a*, *b* and *c* change after this transformation.



new_depth[α] = old_depth[α] – 1

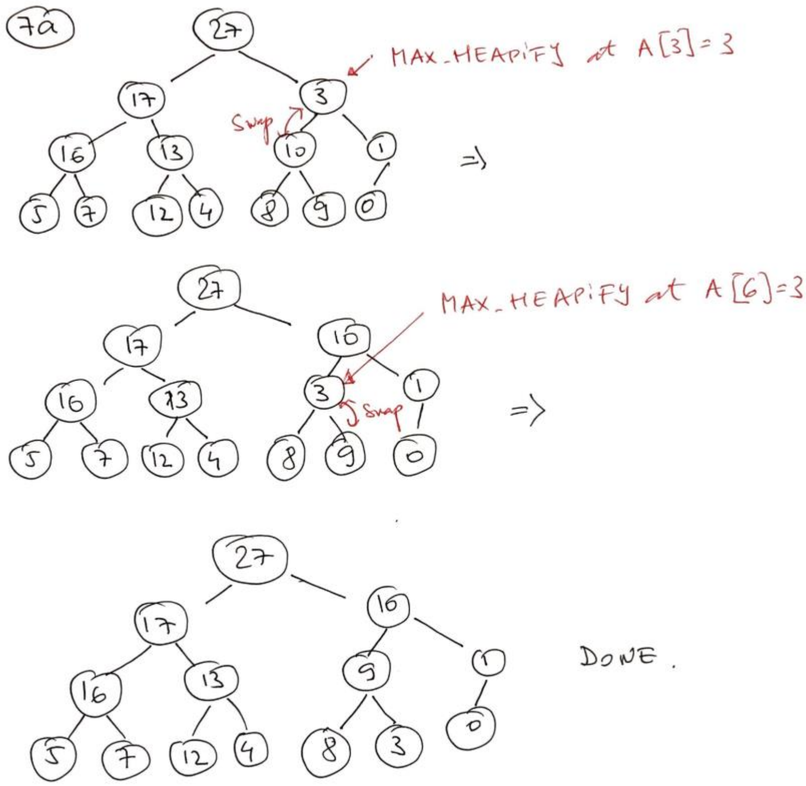
new_depth[β] = old_depth[β]

new_depth[γ] = old_depth[γ] + 1

Extra credit:

7. [20 points]

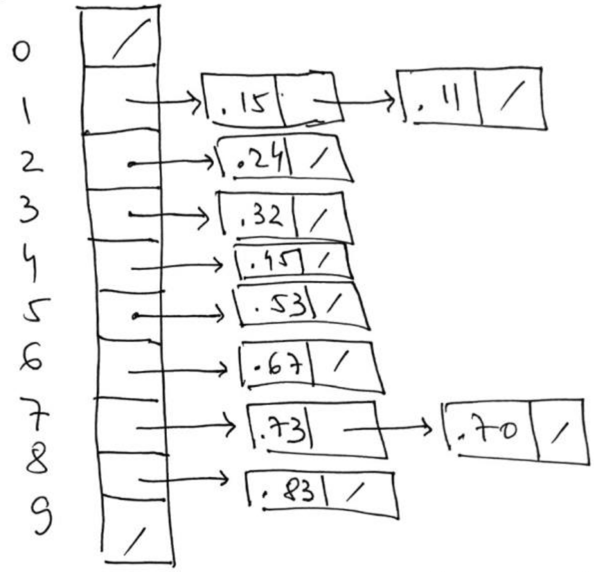
a) [10 points] Illustrate the operation of `MAX-HEAPIFY(A, 3)` on the array `A = [27, 17, 3, 16, 13, 10, 1, 5, 7, 12, 4, 8, 9, 0]`.



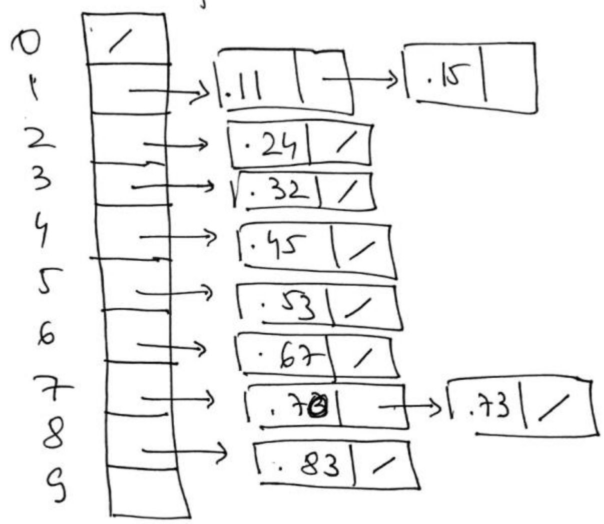
b) [10 points] Illustrate the operation of `BUCKET-SORT` on the array `A = [.73, .15, .11, .67, .32, .24, .83, .53, .70, .45]`.

7b

Step 1: add keys to buckets.



Step 2: sort buckets.



Step 3: concatenate buckets