

CS 477/677 Analysis of Algorithms

Fall 2025

Homework 5 – Solutions

Due date: October 16, 2025

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

(a) [10 points] Illustrate the operation of RADIX_SORT on the following array $A = [23134, 40134, 19134, 31334, 43034, 22454]$. Show the order of the elements after sorting for each digit.

Initial	Digit 0	Digit 1	Digit 2	Digit 3	Digit 4
23134	23134	23134	43034	40134	19134
40134	40134	40134	23134	31334	22454
19134	19134	19134	40134	22454	23134
31334	31334	31334	19134	43034	31334
43034	43034	43034	31334	23134	40134
22454	22454	22454	22454	19134	43034

(b) [10 points] Illustrate the operation of COUNTING_SORT on the following array $A = [8, 4, 5, 1, 3, 5, 8]$. 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
Val	0	1	0	1	1	2	0	0	2

After the second for loop, C becomes:

Idx	0	1	2	3	4	5	6	7	8
Val	0	1	1	2	3	5	5	5	7

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

A = [8, 4, 5, 1, 3, 5, 8].

Idx	1	2	3	4	5	6	7	C array prior	C array after
Val	0	0	0	0	0	0	8	C[8] = 7	C[8] = 6
	0	0	0	0	5	0	8	C[5] = 5	C[5] = 4
	0	3	0	0	5	0	8	C[3] = 2	C[3] = 1
	1	3	0	0	5	0	8	C[1] = 1	C[1] = 0
	1	3	0	5	5	0	8	C[5] = 4	C[5] = 3
	1	3	4	5	5	0	8	C[4] = 3	C[4] = 2
	1	3	4	5	5	8	8	C[8] = 6	C[8] = 5

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

Write pseudocode for an **efficient** algorithm CHECK_HEAP (A) that takes as input an array A of integer values and returns TRUE if the values in the array obey the heap property and returns FALSE otherwise. Analyze the running time of your algorithm.

General solution: for all indices i from 1 to $n/2$ check if $A[i]$ is greater than $A[2i]$ and $A[2i+1]$. Return *true* if all checks are true, return *false* otherwise.

Running time: the algorithm performs $n/2$ basic check operations, only for the elements in the first half of the heap.

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

Write pseudocode for a recursive algorithm RBT_BlackHeight (T) that takes as input a red-black tree and returns the black height of the tree. **Note:** the algorithm cannot use any global variables and cannot take any other parameters than a pointer to a node of the tree (initially the root).

```

Algorithm RBT_BlackHeight (T)
    if T == NIL
        return 0
    else
        if T.left == NIL OR T.left.color == black
            return RBT_BlackHeight(T.left)+1
        else
            return RBT_BlackHeight(T.left)

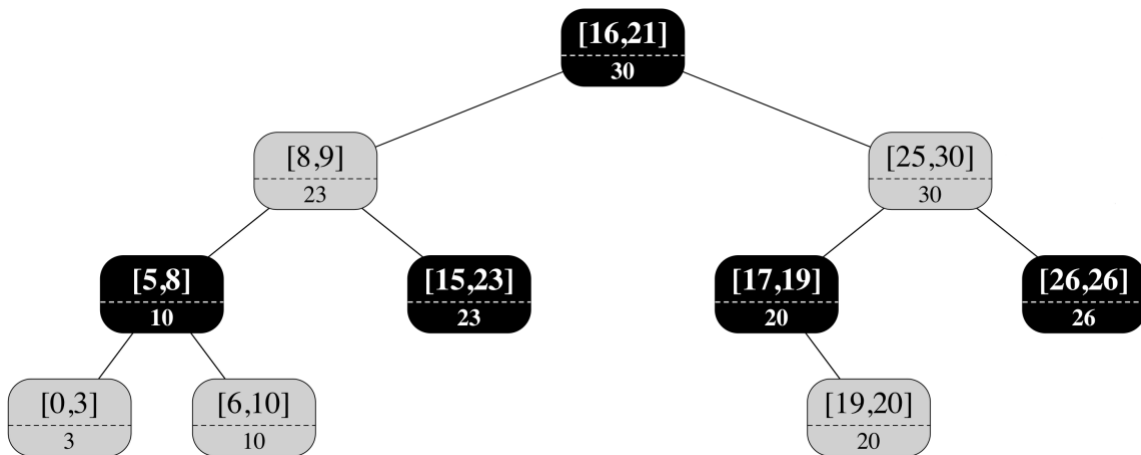
```

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 $i = [10, 14]$.

- 1) Check overlap between $i = [10, 14]$ and root $[16, 21]$ – no overlap
- 2) Compare $\text{low}[i] = 10$ with $\text{max}[8, 9] = 23$: $10 < 23$, potential for overlap in left subtree, move search left to $[8, 9]$
- 3) Check overlap between $i = [10, 14]$ with $[8, 9]$ – no overlap
- 4) Compare $\text{low}[i] = 10$ with $\text{max}[5, 8] = 10$: $10 == 10$, potential for overlap in left subtree, move search left to $[5, 8]$
- 5) Check overlap between $i = [10, 14]$ with $[5, 8]$ – no overlap
- 6) Compare $\text{low}[i] = 10$ with $\text{max}[0, 3] = 3$, no possible overlap in left, move search right to $[6, 10]$
- 7) Check overlap between $i = [10, 14]$ with $[6, 10]$ – overlap ($\text{low}[i] == \text{high}([6, 10])$), return $[6, 10]$

(b) [10 points] Show the tree that results after inserting interval $i = [11, 15]$ 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.



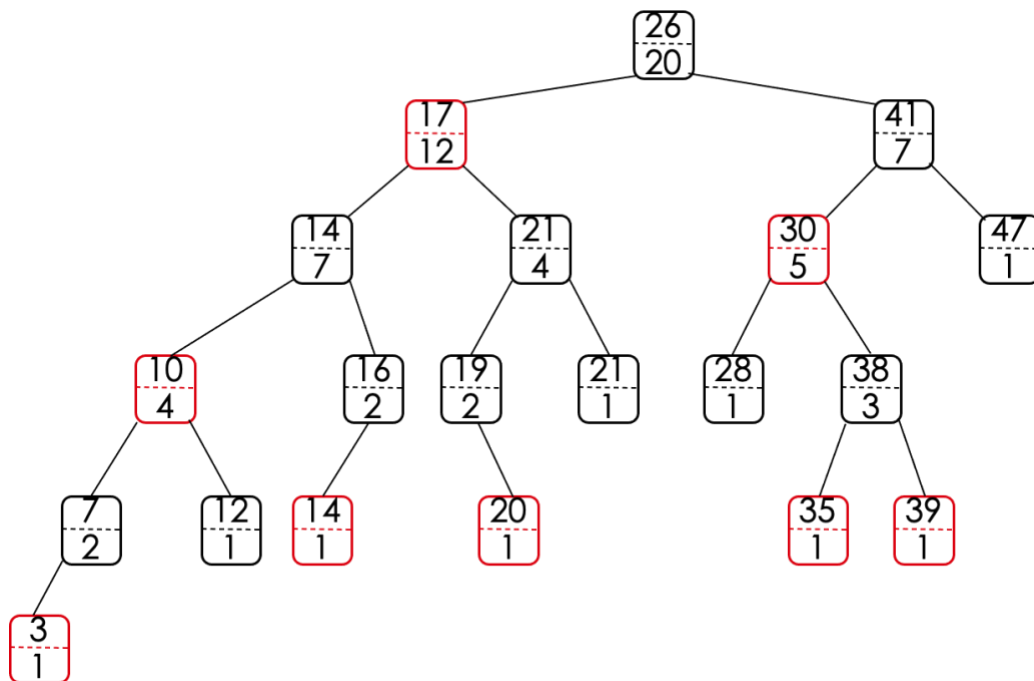
The new node is inserted to the left of $[15, 23]$ as a red node. No RBT properties are affected.

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 $\text{rank}'[26] = 12 + 1 = 13$
- 2) $16 > 13$, thus continue search to the right subtree for the $16 - 13 = 3^{\text{rd}}$ order statistic
- 3) Compute $\text{rank}'[41] = 5 + 1 = 6$
- 4) $3 < 6$, thus continue search to the left subtree for the 3^{rd} order statistic
- 5) Compute $\text{rank}'[30] = 1 + 1 = 2$
- 6) $3 > 2$, this continue search to the right subtree for the $3 - 2 = 1^{\text{st}}$ order statistic
- 7) Compute $\text{rank}'[38] = 1 + 1 = 2$
- 8) $1 < 2$, thus continue search to the left subtree for the 1^{st} order statistic
- 9) Compute $\text{rank}'[35] = 1$, found, return 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 $x.key = 39$.

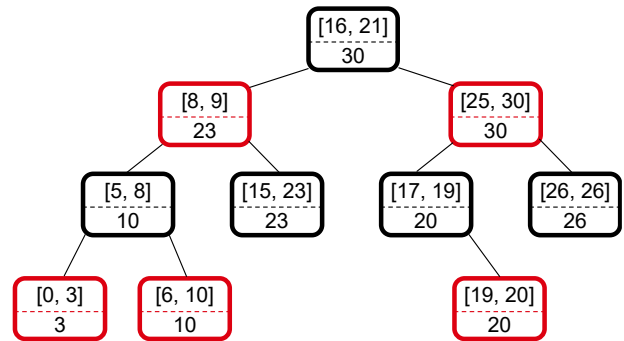


- 1) $\text{rank}[39] = 1$
- 2) go up to parent 38: $\text{rank}[39] = \text{rank}[39] + 1 + 1 = 3$
- 3) go up to parent 30, $\text{rank}[39] = \text{rank}[39] + 1 + 1 = 5$

- 4) go up to parent 41, rank remains unchanged
- 5) go up to parent 26, $\text{rank}[39] = \text{rank}[39] + 12 + 1 = 18$
- 6) Return 18

6. (G-required) [20 points] Write pseudocode

for an algorithm `Find_Shortest_Int(T)` that takes as input a interval tree and returns a pointer to the interval that has the shortest length in the tree. We define the length has the difference between the high and the low end point of the interval. For example, for the tree on the right, the interval with the smallest length is $[26, 26]$, having a length of zero.



```

Algorithm Find_Shortest_Int(T)
// variables leftInt, rightInt
if T == NIL
    return NIL
else
    leftInt = Find_Shortest_Int (T.left)
    rightInt = Find_Shortest_Int (T.right)

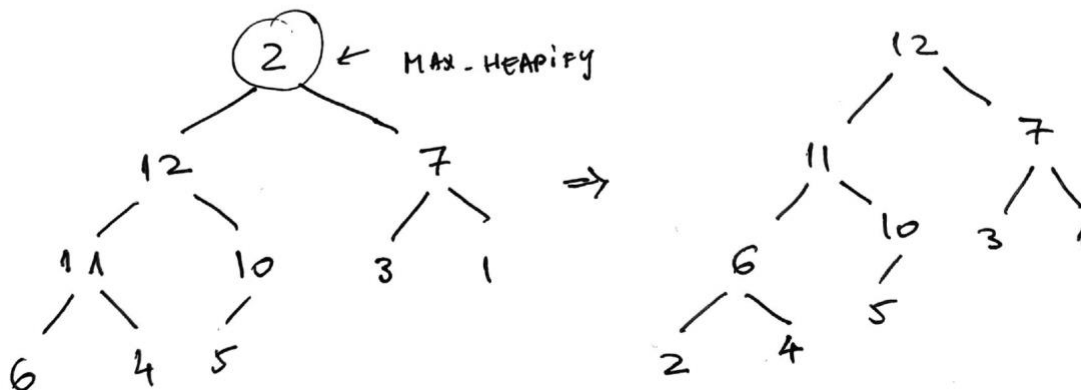
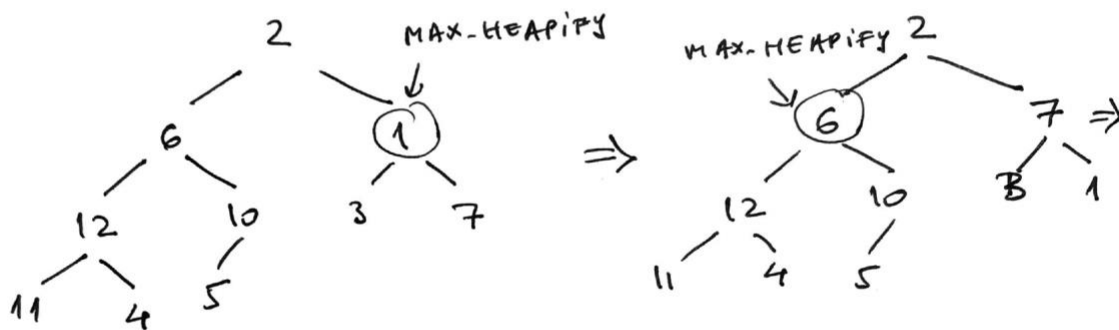
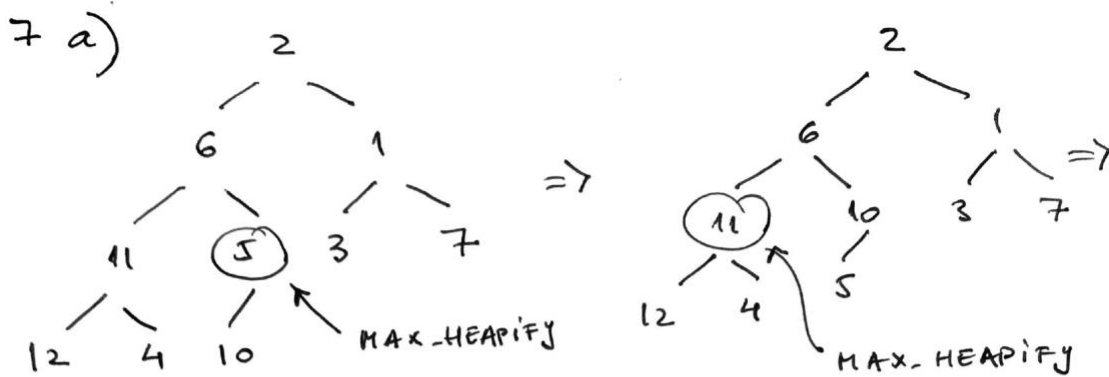
    // Multiple ways to implement this part:
    // checking for leftInt, rightInt not to be NIL:

    return the interval with the smallest length from
    leftInt, rightInt, T
  
```

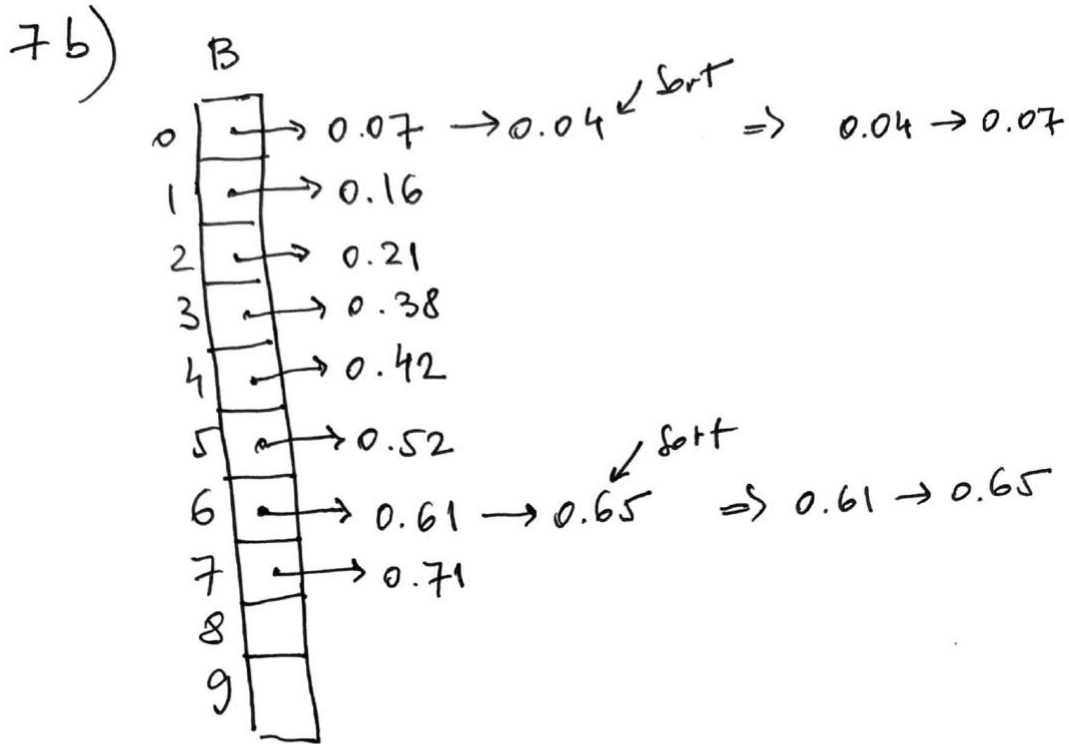
Extra credit:

7. [20 points]

a) [10 points] Illustrate the operation of BUILD-MAX-HEAP(A) on the array $A = [2, 6, 1, 11, 5, 3, 7, 12, 4, 10]$.



b) [10 points] Illustrate the operation of BUCKET-SORT on the array $A = [.61, .07, .04, .52, .21, .16, .71, .42, .65, .38]$.



Concatenate lists:

0.04 → 0.07 → 0.16 → 0.21 → 0.38 → 0.42 → 0.52 →
 → 0.61 → 0.65 → 0.71