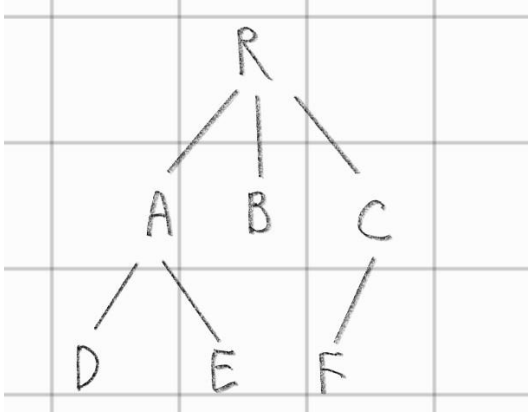
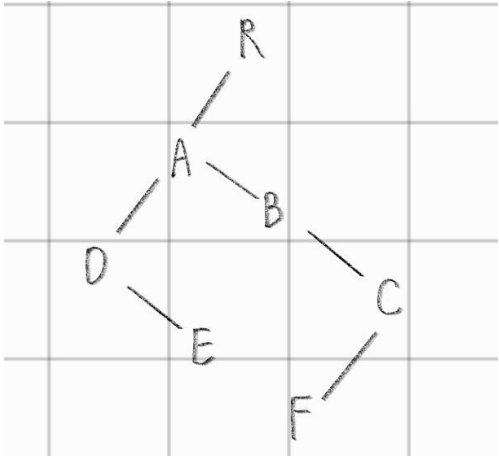


M1522.000900 Data Structure
Spring 2025, Kang
Homework 3 Answer Sheet [1/3]

2023-12753 EunSu Yeo

- ※ Write your answers on the “your answer” columns.
- ※ Do **NOT** write anything on “score” columns.
- ※ Before submission, delete all the blue-colored texts, and convert the file into **PDF-format**.
- ※ Write your proof on the solution sheet and leave the “your answer” column blank.

Question		Your Answer	Points	Score
1	(1)	node.val	2	
		dfs(child, result);	2	
		“)”	2	
	(2)	AB) C) DE)))	4	
2	(1)		7	
	(2)		8	

M1522.000900 Data Structure
Spring 2025, Kang
Homework 3 Answer Sheet [2/3]
 2023-12753 EunSu Yeo

Question	Your Answer	Points	Score																																								
3	<table><tr><td>Node</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td></tr><tr><td>Parent</td><td>12</td><td>0</td><td>4</td><td>2</td><td>/</td><td>4</td><td>4</td><td>8</td></tr></table> <table><tr><td>Node</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td></tr><tr><td>Parent</td><td>4</td><td>8</td><td>4</td><td>10</td><td>/</td><td>12</td><td>12</td><td>12</td></tr></table>	Node	0	1	2	3	4	5	6	7	Parent	12	0	4	2	/	4	4	8	Node	8	9	10	11	12	13	14	15	Parent	4	8	4	10	/	12	12	12	15					
Node	0	1	2	3	4	5	6	7																																			
Parent	12	0	4	2	/	4	4	8																																			
Node	8	9	10	11	12	13	14	15																																			
Parent	4	8	4	10	/	12	12	12																																			
4	<p>1. List of children: $\frac{n(I + 4P) - 2P}{n(I + D + 4P) - 2P}$</p> <p>2. Left-child/Right-sibling: $\frac{I+3P}{I+D+3P}$</p>	10																																									
5	<table><tr><td>41</td><td>25</td><td>11</td><td>3a</td><td>75</td><td>12</td><td>3b</td><td>10</td><td>49</td><td>53</td></tr><tr><td>3a</td><td>41</td><td>25</td><td>11</td><td>3b</td><td>75</td><td>12</td><td>10</td><td>49</td><td>53</td></tr><tr><td>3a</td><td>3b</td><td>41</td><td>25</td><td>11</td><td>10</td><td>75</td><td>12</td><td>49</td><td>53</td></tr><tr><td>3a</td><td>3b</td><td>10</td><td>41</td><td>25</td><td>11</td><td>12</td><td>75</td><td>49</td><td>53</td></tr></table>	41	25	11	3a	75	12	3b	10	49	53	3a	41	25	11	3b	75	12	10	49	53	3a	3b	41	25	11	10	75	12	49	53	3a	3b	10	41	25	11	12	75	49	53	10	
41	25	11	3a	75	12	3b	10	49	53																																		
3a	41	25	11	3b	75	12	10	49	53																																		
3a	3b	41	25	11	10	75	12	49	53																																		
3a	3b	10	41	25	11	12	75	49	53																																		
6	2^m	10																																									

M1522.000900 Data Structure
Spring 2025, Kang
Homework 3 Answer Sheet [3/3]
 2023-12753 EunSu Yeo

Question		Your Answer	Points	Score
7	(1)	$O(n \log n)$	5	
	(2)	$O(n^2)$	5	
	(3)	$O(n \log n)$	5	
	(4)	$O(\frac{n}{\log n})$	5	
8	42, 59, 91		10	

Homework 3 Solution Sheet

2023-12753

EunSu Yeo

1. Question 1 [10 points]

1.1. Q1(1) [3 points]

Solution.

[Write here](#)

1.2. Q1(2) [3 points]

Solution.

[Write here](#)

1.3. Q1(3) [3 points]

Solution.

[Write here](#)

1.4. Q1(4) [3 points]

Solution.

[Write here](#)

2. Question 2 [15 points]

2.1. [7 points]

Solution.

[Write here](#)

2.2. [8 points]

Solution.

[Write here](#)

3. Question 3 [15 points]

Solution.

[Write here](#)

(0,1)

Both sets have equal size (1,1)

Root 1 > 0 → attach 1 under 0

parent[1] = 0

size[0] = 2

(2,3)

Equal size (1,1)

Root 3 > 2 → attach 3 under 2

parent[3] = 2

size[2] = 2

(4,5)

Equal size (1,1)

5 > 4 → attach 5 under 4

parent[5] = 4

size[4] = 2

(6,5)

Find roots: 6 is its own root, 5's root is 4

Sizes: 6 has 1, 4 has 2

Smaller set 6 attaches under 4

parent[6] = 4

size[4] = 3

(3,5)

Root of 3 is 2 (parent[3] = 2)

Root of 5 is 4

Sizes: 2 has 2, 4 has 3

Smaller root 2 attaches under 4

parent[2] = 4

size[4] = 5 (2+3)

(8,9)

Equal size (1,1)

9 > 8 → attach 9 under 8

parent[9] = 8

size[8] = 2

(7,8)

Roots: 7 is root, 8 is root
 Sizes: 7 has 1, 8 has 2
 Smaller root 7 attaches under 8
 $\text{parent}[7] = 8$
 $\text{size}[8] = 3$

(2,7)

Find roots: $\text{parent}[2] = 4 \rightarrow \text{root } 4$
 $\text{parent}[7] = 8 \rightarrow \text{root } 8$
 Sizes: 4 has 5, 8 has 3
 Smaller root 8 attaches under 4
 $\text{parent}[8] = 4$
 $\text{size}[4] = 8$

(10,11)

Equal size (1,1)
 $11 > 10 \rightarrow \text{attach } 11 \text{ under } 10$
 $\text{parent}[11] = 10$
 $\text{size}[10] = 2$

(12,15)

Equal size (1,1)
 $15 > 12 \rightarrow \text{attach } 15 \text{ under } 12$
 $\text{parent}[15] = 12$
 $\text{size}[12] = 2$

(13,15)

Roots: 13 is root, 15's root is 12
 Sizes: 13 has 1, 12 has 2
 Smaller root 13 attaches under 12
 $\text{parent}[13] = 12$
 $\text{size}[12] = 3$

(14,13)

Roots: 14 is root, 13's root is 12
 Sizes: 14 has 1, 12 has 3
 Smaller root 14 attaches under 12
 $\text{parent}[14] = 12$
 $\text{size}[12] = 4$

(6,11)

Roots: 6's root is 4
 11's root is 10
 Sizes: 4 has 8, 10 has 2
 Smaller root 10 attaches under 4
 $\text{parent}[10] = 4$
 $\text{size}[4] = 10$

(0,14)

$\text{size}[0] = 2$
 $\text{size}[12] = 4$
 Weighted union rule: attach the smaller set under the larger set

Since $2 < 4$, the root of 0's set attaches under the root of 12's set
 Therefore, $\text{parent}[0] = 12$

4. Question 4 [10 points]

Solution.

[Write here](#)

In the List of Children representation, each row in a table stores the index, value, and a pointer to the parent node. The child nodes that have this node as their parent are connected via a linked list.

Each linked list node consists of a pointer to the index and a pointer to the next node in the list.

Let n be the total number of nodes.

Data space: nD

Index space: nI (index of the table)

Pointer space:

nP (pointer to parent)

$(2n - 1)P$ (pointers to children)

$(n - 1)P$ (index pointers stored in the linked list)

Total space: $n(I + D + 4P) - 2P$

Overhead (non-data space): $n(I + 4P) - 2P$

Space overhead =

$$\frac{n(I + 4P) - 2P}{n(I + D + 4P) - 2P}$$

In the Leftmost Child/Right Sibling representation, each node has pointers to the leftmost child and the right sibling.

Let n be the total number of nodes.

Data space: nD

Index space: nI

Pointer space:

nP (pointer to parent)

$n \times 2P$ (pointers to leftmost child and right sibling)

Total space: $n(I + D + 3P)$

Overhead (non-data space): $n(I + 3P)$

Space overhead =

$$\frac{n(I + 3P)}{n(I + D + 3P)} = \frac{I + 3P}{I + D + 3P}$$

5. Question 5 [10 points]

[Write here](#)

By using bubble sort in the ppt 16, at each iteration, "Bubble up" the smallest element from the input to the output.

6. Question 6 [10 points]

[Write here](#)

The mergeSort function is called $n = 2^m - 1$ times.

For subarrays of size less than or equal to the threshold, insertionSort is called instead.

In the recursion tree, the number of calls to insertion sort corresponds to the number of leaf nodes.

We can think the Leaf nodes as the points where mergeSort stops and calls insertionSort.

Therefore, the number of insertion sort calls equals the number of leaf nodes, which is 2^m for a complete binary tree.

Hence, the number of insertion sort calls is 2^m

7. Question 7 [20 points]

7.1. [5 points]

Solution.

[Write here](#)

Merge Sort always divides and merges the array regardless of input order, so the best-case time complexity remains $O(n \log n)$.

7.2. [5 points]

Solution.

[Write here](#)

Choosing the minimum value as the pivot results in the worst possible partition each time (highly unbalanced splits), leading to a time complexity of $O(n^2)$.

7.3. [5 points]

Solution.

[Write here](#)

Since the middle index pivot divides the array more evenly even when the input is sorted, the time complexity is $O(n \log n)$

7.4. [5 points]

Solution.

[Write here](#)

Since sorting all n elements requires at least $O(n \log n)$ comparisons, sorting k elements in $O(n)$ limits k .

Finding the **maximum element** in an array takes $O(n)$ time.

Finding the **second largest element** can be done in $O(\log n)$ time using efficient methods such as a tournament tree.

Similarly, finding the third, fourth, ..., k -th largest element can be done in roughly $O(\log n)$ time each.

Therefore, selecting the top k elements can be done in about $O(n + k \log n)$ time.

Hence, the total time complexity is approximately $O(n+k\log n)$, and to keep it $O(n)$, we need $k\log n = O(n)$.

$k = O\left(\frac{n}{\log n}\right)$ may be the answer.

8. Question 8 [10 points]

Solution.

[Write here](#)

As the problem gives us that given array is partitioned by the partition function shown in quicksort, we can think that l and r is fixed to $l = \text{start index of this array}$ and $r = \text{end index of this array}$. So, we need to find the possible pivots.

In partition function, we know that given array is divided into 2 parts which is lower than the pivot and which is greater than the pivot value.

So, we need to divide the given array into 2 parts using assumed pivot.

Due to this step, we can find 42 and 59 can be pivot.

The elements before the 42 is smaller than 42 and the elements after the elements is bigger than the 42 in the given array. So, 42 can be the pivot.

As the same reason 59 and 91 can be the pivot.