School of ECE, UNIST
CSE221 (Data Structures)
Spring 2019

Instructor:
Hyungon Moon

Final, 19:00 - 22:00 (180min) Jun 7, 2019

Name (In English):	
Student ID:	

Instruction:

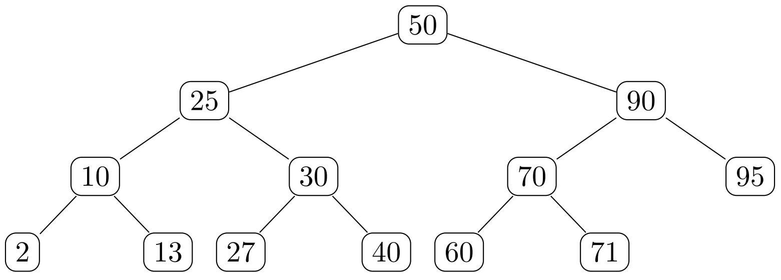
1. Submit all the sheets.
2. No question, no leave for the first 45 minutes of the exam.
3. Write your student ID at every page, bottom left.
4. Your answers should be printed at the designated locations, and written with an **inerasable black or blue** pen.
5. You are not allowed to go to the rest room and come back.

Question	Points	Score
1	20	
2	15	
3	15	
4	20	
5	15	
6	15	
7	10	
8	15	
9	10	
Total:	135	

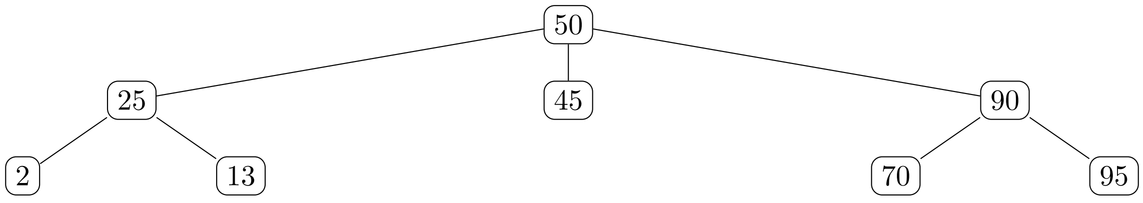
Enjoy!

1. **Binary Search Tree**

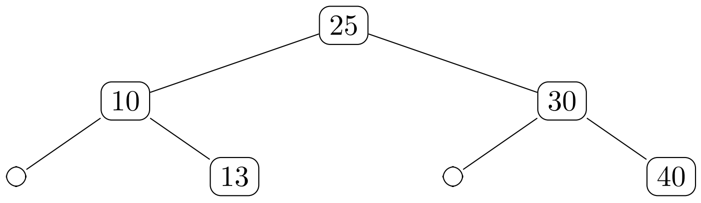
   (a) [5 points] Determine if each of these tree is a Binary Search Tree of not. (1 pt each).



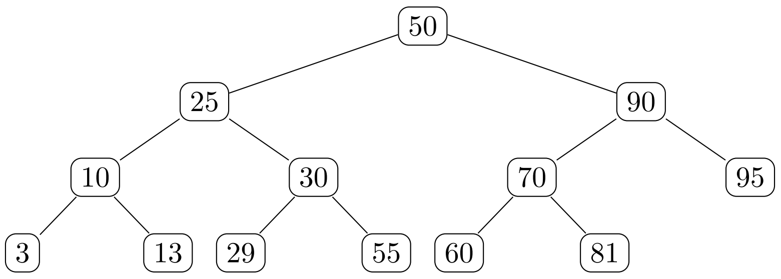
Answer (Yes/No): **Yes**



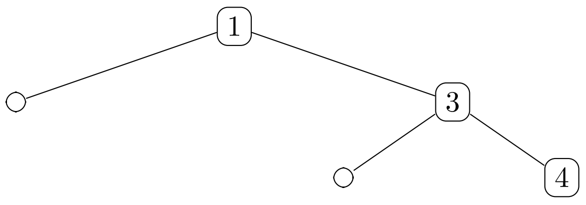
Answer (Yes/No): **No**



Answer (Yes/No): **YES**



Answer (Yes/No): **No**



Answer (Yes/No): **Yes**

Student ID:

Question 1 continues on the next page. . .

(b) [5 points] State the set of properties that a Binary Search Tree satisfies, which is considered as the definition of it. Use the following terms. Presume that the ordering in the set of keys is already well-defined.

- The set of nodes N.
- The set of keys that are associated with the nodes K.
- The left subtree of a node $n \in N$, T_L^n.
- The right subtree of a node $n \in N$, T_R^n.

Solution:

1. For all pairs of $k_1, k_2 \in K$, $k_1 \neq k_2$ (Unique keys).
2. For all $n \in N$, the keys of the nodes in T_L^n are all smaller than the key of n.
3. For all $n \in N$, the keys of the nodes in T_R^n are all larger than the key of n.
4. T_L^n is a Binary Search Tree.
5. T_R^n is a Binary Search Tree.

   Criteria:

   - 1 point per the item.
   - The answer does not to be formal. Just the ones with the correct meaning is considered a correct answer.

(c) [4 points] Fill out the blanks in the following code snippet implementing the search operation for Binary Search Tree. Write your answer in the blanks under the code (2pt each).

```cpp
struct Node_t {
    long key; long data;
    Node_t* parent, *left, *right;
};
Node_t* search_subtree(Node_t* root, long key) const {
    if(___(1)___) return root;
    else if (___(2)___) {
        if(root->left) return search_subtree(root->left, key);
        else return root;
    }
    else {
        if(root->right) return search_subtree(root->right, key);
        else return root;
    }
}
```

- (1): `key == root->key`
- (2): `key < root->key`

Solution: Any answer with the same semantics are considered correct.

Question 1 continues on the next page...

(d) [6 points] Fill out the blanks in the following code snippet implementing the remove operation for Binary Search Tree. Write your answer in the blanks under the code (2pt each).

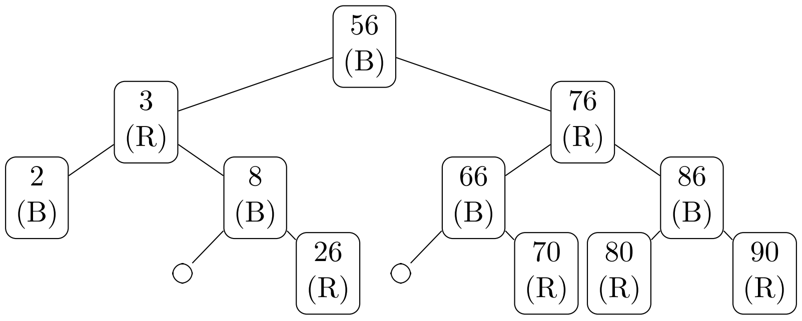
```
struct Node_t {
    long key; long data; Node_t* parent, *left, *right;
};
Node_t* remove_helper(Node_t* root) {
    if(___(1)___) return remove_helper(root->left);
    else return root;
}
Node_t* remove_node(Node_t* n) {
    Node_t* p = nullptr;
    if(___(2)___) {
        Node_t* r = remove_helper(n->right);
        n->key = r->key; n->data = r->data;
        return remove_node(r);
    } else if (___(3)___) {
        n->left->parent = n->parent;
        if(n->parent) {
            if(n == n->parent->left) n->parent->left = n->left;
            else n->parent->right = n->left;
        }
        /* some details omitted */
        p = n->parent;
        delete n;
    } else if (n->right) {
        n->right->parent = n->parent;
        /* some details omitted */
        p = n->parent;
        delete n;
    } else {
        if(n->parent) {
            if(n == n->parent->left) n->parent->left = nullptr;
            else n->parent->right = nullptr;
        }
        Node_t* root; // The pointer to the root of the tree.
        p = n->parent;
        delete n;
    }
    return p;
}
```

- (1): `root->left`
- (2): `n->left && n->right`
- (3): `n->left`

Solution: Any answer with the same semantics are considered correct.

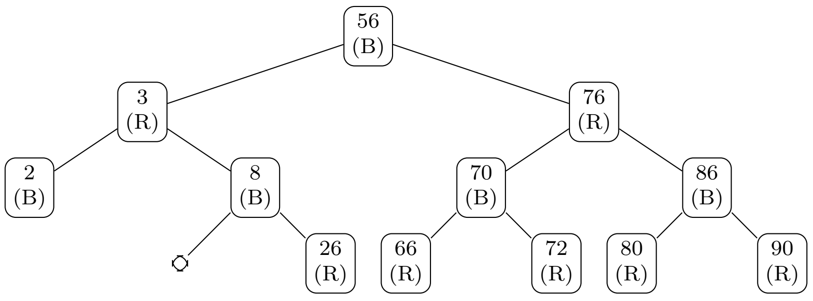
2. **Red Black Tree**

   (a) [6 points] Given this Red Black Tree, illustrate the tree after each of the following operations (No in sequence). In other words, illustrate the tree when we applied each operation to the given tree. R/B in each node represents the color. Represent the color in your illustration in the same way. Note that all external nodes with no data are omitted (and you should also omit). (2 pt each)



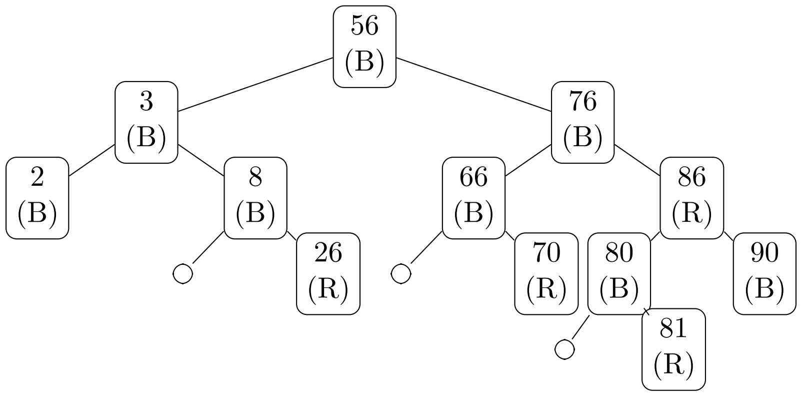
   1) Insert 72.

Solution:



   2) Insert 81

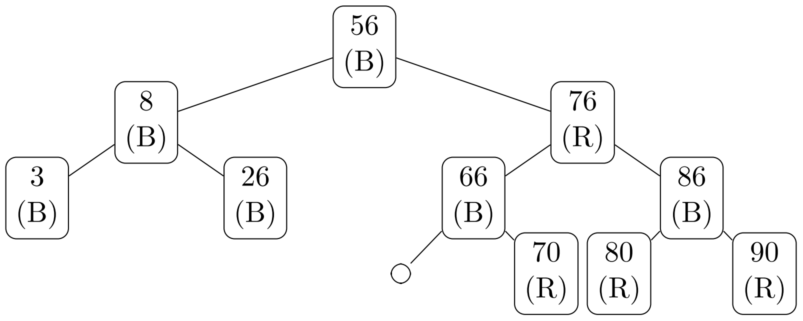
Solution:



Question 2 continues on the next page. . .

3) Remove 2

Solution:



(b) [6 points] Presuming every node of a tree is colored either black or red, under which condition the tree is considered a Red Black Tree? State the three conditions.

Solution:

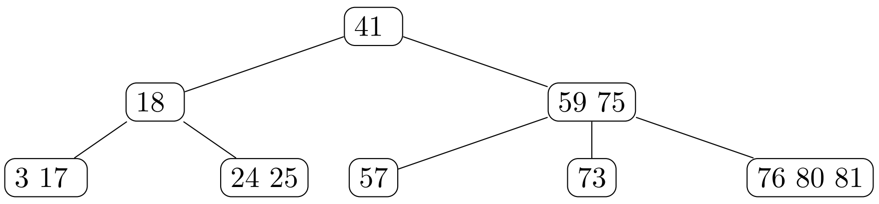
1. The root and all external nodes are black.
2. No path from the root to an external node has two consecutive Red nodes.
3. All paths from the root to external nodes have the same number of the black nodes.

(c) [3 points] Why is the height of a Red Black Tree is $O(\log n)$ in the worst case, when n is the number of nodes in the tree?

Solution:
$Length_{longest} <= 2Length_{shortest}$
Let r be the number of black nodes in a path from the root to an external node.
Then $h = Length_{longest} <= 2r$
The maximal number of nodes such a tree can have is,
$n >= 2^r - 1$
when all nodes are black.
Combining the two we get:
$h <= 2r <= \log 2n + 1$
Therefore
$h = O(\log n)$

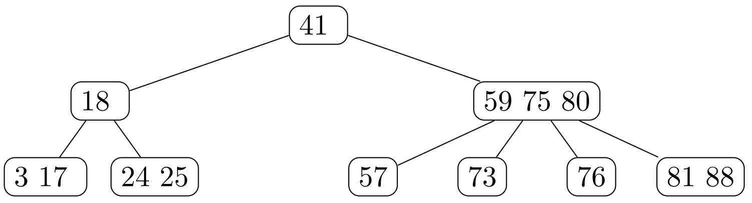
3. **B Tree**

(a) [6 points] Given this B Tree of maximal degree 4 (a.k.a., (2,4) tree), illustrate the result of performing the two given operations (each from the given tree).



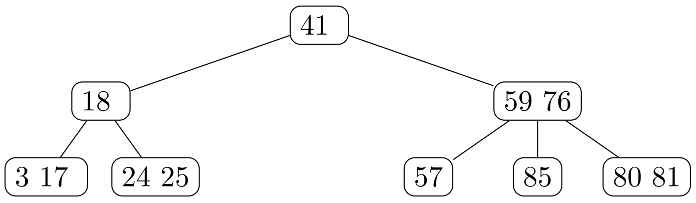
1) Insert 88

Solution:



2) Remove 73

Solution:



(b) [5 points] Show that the height h of a $(2,4)$ tree is $O(\log n)$ where n is the number nodes in the tree.

Solution: Given a tree of height h and n nodes, at level i, there are at least 2^i items. This gives us the following relationship:

$n >= 1 + 2 + \cdots + 2^{h-1} = 2^h = 1$

$h <= \log 2n + 1$

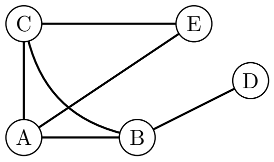
Therefore, we have $h = O(\log n)$

(c) [4 points] What is the expected benefit of using B trees over Binary Search Trees? State one of them. Incorrect discussion will reduce your total score.

Solution: On modern computer systems, the cost of memory access is relatively expensive. B tree reduces the number of nodes being visited, which determines the number of memory (cache line) accesses. Under such an assumption, B tree is expected to have better performance compared to the Binary Search Tree.

4. **Graph**

For the first two questions, refer to this graph.



(a) [5 points] Illustrate the adjacency matrix representation of the given graph.

Solution:

	A	B	C	D	E
A	0	1	1	0	1
B	1	0	1	1	0
C	1	1	0	0	1
D	0	1	0	0	0
E	1	0	1	0	0

(b) [5 points] Illustrate the adjacency list representation of the given graph.

Solution:

A $\boxed{} \rightarrow$ B $\rightarrow$ C $\rightarrow$ E

B $\boxed{} \rightarrow$ A $\rightarrow$ C $\rightarrow$ D

C $\boxed{} \rightarrow$ A $\rightarrow$ B $\rightarrow$ E

D $\boxed{} \rightarrow$ B

E $\boxed{} \rightarrow$ A $\rightarrow$ C

Criteria: 1 pt per row.

Question 4 continues on the next page. . .

(c) [5 points] State the time complexity of finding all incident edges of a vertex v using each of the two representations discussed above and explain why. Consider $deg(x)$ be the degree of vertex x and n be the number of all nodes.

Solution:

Adjacency list: $O(deg(v))$ (1pt)

Why: We can simply traverse one list containing the incident edges of v, which has $deg(v)$ elements. (1.5pt)

Adjacency matrix: $O(n)$. (1pt)

Why: We have to check all elements in one row (or column) in the matrix, whose size is n. (1.5pt)

(d) [5 points] State the time complexity of testing if a vertex v is adjacent to u and explain why.

Solution: Adjacency list: $O(min(deg(v) + deg(u)))$ (1pt)

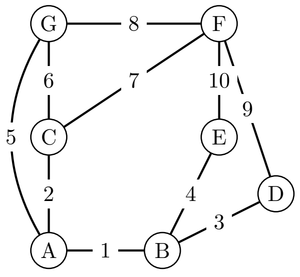
Why: We can simply traverse one list containing the incident edges of v or u, whichever is shorter. (1.5pt)

Adjacency matrix: $O(1)$. (1pt)

Why: We only have to check only one element in the matrix which we can locate using the two vertices with $O(1)$ time. (1.5pt)

5. **Graph Traversal**

Answer the questions refering to this graph.



(a) [5 points] Perform DFS from A and state the sequence of visited node. When visiting incident edges, visit the edge with smaller ID first. In addition, present the list of back edges.

Solution: DFS (3pt): A, B, D, F, C, G, E
back edges (2pt): 2, 4, 5, 8

(b) [5 points] Perform BFS from A,state the sequence of visited node, and list all cross edges. When visiting incident edges, visit the edge with smaller ID first. In addition, present the list of back edges.

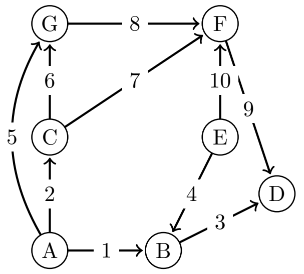
Solution: BFS (3pt): A, B, C, G, D, E, F
cross edges(2pt): 6, 9, 10, 8

(c) [5 points] A way to find cycles in a graph is to augment the DFS algorithm. In particular, each back edge found while DFS gives one cycle. State one cycle per each back edge you have found in (a). Represent it as a sequence of nodes. When visiting incident edges, visit the edge with smaller ID first.

Solution: 2: A - B - D - F - C - A
4: B - D - F - E - B
5: A - B - D - F - C - G - A
8: F - C - G - F
Criteria: answers do not need to include the back edges., 1.25pt each.

6. **Directed Graph**

Answer the questions refering to this directed graph.



(a) [2 points] List all reachable vertices from C.

> **Solution:** C, G, F, D

(b) [8 points] State all possible results of topological sorting from A.

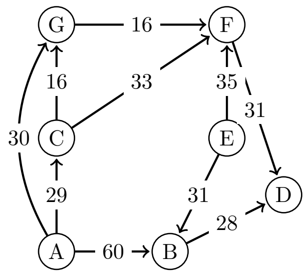
> **Solution:**
>
> 1. A, B, C, G, F, D
> 2. A, C, B, G, F, D
> 3. A, C, G, F, B, D
> 4. A, C, G, B, F, D

(c) [5 points] Given an arbitrary graph represented as $G = (V, E)$ in which V is the set of vertices and E is the set of edges, describe an $O(|V| + |E|)$ algorithm of testing if the graph is strongly connected or not and explain why it works. To explain, describe how you can find the path between a pair of vertices (u, v).

> **Solution:** Perform DFS using an arbitrary vertex A, and again with inverted edges. To find a path between u and v, we can think of two cases. First, such a path could be found from the former DFS tree. Second, one can obtain a path from u to A using the second DFS tree by reverting the path from A to u, and then concatenate with the path from A to v from the former DFS tree.
>
> Criteria
>
> - Algorithm: 4pt
> - Path: 1pt
> - One cannot replace the DFS with topological sort from this discussion and will get 2pt.

7. **Dijkstra**

We'd like to find the shortest path from A to D.



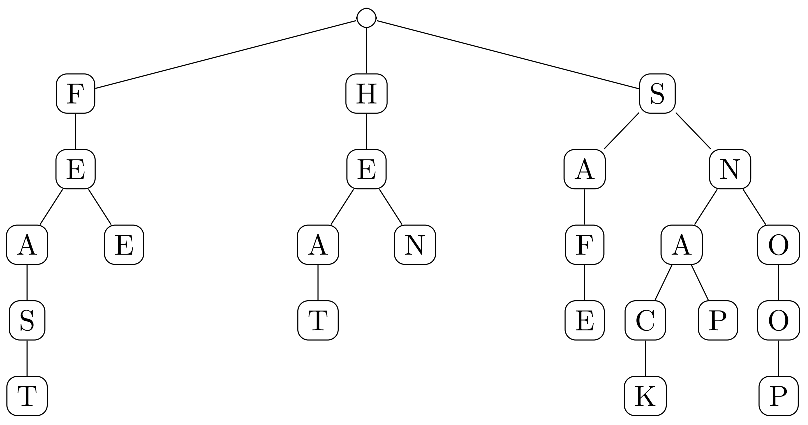
(a) [10 points] Fill out the table built while running the Dijkstra's algorithm to find the shortest path from A. At the first column (labeled S), put the vertices that we already have the length of shortest path. For each column for the vertices, fill out the length of the shortest path and the path itself.

S	A	B	C	D	E	F	G
$\{\}$	0;(A)	inf	inf	inf	inf	inf	inf
$\{A\}$	0;(A)	60;(A,B)	29;(A,C)	inf	inf	inf	30;(A,G)
$\{A, C\}$	0;(A)	60;(A,B)	29;(A,C)	inf	inf	62;(A,C,F)	30;(A,G)
$\{A, C, G\}$	0;(A)	60;(A,B)	29;(A,C)	inf	inf	46;(A,G,F)	30;(A,G)
$\{A, C, F, G\}$	0;(A)	**60;(A,B)**	29;(A,C)	**77;(A,G,F,D)**	**inf**	**46;(A,G,F)**	30;(A,G)
$\{A, C, B, F, G\}$	0;(A)	**60;(A,B)**	29;(A,C)	**77;(A,G,F,D)**	**inf**	**46;(A,G,F)**	30;(A,G)
$\{A, C, B, D, F, G\}$	0;(A)	**60;(A,B)**	29;(A,C)	**77;(A,G,F,D)**	**inf**	**46;(A,G,F)**	30;(A,G)

8. **String and Pattern Matching**

(a) [5 points] Illustrate the standard trie that contains this set S of strings.
$S = \{$ HEAT, FEAST, FEE, SAFE, SNAP, SNACK, HEN, SNOOP$\}$

Solution:



Criteria: No partial

(b) [5 points] We'd like to run the Knuth-Norris-Pratt (KMP) algorithm to find the first location of a pattern P from a string T shown below. Fill out the table representing the failure function, and present the amounts of shifts we are making as a list of shift amounts. For example, the first shift you will find will be due to the mismatch between $T_1 = b$ and $P_1 = c$ and you will shift the pattern by 1.
$T = (a, b, a, c, a, a, b, a, c, c, a, b, a, c, a, c, a, a, b, b)$
$P = (a, c, a, c, a, a)$

Figure 1: Failure Function

j	0	1	2	3	4	5
$P[j]$	a	c	a	c	a	a
$F[j]$	**0**	**0**	**1**	**2**	**3**	**1**

The sequence of shift amounts.

Solution: 1, 1, 2, 1, 1, 1, 2, 1, 1, 1
Criteria

- Table: 2pt (no partial)
- Amounts: 3pt (no partial)

(c) [5 points] Sketch the proof of why the KMP has the worst case time complexity of $O(n)$.

Solution: While running the KMP algorithm, we are shifting the pattern only when the comparison fails. Let's say that $m_0, m_1, \cdots, m_k$ be the sequence of

Question 8 continues on the next page...

comparison indices when it fails. Then for all $0 < i <= k$, when shifting at the m_i's comparison, the amount of shift is determine by the index of the pattern $j = m_i - m_{i-1}$. The shift amount $P[j-1]$ is always smaller than $\frac{j}{2}$. Combining these, we find that the shift amount at m_i is $S_i = P[j-1] <= \frac{j}{2} = \frac{m_i - m_{i-1}}{2}$. As the number of total shifts $\Sigma S_i = \frac{m_1 - m_0}{2} + \cdots + \frac{m_k - m_{k-1}}{2} = \frac{m_k}{2}$. From this we have $m = 2n$ where $n = \Sigma S_i$ is the length of the target string and the m is the number of total comparisons.

9. **Sorting**

(a) [5 points] Fill out the blanks in the following function implementing the Quick Sort. Write your answers below.

```
void qsort(int a[], int left, int right) {
    if(left >= right) return;
    int pivot = a[left];
    int i = left, j = right;
    int temp;
    while(i < j) {
        while (a[i] <= pivot) i += 1;
        while (a[j] > pivot) j -= 1;
        if (___(1)___) {
            temp = a[i];
            a[i] = a[j];
            a[j] = temp;
            i+=1;
            j-=1;
        }
    }
    ___(2)___;
    ___(3)___;
    qsort(a,left,___(4)___);
    qsort(a,___(5)___,right);
}
```

(1) `i < j`
(2) `a[left] = a[j]`
(3) `a[j] = pivot`
(4) `j-1`
(5) `j+1`

(b) [5 points] State the worst case time complexity of Quick sort, and show an example of the worst case using an input integer array whose length is longer than 5 and the sequence of pivots we will be choosing.

Solution: $O(n^2)$
Any example that results in partitioning into an empty and $n-1$ array at every step works. For example:
$A = 6, 5, 4, 3, 2, 1$ and always pick the leftmost as the pivot.
Criteria

- Complexity: 2pt
- Example:3 pt (list: 1.5pt, pivot: 1.5pt)

The end of questions.