Answers

Question 1

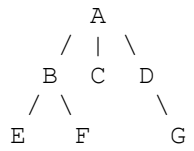
What data structure is primarily used in Breadth-First Traversal?

Answer: B. Queue

Because BFS processes nodes in **First In, First Out (FIFO)** order.

Question 2

Tree:



BFS Order

Answer:

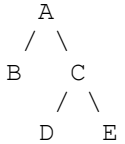
```
A B C D E F G
```

Explanation:

- Visit A
- Then B, C, D
- Then E, F, G

Question 3

Tree:



Queue Simulation

Start

Queue: `[A]`

Iteration 1

Poll A

Add B, C

Queue: `[B, C]`

Iteration 2

Poll B

No children

Queue: `[C]`

Iteration 3

Poll C

Add D, E

Queue:`[D, E]`

Iteration 4

Poll D

Queue:`[E]`

Iteration 5

Poll E

Queue: `[]`

Question 4

Completed Code

```
public void printAll(Person root){

    if (root == null){
        return;
    }

    Queue<Person> todo = new ArrayDeque<Person>();

    todo.offer(root);

    while (!todo.isEmpty()){

        Person p = todo.poll();

        UI.println(p.getName());

        for (Person child : p.getChildren()){
            todo.offer(child);
        }
    }
}
```

Question 5

Explanation

Breadth-First Traversal visits nodes **level by level**.

A queue is used to remember which nodes must be visited next.

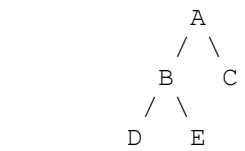
- `offer()` adds nodes to the back of the queue
- `poll()` removes nodes from the front

This FIFO behavior ensures earlier-discovered nodes are processed first.

That is why BFS visits all nodes on one level before moving deeper.

Question 6

Tree:



Correct BFS Order

Answer: B

```
A B C D E
```

Question 7

Errors

Original Incorrect Code

```
Queue<Person> todo = new ArrayDeque<Person>();

while (!todo.isEmpty()) {
    Person p = todo.pop();

    for (Person child : p.getChildren()) {
        todo.push(child);
    }
}
```

Problems

Error 1

`todo.pop(). // pop()` is for stacks.

Should be:

`todo.poll()`

Error 2

`todo.push(child) // push()` is stack behavior.

Should be:

`todo.offer(child)`

Corrected

```
Queue<Person> todo = new ArrayDeque<Person>();

while (!todo.isEmpty()) {
    Person p = todo.poll();

    for (Person child : p.getChildren()) {
        todo.offer(child);
    }
}
```

Question 8

Time Complexity

Answer: `O(n)`:

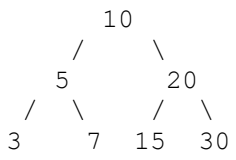
Each node is:

- added once to the queue
- removed once from the queue

So the total work is proportional to the number of nodes.

Question 9

Tree:



BFS Traversal

Start

Queue: `[10]`

Visit 10

Add 5, 20

Queue: `[5, 20]`

Visit 5

Add 3, 7

Queue: `[20, 3, 7]`

Visit 20

Add 15, 30

Queue: `[3, 7, 15, 30]`

Visit 3

Queue: `[7, 15, 30]`

Visit 7

Queue: `[15, 30]`

Visit 15

Queue: `[30]`

Visit 30

Queue: `[]`

Final BFS Order

```
10 5 20 3 7 15 30
```

Question 10

BFS Method

```
public void printAll(Person root){

    if (root == null){
        return;
    }

    Queue<Person> todo = new ArrayDeque<Person>();
```

```
    todo.offer(root);

    while (!todo.isEmpty()){

        Person p = todo.poll();

        UI.println(p.getName());

        for (Person child : p.getChildren()){
            todo.offer(child);
        }
    }
}
```

Question 11

contains Method

```
public boolean contains(Person root, String target){

    if (root == null){
        return false;
    }

    Queue<Person> todo = new ArrayDeque<Person>();

    todo.offer(root);

    while (!todo.isEmpty()){

        Person p = todo.poll();

        if (p.getName().equals(target)){
            return true;
        }

        for (Person child : p.getChildren()){
            todo.offer(child);
        }
    }

    return false;
}
```

Question 12

Count Nodes

```
public int countNodes(Person root){

    if (root == null){
        return 0;
    }
```

```
    int count = 0;

    Queue<Person> todo = new ArrayDeque<Person>();

    todo.offer(root);

    while (!todo.isEmpty()){

        Person p = todo.poll();

        count++;

        for (Person child : p.getChildren()){
            todo.offer(child);
        }
    }

    return count;
}
```

Question 13

Count Levels

```
public int countLevels(Person root){

    if (root == null){
        return 0;
    }

    Queue<Person> todo = new ArrayDeque<Person>();

    todo.offer(root);

    int levels = 0;

    while (!todo.isEmpty()){

        int size = todo.size();

        for (int i = 0; i < size; i++){

            Person p = todo.poll();

            for (Person child : p.getChildren()){
                todo.offer(child);
            }
        }

        levels++;
    }

    return levels;
}
```

Question 14

Why Queue is Necessary

A queue guarantees nodes are processed in the same order they are discovered.

This creates level-by-level traversal.

If a stack were used instead:

- the newest node would be processed first
- traversal would go deep before wide

That becomes Depth-First Search (DFS).

Question 15

Feature	BFS	DFS
Main Data Structure	Queue	Stack / Recursion
Traversal Style	Level by level	Go deep first
Finds shortest path in unweighted graph?	Yes	Not guaranteed
Uses recursion commonly?	Rarely	Often