Code Snippets:

```
public class Test {
   public static void main(String[] args) {
      int arr[] = {5, 1, 4, 2, 8};
      for(int i=0;i<arr.length-1;i++){
         for(int j=0;j<arr.length-i-1;j++){
            if(arr[j] > arr[j+1]){
               int temp = arr[j];
               arr[j] = arr[j+1];
               arr[j+1] = temp;
            }
         }
      }
      for(int n: arr) System.out.print(n+" ");
   }
}
```

Options:

A) 8 5 4 2 1

B) 1 2 4 5 8

C) 5 1 2 4 8

D) 2 1 4 5 8

Q2. Insertion Sort Pass

java

Copy code

```
int arr[] = {9, 5, 1, 4, 3};
for(int i=1; i<arr.length; i++){
   int key = arr[i];
   int j=i-1;
   while(j>=0 && arr[j]>key){
```

```
      arr[j+1]=arr[j];
      j--;
   }
   arr[j+1]=key;
}
System.out.print(arr[2]);
```

Options:

A) 1

B) 3

C) 4

D) 5

Q3. Quick Sort Partition Index

```
java
Copy code
int arr[] = {10, 7, 8, 9, 1, 5};
int pivot = arr[arr.length-1];
int i=-1;
for(int j=0;j<arr.length-1;j++){
   if(arr[j] < pivot){
      i++;
      int temp=arr[i]; arr[i]=arr[j]; arr[j]=temp;
   }
}
System.out.println(i+1);
```

Options:

A) 2

B) 3

C) 4

D) 5

Q4. Merge Sort First Split

java

Copy code

```
int arr[] = {38, 27, 43, 3, 9, 82, 10};
int mid = (0+arr.length-1)/2;
System.out.println(mid);
```

Options:

A) 3

B) 2

C) 4

D) 5

🔹 BST Snippets

Q5. BST Inorder Traversal

java

Copy code

```
class Node {
  int key; Node left, right;
  Node(int k){ key=k; }
}
class Test {
  static void inorder(Node root){
    if(root!=null){
      inorder(root.left);
      System.out.print(root.key+" ");
      inorder(root.right);
    }
  }
  public static void main(String[] args){
```

```
    Node root=new Node(50);
    root.left=new Node(30);
    root.right=new Node(70);
    root.left.left=new Node(20);
    root.left.right=new Node(40);
    inorder(root);
  }
}
```

Options:

A) 20 30 40 50 70

B) 50 30 20 40 70

C) 50 70 30 20 40

D) 30 20 40 50 70

Q6. BST Search

java

Copy code

```
Node root = new Node(15);
root.left = new Node(10);
root.right = new Node(20);
System.out.println(search(root, 25));

boolean search(Node root, int key){
  if(root==null) return false;
  if(root.key==key) return true;
  if(key<root.key) return search(root.left,key);
  return search(root.right,key);
}
```

Options:

A) true

B) false

C) compile error

D) runtime error

Q7. BST Delete Node (Leaf)

java

Copy code

```
// Delete node 20
Node root = new Node(30);
root.left = new Node(20);
root.right = new Node(40);
root = delete(root,20);
inorder(root);
```

Options:

A) 20 30 40

B) 30 40

C) 20 40

D) 30

Q8. Height of BST

java

Copy code

```
int height(Node root){
   if(root==null) return 0;
   return 1+Math.max(height(root.left), height(root.right));
}
```

If BST is:

markdown

Copy code

```
    10
   / \
  8   15
 /
5
```

Options:

A) 2

B) 3

C) 4

D) 1

Q9. Selection Sort First Pass

```
int arr[] = {29, 10, 14, 37, 13};
for(int i=0; i<arr.length-1; i++){
   int min=i;
   for(int j=i+1;j<arr.length;j++){
      if(arr[j]<arr[min]) min=j;
   }
   int temp=arr[min]; arr[min]=arr[i]; arr[i]=temp;
}
System.out.println(arr[0]);
```

Options:

A) 29

B) 10

C) 13

D) 14

Q10. Time Complexity Bubble Sort (Worst Case)

```
for(int i=0; i<n-1; i++){
   for(int j=0; j<n-i-1; j++){
      if(arr[j]>arr[j+1]){
```

```
        // swap
      }
    }
}
```

Options:
A) O(n)
B) O(n log n)
C) $O(n^2)$
D) O(log n)

Q11. Quick Sort Best Case Complexity

Options:
A) $O(n^2)$
B) O(n log n)
C) O(n)
D) O(log n)

Q12. Merge Sort Recurrence

For Merge Sort, recurrence relation is:
Options:
A) T(n) = T(n-1) + O(1)
B) T(n) = 2T(n/2) + O(n)
C) T(n) = T(n/2) + O(1)
D) $T(n) = T(n) + O(n^2)$

Q13. Stability of Sorting

Which of the following sorting algorithms is **stable**?

Options:
A) Selection Sort
B) Quick Sort
C) Merge Sort
D) Heap Sort

Q14. Sorting Already Sorted Array with Insertion Sort

```
int arr[]={1,2,3,4,5};
```

insertionSort(arr);

System.out.println("Time Complexity?");

Options:
A) O(n)
B) O(n log n)
C) $O(n^2)$
D) O(log n)

- **Binary Search Tree (BST) MCQ**

Q15. BST Property

In a BST:
Options:
A) Left child > parent, Right child < parent
B) Left child < parent, Right child > parent
C) Both children < parent
D) Both children > parent

Q16. Search Complexity in Balanced BST

Options:
A) $O(n^2)$
B) O(log n)
C) O(n)
D) O(1)

Q17. BST Traversals

Which traversal of BST gives sorted order of elements?

Options:
A) Preorder
B) Postorder
C) Inorder
D) Level Order

Q18. Minimum Value in BST

Node minValue(Node root){

```
  while(root.left!=null){
    root=root.left;
  }
  return root;
}
```

If tree is:

```
    50
   /  \
  30   70
 / \
20 40
```

Options:
A) 20
B) 30
C) 40
D) 50

Q19. Height of Empty BST

```
int height(Node root){
  if(root==null) return 0;
  return 1+Math.max(height(root.left), height(root.right));
}
```

Options:
A) -1
B) 0
C) 1
D) Undefined

Q20. Deletion of Node with Two Children

When deleting a node with two children in BST:

Options:
A) Replace with left child
B) Replace with right child
C) Replace with inorder predecessor or inorder successor
D) Node cannot be deleted

Answers:

1. Answer: B) 1 2 4 5 8
2. Answer: C) 4
3. Answer: A) 2
   (Partition index returned is i+1)
4. Answer: A) 3
   (Mid of 0 and 6 = 3)
5. Answer: A) 20 30 40 50 70
6. Answer: B) false
7. Answer: B) 30 40
8. Answer: B) 3
9. Answer: B) 10
10. Answer: C) $O(n^2)$
11. Answer: B) O(n log n)
12. Answer: B) T(n) = 2T(n/2) + O(n)
13. Answer: C) Merge Sort
14. Answer: A) O(n)
15. Answer: B) Left child < parent, Right child > parent
16. Answer: B) O(log n)
17. Answer: C) Inorder
18. Answer: A) 20
19. Answer: B) 0
20. Answer: C) Replace with inorder predecessor or inorder successor