**1. Selection Sort**

**Question 1:** What is the time complexity of Selection Sort in the worst case?

**public** **class** SelectionSortExample {

**public** **static** **void** selectionSort(**int**[] arr) {

**for** (**int** i = 0; i < arr.length - 1; i++) {

**int** minIdx = i;

**for** (**int** j = i + 1; j < arr.length; j++) {

**if** (arr[j] < arr[minIdx]) {

minIdx = j;

}

}

**int** temp = arr[minIdx];

arr[minIdx] = arr[i];

arr[i] = temp;

}

}

}

a) O(n)  
b) O(n log n)  
c) O(n²)  
d) O(log n)

**Answer:** c) O(n²)

**2. Bubble Sort**

**Question 2:** What does the following Bubble Sort code do if it is modified with a flag to stop early if no swaps are made in a pass?

**public** **class** BubbleSortExample {

**public** **static** **void** bubbleSort(**int**[] arr) {

**boolean** swapped;

**for** (**int** i = 0; i < arr.length - 1; i++) {

swapped = **false**;

**for** (**int** j = 0; j < arr.length - 1 - i; j++) {

**if** (arr[j] > arr[j + 1]) {

**int** temp = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = temp;

swapped = **true**;

}

}

**if** (!swapped) **break**;

}

}

a) Reduces time complexity to O(n)  
b) Guarantees a worst-case time complexity of O(n²)  
c) Improves space complexity to O(1)  
d) Converts Bubble Sort to a stable sort

**Answer:** b) Guarantees a worst-case time complexity of O(n²)

**3. Insertion Sort**

**Question 3:** How does the following Insertion Sort code handle the sorting?

**public** **class** InsertionSortExample {

**public** **static** **void** insertionSort(**int**[] arr) {

**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;

}

}

}

a) Time complexity is O(n log n) in all cases  
b) It uses additional memory proportional to the size of the array  
c) It is efficient for small or nearly sorted arrays  
d) It is not a stable sorting algorithm

**Answer:** c) It is efficient for small or nearly sorted arrays

**4. Merge Sort**

**Question 4:** What is the time complexity of Merge Sort?

**public** **class** MergeSortExample {

**public** **static** **void** mergeSort(**int**[] arr) {

**if** (arr.length < 2) **return**;

**int** mid = arr.length / 2;

**int**[] left = **new** **int**[mid];

**int**[] right = **new** **int**[arr.length - mid];

System.*arraycopy*(arr, 0, left, 0, mid);

System.*arraycopy*(arr, mid, right, 0, arr.length - mid);

*mergeSort*(left);

*mergeSort*(right);

*merge*(arr, left, right);

}

**private** **static** **void** merge(**int**[] arr, **int**[] left, **int**[] right) {

**int** i = 0, j = 0, k = 0;

**while** (i < left.length && j < right.length) {

**if** (left[i] <= right[j]) arr[k++] = left[i++];

**else** arr[k++] = right[j++];

}

**while** (i < left.length) arr[k++] = left[i++];

**while** (j < right.length) arr[k++] = right[j++];

}

}

a) O(n)  
b) O(n log n)  
c) O(n²)  
d) O(log n)

**Answer:** b) O(n log n)

**5. Quick Sort**

**Question 5:** What is the worst-case time complexity of Quick Sort?

**public** **class** QuickSortExample {

**public** **static** **void** quickSort(**int**[] arr, **int** low, **int** high) {

**if** (low < high) {

**int** pi = *partition*(arr, low, high);

*quickSort*(arr, low, pi - 1);

*quickSort*(arr, pi + 1, high);

}

}

**private** **static** **int** partition(**int**[] arr, **int** low, **int** high) {

**int** pivot = arr[high];

**int** i = low - 1;

**for** (**int** j = low; j < high; j++) {

**if** (arr[j] <= pivot) {

i++;

**int** temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

}

**int** temp = arr[i + 1];

arr[i + 1] = arr[high];

arr[high] = temp;

**return** i + 1;

}

}

a) O(n)  
b) O(n log n)  
c) O(n²)  
d) O(log n)

**Answer:** c) O(n²)

**6. Linear Search**

**Question 6:** What is the average-case time complexity of Linear Search?

**public** **class** LinearSearchExample {

**public** **static** **int** linearSearch(**int**[] arr, **int** target) {

**for** (**int** i = 0; i < arr.length; i++) {

**if** (arr[i] == target) **return** i;

}

**return** -1;

}

}

a) O(1)  
b) O(log n)  
c) O(n)  
d) O(n²)

**Answer:** c) O(n)

**7. Binary Search**

**Question 7:** What is the precondition for Binary Search to work correctly?

**public** **class** BinarySearchExample {

**public** **static** **int** binarySearch(**int**[] arr, **int** target) {

**int** left = 0, right = arr.length - 1;

**while** (left <= right) {

**int** mid = left + (right - left) / 2;

**if** (arr[mid] == target) **return** mid;

**if** (arr[mid] < target) left = mid + 1;

**else** right = mid - 1;

}

**return** -1;

}

}

a) The array must be unsorted  
b) The array must be sorted  
c) The array can be sorted or unsorted  
d) The array must be of fixed size

**Answer:** b) The array must be sorted

**8. Selection Sort**

**Question 8:** In Selection Sort, how many times is the arr[i] element compared with other elements?

**public** **class** SelectionSortExample {

**public** **static** **void** selectionSort(**int**[] arr) {

**for** (**int** i = 0; i < arr.length - 1; i++) {

**int** minIdx = i;

**for** (**int** j = i + 1; j < arr.length; j++) {

**if** (arr[j] < arr[minIdx]) {

minIdx = j;

}

}

**int** temp = arr[minIdx];

arr[minIdx] = arr[i];

arr[i] = temp;

}

}

}

a) It is compared with every other element in each iteration  
b) It is compared with half of the elements  
c) It is compared only once  
d) It is not compared with any other elements

**Answer:** a) It is compared with every other element in each iteration

**9. Bubble Sort**

**Question 9:** What effect does setting the swapped flag in Bubble Sort have on performance?

**public** **class** BubbleSortExample {

**public** **static** **void** bubbleSort(**int**[] arr) {

**boolean** swapped;

**for** (**int** i = 0; i < arr.length - 1; i++) {

swapped = **false**;

**for** (**int** j = 0; j < arr.length - 1 - i; j++) {

**if** (arr[j] > arr[j + 1]) {

**int** temp = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = temp;

swapped = **true**;

}

}

**if** (!swapped) **break**;

}

}

}

a) It improves worst-case time complexity  
b) It eliminates the need for nested loops  
c) It reduces the number of passes if no swaps occur  
d) It guarantees O(n) time complexity in all cases

**Answer:** c) It reduces the number of passes if no swaps occur

**10. Merge Sort**

**Question 10:** What is the space complexity of Merge Sort?

**public** **class** MergeSortExample {

**public** **static** **void** mergeSort(**int**[] arr) {

**if** (arr.length < 2) **return**;

**int** mid = arr.length / 2;

**int**[] left = **new** **int**[mid];

**int**[] right = **new** **int**[arr.length - mid];

System.*arraycopy*(arr, 0, left, 0, mid);

System.*arraycopy*(arr, mid, right, 0, arr.length - mid);

*mergeSort*(left);

*mergeSort*(right);

*merge*(arr, left, right);

}

**private** **static** **void** merge(**int**[] arr, **int**[] left, **int**[] right) {

**int** i = 0, j = 0, k = 0;

**while** (i < left.length && j < right.length) {

**if** (left[i] <= right[j]) arr[k++] = left[i++];

**else** arr[k++] = right[j++];

}

**while** (i < left.length) arr[k++] = left[i++];

**while** (j < right.length) arr[k++] = right[j++];

}

}

a) O(1)  
b) O(n)  
c) O(n log n)  
d) O(n²)

**Answer:** b) O(n)