1. Which of the following sorting algorithms has the worst time complexity in the average and worst case?

a) Bubble Sort

**b) Insertion Sort**

c) Selection Sort

d) Merge Sort

2. #include <iostream>

using namespace std;

void bubbleSort(int arr[], int n) {

for (int i = 0; i < n - 1; i++) {

for (int j = 0; j < n - i - 1; j++) {

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

int temp = arr[j];

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

arr[j + 1] = temp;

}

}

}

}

int main() {

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

int n = sizeof(arr) / sizeof(arr[0]);

bubbleSort(arr, n);

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

cout << arr[i] << " ";

}

return 0;

}

What will be the output when the above code is executed?

a) 3 4 5 1 1

**b) 1 1 3 4 5**

c) 5 4 3 1 1

d) 1 3 1 4 5

3. The time complexity of the bubble sort algorithm for sorting an array of n elements is:

a) O(1)

b) O(log n)

**c) O(n^2)**

d) O(n log n)

4. #include <iostream>

using namespace std;

void insertionSort(int arr[], int n) {

for (int i = 1; i < n; 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;

}

}

int main() {

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

int n = sizeof(arr) / sizeof(arr[0]);

insertionSort(arr, n);

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

cout << arr[i] << " ";

}

return 0;

}

What will be the output when the above code is executed?

**a) 1 2 3 4 5**

b) 5 4 3 2 1

c) 2 3 4 5 1

d) 1 5 4 3 2

5. The insertion sort algorithm works by:

a) Repeatedly dividing the array into subarrays and then merging them.

**b) Inserting elements from unsorted part into the sorted part of the array.**

c) Swapping elements to arrange them in ascending order.

d) Selecting the minimum element and moving it to the front of the array.

6. #include <iostream>

using namespace std;

void selectionSort(int arr[], int n) {

for (int i = 0; i < n - 1; i++) {

int minIndex = i;

for (int j = i + 1; j < n; j++) {

if (arr[j] < arr[minIndex]) {

minIndex = j;

}

}

int temp = arr[i];

arr[i] = arr[minIndex];

arr[minIndex] = temp;

}

}

int main() {

int arr[] = {4, 7, 2, 9, 1};

int n = sizeof(arr) / sizeof(arr[0]);

selectionSort(arr, n);

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

cout << arr[i] << " ";

}

return 0;

}

What will be the output when the above code is executed?

**a) 1 2 4 7 9**

b) 9 7 4 2 1

c) 4 7 2 9 1

d) 1 4 7 2 9

7. In the selection sort algorithm, how many passes are required to sort an array of n elements in the worst-case scenario?

a) 1

b) n

**c) n - 1**

d) log(n)

8. Select the appropriate code that performs bubble sort.

**a)**

**for(int j=arr.length-1; j>=0; j--)**

**{**

**for(int k=0; k<j; k++)**

**{**

**if(arr[k] > arr[k+1])**

**{**

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

**arr[k] = arr[k+1];**

**arr[k+1] = temp;**

**}**

**}**

**}**

b)

for(int j=arr.length-1; j>=0; j--)

{

for(int k=0; k<j; k++)

{

if(arr[k] < arr[k+1])

{

int temp = arr[k];

arr[k] = arr[k+1];

arr[k+1] = temp;

}

}

}

c)

for(int j=arr.length; j>=0; j--)

{

for(int k=0; k<j; k++)

{

if(arr[k] > arr[k+1])

{

int temp = arr[k];

arr[k] = arr[k+1];

arr[k+1] = temp;

}

}

}

d)

for(int j=arr.length; j>=0; j--)

{

for(int k=0; k<j; k++)

{

if(arr[k] > arr[k+2])

{

int temp = arr[k];

arr[k] = arr[k+1];

arr[k+1] = temp;

}

}

}

9. The time complexity of the selection sort algorithm for sorting an array of n elements is:

a) O(1)

b) O(log n)

c) O(n)

**d) O(n^2)**

10. Which sorting algorithm is known for its efficiency and is often used for large datasets due to its average-case time complexity of O(n log n)?

a) Bubble Sort

b) Insertion Sort

**c) Merge Sort**

d) Selection Sort

11. How can you improve the best case efficiency in bubble sort? (The input is already sorted)

a)

boolean swapped = false;

for(int j=arr.length-1; j>=0 && swapped; j--)

{

swapped = true;

for(int k=0; k<j; k++)

{

if(arr[k] > arr[k+1])

{

int temp = arr[k];

arr[k] = arr[k+1];

arr[k+1] = temp;

swapped = false;

}

}

}

b)

boolean swapped = true;

for(int j=arr.length-1; j>=0 && swapped; j--)

{

swapped = false;

for(int k=0; k<j; k++)

{

if(arr[k] > arr[k+1])

{

int temp = arr[k];

arr[k] = arr[k+1];

arr[k+1] = temp;

}

}

}

**c)**

**boolean swapped = true;**

**for(int j=arr.length-1; j>=0 && swapped; j--)**

**{**

**swapped = false;**

**for(int k=0; k<j; k++)**

**{**

**if(arr[k] > arr[k+1])**

**{**

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

**arr[k] = arr[k+1];**

**arr[k+1] = temp;**

**swapped = true;**

**}**

**}**

**}**

d)

boolean swapped = true;

for(int j=arr.length-1; j>=0 && swapped; j--)

{

for(int k=0; k<j; k++)

{

if(arr[k] > arr[k+1])

{

int temp = arr[k];

arr[k] = arr[k+1];

arr[k+1] = temp;

swapped = true;

}

}

}

.

12. The time complexity of the merge sort algorithm for sorting an array of n elements is:

a) O(1)

b) O(log n)

**c) O(n log n)**

d) O(n^2)

13. Consider the code given below, which runs insertion sort:

void insertionSort(int arr[], int array\_size)

{

int i, j, value;

for (i = 1; i < array\_size; i++)

{

value = arr[i];

j = i;

while (\_\_\_\_\_\_\_\_ )

{

arr[j] = arr[j − 1];

j = j − 1;

}

arr[j] = value;

}

}

Which condition will correctly implement the while loop?

a) (j > 0) || (arr[j − 1] > value)

**b) (j > 0) && (arr[j − 1] > value)**

c) (j > 0) && (arr[j + 1] > value)

d) (j > 0) && (arr[j + 1] < value)

14. In the insertion sort algorithm, elements are moved within the sorted part of the array until they reach their correct position.

a) True

**b) False**

15. Which condition will correctly implement the if condition if we want array to be sorted in descendng order?

boolean swapped = true;

for(int j=arr.length-1; j>=0 && swapped; j--)

{

swapped = false;

for(int k=0; k<j; k++)

{

if(\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)

{

int temp = arr[k];

arr[k] = arr[k+1];

arr[k+1] = temp;

swapped = true;

}

}

}

a) arr[k] > arr[k+1]

**b) arr[k] < arr[k+1] \*\*\*\***

c) arr[k] = arr[k+1]

d) arr[k] != arr[k+1]

16. The selection sort algorithm works by:

**a) Finding the minimum (or maximum) element and moving it to the beginning (or end) of the sorted part of the array.**

b) Repeatedly swapping adjacent elements if they are in the wrong order.

c) Inserting elements from the unsorted part into the sorted part of the array.

d) Dividing the array into two halves, sorting each half, and then merging them.

17. Which condition will correctly implement the if condition if we want array to be sorted in descendng order?

boolean swapped = true;

for(int j=arr.length-1; j>=0 && swapped; j--)

{

swapped = false;

for(int k=0; k<j; k++)

{

if(\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)

{

int temp = arr[k];

arr[k] = arr[k+1];

arr[k+1] = temp;

swapped = true;

}

}

}

**a) arr[k] > arr[k+1]**

b) arr[k] < arr[k+1]

c) arr[k] = arr[k+1]

d) arr[k] != arr[k+1]

18. The time complexity of the heap sort algorithm for sorting an array of n elements is:

a) O(1)

**b) O(n log n)**

c) O(n^2)

d) O(log n)

19. What will be the statment at the balnk space in given code?

void insertionSort(int arr[], int n) {

for (int i = 1; i < n; i++) {

int key = arr[i];

int j = i - 1;

while (j >= 0 && arr[j] > key) {

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

j--;

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

}

}

a) j++;

b) arr[j - 1] = key;

**c) arr[j + 1] = key;**

d) arr[j] = key;

20. The main advantage of merge sort over other sorting algorithms is its:

a) Simplicity of implementation.

**b) Efficiency with large datasets and consistent time complexity.**

c) Stability in maintaining the order of equal elements.

d) Ability to sort the array in-place without using additional memory.