“Merge Sort”.

1. Merge sort uses which of the following technique to implement sorting?
a) backtracking
b) greedy algorithm
c) divide and conquer
d) dynamic programming
View Answer
Answer: c
Explanation: Merge sort uses divide and conquer in order to sort a given array. This is because it divides the array into two halves and applies merge sort algorithm to each half individually after which the two sorted halves are merged together.

2. What is the average case time complexity of merge sort?
a) O(n log n)
b) $O(n^2)$
c) $O(n^2 \log n)$
d) $O(n \log n^2)$
View Answer
Answer: a
Explanation: The recurrence relation for merge sort is given by T(n) = 2T(n/2) + n. It is found to be equal to O(n log n) using the master theorem.

3. What is the auxiliary space complexity of merge sort?
a) O(1)
b) O(log n)
c) O(n)
d) O(n log n)
View Answer
Answer: c
Explanation: An additional space of O(n) is required in order to merge two sorted arrays. Thus merge sort is not an in place sorting algorithm.

4. Merge sort can be implemented using O(1) auxiliary space.
a) true
b) false
View Answer

Answer: a
Explanation: Standard merge sort requires O(n) space to merge two sorted arrays. We can optimize this merging process so that it takes only constant space. This version is known as in place merge sort.

5. What is the worst case time complexity of merge sort?
a) O(n log n)
b) $O(n^2)$
c) $O(n^2 \log n)$
d) $O(n \log n^2)$
View Answer
Answer: a
Explanation: The time complexity of merge sort is not affected by worst case as its algorithm has to implement the same number of steps in any case. So its time complexity remains to be O(n log n).

6. Which of the following method is used for sorting in merge sort?
a) merging
b) partitioning
c) selection
d) exchanging
View Answer
Answer: a
Explanation: Merge sort algorithm divides the array into two halves and applies merge sort algorithm to each half individually after which the two sorted halves are merged together. Thus its method of sorting is called merging.

7. What will be the best case time complexity of merge sort?
a) O(n log n)
b) $O(n^2)$
c) $O(n^2 \log n)$
d) $O(n \log n^2)$
View Answer
Answer: a
Explanation: The time complexity of merge sort is not affected in any case as its algorithm has to implement the same number of steps. So its time complexity remains to be O(n log n) even in the best case.

8. Which of the following is not a variant of merge sort?
a) in-place merge sort
b) bottom up merge sort

c) top down merge sort
d) linear merge sort
View Answer
Answer: d
Explanation: In-place, top down and bottom up merge sort are different variants of merge sort. Whereas linear merge sort is not a possible variant as it is a comparison based sort and the minimum time complexity of any comparison based sort is O(n log n).

9. Choose the incorrect statement about merge sort from the following?
a) it is a comparison based sort
b) it is an adaptive algorithm
c) it is not an in place algorithm
d) it is stable algorithm
View Answer
Answer: b
Explanation: Merge sort is not an adaptive sorting algorithm. This is because it takes O(n log n) time complexity irrespective of any case.

10. Which of the following is not in place sorting algorithm by default?
a) merge sort
b) quick sort
c) heap sort
d) insertion sort
View Answer
Answer: a
Explanation: Quick sort, heap sort, and insertion sort are in-place sorting algorithms, whereas an additional space of O(n) is required in order to merge two sorted arrays. Even though we have a variation of merge sort (to do in-place sorting), it is not the default option. So, among the given choices, merge sort is the most appropriate answer.

11. Which of the following is not a stable sorting algorithm?
a) Quick sort
b) Cocktail sort
c) Bubble sort
d) Merge sort
View Answer
Answer: a
Explanation: Out of the given options quick sort is the only algorithm which is not stable. Merge sort is a stable sorting algorithm.

12. Which of the following stable sorting algorithm takes the least time when applied to an almost sorted array?
a) Quick sort
b) Insertion sort
c) Selection sort
d) Merge sort
View Answer
Answer: d
Explanation: Insertion sort takes linear time to sort a partially sorted array. Though merge and quick sort takes O(n*logn) complexity to sort, merge sort is stable. Hence, Merge sort takes less time to sort partially sorted array.

13. Merge sort is preferred for arrays over linked lists.
a) true
b) false
View Answer
Answer: b
Explanation: Merge sort is preferred for linked list over arrays. It is because in a linked list the insert operation takes only O(1) time and space which implies that we can implement merge operation in constant time.

14. Which of the following sorting algorithm makes use of merge sort?
a) tim sort
b) intro sort
c) bogo sort
d) quick sort
View Answer
Answer: a
Explanation: Tim sort is a hybrid sorting algorithm as it uses more than one sorting algorithm internally. It makes use of merge sort and insertion sort.

15. Choose the correct code for merge sort.
a)

```
void merge_sort(int arr[], int left, int right)
{
    if (left > right)
    {

        int mid = (right-left)/2;
        merge_sort(arr, left, mid);
        merge_sort(arr, mid+1, right);
```

```
        merge(arr, left, mid, right); //function to merge sorted arrays
    }
}
```

b)

```
void merge_sort(int arr[], int left, int right)
{
    if (left < right)
    {

        int mid = left+(right-left)/2;
        merge_sort(arr, left, mid);
        merge_sort(arr, mid+1, right);

        merge(arr, left, mid, right); //function to merge sorted arrays
    }
}
```

c)

```
void merge_sort(int arr[], int left, int right)
{
    if (left < right)
    {

        int mid = left+(right-left)/2;
merge(arr, left, mid, right); //function to merge sorted arrays
        merge_sort(arr, left, mid);
        merge_sort(arr, mid+1, right);


    }
}
```

d)

```
void merge_sort(int arr[], int left, int right)
{
    if (left < right)
    {
```

```
        int mid = (right-left)/2;
merge(arr, left, mid, right); //function to merge sorted arrays
        merge_sort(arr, left, mid);
        merge_sort(arr, mid+1, right);


    }
}
```

View Answer

Answer: b

Explanation: Merge sort first sorts the two halves of the array individually. Then it merges the two sorted halves in order to obtain sorted array.

16. Which of the following sorting algorithm does not use recursion?

a) quick sort

b) merge sort

c) heap sort

d) bottom up merge sort

View Answer

Answer: d

Explanation: Bottom up merge sort uses the iterative method in order to implement sorting. It begins by merging a pair of adjacent array of size 1 each and then merge arrays of size 2 each in the next step and so on.

“In-place Merge Sort”.

1. Merge sort uses which of the following algorithm to implement sorting?

a) backtracking

b) greedy algorithm

c) divide and conquer

d) dynamic programming

View Answer

Answer: c

Explanation: Merge sort uses the technique of divide and conquer in order to sort a given array. It divides the array into two halves and apply merge sort algorithm to each half individually after which the sorted versions of these halves are merged together.

2. What is the average case time complexity of standard merge sort?
a) O(n log n)
b) $O(n^2)$
c) $O(n^2 \log n)$
d) $O(n \log n^2)$
View Answer
Answer: a
Explanation: The recurrence relation for merge sort is given by T(n) = 2T(n/2) + n. This can be solved using master's theorem and is found equal to O(n log n).

3. What is the auxiliary space complexity of standard merge sort?
a) O(1)
b) O(log n)
c) O(n)
d) O(n log n)
View Answer
Answer: c
Explanation: The merging of two sorted arrays requires an additional space of n due to which the auxiliary space requirement of merge sort is O(n). Thus merge sort is not an in place sorting algorithm.

4. What is the space complexity of in place merge sort?
a) O(1)
b) O(n)
c) O(log n)
d) O(n log n)
View Answer
Answer: c
Explanation: Space complexity of in place version of merge sort is O(log n) which is used for storing call stack formed due to recursion. Note that the algorithms with space complexity as O(log n) also qualifies as in place algorithms as the value of log n is close to 1.

5. What is the average time complexity of in place merge sort when we use the following function for merging?


```
void in_place_merge(int arr[], int l, int middle, int r)
```

```
{
         int start2 = middle + 1;
         if (arr[middle] <= arr[start2])
        {
                  return;
         }
         while (l <= middle && start2 <= r)
        {
                  if (arr[l] <= arr[start2])
               {
                           l++;
                  }
                  else
               {
                           int val = arr[start2];
                           int index = start2;
                           while (index != l)
                     {
                                    arr[index] = arr[index - 1];
                                    index--;
                           }
                           arr[l] = val;
                          l++;
                           middle++;
                           start2++;
                  }
         }
}
```

a) O(n log n)
b) O(n^2)
c) O(n^2 log n)
d) O(n log n^2)
View Answer
Answer: b
Explanation: The merge function in the in place merge sort takes O(n^2) time so the recurrence relation becomes T(n)=2T(n/2)+n^2. This can be solved using master's theorem and is found equal to O(n^2).

6. Merge sort uses which of the following method to implement sorting?
a) merging
b) partitioning
c) selection

d) exchanging
View Answer
Answer: a
Explanation: Merge sort implements sorting by merging the sorted versions of smaller parts of the array. Thus its method of sorting is called merging.

7. In place merge sort has same time complexity as standard merge sort.
a) true
b) false
View Answer
Answer: b
Explanation: In place version of merge sort has a greater time complexity as compared to its standard version. It is because the merging in in-place merge sort takes place in $O(n^2)$ time whereas in standard version it takes O(n) time.

8. In-place merge sort is a stable sort.
a) true
b) false
View Answer
Answer: a
Explanation: In-place merge sort like standard merge sort is a stable sort. This implies that the relative position of equal valued elements in the input and sorted array remain same.

9. Choose the incorrect statement about merge sort from the following?
a) both standard merge sort and in-place merge sort are stable
b) standard merge sort has greater time complexity than in-place merge sort
c) standard merge sort has greater space complexity than in-place merge sort
d) in place merge sort has O(log n) space complexity
View Answer
Answer: b
Explanation: Time complexity of standard merge sort is O(n log n) and that of in-place merge sort is $O(n^2)$. So the time complexity of in-place merge sort is more than that of standard merge sort.

10. Choose the correct function from the following that implements merging in in-place merge sort.
a)

```
void in_place_merge(int arr[], int l, int middle, int r)
{
          int start2 = middle + 1;
```

```
         if (arr[middle] <= arr[start2])
        {
                  return;
         }
         while (l <= middle+1 && start2 <= r)
        {
                  if (arr[l] <= arr[start2])
                {
                          l++;
                  }
                  else
                {
                          int val = arr[start2];
                          int index = start2;
                          while (index != l)
                       {
                                  arr[index] = arr[index - 1];
                                  index--;
                          }
                          arr[l] = val;
                         l++;
                          middle++;
                          start2++;
                  }
         }
}
```

b)

```
void in_place_merge(int arr[], int l, int middle, int r)
{
         int start2 = middle + 1;
         if (arr[middle] <= arr[start2])
        {
                  return;
         }
         while (l <= middle && start2 <= r)
        {
                  if (arr[l] <= arr[start2])
                {
                          l++;
                  }
                  else
```

```
                {
                        int val = arr[start2];
                        int index = start2;
                        while (index != l)
                    {
                                arr[index] = arr[index - 1];
                                index--;
                        }
                        arr[l] = val;
                       l++;
                        middle++;
                        start2++;
                  }
          }
}
```

c)

```
void in_place_merge(int arr[], int l, int middle, int r)
{
          int start2 = middle + 1;
          if (arr[middle] <= arr[start2])
        {
                  return;
          }
          while (l <= middle+1 && start2 <= r)
        {
                  if (arr[l] >= arr[start2])
                {
                        l++;
                  }
                  else
                {
                        int val = arr[start2];
                        int index = start2;
                        while (index != l)
                    {
                                arr[index] = arr[index - 1];
                                index--;
                        }
                        arr[l] = val;
                    l++;
                        middle++;
```

```
                        start2++;
                }
        }
}
```

d)

```
void in_place_merge(int arr[], int l, int middle, int r)
{
        int start2 = middle + 1;
        if (arr[middle] >= arr[start2])
       {
                return;
        }
        while (l <= middle && start2 <= r)
       {
                if (arr[l] <= arr[start2])
              {
                        l++;
                }
                else
              {
                        int val = arr[start2];
                        int index = start2;
                        while (index != r)
                     {
                                arr[index] = arr[index - 1];
                                Index++;
                        }
                        arr[l] = val;
                       l++;
                        middle++;
                        start2++;
                }
        }
}
```

View Answer

Answer: b

Explanation: Merging in in-place merge sort takes place in the original array itself. We first compare the first elements of the two halves. If the first element of second half is greater then we just increment the pointer of the first element of first half. Otherwise we shift elements to the right.

“Bottom-Up Mergesort”.

1. Merge sort uses which of the following algorithm to implement sorting?
a) backtracking
b) greedy algorithm
c) divide and conquer
d) dynamic programming
View Answer
Answer: c
Explanation: Merge sort uses the technique of divide and conquer in order to sort a given array. It divides the array into two halves and applies merge sort algorithm to each half individually after which the sorted versions of these halves are merged together.

2. What is the average case time complexity of standard merge sort?
a) O(n log n)
b) $O(n^2)$
c) $O(n^2 \log n)$
d) $O(n \log n^2)$
View Answer
Answer: a
Explanation: The recurrence relation for merge sort is given by T(n) = 2T(n/2) + n. This can be solved using master’s theorem and is found equal to O(n log n).

3. What is the auxiliary space complexity of standard merge sort?
a) O(1)
b) O(log n)
c) O(n)
d) O(n log n)
View Answer
Answer: c
Explanation: The merging of two sorted arrays requires an additional space of n due to which the auxiliary space requirement of merge sort is O(n). Thus merge sort is not an in place sorting algorithm.

4. What is the auxiliary space complexity of bottom up merge sort?
a) O(1)
b) O(n)

c) O(log n)
d) O(n log n)
View Answer
Answer: b
Explanation: The auxiliary space complexity of bottom up merge sort is same as standard merge sort as both uses the same algorithm for merging the sorted arrays which takes o(n) space. But bottom up merge sort does not need to maintain a call stack.

5. What is the average time complexity of bottom up merge sort?
a) O(n log n)
b) $O(n^2)$
c) $O(n^2 \log n)$
d) $O(n \log n^2)$
View Answer
Answer: a
Explanation: The merge function in the bottom up merge sort takes O(n) time which is placed inside the for loop. The loop runs for O(log n) time, thus the overall time complexity of the code becomes O(n log n).

Check this: Programming MCQs | Computer Science MCQs

6. Merge sort uses which of the following method to implement sorting?
a) merging
b) partitioning
c) selection
d) exchanging
View Answer
Answer: a
Explanation: Merge sort implements sorting by merging the sorted versions of smaller parts of the array. Thus its method of sorting is called merging.

7. Bottom up merge sort uses recursion.
a) true
b) false
View Answer
Answer: b
Explanation: Bottom up merge sort uses the iterative method in order to implement sorting. It begins by merging a pair of adjacent array of size 1 each and then merge arrays of size 2 each in the next step and so on.

8. Bottom up merge sort is a stable sort.
a) true

b) false
View Answer
Answer: a
Explanation: Bottom up merge sort like standard merge sort is a stable sort. This implies that the relative position of equal valued elements in the input and sorted array remain same.

9. Choose the correct statement about bottom up merge sort from the following?
a) bottom up merge sort has greater time complexity than standard merge sort
b) bottom up merge sort has lesser time complexity than standard merge sort
c) bottom up merge sort saves auxiliary space required on call stack
d) bottom up merge sort uses recursion.
View Answer
Answer: c
Explanation: Bottom up merge sort unlike standard merge sort uses an iterative algorithm for sorting. Thus, it saves auxiliary space required by the call stack.

10. Choose the correct option from the following that represents bottom up merge sort function?
a)

```
void mergesort(int Arr[], int temp[], int low, int high)
{
         for (int m = 1; m <= high - low; m = 2*m)
         {
                  for (int i = low; i < high; i += 2*m)
                  {
                           int left = i;
                           int mid = i + m - 1;
                           int right = min(i + 2*m - 1, high);
                           merge(Arr, temp, left, mid, right);
                  }
         }
}
```

b)

```
void mergesort(int Arr[], int temp[], int low, int high)
{
         for (int m = 1; m <= high - low; m = 2*m)
         {
                  for (int i = low; i < high; i += m)
                  {
```

```
                        int left = i;
                        int mid = i + m - 1;
                        int right = min(i + 2*m - 1, high);
                        merge(Arr, temp, left, mid, right);
                }
        }
}
```

c)

```
void mergesort(int Arr[], int temp[], int low, int high)
{
        for (int m = 1; m <= high - low; m = m)
        {
                for (int i = low; i < high; i += 2*m)
                {
                        int left = i;
                        int mid = i + m - 1;
                        int right = min(i + 2*m - 1, high);
                        merge(Arr, temp, left, mid, right);
                }
        }
}
```

d)

```
void mergesort(int Arr[], int temp[], int low, int high)
{
        for (int m = 1; m <= high - low; m = 2*m)
        {
                for (int i = low; i < high; i += 2*m)
                {
                        int left = i;
                        int mid = i + m - 1;
                        int right = min(i + m - 1, high);
                        merge(Arr, temp, left, mid, right);
                }
        }
}
```

View Answer

Answer: a

Explanation: Bottom up merge sort uses iterative method in order to implement

sorting. It begins by merging a pair of adjacent array of size 1 each and then merge arrays of size 2 each in the next step and so on. The process of merging the sorted arrays is same as in standard merge sort.