1000-data-structures-algorithms-ii-questions-answers

1. Where is linear searching used?
a) When the list has only a few elements
b) When performing a single search in an unordered list
c) Used all the time
d) When the list has only a few elements and When performing a single search in an unordered list

Answer: d
Explanation: It is practical to implement linear search in the situations mentioned in When the list has only a few elements and When performing a single search in an unordered list, but for larger elements the complexity becomes larger and it makes sense to sort the list and employ binary search or hashing.

b)

```
int unorderedLinearSearch(int arr[], int size, int data)
{
    int index;
    for(int i = 0; i < size; i++)
    {
        if(arr[i] == data)
        {
            index = i;
            break;
        }
    }
    return index;
}
```

c)

```
int unorderedLinearSearch(int arr[], int size, int data)
{
    int index;
    for(int i = 0; i < size; i++)
    {
        if(arr[i] == data)
        {
            break;
        }
    }
    return index;
}
```

d)

```
int unorderedLinearSearch(int arr[], int size, int data)
{
    int index;
    for(int i = 0; i <= size; i++)
    {
        if(arr[i] == data)
        {
            index = i;
            break;
        }
    }
    return index;
}
```

3. What is the best case for linear search?
a) O(nlogn)
b) O(logn)
c) O(n)

d) O(1)

Answer: d
Explanation: The element is at the head of the array, hence O(1).

4. What is the worst case for linear search?
a) O(nlogn)
b) O(logn)
c) O(n)
d) O(1)

Answer: c
Explanation: Worst case is when the desired element is at the tail of the array or not present at all, in this case you have to traverse till the end of the array, hence the complexity is O(n).

b)

```
int unorderedLinearSearch(int arr[], int size, int data)
{
    int index;
    for(int i = 0; i < size-1; i++)
    {
        if(arr[i] == data)
        {
            index = i;
            break;
        }
    }
    return index;
}
```

c)

```
public int linearSearch(int arr[],int key,int size)
{
       int index = -1;
           int i = 0;
       while(size > 0)
       {
             if(data[i] == key)
             {
                  index = i;
             }
             if(data[i] > key))
             {
                 index = i;
                 break;
             }
             i++;
       }
       return index;
}
```

d)

```
public int linearSearch(int arr[],int key,int size)
{
       int index = -1;
           int i = 0;
       while(size > 0)
       {
             if(data[i] == key)
             {
                  index = i;
             }
             if(data[i] > key))
```

```
            {
                 break;
            }
            i++;
       }
       return index;
}
```

6. What is the best case and worst case complexity of ordered linear search?
a) O(nlogn), O(logn)
b) O(logn), O(nlogn)
c) O(n), O(1)
d) O(1), O(n)

Answer: d
Explanation: Although ordered linear search is better than unordered when the element is not present in the array, the best and worst cases still remain the same, with the key element being found at first position or at last position.

b)

```
public int linearSearch(int arr[],int key,int size)
{
       int index = -1;
           int i = 0;
       while(size > 0)
       {
             if(data[i] == key)
             {
                break;
             }
             if(data[i] > key))
             {
                 index = i;
             }
             i++;
        }
        return index;
}
```

c)

```
public int linearSearch(int arr[],int key,int size)
{
       int index = -1;
           int i = 0;
       while(size > 0)
       {
             if(data[i] == key)
             {
                break;
             }
             if(data[i] > key))
             {
                 break;
                 index=i;
             }
             i++;
        }
        return index;
}
```

d)

```
public void linSearch(int[] arr, int first, int last, int key)
{
        if(first == last)
        {
```

```
                System.out.print("-1");
        }
        else
        {
                if(arr[first] == key)
                {
                        System.out.print(first);
                }
                else
                {
                        linSearch(arr, first+1, last, key);
                }
        }
}
```

8. What does the following piece of code do?

```
      public void linSearch(int[] arr, int first, int last, int key)
      {
                if(first == last)
                {
                        System.out.print("-1");
                }
                else
                {
                        if(arr[first] == key)
                        {
                                System.out.print(first);
                        }
                        else
                        {
                                linSearch(arr, first+1, last-1, key);
                        }
                }
      }
```

a) Print the duplicate elements in the array
b) Print the element with maximum frequency
c) Print the unique elements in the array
d) Prints the element with minimum frequnecy

Answer: a
Explanation: The print statement is executed only when the items are equal and their indices are not.

b)

```
public void linSearch(int[] arr, int first, int last, int key)
{
        if(first == last)
        {
                System.out.print("-1");
        }
        else
        {
                if(arr[first] == key)
                {
                        System.out.print(last);
                }
                else
                {
                        linSearch(arr, first+1, last, key);
                }
        }
}
```

c)

```
public void linSearch(int[] arr, int first, int last, int key)
{
        if(first == last)
        {
                System.out.print("-1");
        }
        else
        {
                if(arr[first] == key)
                {
                        System.out.print(first);
                }
                else
                {
                        linSearch(arr, first+1, last+1, key);
                }
        }
}
```

d)

```
for (int i = 0; i < arr.length-1; i++)
{
    for (int j = i+1; j < arr.length; j++)
    {
        if( (arr[i].equals(arr[j])) && (i != j) )
        {
            System.out.println(arr[i]);
        }
    }
}
```

10. Which of the following is a disadvantage of linear search?
a) Requires more space
b) Greater time complexities compared to other searching algorithms
c) Not easy to understand
d) Not easy to implement

Answer: b
Explanation: The complexity of linear search as the name suggests is O(n) which is much greater than other searching techniques like binary search(O(logn)). Linear search is easy to implement and understand than other searching techniques.

1. Is there any difference in the speed of execution between linear serach(recursive) vs linear search(Iterative)?
a) Both execute at same speed
b) Linear search(recursive) is faster
c) Linear search(Iterative) is faster
d) Cant be said

Answer: c
Explanation: The Iterative algorithm is faster than the latter as recursive algorithm has overheads like calling function and registering stacks repeatedly.

2. Is the space consumed by the linear search(recursive) and linear search(iterative) same?
a) No, recursive algorithm consumes more space
b) No, recursive algorithm consumes less space
c) Yes
d) Nothing can be said

Answer: a
Explanation: The recursive algorithm consumes more space as it involves the usage the stack space(calls the function numerous times).

3. What is the worst case runtime of linear search(recursive) algorithm?
a) O(n)
b) O(logn)
c) $O(n^2)$
d) O(nx)

Answer: a
Explanation: In the worst case scenario, there might be a need of calling the stack n times. Therfore O(n).

4. Linear search(recursive) algorithm used in _____
a) When the size of the dataset is low
b) When the size of the dataset is large
c) When the dataset is unordered
d) Never used

Answer: a
Explanation: It is used when the size of the dataset is low as its runtime is O(n) which is more when compared to the binary search O(logn).

5. The array is as follows: 1,2,3,6,8,10. At what time the element 6 is found? (By using linear search(recursive) algorithm)
a) 4th call
b) 3rd call
c) 6th call
d) 5th call

Answer: a
Explanation: Provided that the search starts from the first element, the function calls itself till the element is found. In this case, the element is found in 4th call.

6. The array is as follows: 1,2,3,6,8,10. Given that the number 17 is to be searched. At which call it tells that there's no such element? (By using linear search(recursive) algorithm)
a) 7th call
b) 9th call
c) 17th call
d) The function calls itself infinite number of times

Answer: a
Explanation: The function calls itself till the element is found. But at the 7th call it terminates as goes outside the array.

7. What is the best case runtime of linear search(recursive) algorithm on an ordered set of elements?
a) O(1)
b) O(n)
c) O(logn)
d) O(nx)

Answer: a
Explanation: The best case occurs when the given element to be found is at the first position. Therefore O(1) is the correct answer.

b)

```
for(i=0;i<n;i++)
{
        if(a[i]==key)
        printf("element found");
}
```

c)

```
        LinearSearch(int[] a, n,key)
        {
                if(n<1)
                return False
                if(a[n]==key)
                return True
                else
                LinearSearch(a,n-1,key)
        }
```

d)

```
        LinearSearch(int[] a, n,key)
        {
                if(n<1)
                return True
                if(a[n]==key)
                return False
                else
                LinearSearch(a,n-1,key)
        }
```

9. Can linear search recursive algorithm and binary search recursive algorithm be performed on an unordered list?
a) Binary search can't be used
b) Linear search can't be used
c) Both cannot be used
d) Both can be used

Answer: a
Explanation: As binary search requires comparison, it is required that the list be ordered. Whereas this doesn't matter for linear search.

10. What is the recurrence relation for the linear search recursive algorithm?
a) T(n-2)+c
b) 2T(n-1)+c
c) T(n-1)+c
d) T(n+1)+c

Answer: c
Explanation: After each call in the recursive algorithm, the size of n is reduced by 1. Therefore the optimal solution is T(n-1)+c.

1. What is the advantage of recursive approach than an iterative approach?
a) Consumes less memory
b) Less code and easy to implement
c) Consumes more memory
d) More code has to be written

Answer: b
Explanation: A recursive approach is easier to understand and contains fewer lines of code.

b)

```
public static int recursive(int arr[], int low, int high, int key)
{
        int mid = low + (high - low)/2;
        if(arr[mid] == key)
        {
                return mid;
        }
        else if(arr[mid] < key)
        {
                return recursive(arr,mid+1,high,key);
```

```
        }
        else
        {
                return recursive(arr,low,mid-1,key);
        }
}
```

c)

```
public static int recursive(int arr[], int low, int high, int key)
{
        int mid = low + (high + low)/2;
        if(arr[mid] == key)
        {
                return mid;
        }
        else if(arr[mid] < key)
        {
                return recursive(arr,mid-1,high,key);
        }
        else
        {
                return recursive(arr,low,mid+1,key);
        }
}
```

d)

```
public static int recursive(int arr[], int low, int high, int key)
{
        int mid = low + (high - low)/2;
        if(arr[mid] == key)
        {
                return mid;
        }
        else if(arr[mid] < key)
        {
                return recursive(arr,mid,high,key);
        }
        else
        {
                return recursive(arr,low,mid-1,key);
        }
}
```

3. Given an input arr = {2,5,7,99,899}; key = 899; What is the level of recursion?
a) 5
b) 2
c) 3
d) 4

Answer: c
Explanation: level 1: mid = 7
level 2: mid = 99
level 3: mid = 899(this is the key).

4. Given an array arr = {45,77,89,90,94,99,100} and key = 99; what are the mid values(corresponding array elements) in the first and second levels of recursion?
a) 90 and 99
b) 90 and 94
c) 89 and 99
d) 89 and 94

Answer: a
Explanation: At first level key = 90

At second level key= 99
Here 90 and 99 are mid values.

5. What is the worst case complexity of binary search using recursion?
a) O(nlogn)
b) O(logn)
c) O(n)
d) O(n^2)

Answer: b
Explanation: Using the divide and conquer master theorem.

6. What is the average case time complexity of binary search using recursion?
a) O(nlogn)
b) O(logn)
c) O(n)
d) O(n^2)

Answer: b
Explanation: T(n) = T(n/2) + 1, Using the divide and conquer master theorem.

7. Which of the following is not an application of binary search?
a) To find the lower/upper bound in an ordered sequence
b) Union of intervals
c) Debugging
d) To search in unordered list

Answer: d
Explanation: In Binary search, the elements in the list should be sorted. It is applicable only for ordered list. Hence Binary search in unordered list is not an application.

b)

```
public static int recursive(int arr[], int low, int high, int key)
{
        int mid = low + ((high - low)/2)+1;
        if(arr[mid] == key)
        {
                return mid;
        }
        else if(arr[mid] < key)
        {
                return recursive(arr,mid,high,key);
        }
        else
        {
                return recursive(arr,low,mid-1,key);
        }
}
```

c)

```
public static int iterative(int arr[], int key)
{
        int low = 0;
        int mid = 0;
        int high = arr.length-1;
        while(low <= high)
        {
                mid = low + (high + low)/2;
                if(arr[mid] == key)
                {
                        return mid;
```

```
                }
                else if(arr[mid] < key)
                {
                        low = mid - 1;
                }
                else
                {
                        high = mid + 1;
                }
        }
        return -1;
}
```

d)

```
public static int iterative(int arr[], int key)
{
        int low = 0;
        int mid = 0;
        int high = arr.length-1;
        while(low <= high)
        {
                mid = low + (high - low)/2;
                if(arr[mid] == key)
                {
                        return mid;
                }
                else if(arr[mid] < key)
                {
                        low = mid + 1;
                }
                else
                {
                        high = mid - 1;
                }
        }
        return -1;
}
```

9. Binary Search can be categorized into which of the following?
a) Brute Force technique
b) Divide and conquer
c) Greedy algorithm
d) Dynamic programming

Answer: b
Explanation: Since 'mid' is calculated for every iteration or recursion, we are diving the array into half and then try to solve the problem.

10. Given an array arr = {5,6,77,88,99} and key = 88; How many iterations are done until the element is found?
a) 1
b) 3
c) 4
d) 2

Answer: d
Explanation: Iteration1 : mid = 77; Iteration2 : mid = 88;

11. Given an array arr = {45,77,89,90,94,99,100} and key = 100; What are the mid values(corresponding array elements) generated in the first and second iterations?
a) 90 and 99
b) 90 and 100
c) 89 and 94
d) 94 and 99

Answer: a
Explanation: Trace the input with the binary search iterative code.

12. What is the time complexity of binary search with iteration?
a) O(nlogn)
b) O(logn)
c) O(n)
d) O(n^2)

Answer: b
Explanation: T(n) = T(n/2) + theta(1)
Using the divide and conquer master theorem, we get the time complexity as O(logn).

1. In which of the cases uniform binary search fails compared to binary search?
a) A table lookup is generally faster than an addition and a shift
b) Many searches will be performed on the same array
c) Many searches will be performed on several arrays of the same length
d) Complexity of code

Answer: d
Explanation: Uniform binary search code is more complex to implement than binary search as it involves mid points to be computed in hand before search.

b)

```
public static void make_delta(int N)
{
       int power = 1;
       int i = 0;
       do
       {
            int half = power;
            power <<= 1;
            delta[i] = (N + half) / power;
       }
       while (delta[i++] != 0);
}
```

c)

```
public static void make_delta(int N)
{
       int power = 0;
       int i = 0;
       do
       {
            int half = power;
            power <<= 1;
            delta[i] = (N + half) / power;
       }
       while (delta[i++] != 0);
}
```

d)

```
public static void make_delta(int N)
{
       int power = 1;
       int i = 0;
       do
       {
            int half = power;
            power >>= 1;
            delta[i] = (N + half) / power;
```

```
        }
        while (delta[i++] != 0);
}
```

3. Given delta[4] is a global array and number of elements in the sorted array is 10, what are the values in the delta array?
a) 4, 3, 1, 0
b) 5, 3, 1, 0
c) 4, 2, 1, 1
d) 5, 2, 1, 1

Answer: b
Explanation: Trace with respect to the make_delta function, always note that the last element is always 0.

b)

```
public static void make_delta(int N)
{
        int power = 1;
        int i = 0;
        do
        {
             int half = power;
             power <<= 1;
             delta[i] = (N - half) / power;
        }
        while (delta[i++] != 0);
}
```

c)

```
public static int unisearch(int key)
{
        int i = delta[0] - 1;
        int j = 0;
        while (true)
        {
             if (key == arr[i])
                 return i;
             else if (delta[j] == 0)
                 return -1;
             else
             {
                 if (key < arr[i])
                     i += delta[++j];
                 else
                     i -= delta[++j];
             }
        }
}
```

d)

```
public static int unisearch(int key)
{
        int i = delta[1] - 1;
        int j = 0;
        while (true)
        {
             if (key == arr[i])
                 return i;
             else if (delta[j] == 0)
                 return -1;
             else
             {
                 if (key < arr[i])
                     i -= delta[++j];
```

```
                else
                    i += delta[++j];
            }
        }
}
```

5. What is the time complexity of uniform binary search?
a) O(nlogn)
b) O(logn)
c) O(n)
d) O(n^2)

Answer: b
Explanation: With every iteration we are dividing the array into two parts(though not equal halves), the complexity remains same as the normal binary search.Answer: b
Explanation: Tracing with the above code, comparison #1: i=4, comparison #2: i=7, comparison #3: i=8

7. How can Jump Search be improved?
a) Start searching from the end
b) Begin from the kth item, where k is the step size
c) Cannot be improved
d) Step size should be other than sqrt(n)

Answer: b
Explanation: This gives a very slight improvement as you are skipping the first k elements.

8. Which of the following false about Jump Search?
a) Jump Search is better than Linear Search
b) Useful when jumping back is more costly than jumping forward
c) Jump Search is worse than Binary Search
d) Jump search starts from the index 0 even though specified index is k

Answer: d
Explanation: Linear search has O(n) complexity and Binary search has O(logn) complexity, in Jump search you have to jump backwards only once, hence it is preferable if jumping backwards is costly. Jump search starts from index k (specified index) and searches for the element. It won't start searching from index 0.

1. Jump search algorithm requires which of the following condition to be true?
a) array should be sorted
b) array should have not be sorted
c) array should have a less than 64 elements
d) array should be partially sorted

Answer: a
Explanation: Jump sort requires the input array to be sorted. The algorithm would fail to give the correct result if array is not sorted.

2. Jumps are made in the jump search algorithm until _____
a) element having value less than that of the required element is found
b) element having value equal to the median of values of the array is found
c) element having value greater than that of the required element is found
d) middle element is found equal to the element being searched

Answer: c
Explanation: In jump search algorithm jumps are made until element having value greater than the value of element being searched is found. After this linear search is performed in backwards direction.

3. Which of the following step is taken after finding an element having value greater than the element being searched?

a) linear search takes place in the forward direction
b) linear search takes place in the backward direction
c) binary search takes place in the forward direction
d) binary search takes place in a backward direction

Answer: b
Explanation: First an element having value greater than the element being searched is found. After this linear search is performed in a backward direction.

4. How many jumps will be made in the worst case of jump search(let block jumped =k)?
a) n*k
b) n/k
c) k/n
d) n+k

Answer: b
Explanation: Worst case occurs when the value to be searched is in the last section of the array. So, in this case the number of jumps will be n/k.

5. What will be the maximum number of comparisons that can be made in jump search algorithm (assuming k to be blocks jumped)?
a) k
b) n/k
c) k-1
d) k-1

Answer: c
Explanation: Worst case occurs when the element being searched is present just after the element that has been compared while making the last jump. So, in this case k-1 comparisons will have to be made.

6. What is the value of jump taken for maximum efficiency while implementing jump search?
a) n/2
b) n^2
c) $n^{1/2}$
d) log n

Answer: c
Explanation: Total number of comparisons required will be n/k + k-1 in worst case. This function will be minimum for $k=n^{1/2}$. So this value of jump will be the best for implementing jump search.

7. What is the auxiliary space requirement of the jump search?
a) O(n)
b) O(log n)
c) O($n^{1/2}$)
d) O(1)

Answer: d
Explanation: Jump search does not require any additional space for searching the required element. Thus its auxiliary space requirement will be O(1).

8. Which of the following searching algorithm is fastest?
a) jump search
b) binary search
c) linear search
d) all are equally fast

Answer: b

Explanation: Binary search has the least time complexity (equal to log n) out of the given searching algorithms. This makes binary search preferable in most cases.

9. In which of the following case jump search will be preferred over binary search?
a) jumping backwards takes significantly more time than jumping forward
b) jumping forward takes significantly more time than jumping backwards
c) when the given array is very large in size
d) when the given array is very small in size

Answer: a
Explanation: Jump search only needs to jump backwards once, while a binary can jump backwards up to log n times. Thus jump search will be preferred over binary search if jumping backwards is expensive.

10. Best case of jump search will have time complexity of _____
a) O(1)
b) O(n)
c) O(log n)
d) O(n log n)

Answer: a
Explanation: Best case of jump search will be when the first element of the array is the element that is being searched. In this case only one comparison will be required. Thus it will have a time complexity of O(1).

b)

```
int jumpSearch(int arr[], int x, int n)
{

    int step = n*n;
    int prev = 0;
    while (arr[min(step, n)-1] < x)
    {
        prev = step;
        step += sqrt(n);
        if (prev >= n)
            return -1;
    }


    while (arr[prev] < x)
    {
        prev++;


        if (prev == min(step, n))
            return -1;
    }

    if (arr[prev] == x)
        return prev;

    return -1;
}
```

c)

```
int jumpSearch(int arr[], int x, int n)
{

    int step = sqrt(n);
    int prev = 0;
    while (arr[min(step, n)-1] < x)
    {
        prev = step;
        step += sqrt(n);
```

```
        if (prev >= n)
            return -1;
    }


    while (arr[prev] < x)
    {
        prev++;


        if (prev == min(step, n))
            return -1;
    }

    if (arr[prev] == x)
        return prev;

    return -1;
}
```

d)

```
int jumpSearch(int arr[], int x, int n)
{

    int step = sqrt(n);
    int prev = 0;
    while (arr[min(step, n)-1] < x)
    {
        prev = step;
        step += sqrt(n);
        if (prev == n)
            return -1;
    }


    while (arr[prev] < x)
    {
        prev++;


        if (prev == min(step, n))
            return -1;
    }

    if (arr[prev] == x)
        return prev;

    return -1;
}
```

Answer: b

Explanation: Linear search has a time complexity of O(n) and the time complexity of jump search is $O(n^{1/2})$. So jump search is better than linear search in terms of time complexity.Answer: b

Explanation: The time complexity of jump search is $O(n^{1/2})$ in worst and average case. It is due to the fact that size of optimal jump is $n^{1/2}$.

1. Which algorithmic technique does Fibonacci search use?
a) Brute force
b) Divide and Conquer
c) Greedy Technique
d) Backtracking

Answer: b
Explanation: With every iteration, we divide the given array into two sub arrays(not necessarily equal).

2. Choose the recursive formula for the Fibonacci series.(n>=1)
a) F(n) = F(n+1) + F(n+2)
b) F(n) = F(n) + F(n+1)
c) F(n) = F(n-1) + F(n-2)
d) F(n) = F(n-1) – F(n-2)

Answer: c
Explanation: None.

b)

```
public static int fibSearch(final int key, final int[] a)
{
        int low = 0;
        int high = a.length - 1;
        int fibCurrent = 1;
        int fibPrev = 1;
        int N = a.length;
        while (low <= high)
        {
            while(fibCurrent < N)
            {
                int tmp = fibCurrent + fibPrev;
                fibPrev = fibCurrent;
                fibCurrent = tmp;
                N = N - (fibCurrent - fibPrev);
            }
            final int mid = low + (high - low) - (fibCurrent + fibPrev);
            if      (key < a[mid]) high = mid - 1;
            else if (key > a[mid]) low = mid + 1;
            else return mid;
        }
        return -1;
}
```

c)

```
public static int fibSearch(final int key, final int[] a)
{
        int low = 0;
        int high = a.length - 1;
        int fibCurrent = 1;
        int fibPrev = 1;
        int N = a.length;
        while (low <= high)
        {
            int tmp = fibCurrent + fibPrev;
            fibPrev = fibCurrent;
            fibCurrent = tmp;
            N = N - (fibCurrent - fibPrev);
            final int mid = low + (high - low) - (fibCurrent + fibPrev);
            if      (key < a[mid]) high = mid - 1;
            else if (key > a[mid]) low = mid + 1;
            else return mid;
        }
        return -1;
}
```

d)

```
public static int fibSearch(final int key, final int[] a)
{
        int low = 0;
        int high = a.length - 1;
        int fibCurrent = 1;
        int fibPrev = 1;
        int N = a.length;
```

```
        while (low <= high)
        {
            while(fibCurrent < N)
            {
                int tmp = fibCurrent + fibPrev;
                fibPrev = fibCurrent;
                fibCurrent = tmp;
                N = N - (fibCurrent - fibPrev);
            }
            final int mid = low + (high - low) - (fibCurrent + fibPrev);
            if      (key < a[mid]) low = mid + 1;
            else if (key > a[mid]) high = mid - 1;
            else return mid;
        }
}
```

4. What is the time complexity of Fibonacci Search?
a) O(logn)
b) O(n)
c) O(n^2)
d) O(nlogn)

Answer: a
Explanation: Since it divides the array into two parts, although not equal, its time complexity is O(logn), it is better than binary search in case of large arrays.

5. Which of the following is not an advantage of Fibonacci Search?
a) When the element being searched for has a non uniform access storage
b) Can be used in magnetic tapes
c) Can be used for large arrays which do not fit in the CPU cache or in the RAM
d) It can be applied efficiently on unsorted arrays

Answer: d
Explanation: When the speed of access depends on the location previously accessed, Fibonacci search is better compared to binary search as it performs well on those locations which have lower dispersion. Fibonacci search won't work on unsorted arrays. The input should be a sorted array or array should be sorted before Fibonacci search.

6. What is the length of the step in jump search?
a) n
b) n/2
c) sqrt(n)
d) 1

Answer: c
Explanation: If the step size is 1, it becomes a linear search, if it is n, we reach the end of the list in just on step, if it is n/2, it becomes similar to binary search, therefore the most efficient step size is found to be sqrt(n).

b)

```
public static int fibSearch(final int key, final int[] a)
{
        int low = 0;
        int high = a.length - 1;
        int fibCurrent = 1;
        int fibPrev = 1;
        int N = a.length;
        while (low <= high)
        {
            while(fibCurrent < N)
            {
                int tmp = fibCurrent + fibPrev;
                fibPrev = fibCurrent;
                fibCurrent = tmp;
```

```
            N = N - (fibCurrent - fibPrev);
        }
        final int mid = low + (high - low) - (fibCurrent + fibPrev);
        if      (key < a[mid]) low = mid - 1;
        else if (key > a[mid]) high = mid - 1;
        else return mid;
    }
}
```

c)

```
public int jumpSearch(int arr[], int key)
{
        int size = arr.length;
        int step = floor(sqrt(size));
        int prev = 0;
        while (arr[(step > size ? step : size)] < key)
        {
                prev = step;
                step += floor(sqrt(size));
                if (step >= size)
                {
                        return -1;
                }
        }
        while (arr[prev] < key)
        {
                prev++;
                if (prev == (step < size ? step : size))
                {
                        return -1;
                }
        }
        if (arr[prev] == key)
        {
                return prev;
        }
        return -1;
}
```

d)

```
public int jumpSearch(int arr[], int key)
{
        int size = arr.length;
        int step = floor(sqrt(size));
        int prev = 0;
        while (arr[(step < size ? step : size)] < key)
        {
                prev = step;
                step += floor(sqrt(size));
                if (step >= size)
                {
                        return -1;
                }
        }
        while (arr[prev] < key)
        {
                prev++;
                if (prev == (step < size ? step : size))
                {
                        return -1;
                }
        }
        if (arr[prev] == key)
        {
                return prev;
        }
        return -1;
```

```
}
```

8. What is the time complexity of Jump Search?
a) O(logn)
b) O(n)
c) O(sqrt(n))
d) O(nlogn)

Answer: c
Explanation: Since the size of the step is sqrt(n), the complexity is also obviously O(sqrt(n)).

1. Exponential search algorithm requires which of the following condition to be true?
a) array should be sorted
b) array should have not be sorted
c) array should have a less than 128 elements
d) array should be partially sorted

Answer: a
Explanation: Exponential sort requires the input array to be sorted. The algorithm would fail to give the correct result if array is not sorted.

2. Which of the following searching algorithm is used with exponential sort after finding the appropriate range?
a) Linear search
b) Binary search
c) Jump search
d) Fibonacci Search

Answer: b
Explanation: In exponential search, we first find a range where the required elements should be present in the array. Then we apply binary search in this range.

3. Exponential search has _____
a) neither an exponential space complexity nor exponential time complexity
b) exponential time complexity but a linear space complexity
c) exponential space complexity but a linear time complexity
d) both exponential time and space complexity

Answer: a
Explanation: Exponential search has neither an exponential space complexity nor exponential time complexity. It is named exponential search because it searches for an element in an exponential manner.

5. What is the time complexity of exponential sort?
a) O(n)
b) O(2n)
c) O(n log n)
d) O(log n)

Answer: d
Explanation: In exponential search, we first find a range where the required elements should be present in the array. Then we apply binary search in this range. This takes O(log n) time in the worst case.

6. What is the auxiliary space requirement of an exponential sort when used with iterative binary search?
a) O(n)
b) $O(2^n)$
c) O(1)
d) O(log n)

Answer: c

Explanation: Exponential search does not require any auxiliary space for finding the element being searched. So it has a constant auxiliary space O(1).

7. What is the auxiliary space requirement of the exponential sort when used with recursive binary search?
a) O(n)
b) $O(2^n)$
c) O(1)
d) O(log n)

Answer: d
Explanation: Exponential search requires an auxiliary space of log n when used with recursive binary search. This space is required for the recursion call stack space.

8. Which of the following searching algorithm is fastest?
a) jump search
b) exponential search
c) linear search
d) all are equally fast

Answer: b
Explanation: Exponential search has the least time complexity (equal to log n) out of the given searching algorithms. This makes exponential search preferable in most cases.

9. In which of the following case jump search will be preferred over exponential search?
a) jumping backwards takes significantly more time than jumping forward
b) jumping forward takes significantly more time than jumping backwards
c) when the given array is very large in size
d) when the given array is very small in size

Answer: a
Explanation: Jump search only needs to jump backwards once, while an exponential search can jump backwards up to log n times. Thus jump search will be preferred if jumping backwards is expensive.

10. Best case of the exponential search will have time complexity of?
a) O(1)
b) O(n)
c) O(log n)
d) O(n log n)

Answer: a
Explanation: Best case of the exponential search will be when the first element of the array is the element that is being searched. In this case, only one comparison will be required. Thus it will have a time complexity of O(1).

b)

```
int expSearch(int arr[], int n, int x)
{
        if (arr[0] == x)
                return 0;
        int i = 1;
        while (i < n && arr[i] <= x)
                i = i*2;

return binarySearch(arr, i/2, min(i, n-1), x);
//applies binary search in the calculated range
}
```

c)

```
int expSearch(int arr[], int n, int x)
```

```
{
        if (arr[0] == x)
                return 0;
        int i = 1;
        while (i < n && arr[i] <= x)
                i = i*2;
        return binarySearch(arr, i, min(i, n-1), x);
//applies binary search in the calculated range
}
```

d)

```
int expSearch(int arr[], int n, int x)
{

        if (arr[0] == x)
                return 0;


        int i = 1;
        while (i < n && arr[i] <= x)
                i = i/2;

return binarySearch(arr, i/2, min(i, n-1), x);
//applies binary search in the calculated range
}
```

Answer: b
Explanation: The worst case time complexity of jump search and exponential searches are O(n1/2) and O(log n) respectively. So exponential search is better in terms of time complexity.Answer: a
Explanation: Exponential search first finds the range where binary search needs to be applied. So when the element is present near the starting point of the array then exponential search performs better than standard binary search.

14. Choose the incorrect statement about exponential search from the following.
a) Exponential search is an in place algorithm
b) Exponential search has a greater time complexity than binary search
c) Exponential search performs better than binary search when the element being searched is present near the starting point of the array
d) Jump search has a greater time complexity than an exponential search

Answer: b
Explanation: Time complexity of exponential search and binary search are the same. But exponential search performs better than binary search when the element being searched is present near the starting point of the array.

15. Which of the following is not an alternate name of exponential search?
a) Logarithmic search
b) Doubling search
c) Galloping search
d) Struzik search

Answer: a
Explanation: Logarithmic search is not an alternate name of the exponential search. Some alternate names of exponential search are Doubling search, Galloping search and Struzik search.

1. Which of the following is the most desirable condition for interpolation search?
a) array should be sorted
b) array should not be sorted but the values should be uniformly distributed
c) array should have a less than 64 elements
d) array should be sorted and the values should be uniformly distributed

Answer: d
Explanation: Desirable condition for interpolation search is that the array should be sorted and the values should be

uniformly distributed. The algorithm would fail to give the correct result if array is not sorted.

2. Interpolation search is a variation of?
a) Linear search
b) Binary search
c) Jump search
d) Exponential search

Answer: b
Explanation: Interpolation search is a variation of binary search which gives the best result when the array has uniformly distributed values. Interpolation search goes to different positions depending on the value being searched whereas binary search always goes to the middle element.

3. Interpolation search performs better than binary search when?
a) array has uniformly distributed values but is not sorted
b) array is sorted and has uniform distribution of values
c) array is sorted but the values are not uniformly distributed
d) array is not sorted

Answer: b
Explanation: Interpolation search is an improvement over a binary search for the case when array is sorted and has uniformly distributed values. Binary search performs better when the values are not distributed uniformly.

4. In which of the following case jump search performs better than interpolation search?
a) When array has uniformly distributed values but is not sorted
b) when array is sorted and has uniform distribution of values
c) when array is sorted but the values increases exponentially
d) when array is not sorted

Answer: c
Explanation: In case of non uniform distribution of values the time complexity of interpolation search is O(n) whereas the average time complexity of jump search is $O(n^{1/2})$*. So in such a case jump search has a better performance.*

5. What is the time complexity of interpolation search when the input array has uniformly distributed values and is sorted?
a) O(n)
b) O(log log n)
c) O(n log n)
d) O(log n)

Answer: b
Explanation: Interpolation search goes to different positions in the array depending on the value being searched. It is an improvement over binary search and has a time complexity of O(log log n).

6. What is the auxiliary space requirement of interpolation search?
a) O(n)
b) $O(2^n)$
c) O(1)
d) O(log n)

Answer: c
Explanation: Interpolation search does not require any auxiliary space for finding the element being searched. So it has a constant auxiliary space O(1).

7. What is the time complexity of exponential search when the input array is sorted but the values are not uniformly distributed?
a) $O(n^{1/2})$

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

Answer: c
Explanation: When an array has non uniformly distributed values then in that case the algorithm of interpolation search fails to work efficiently. As a result, it has a time complexity of O(n) in such a case.

8. Which of the following searching algorithm is fastest when the input array is sorted and has uniformly distributed values?
a) jump search
b) exponential search
c) binary search
d) interpolation search

Answer: d
Explanation: Interpolation search has a time complexity of O(log log n) when the array is sorted and has uniformly distributed values. It has the least time complexity out of the given options for such a case.

9. Which of the following searching algorithm is fastest when the input array is sorted but has non uniformly distributed values?
a) jump search
b) linear search
c) binary search
d) interpolation search

Answer: c
Explanation: Interpolation search has a time complexity of O(n) when the array does not have uniformly distributed values. So in such a case binary search has the least time complexity out of the given options.

10. Which of the following searching algorithm is fastest when the input array is not sorted but has uniformly distributed values?
a) jump search
b) linear search
c) binary search
d) interpolation search

Answer: b
Explanation: Out of the given options linear search is the only searching algorithm which can be applied to arrays which are not sorted. It has a time complexity of O(n) in the worst case.Answer: a
Explanation: Interpolation search has an auxiliary space complexity of O(1). So it qualifies as an in place algorithm.Answer: b
Explanation: The worst case time complexity of interpolation search and exponential search are O(n) and O(log n) respectively. So exponential search is better when the worst case scenario is considered. Answer: c
*Explanation: For calculating the position after each iteration in interpolation search we use the formula low + ((x – A[low]) * (high – low)) / (A[high] – A[low]). Then the value at the calculated position is compared with the element being searched.*

14. What are the updated values of high and low in the array if the element being searched is greater than the value at calculated index in interpolation search? (pos = current position)
a) low = pos + 1, high remains unchanged
b) high = pos – 1, low remains unchanged
c) low = low +1, high = high – 1
d) low = pos +1, high = pos – 1

Answer: a
Explanation: When the element being searched is greater than the value at the calculated position then in that case we

update low and high remains unaltered. Updated value of low is low = pos + 1.

15. What are the updated values of high and low in the array if the element being searched is lower than the value at calculated index in interpolation search? (pos = current position)
a) low = pos + 1, high remains unchanged
b) high = pos – 1, low remains unchanged
c) low = low +1, high = high – 1
d) low = pos +1, high = pos – 1

Answer: b
Explanation: When the element being searched is lower than the value at the calculated position then in that case we update high and low remains unaltered. Updated value of high is high = pos – 1.

1. Which of the following is a sub string of "SANFOUNDRY"?
a) SANO
b) FOUND
c) SAND
d) FOND

Answer: b
Explanation: A sub string is a subset of another string. So "FOUND" is the only possible sub string out of the given options.

2. What will be the output of the following code?

```
#include<bits/stdc++.h>
using namespace std;

void func(char* str2, char* str1)
{
        int m = strlen(str2);
        int n = strlen(str1);
        for (int i = 0; i <= n - m; i++)
        {
                int j;


                for (j = 0; j < m; j++)
                        if (str1[i + j] != str2[j])
                                break;

                if (j == m)
                        cout << i << endl;
        }
}

int main()
{
        char str1[] = "1253234";
        char str2[] = "323";
        func(str2, str1);
        return 0;
}
```

a) 1
b) 2
c) 3
d) 4

Answer: c
Explanation: The given code describes the naive method of finding a pattern in a string. So the output will be 3 as the given sub string begins at that index in the pattern.

3. What will be the worst case time complexity of the following code?

```
#include<bits/stdc++.h>
using namespace std;

void func(char* str2, char* str1)
{
        int m = strlen(str2);
        int n = strlen(str1);
        for (int i = 0; i <= n - m; i++)
        {
                int j;


                for (j = 0; j < m; j++)
                        if (str1[i + j] != str2[j])
                                break;

                if (j == m)
                        cout << i << endl;
        }
}

int main()
{
        char str1[] = "1253234";
        char str2[] = "323";
        func(str2, str1);
        return 0;
}
```

a) O(n)
b) O(m)
c) O(m * n)
d) O(m + n)

Answer: c
*Explanation: The given code describes the naive method of pattern searching. By observing the nested loop in the code we can say that the time complexity of the loop is O(m*n).*

4. What will be the auxiliary space complexity of the following code?

```
#include<bits/stdc++.h>
using namespace std;

void func(char* str2, char* str1)
{
        int m = strlen(str2);
        int n = strlen(str1);
        for (int i = 0; i <= n - m; i++)
        {
                int j;

                for (j = 0; j < m; j++)
                        if (str1[i + j] != str2[j])
                                break;

                if (j == m)
                        cout << i << endl;
        }
}

int main()
{
        char str1[] = "1253234";
        char str2[] = "323";
        func(str2, str1);
```

```
        return 0;
}
```

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

Answer: b
Explanation: The given code describes the naive method of pattern searching. Its auxiliary space requirement is O(1).

5. What is the worst case time complexity of KMP algorithm for pattern searching (m = length of text, n = length of pattern)?
a) O(n)
b) O(n*m)
c) O(m)
d) O(log n)

Answer: c
Explanation: KMP algorithm is an efficient pattern searching algorithm. It has a time complexity of O(m) where m is the length of text.

6. What will be the best case time complexity of the following code?

```
#include<bits/stdc++.h>
using namespace std;
void func(char* str2, char* str1)
{
        int m = strlen(str2);
        int n = strlen(str1);

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


                for (j = 0; j < m; j++)
                        if (str1[i + j] != str2[j])
                                break;

                if (j == m)
                        cout << i << endl;
        }
}

int main()
{
        char str1[] = "1253234";
        char str2[] = "323";
        func(str2, str1);
        return 0;
}
```

a) O(n)
b) O(m)
c) O(m * n)
d) O(m + n)

Answer: b
Explanation: The given code describes the naive method of pattern searching. The best case of the code occurs when the first character of the pattern does not appear in the text at all. So in such a case, only one iteration is required thus time complexity will be O(m).

7. What is the time complexity of Z algorithm for pattern searching (m = length of text, n = length of pattern)?
a) O(n + m)
b) O(m)
c) O(n)
d) O(m * n)

Answer: a
Explanation: Z algorithm is an efficient pattern searching algorithm as it searches the pattern in linear time. It has a time complexity of O(m + n) where m is the length of text and n is the length of the pattern.

8. What is the auxiliary space complexity of Z algorithm for pattern searching (m = length of text, n = length of pattern)?
a) O(n + m)
b) O(m)
c) O(n)
d) O(m * n)

Answer: b
Explanation: Z algorithm is an efficient pattern searching algorithm as it searches the pattern in linear time. It an auxiliary space of O(m) for maintaining Z array. Answer: a
Explanation: The auxiliary space complexity required by naive pattern searching algorithm is O(1). So it qualifies as an in place algorithm.Answer: a
*Explanation: The worst case time complexity of Rabin Karp algorithm is O(m*n) but it has a linear average case time complexity. So Rabin Karp and naive pattern searching algorithm have the same worst case time complexity.*

1. How many passes does an insertion sort algorithm consist of?
a) N
b) N-1
c) N+1
d) N^2

Answer: b
Explanation: An insertion algorithm consists of N-1 passes when an array of N elements is given.

2. Which of the following algorithm implementations is similar to that of an insertion sort?
a) Binary heap
b) Quick sort
c) Merge sort
d) Radix sort

Answer: a
Explanation: Insertion sort is similar to that of a binary heap algorithm because of the use of temporary variable to swap.

3. What is the average case running time of an insertion sort algorithm?
a) O(N)
b) O(N log N)
c) O(log N)
d) $O(N^2)$

Answer: d
Explanation: The average case analysis of a tight bound algorithm is mathematically achieved to be $O(N^2)$.Answer: a
Explanation: Each swap removes only one inversion, so $O(N^2)$ swaps are required.

5. What is the average number of inversions in an array of N distinct numbers?
a) N(N-1)/4
b) N(N+1)/2

c) N(N-1)/2
d) N(N-1)/3

Answer: a
Explanation: The total number of pairs in a list L is N(N-1)/2. Thus, an average list has half this amount, or N(N-1)/4 inversions.

6. What is the running time of an insertion sort algorithm if the input is pre-sorted?
a) $O(N^2)$
b) O(N log N)
c) O(N)
d) O(M log N)

Answer: c
Explanation: If the input is pre-sorted, the running time is O(N), because the test in the inner for loop always fails immediately and the algorithm will run quickly.Answer: b
Explanation: The number of passes is given by N-1. Here, N=6. Therefore,
6-1=5 passes.Answer: d
Explanation: After swapping elements in the second pass, the array will look like, 8, 34, 64, 51, 32, 21.

9. Which of the following real time examples is based on insertion sort?
a) arranging a pack of playing cards
b) database scenarios and distributes scenarios
c) arranging books on a library shelf
d) real-time systems

Answer: a
Explanation: Arranging a pack of cards mimics an insertion sort. Database scenario is an example for merge sort, arranging books is a stack and real-time systems uses quick sort.

10. In C, what are the basic loops required to perform an insertion sort?
a) do- while
b) if else
c) for and while
d) for and if

Answer: c
Explanation: To perform an insertion sort, we use two basic loops- an outer for loop and an inner while loop.Answer: a
Explanation: Binary search can be used in an insertion sort algorithm to reduce the number of comparisons. This is called a Binary insertion sort.

12. Which of the following options contain the correct feature of an insertion sort algorithm?
a) anti-adaptive
b) dependable
c) stable, not in-place
d) stable, adaptive

Answer: d
Explanation: An insertion sort is stable, adaptive, in-place and incremental in nature.

13. Which of the following sorting algorithms is the fastest for sorting small arrays?
a) Quick sort
b) Insertion sort
c) Shell sort
d) Heap sort

Answer: b
Explanation: For sorting small arrays, insertion sort runs even faster than quick sort. But, it is impractical to sort large

arrays.

14. For the best case input, the running time of an insertion sort algorithm is?
a) Linear
b) Binary
c) Quadratic
d) Depends on the input

Answer: a
Explanation: The best case input for an insertion sort algorithm runs in linear time and is given by O(N).

15. Which of the following examples represent the worst case input for an insertion sort?
a) array in sorted order
b) array sorted in reverse order
c) normal unsorted array
d) large array

Answer: b
Explanation: The worst case input for an insertion sort algorithm will be an array sorted in reverse order and its running time is quadratic.

1. Which of the following is correct with regard to insertion sort?
a) insertion sort is stable and it sorts In-place
b) insertion sort is unstable and it sorts In-place
c) insertion sort is stable and it does not sort In-place
d) insertion sort is unstable and it does not sort In-place

Answer: a
Explanation: During insertion sort, the relative order of elements is not changed. Therefore, it is a stable sorting algorithm. And insertion sort requires only O(1) of additional memory space. Therefore, it sorts In-place.

2. Which of the following sorting algorithm is best suited if the elements are already sorted?
a) Heap Sort
b) Quick Sort
c) Insertion Sort
d) Merge Sort

Answer: c
Explanation: The best case running time of the insertion sort is O(n). The best case occurs when the input array is already sorted. As the elements are already sorted, only one comparison is made on each pass, so that the time required is O(n).

3. The worst case time complexity of insertion sort is $O(n^2)$. What will be the worst case time complexity of insertion sort if the correct position for inserting element is calculated using binary search?
a) O(nlogn)
b) $O(n^2)$
c) O(n)
d) O(logn)

Answer: b
Explanation: The use of binary search reduces the time of finding the correct position from O(n) to O(logn). But the worst case of insertion sort remains $O(n^2)$ because of the series of swapping operations required for each insertion. Answer: a
Explanation: In the incremental algorithms, the complicated structure on n items is built by first building it on $n - 1$ items. And then we make the necessary changes to fix things in adding the last item. Insertion sort builds the sorted sequence one element at a time. Therefore, it is an example of an incremental algorithm.

5. 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)

Answer: b
Explanation: In insertion sort, the element is A[j] is inserted into the correct position in the sorted sequence A[1… j – 1]. So, condition given in (j > 0) && (arr[j − 1] > value) will implement while loop correctly.

6. Which of the following is good for sorting arrays having less than 100 elements?
a) Quick Sort
b) Selection Sort
c) Merge Sort
d) Insertion Sort

Answer: d
Explanation: The insertion sort is good for sorting small arrays. It sorts smaller arrays faster than any other sorting algorithm.

7. Consider an array of length 5, arr[5] = {9,7,4,2,1}. What are the steps of insertions done while running insertion sort on the array?
a) 7 9 4 2 1 4 7 9 2 1 2 4 7 9 1 1 2 4 7 9
b) 9 7 4 1 2 9 7 1 2 4 9 1 2 4 7 1 2 4 7 9
c) 7 4 2 1 9 4 2 1 9 7 2 1 9 7 4 1 9 7 4 2
d) 7 9 4 2 1 2 4 7 9 1 4 7 9 2 1 1 2 4 7 9

Answer: a
Explanation: The steps performed while running insertion sort on given array are:
Initial : 9 7 4 2 1 key = 7
7 9 4 2 1 key = 4
4 7 9 2 1 key = 2
2 4 7 9 1 key = 1
1 2 4 7 9

In each step, the key is the element that is compared with the elements present at the left side to it.

Answer: b
Explanation: In insertion sort, after m passes through the array, the first m elements are in sorted order but they are whatever the first m elements were in the unsorted array.

9. In insertion sort, the average number of comparisons required to place the 7^{th} element into its correct position is

a) 9

b) 4
c) 7
d) 14

Answer: b
Explanation: On average (k + 1) / 2 comparisons are required to place the k^{th} element into its correct position. Therefore, average number of comparisons required for 7th element = (7 + 1)/2 = 4.

10. Which of the following is not an exchange sort?
a) Bubble Sort
b) Quick Sort
c) Partition-exchange Sort
d) Insertion Sort

Answer: d
Explanation: In Exchange sorts, we compare each element of an array and swap those elements that are not in their proper position. Bubble Sort and Quick Sort are exchange sorts. Quick Sort is also called as Partition-exchange Sort. Insertion sort is not an exchange sort.

1. What is an in-place sorting algorithm?
a) It needs O(1) or O(logn) memory to create auxiliary locations
b) The input is already sorted and in-place
c) It requires additional storage
d) It requires additional space

Answer: a
Explanation: Auxiliary memory is required for storing the data temporarily.

2. In the following scenarios, when will you use selection sort?
a) The input is already sorted
b) A large file has to be sorted
c) Large values need to be sorted with small keys
d) Small values need to be sorted with large keys

Answer: c
Explanation: Selection is based on keys, hence a file with large values and small keys can be efficiently sorted with selection sort.

3. What is the worst case complexity of selection sort?
a) O(nlogn)
b) O(logn)
c) O(n)
d) $O(n^2)$

Answer: d
Explanation: Selection sort creates a sub-list, LHS of the 'min' element is already sorted and RHS is yet to be sorted. Starting with the first element the 'min' element moves towards the final element.

b)

```
        int min;
        for(int j=0; j<arr.length-1; j++)
        {
                min = j;
                for(int k=j+1; k<=arr.length-1; k++)
                {
                        if(arr[k] < arr[min])
                                min = k;
                }
                int temp = arr[min];
```

```
                arr[min] = arr[j];
                arr[j] = temp;
        }
```

c)

```
        int min;
        for(int j=0; j<arr.length-1; j++)
        {
                min = j;
                for(int k=j+1; k<=arr.length; k++)
                {
                        if(arr[k] < arr[min])
                                min = k;
                }
                int temp = arr[min];
                arr[min] = arr[j];
                arr[j] = temp;
        }
```

d)

```
        int min;
        for(int j=0; j<arr.length-1; j++)
        {
                min = j;
                for(int k=j+1; k<=arr.length-1; k++)
                {
                        if(arr[k] > arr[min])
                                min = k;
                }
                int temp = arr[min];
                arr[min] = arr[j];
                arr[j] = temp;
        }
```

5. What is the advantage of selection sort over other sorting techniques?
a) It requires no additional storage space
b) It is scalable
c) It works best for inputs which are already sorted
d) It is faster than any other sorting technique

Answer: a
Explanation: Since selection sort is an in-place sorting algorithm, it does not require additional storage.

6. What is the average case complexity of selection sort?
a) O(nlogn)
b) O(logn)
c) O(n)
d) $O(n^2)$

Answer: d
Explanation: In the average case, even if the input is partially sorted, selection sort behaves as if the entire array is not sorted. Selection sort is insensitive to input.

7. What is the disadvantage of selection sort?
a) It requires auxiliary memory
b) It is not scalable
c) It can be used for small keys
d) It takes linear time to sort the elements

Answer: b
Explanation: As the input size increases, the performance of selection sort decreases.

8. The given array is arr = {3,4,5,2,1}. The number of iterations in bubble sort and selection sort respectively are _____

a) 5 and 4
b) 4 and 5
c) 2 and 4
d) 2 and 5

Answer: a
Explanation: Since the input array is not sorted, bubble sort takes 5 iterations and selection sort takes 4(n-1) iterations.

9. The given array is arr = {1,2,3,4,5}. (bubble sort is implemented with a flag variable)The number of iterations in selection sort and bubble sort respectively are _____
a) 5 and 4
b) 1 and 4
c) 0 and 4
d) 4 and 1

Answer: d
Explanation: Selection sort is insensitive to input, hence 4(n-1) iterations. Whereas bubble sort iterates only once to set the flag to 0 as the input is already sorted.

10. What is the best case complexity of selection sort?
a) O(nlogn)
b) O(logn)
c) O(n)
d) $O(n^2)$

Answer: d
Explanation: The best, average and worst case complexities of selection sort is $O(n^2)$.
$(n-1) + (n-2) + (n-3) + ... + 1 = (n(n-1))/2 \sim (n^2)/2$.

1. What is an external sorting algorithm?
a) Algorithm that uses tape or disk during the sort
b) Algorithm that uses main memory during the sort
c) Algorithm that involves swapping
d) Algorithm that are considered 'in place'

Answer: a
Explanation: As the name suggests, external sorting algorithm uses external memory like tape or disk.

2. What is an internal sorting algorithm?
a) Algorithm that uses tape or disk during the sort
b) Algorithm that uses main memory during the sort
c) Algorithm that involves swapping
d) Algorithm that are considered 'in place'

Answer: b
Explanation: As the name suggests, internal sorting algorithm uses internal main memory.

3. What is the worst case complexity of bubble sort?
a) O(nlogn)
b) O(logn)
c) O(n)
d) $O(n^2)$

Answer: d
Explanation: Bubble sort works by starting from the first element and swapping the elements if required in each

iteration.

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-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;
                }
        }
}
```

d)

```
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;
                }
        }
}
```

5. What is the average case complexity of bubble sort?
a) O(nlogn)
b) O(logn)
c) O(n)
d) O(n^2)

Answer: d
Explanation: Bubble sort works by starting from the first element and swapping the elements if required in each iteration even in the average case.

6. Which of the following is not an advantage of optimised bubble sort over other sorting techniques in case of sorted elements?
a) It is faster
b) Consumes less memory
c) Detects whether the input is already sorted
d) Consumes less time

Answer: c

Explanation: Optimised Bubble sort is one of the simplest sorting techniques and perhaps the only advantage it has over other techniques is that it can detect whether the input is already sorted. It is faster than other in case of sorted array and consumes less time to describe whether the input array is sorted or not. It consumes same memory than other sorting techniques. Hence it is not an advantage.

7. The given array is arr = {1, 2, 4, 3}. Bubble sort is used to sort the array elements. How many iterations will be done to sort the array?
a) 4
b) 2
c) 1
d) 0

Answer: a
Explanation: Even though the first two elements are already sorted, bubble sort needs 4 iterations to sort the given array.

b)

```
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;
                }
        }
}
```

c)

```
        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;
                        }
                }
        }
```

d)

```
        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;
                        }
                }
        }
```

9. What is the best case efficiency of bubble sort in the improvised version?
a) O(nlogn)
b) O(logn)
c) O(n)
d) $O(n^2)$

Answer: c
Explanation: Some iterations can be skipped if the list is sorted, hence efficiency improves to O(n).

10. The given array is arr = {1,2,4,3}. Bubble sort is used to sort the array elements. How many iterations will be done to sort the array with improvised version?
a) 4
b) 2
c) 1
d) 0

Answer: b
Explanation: Only 2 elements in the given array are not sorted, hence only 2 iterations are required to sort them.

1. Merge sort uses which of the following technique to implement sorting?
a) backtracking
b) greedy algorithm
c) divide and conquer
d) dynamic programming

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) $O(n \log n^2)$

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)

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.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) $O(n \log n^2)$

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

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) $O(n \log n^2)$

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

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

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

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

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

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.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

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.

b)

```
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
    }
}
```

c)

```
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
    }
}
```

d)

```
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);


    }
}
```

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

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.

1. Merge sort uses which of the following algorithm to implement sorting?
a) backtracking
b) greedy algorithm
c) divide and conquer
d) dynamic programming

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) $O(n \log n^2)$

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)

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)

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)

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

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.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.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

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.

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+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++;
                }
        }
}
```

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 && 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+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++;
                }
        }
}
```

1. Merge sort uses which of the following algorithm to implement sorting?
a) backtracking
b) greedy algorithm
c) divide and conquer
d) dynamic programming

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) $O(n \log n^2)$

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)

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)

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)

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).

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

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.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.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.

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.

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 += 2*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 = 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);
                }
        }
}
```

d)

```
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);
                }
        }
}
```

1. Which of the following sorting algorithms is the fastest?
a) Merge sort
b) Quick sort
c) Insertion sort
d) Shell sort

Answer: b
Explanation: Quick sort is the fastest known sorting algorithm because of its highly optimized inner loop.Answer: a
Explanation: In quick sort, the array is divided into sub-arrays and then it is sorted (divide-and-conquer strategy).

3. What is the worst case time complexity of a quick sort algorithm?
a) O(N)
b) O(N log N)
c) O(N^2)
d) O(log N)

Answer: c
Explanation: The worst case performance of a quick sort algorithm is mathematically found to be O(N^2).

4. Which of the following methods is the most effective for picking the pivot element?
a) first element

b) last element
c) median-of-three partitioning
d) random element

Answer: c
Explanation: Median-of-three partitioning is the best method for choosing an appropriate pivot element. Picking a first, last or random element as a pivot is not much effective.Answer: d
Explanation: Left element=8, right element=0,
Centre=[position(left+right)/2]=6.

6. Which is the safest method to choose a pivot element?
a) choosing a random element as pivot
b) choosing the first element as pivot
c) choosing the last element as pivot
d) median-of-three partitioning method

Answer: a
Explanation: This is the safest method to choose the pivot element since it is very unlikely that a random pivot would consistently provide a poor partition.

7. What is the average running time of a quick sort algorithm?
a) $O(N^2)$
b) O(N)
c) O(N log N)
d) O(log N)

Answer: c
Explanation: The best case and average case analysis of a quick sort algorithm are mathematically found to be O(N log N).

8. Which of the following sorting algorithms is used along with quick sort to sort the sub arrays?
a) Merge sort
b) Shell sort
c) Insertion sort
d) Bubble sort

Answer: c
Explanation: Insertion sort is used along with quick sort to sort the sub arrays.
It is used only at the end.Answer: a
Explanation: Quick sort uses join operation since join is a faster operation than merge.

10. How many sub arrays does the quick sort algorithm divide the entire array into?
a) one
b) two
c) three
d) four

Answer: b
Explanation: The entire array is divided into two partitions, 1st sub array containing elements less than the pivot element and 2nd sub array containing elements greater than the pivot element.

11. Which is the worst method of choosing a pivot element?
a) first element as pivot
b) last element as pivot
c) median-of-three partitioning
d) random element as pivot

Answer: a

Explanation: Choosing the first element as pivot is the worst method because if the input is pre-sorted or in reverse order, then the pivot provides a poor partition.

12. Which among the following is the best cut-off range to perform insertion sort within a quick sort?
a) N=0-5
b) N=5-20
c) N=20-30
d) N>30

Answer: b
Explanation: A good cut-off range is anywhere between N=5 and N=20 to avoid nasty degenerate cases.

1. Quick sort is a _____
a) greedy algorithm
b) divide and conquer algorithm
c) dynamic programming algorithm
d) backtracking algorithm

Answer: b
Explanation: Quick sort is a divide and conquer algorithm. Quick sort first partitions a large array into two smaller sub-arrays. And then recursively sorts the sub-arrays.

2. What is the worst case time complexity of the Quick sort?
a) O(nlogn)
b) O(n)
c) O(n^3)
d) O(n^2)

Answer: d
Explanation: The worst case running time for Quick sort is O(n^2). In Quick sort, the worst case behaviour occurs when the partitioning routine produces two sub-arrays one with n – 1 element and other with 0 elements.

3. Apply Quick sort on a given sequence 7 11 14 6 9 4 3 12. What is the sequence after first phase, pivot is first element?
a) 6 4 3 7 11 9 14 12
b) 6 3 4 7 9 14 11 12
c) 7 6 14 11 9 4 3 12
d) 7 6 4 3 9 14 11 12

Answer: b
Explanation: Let's apply Quick sort on the given sequence,
For first phase, pivot = 7

```
7   11   14   6   9   4   3   12
     i                       j
7   11   14   6   9   4   3   12
     i                   j
7   3    14   6   9   4   11   12
         i           j
7   3    4   6   9   14   11   12
             i   j
7   3    4   6   9   14   11   12
             j   i
6   3    4   7   9   14   11   12
```

4. The best case behaviour occurs for quick sort is, if partition splits the array of size n into _____
a) n/2 : (n/2) – 1
b) n/2 : n/3

c) n/4 : 3n/2
d) n/4 : 3n/4

Answer: a
Explanation: The best case analysis of quick sort occurs when the partition splits the array into two subarrays, each of size no more than n/2 since one is of size n/2 and one of size (n/2) – 1.Answer: b
Explanation: In stable sorting algorithm the records with equal keys appear in the same order in the sorted sequence as they appear in the input unsorted sequence. Quick sort does not preserve the relative order of equal sort items. Therefore, Quick sort is not a stable sort.

6. Consider the Quick sort algorithm in which the partitioning procedure splits elements into two sub-arrays and each sub-array contains at least one-fourth of the elements. Let T(n) be the number of comparisons required to sort array of n elements. Then T(n)<=?
a) T(n) <= 2 T(n/4) + cn
b) T(n) <= T(n/4) + T(3n/4) + cn
c) T(n) <= 2 T(3n/4) + cn
d) T(n) <= T(n/3) + T(3n/4) + cn

Answer: b
Explanation: If there are n/4 elements in one sub-array then T(n/4) comparisons are needed for this sub-array. And T(3n/4) comparison are required for the rest 4n/5 elements, and cn is time required for finding the pivot. If there are more than n/4 elements in one sub-array then other sub-array will have less than 3n/4 elements and time complexity will be less than T(n/4) + T(3n/4) + cn.

7. Consider the Quick sort algorithm which sorts elements in ascending order using the first element as pivot. Then which of the following input sequence will require a maximum number of comparisons when this algorithm is applied on it?
a) 22 25 56 67 89
b) 52 25 76 67 89
c) 22 25 76 67 50
d) 52 25 89 67 76

Answer: a
Explanation: If the input sequence is already sorted then worst case behaviour occurs for the Quick sort algorithm which use the first element as pivot. Therefore, the input sequence given in 22 25 56 67 89 will require a maximum number of comparisons.

8. A machine needs a minimum of 200 sec to sort 1000 elements by Quick sort. The minimum time needed to sort 200 elements will be approximately _____
a) 60.2 sec
b) 45.54 sec
c) 31.11 sec
d) 20 sec

Answer: c
*Explanation: The Quick sort requires nlog2n comparisons in best case, where n is size of input array. So, 1000 * log21000 ≈ 9000 comparisons are required to sort 1000 elements, which takes 200 sec. To sort 200 elements minimum of 200 * log2200 ≈ 1400 comparisons are required. This will take 200 * 1400 / 9000 ≈ 31.11 sec.*

9. Which one of the following sorting algorithm is best suited to sort an array of 1 million elements?
a) Bubble sort
b) Insertion sort
c) Merge sort
d) Quick sort

Answer: d
Explanation: The Quick sort is best suited to sort the array of 1 million elements. The practical implementations of Quick

sort use randomised version. In practice randomised Quick sort algorithms rarely shows worst case behaviour and is almost always O(nlogn). And Quick sort requires little additional space and exhibits good cache locality.

10. Quick sort is a space-optimised version of ____
a) Bubble sort
b) Selection sort
c) Insertion sort
d) Binary tree sort

Answer: d
Explanation: Quick sort is a space-optimised version of the binary tree sort. In binary sort tree, the elements are inserted sequentially into the binary search tree and Quick sort organises elements into a tree that is implied by the recursive calls.

1. QuickSort can be categorized into which of the following?
a) Brute Force technique
b) Divide and conquer
c) Greedy algorithm
d) Dynamic programming

Answer: b
Explanation: First you divide(partition) the array based on the pivot element and sort accordingly.

b)

```
public static void quickSort(int[] arr, int low, int high)
{
        int pivot;
        if(high>low)
        {
                pivot = partition(arr, low, high);
                quickSort(arr, low, pivot-1);
                quickSort(arr, pivot+1, high);
        }
}
```

c)

```
public static void quickSort(int[] arr, int low, int high)
{
        int pivot;
        if(high<low)
        {
                pivot = partition(arr, low, high);
                quickSort(arr, low, pivot-1);
                quickSort(arr, pivot+1, high);
        }
}
```

d)

```
public static void quickSort(int[] arr, int low, int high)
{
        int pivot;
        if(high>low)
        {
                pivot = partition(arr, low, high);
                quickSort(arr, low, pivot);
                quickSort(arr, pivot, high);
        }
}
```

3. What is the worst case complexity of QuickSort?
a) O(nlogn)
b) O(logn)

c) O(n)
d) O(n^2)

Answer: d
Explanation: When the input array is already sorted.

4. What is a randomized QuickSort?
a) The leftmost element is chosen as the pivot
b) The rightmost element is chosen as the pivot
c) Any element in the array is chosen as the pivot
d) A random number is generated which is used as the pivot

Answer: c
Explanation: QuickSort is randomized by placing the input data in the randomized fashion in the array or by choosing a random element in the array as a pivot.

b)

```
public static void quickSort(int[] arr, int low, int high)
{
        int pivot;
        if(high>low)
        {
                pivot = partition(arr, low, high);
                quickSort(arr, low, pivot);
                quickSort(arr, pivot+2, high);
        }
}
```

c)

```
private static int partition(int[] arr, int low, int high)
{
        int left, right, pivot_item = arr[low];
        left = low;
        right = high;
        while(left > right)
        {
                while(arr[left] <= pivot_item)
                {
                        left++;
                }
                while(arr[right] > pivot_item)
                {
                        right--;
                }
                if(left < right)
                {
                        swap(arr, left, right);
                }
        }
        arr[low] = arr[right];
        arr[right] = pivot_item;
        return right;
}
```

d)

```
private static int partition(int[] arr, int low, int high)
{
        int left, right, pivot_item = arr[low];
        left = low;
        right = high;
        while(left <= right)
        {
                while(arr[left] <= pivot_item)
```

```
                {
                        left++;
                }
                while(arr[right] > pivot_item)
                {
                        right--;
                }
                if(left < right)
                {
                        swap(arr, left, right);
                }
        }
        arr[low] = arr[right];
        arr[right] = pivot_item;
        return right;
}
```

6. What is the best case complexity of QuickSort?
a) O(nlogn)
b) O(logn)
c) O(n)
d) O(n^2)

Answer: a
Explanation: The array is partitioned into equal halves, using the Divide and Conquer master theorem, the complexity is found to be O(nlogn).

7. The given array is arr = {2,3,4,1,6}. What are the pivots that are returned as a result of subsequent partitioning?
a) 1 and 3
b) 3 and 1
c) 2 and 6
d) 6 and 2

Answer: a
Explanation: The call to partition returns 1 and 3 as the pivot elements.

8. What is the average case complexity of QuickSort?
a) O(nlogn)
b) O(logn)
c) O(n)
d) O(n^2)

Answer: a
Explanation: The position of partition(split) is unknown, hence all(n) possibilities are considered, the average is found by adding all and dividing by n.

9. The given array is arr = {2,6,1}. What are the pivots that are returned as a result of subsequent partitioning?
a) 1 and 6
b) 6 and 1
c) 2 and 6
d) 1

Answer: d
Explanation: There is only one pivot with which the array will be sorted, the pivot is 1.

10. Which of the following is not true about QuickSort?
a) in-place algorithm
b) pivot position can be changed
c) adaptive sorting algorithm
d) can be implemented as a stable sort

Answer: b
Explanation: Once a pivot is chosen, its position is finalized in the sorted array, it cannot be modified.

1. Quick sort uses which of the following algorithm to implement sorting?
a) backtracking
b) greedy algorithm
c) divide and conquer
d) dynamic programming

Answer: c
Explanation: Quick sort uses the technique of divide and conquer in order to sort a given array. It divides the array into two parts about the pivot and then apply a quick sort to both the parts.

2. What is a randomized quick sort?
a) quick sort with random partitions
b) quick sort with random choice of pivot
c) quick sort with random output
d) quick sort with random input

Answer: b
Explanation: Randomized quick sort chooses a random element as a pivot. It is done so as to avoid the worst case of quick sort in which the input array is already sorted.

3. What is the purpose of using randomized quick sort over standard quick sort?
a) so as to avoid worst case time complexity
b) so as to avoid worst case space complexity
c) to improve accuracy of output
d) to improve average case time complexity

Answer: a
Explanation: Randomized quick sort helps in avoiding the worst case time complexity of O(n2) which occurs in case when the input array is already sorted. However the average case and best case time complexities remain unaltered.

4. What is the auxiliary space complexity of randomized quick sort?
a) O(1)
b) O(n)
c) O(log n)
d) O(n log n)

Answer: c
Explanation: Auxiliary space complexity of randomized quick 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 randomized quick sort?
a) O(n log n)
b) $O(n^2)$
c) $O(n^2 \log n)$
d) $O(n \log n^2)$

Answer: a
Explanation: The average case time complexity of randomized quick sort is same as that of standard quick sort as randomized quick sort only helps in preventing the worst case. It is equal to O(n log n).

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

c) selection
d) exchanging

Answer: b
Explanation: Quick sort makes partitions of the input array about the pivot in order to implement sorting. Thus its method of sorting is called partitioning.Answer: a
Explanation: In-place algorithms requires constant or very less auxiliary space. Quick sort qualifies as an in place sorting algorithm as it has a very low auxiliary space requirement of O(log n).Answer: b
Explanation: Randomized quick sort like standard quick sort is also not a stable sorting algorithm. It is because the elements with the same values are not guaranteed to appear in the same relative order in the output sorted array.

9. What is the best case time complexity randomized quick sort?
a) O(log n)
b) O(n log n)
c) O(n^2)
d) O(n^2 log n)

Answer: b
Explanation: Best case time complexity is given in the case when there is equal partitioning of the array about the pivot. It is given by the relation T(n) = 2T(n/2) + n which gives the result O(n log n).

10. Which of the following is incorrect about randomized quicksort?
a) it has the same time complexity as standard quick sort
b) it has the same space complexity as standard quick sort
c) it is an in-place sorting algorithm
d) it cannot have a time complexity of O(n^2) in any case.

Answer: d
Explanation: Randomized quick sort prevents the worst case complexity of O(n^2) in most of the cases. But in some rare cases the time complexity can become O(n^2). The probability of such a case is however very low.

b)

```
void partition_random(int arr[], int low, int high)
{
    srand(time(NULL));
    int random = low + rand() % (high - low);
    swap(arr[random], arr[high]);
}
```

c)

```
void partition_random(int arr[], int low, int high)
{
    srand(time(NULL));
    int random = high + rand() % (high - low);
    swap(arr[random], arr[high]);
}
```

d)

```
void partition_random(int arr[], int low, int high)
{
    srand(1);
    int random = low + rand() % (high - low);
    swap(arr[random], arr[high]);
}
```

12. What is the worst case time complexity of randomized quicksort?
a) O(n)
b) O(n log n)

c) $O(n^2)$
d) $O(n^2 \log n)$

Answer: c
Explanation: Randomized quicksort prevents the worst case complexity of $O(n^2)$ in most of the cases. But in some rare cases the time complexity can become $O(n^2)$. The probability of such a case is however very low.

1. Quick sort uses which of the following algorithm to implement sorting?
a) backtracking
b) greedy algorithm
c) divide and conquer
d) dynamic programming

Answer: c
Explanation: Quick sort uses the technique of divide and conquer in order to sort a given array. It divides the array into two parts about the pivot and then applies quick sort to both the parts.

2. What is the median of three techniques in quick sort?
a) quick sort with random partitions
b) quick sort with random choice of pivot
c) choosing median element as pivot
d) choosing median of first, last and middle element as pivot

Answer: d
Explanation: In the median of three technique the median of first, last and middle element is chosen as the pivot. It is done so as to avoid the worst case of quick sort in which the time complexity shoots to $O(n^2)$.

3. What is the purpose of using a median of three quick sort over standard quick sort?
a) so as to avoid worst case time complexity
b) so as to avoid worst case space complexity
c) to improve accuracy of output
d) to improve average case time complexity

Answer: a
Explanation: Median of three quick sort helps in avoiding the worst case time complexity of $O(n^2)$ which occurs in case when the input array is already sorted. However, the average case and best case time complexities remain unaltered.

4. What is the auxiliary space complexity of a median of three quick sort?
a) O(1)
b) O(n)
c) O(log n)
d) O(n log n)

Answer: c
Explanation: Auxiliary space complexity of median of three quick 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 the median of three quick sort?
a) O(n log n)
b) $O(n^2)$
c) $O(n^2 \log n)$
d) $O(n \log n^2)$

Answer: a
Explanation: The average case time complexity of median of three quick sort is the same as that of a standard quick sort

as randomized quick sort only helps in preventing the worst case. It is equal to O(n log n).

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

Answer: b
Explanation: Quick sort makes partitions of the input array about the pivot in order to implement sorting. Thus its method of sorting is called partitioning.Answer: a
Explanation: In-place algorithms require constant or very less auxiliary space. Median of three quick sort qualifies as an in-place sorting algorithm. It has a very low auxiliary space requirement of O(log n).Answer: b
Explanation: Median of three quick sort like standard quick sort is also not a stable sorting algorithm. It is because the elements with the same values are not guaranteed to appear in the same relative order in the output sorted array.

9. What is the best case time complexity Median of three quick sort?
a) O(log n)
b) O(n log n)
c) O(n^2)
d) O(n^2 log n)

Answer: b
Explanation: Best case time complexity is given in the case when there is equal partitioning of the array about the pivot. It is given by the relation T(n) = 2T(n/2) + n which gives the result O(n log n).

b)

```
int Median(arr, left, right)
{
    int mid;
    mid = (left + right)/2
    if (arr[right] < arr[left]);
        Swap(arr, left, right); //to swap arr[left],arr[right]
    if (arr[mid] < arr[left]);
        Swap(arr, mid, left);//to swap arr[left],arr[mid]
    if (arr[right] < arr[mid]);
        Swap(arr, right, mid);// to swap arr[right],arr[mid]
    return mid;
}
```

c)

```
int Median(arr, left, right)
{
    int mid;
    mid = (left + right)/2
    if (arr[right] > arr[left]);
        Swap(arr, left, right); //to swap arr[left],arr[right]
    if (arr[mid] < arr[left]);
        Swap(arr, mid, left);//to swap arr[left],arr[mid]
    if (arr[right] < arr[mid]);
        Swap(arr, right, mid);// to swap arr[right],arr[mid]
    return mid;
}
```

d)

```
int Median(arr, left, right)
{
    int mid;
    mid = (left + right)/2
    if (arr[left] < arr[right]);
```

```
        Swap(arr, left, right); //to swap arr[left],arr[right]
    if (arr[left] < arr[mid]);
        Swap(arr, mid, left);//to swap arr[left],arr[mid]
    if (arr[right] < arr[mid]);
        Swap(arr, right, mid);// to swap arr[right],arr[mid]
    return mid;
}
```

11. What will be the pivot for the array arr={8,2,4,9} for making the first partition when a median of three quick sort is implemented?
a) 8
b) 2
c) 4
d) 9

Answer: a
Explanation: While making the first partition the first, last and middle elements will be 8, 9 and 2 respectively. Thus the median element will be 8.

1. What is the other name for a shell sort algorithm?
a) Diminishing increment sort
b) Diminishing decrement sort
c) Insertion sort
d) Selection sort

Answer: a
Explanation: The other name for a shell sort algorithm is diminishing decrement sort as the distance between comparisons decreases as the algorithm runs until the last phase.

2. The worst case running time of shell sort, using Shell's increments is?
a) O(N)
b) O(N log N)
c) O(log N)
d) $O(N^2)$

Answer: d
Explanation: The lower bound of a shell sort algorithm is mathematically found to be $O(N^2)$.

3. Who invented the shell sort algorithm?
a) John Von Neumann
b) Donald Shell
c) Tony Hoare
d) Alan Shell

Answer: b
Explanation: Shell sort algorithm is invented by Donald shell. Merge sort is invented by John Von Neumann. Quick sort is invented by Tony Hoare.Answer: a
Explanation: Shell sort broke the quadratic time barrier as it works by comparing elements that are distant.

5. Shell sort algorithm is an example of?
a) External sorting
b) Internal sorting
c) In-place sorting
d) Bottom-up sorting

Answer: b
Explanation: Shell sort is an example of internal sorting because sorting of elements is done internally using an array.Answer: a
Explanation: Shell sort uses an increment sequence h1, h2, h3… and this sequence will work as long as h1=1.

7. Which of the following sorting algorithms is closely related to shell sort?
a) Selection sort
b) Merge sort
c) Insertion sort
d) Bucket sort

Answer: c
Explanation: Shell sort performs an insertion sort on hk independent arrays. It is mainly a variation of insertion sort.

8. Why is Shell sort called as a generalization of Insertion sort?
a) Shell sort allows an exchange of far items whereas insertion sort moves elements by one position
b) Improved lower bound analysis
c) Insertion is more efficient than any other algorithms
d) Shell sort performs internal sorting

Answer: a
Explanation: Shell sort is an extension of insertion sort because it swaps elements at far distances and at a faster rate.Answer: c
Explanation: The general strategy to hk sort is for each position, i, in hk,, hk+1,...., N-1, place the element in the correct spot among i, i-hk,i-2hk, etc.

10. Which of the following statements is the basic for loop for a shell sort algorithm?
a) for(increment=N/2;increment>0;increment/=2)
b) for(i=1;i<n;i++)
c) for(i=n/2;i>=0;i- -)
d) for(i=0;i< n;i++;numelements- -)

Answer: a
Explanation: for(increment=N/2;increment>0;increment/=2) represents shell sort, for(i=1;i<n;i++) represents insertion sort, for(i=n/2;i>=0;I- -) represents heap sort, for(i=0;i<n;i++;numelements- -) merge sort.

11. On how many increment sequences does the worst case analysis of shell sort depends?
a) one
b) two
c) three
d) four

Answer: c
Explanation: The worst case analysis of shell sort depends on two increment sequences- using Shell's increments, Sedgewick's and Hibbard's increments.

12. What is the worst case running time of shell sort using Hibbard's increments?
a) O(N)
b) $O(N^2)$
c) $O(N^{1/2})$
d) $O(N^{3/2})$

Answer: d
Explanation: Mathematically, the lower bound analysis for shell sort using Hibbard's increments is $O(N^{3/2})$.

13. What is the general form of Shell's increments?
a) 1,2,3,…,n
b) 1,3,7,….,2k-1
c) 1,3,5,7,….,k-1
d) 1,5,10,15,…, k-1

Answer: b

Explanation: Shell's increments are of the form 1,3,7,….,2k-1. The key difference is that the consecutive elements have no common factors.

14. What is the worst case analysis of shell sort using Shell's increments?
a) O(N)
b) $O(N^2)$
c) $O(N^{1/2})$
d) $O(N^{3/2})$

Answer: b
Explanation: The worst case analysis is mathematically found to be $O(N^2)$. The proof is rather complicated.

15. What is the worst case analysis of Shell sort using Sedgewick's increments?
a) $O(N^2)$
b) $O(N^{3/2})$
c) $O(N^{4/3})$
d) $O(N^{5/4})$

Answer: c
Explanation: The worst case analysis of Shell sort using Sedgewick's increments is mathematically calculated to be $O(N^{4/3})$.

1. Shell sort is also known as _____
a) diminishing decrement sort
b) diminishing increment sort
c) partition exchange sort
d) diminishing insertion sort

Answer: b
Explanation: Shell sort is also known as diminishing increment sort since each pass is defined by an increment h such that only the records which are h units apart will be sorted.Answer: b
Explanation: In Shell sort, the relative order of elements with equal values may change. Therefore, it is not a stable sorting algorithm. Shell sort is an in-place sorting algorithm as it requires O(1) auxiliary space.

3. Shell sort is applied on the elements 27 59 49 37 15 90 81 39 and the chosen decreasing sequence of increments is (5,3,1). The result after the first iteration will be
a) 27 59 49 37 15 90 81 39
b) 27 59 37 49 15 90 81 39
c) 27 59 39 37 15 90 81 49
d) 15 59 49 37 27 90 81 39

Answer: c
Explanation: Given elements 27 59 49 37 15 90 81 39,
First Iteration (span = 5):

So, the sequence after first iteration will be, 27 59 39 37 15 90 81 49.

4. Consider the following code snippet, which implements the Shell sort algorithm.

```
shellSort( int elements[], int num_elements , int incrmnts[], int num_incrmnts)
{
```

```
int incr, j, k, span, y;
for(incr = 0; incr ;&lt num_incrmnts; incr++)
{
        span = incrmnts[incr]; data-structure-questions-answers-shell-sort
        for( j = span; j &lt; num_elements; j++)
        {
                k = j;
                y = elements[j];
                while (________ )
                {
                        elements [ k]  = elements[k - span];
                        k = k - span;
                }
                elements[k] = y;
        }
}
```

Which condition will correctly implement the while loop?
a) k >= j && y < elements[k- span]
b) k >= span || y < elements[k + span]
c) k >= j || y < elements[k + span]
d) k >= span && y < elements[k- span]

Answer: d
Explanation: In Shell sort, for increment = h we sort the sub-arrays that start at arbitrary element and include every hth element.
So, if h = 4 the algorithms sorts:
Sub-array formed with elements at positions 1, 5, 9, 13 … in original array
Sub-array formed with elements at positions 2, 6, 10, 14 … in original array
Sub-array formed with elements at positions 3, 7, 11, 15 … in original array
Sub-array formed with elements at positions 4, 8, 12, 16 … in original array
Therefore, the condition given in option k >= span && y < elements[k- span] will implement while loop correctly.

5. Shell sort is an improvement on ____
a) insertion sort
b) selection sort
c) binary tree sort
d) quick sort

Answer: a
Explanation: Shell sort is an improvement on insertion sort that allows the exchange of elements that are far apart. Shell sort algorithm sorts separate sub-arrays of the original input array. These sub-arrays contains every hth element of the original array.

6. An array that is first 7-sorted, then 5-sorted becomes _____
a) 7-ordered
b) 5-ordered
c) both 2-ordered and 5-ordered
d) both 7-ordered and 5-ordered

Answer: d
Explanation: An array that is 7-sorted, becomes 7-ordered. And an array that is 5-sorted, becomes 5-ordered. If k-ordered array is h-sorted, it remains k-ordered. Thus, an array that is first 7-sorted, then 5-sorted becomes both 7-ordered and 5-ordered.

7. If Hibbard increments (h1= 1, h2= 3, h3= 7, …, hk = 2^k–1) are used in a Shell sort implementation, then the best case time complexity will be _____
a) O(nlogn)
b) O(n)
c) O(n^2)

d) O(logn)

Answer: a
Explanation: The best case occurs when the array is already sorted. In best case the number of comparison for each of the increments-based insertion sorts is equal to length of array.
*Here 2k –1 < n, where n is the number of records. So k < log(n+1), thus the sorting time in best case is less the n * log(n+1). Therefore best case time complexity is O(nlogn).*

8. Records R1, R2, R3,.. RN with keys K1, K2, K3,.. KN are said to be h-ordered, if _____
a) Ki <= Ki+h for 1<= i*h <= N
b) Kh <= Ki+h for 1<= i <= N
c) Ki <= Kh for 1<= i <= h
d) Ki <= Ki+h for 1<= i <= N-h

Answer: d
Explanation: Records are h-ordered if every hth element (starting anywhere) yields a sorted array. Therefore, given records with keys K1, K2, K3,.. KN are said to be h-ordered, if Ki <= Ki+h for 1<= i <= N-h.Answer: b
Explanation: Insertion sort is more efficient than Shell sort if the length of array is small because insertion sort is quite simple to program and involve very few actions other than comparisons and replacements on each pass.

10. Which of the following is true?
a) Shell sort's passes completely sort the elements before going on to the next-smallest gap while Comb sort's passes do not completely sort the elements
b) Shell sort's passes do not completely sort the elements before going on to the next-smallest gap like in Comb sort
c) Comb sort's passes completely sort the elements before going on to the next-smallest gap like in Shell sort
d) Shell sort's passes do not completely sort the elements before going on to the next-smallest gap while Comb sort's passes completely sort the elements

Answer: a
Explanation: Both Shell sort and Comb sort have repeated sorting passes with decreasing gaps. Unlike Comb sort, in Shell sort the array is sorted completely in each pass before going on to the next-smallest gap.

1. On which algorithm is heap sort based on?
a) Fibonacci heap
b) Binary tree
c) Priority queue
d) FIFO

Answer: c
Explanation: Heap sort is based on the algorithm of priority queue and it gives the best sorting time.

2. In what time can a binary heap be built?
a) O(N)
b) O(N log N)
c) O(log N)
d) $O(N^2)$

Answer: a
Explanation: The basic strategy is to build a binary heap of N elements which takes O(N) time.Answer: b
Explanation: Heap sort is slower than Shell sort because Shell sort uses Sedgewick's increment sequence.Answer: d
Explanation: Constructing a max heap using the elements 97,53,59,26,41,58,31 will cause the heap to look like that.

5. In what position does the array for heap sort contains data?
a) 0
b) 1
c) -1
d) anywhere in the array

Answer: a
Explanation: The array for heap sort contains data at position 0 whereas for a binary heap, array begins at 1. This is the reason for its complexity.

6. In heap sort, after deleting the last minimum element, the array will contain elements in?
a) increasing sorting order
b) decreasing sorting order
c) tree inorder
d) tree preorder

Answer: b
Explanation: By logic, after deleting minimum element, the heap will contain elements in decreasing sorting order. We can change this by altering the ordering property.

7. What is the typical running time of a heap sort algorithm?
a) O(N)
b) O(N log N)
c) O(log N)
d) $O(N^2)$

Answer: b
Explanation: The total running time of a heap sort algorithm is mathematically found to be O(N log N).

8. How many arrays are required to perform deletion operation in a heap?
a) 1
b) 2
c) 3
d) 4

Answer: b
Explanation: To perform deletion operation in a heap, we require 2 arrays and that occupies extra memory space and hence increase in running time.

9. What is the time taken to perform a delete min operation?
a) O(N)
b) O(N log N)
c) O(log N)
d) $O(N^2)$

Answer: c
Explanation: The time taken to perform a deletion of a minimum element is mathematically found to be O(log N).Answer: a
Explanation: Heap sort uses fewer comparisons than other sorting algorithms and hence it is an extremely stable algorithm.

11. What is the average number of comparisons used in a heap sort algorithm?
a) N log N-O(N)
b) O(N log N)-O(N)
c) O(N log N)-1
d) 2N log N + O(N)

Answer: d
Explanation: The average number of comparisons in a heapsort algorithm is mathematically found to be 2N log N + O(N).

12. What is the time taken to copy elements to and from two arrays created for deletion?
a) O(N)
b) O(N log N)

c) O(log N)
d) O(N^2)

Answer: a
Explanation: The time taken to copy elements to and from the main array and extra array is found to be O(N).

13. What is the average number of comparisons used to heap sort a random permutation of N distinct items?
a) 2N log N-O(N)
b) 2N log N-O(N log N)
c) 2N log N-O(N log log N)
d) 2N log N-O(log N)

Answer: c
Explanation: According to a theorem, the average number of comparisons used to heap sort a random permutation of N distinct items is found to be 2N log N-O(N log log N).

1. Heap sort is an implementation of _____ using a descending priority queue.
a) insertion sort
b) selection sort
c) bubble sort
d) merge sort

Answer: b
Explanation: Heap sort is an implementation of selection sort using the input array as a heap representing a descending priority queue. Heap sort algorithm is divided into two phase. In first phase the max-heap is created and the second phase (selection phase) deletes the elements from the priority queue using siftdown operation.

2. Which one of the following is false?
a) Heap sort is an in-place algorithm
b) Heap sort has O(nlogn) average case time complexity
c) Heap sort is stable sort
d) Heap sort is a comparison-based sorting algorithm

Answer: c
Explanation: Heap sort is a comparison based sorting algorithm and has time complexity O(nlogn) in the average case. Heap sort is an in-place algorithm as it needs O(1) of auxiliary space. Heap sort uses heap and operations on heap can change the relative order of items with the same key values. Therefore, Heap sort is not a stable sort.Answer: d
Explanation: In max-heap element at each node is smaller than or equal to the element at its parent node. On applying the heapify procedure on item at position 2, it will be in position 9 as shown below.

4. The descending heap property is _____
a) A[Parent(i)] = A[i]
b) A[Parent(i)] <= A[i]
c) A[Parent(i)] >= A[i]
d) A[Parent(i)] > 2 * A[i]

Answer: c
Explanation: The max-heap is also known as descending heap. Max-heap of size n is an almost complete binary tree of n nodes such that the element at each node is less than or equal to the element at its parent node.

5. What is its wort case time complexity of Heap sort?
a) O(nlogn)
b) O(n^2logn)
c) O(n^2)
d) O(n^3)

Answer: a
Explanation: In Heap sort, the call to procedure build_Max-heap takes O(n) time and each of O(n) calls to the function max_Heapify takes O(logn) time. So the worst case complexity of Heap sort is O(nlogn).Answer: b
Explanation: Quick sort is more efficient than Heap sort because experiments indicate that Heap sort requires twice as much time as Quick sort for randomly sorted input.

7. Choose the correct option to fill? X so that the code given below implements the Heap sort.

```
#include <stdio.h>
void heapify(int arr[], int n, int i)
{
    int largest = i; // Initialize largest as root
    int l = 2*i + 1; // left = 2*i + 1
    int r = 2*i + 2; // right = 2*i + 2
    if (l < n && arr[l] > arr[largest])
        largest = l;
```

```
    if (r < n && arr[r] > arr[largest])
        largest = r;
    if (largest != i)
    {
        swap(arr[i], arr[largest]);
        heapify(arr, n, largest);
    }
}
void heapSort(int arr[], int n)
{
    for (int i = n / 2 - 1; i >= 0; i--)
        heapify(arr, n, i);
    for (int i=n-1; i>=0; i--)
    {
        X;
        heapify(arr, i, 0);
    }
}
void printArray(int arr[], int n)
{
    for (int i=0; i<n; ++i)
        printf("%d",arr[i]);
    printf("\n");
}
int main()
{
    int arr[] = {12, 11, 13, 5, 6, 7};
    int n = sizeof(arr)/sizeof(arr[0]);
    heapSort(arr, n);
    printf("Sorted array is \n");
    printArray(arr, n);
}
```

a) swap(arr[0], arr[n])
b) swap(arr[i], arr[n])
c) swap(arr[0], arr[i])
d) swap(arr[i], arr[2*i])

Answer: c
Explanation: Steps in heap sort are : (i) Build the max-heap, (ii) Swap the root element with the last element of the heap, (iii) Reduce the size of heap by 1 and heapify the root element, (iv) Repeat the steps form step number (v) until all the elements are sorted. Therefore the correct option is swap(arr[0], arr[i]).

8. Which one of the following is a variation of Heap sort?
a) Comb sort
b) Smooth sort
c) Binary tree sort
d) Shell sort

Answer: b
Explanation: Smooth sort is a variation of Heap sort. Smooth sort has O(nlogn) worst case time complexity like Heap sort. But Smooth sort takes O(n) time to sort the nearly sorted input array.

9. Introsort algorithm is combination of _____
a) Quick sort and Heap sort
b) Quick sort and Shell sort
c) Heap sort and Merge sort
d) Heap sort and insertion sort

Answer: a
Explanation: Introsort is a hybrid sorting algorithm that combines Quick sort and Heap sort to retain advantages of both. It has worst case speed of Heap sort and average case speed of Quick sort.

10. How many elements can be sorted in O(logn) time using Heap sort?
a) O(1)
b) O(n/2)
c) O(logn/log(logn))
d) O(logn)

Answer: c
Explanation: The time complexity of Heap sort is O(klogk) for k input elements,
for k = logn/log(logn),
*O(klogk) = O(logn/log(logn) * log(logn/log(logn)))*
*∴ O(klogk) = O(logn/log(logn) * (log(logn) – log(log(logn))))*
= O(logn)
Hence the correct option is O(logn/log(logn)).

1. Which of the following sorting algorithm is used by C++ internally?
a) quicksort
b) introsort
c) merge sort
d) heap sort

Answer: b
Explanation: Introsort is the in built sorting algorithm used by C++. It is an example of a hybrid sorting algorithm which means it uses more than one sorting algorithm as a routine.

2. Which of the following sorting algorithm is not a constituent of introsort?
a) selection sort
b) quicksort
c) insertion sort
d) heap sort

Answer: a
Explanation: Introsort is a hybrid sorting algorithm which means it uses more than one sorting algorithm as a routine. It may use quick sort or heap sort or insertion sort depending on the given situation.

3. Introsort begins sorting the given array by using which of the following sorting algorithm?
a) selection sort
b) quick sort
c) insertion sort
d) heap sort

Answer: b
Explanation: Introsort begins sorting any given array by using quick sort. Then it may switch to heap sort or insertion sort or may stick to quick sort depending upon the size of the partition.

4. Which of the following sorting algorithm is NOT stable?
a) Introsort
b) Brick sort
c) Bubble sort
d) Merge sort

Answer: a
Explanation: Out of the given options introsort is the only algorithm which is not stable. As it may use quick sort in some case to perform sorting which is itself not stable. Thus stability of introsort is not guaranteed.

5. Which of the following sorting algorithm is in-place?
a) intro sort
b) merge sort
c) counting sort

d) radix sort

Answer: a
Explanation: Introsort may use quick sort or heap sort or insertion sort internally in order to sort the given input. All of the three algorithms are in place, thus making introsort to be an in-place sorting algorithm.Answer: a
Explanation: Quicksort, heap sort and insertion sort are comparison based sorts. Thus overall introsort also becomes a comparison based sort.

7. What is the best case time complexity of introsort?
a) O(n)
b) O(n log n)
c) O(n^2)
d) O(log n)

Answer: b
Explanation: Introsort is mainly governed by heap sort and quick sort. As the best case of both heap sort and quick sort is O(n log n) so the best case of introsort also becomes O(n log n).

8. What is the worst case time complexity of introsort?
a) O(n)
b) O(n log n)
c) O(n^2)
d) O(log n)

Answer: b
Explanation: Worst case time complexity of quicksort is avoided when we implement introsort. Introsort switches to heap sort when there is a possibility of crossing the maximum depth limit.

9. What is the average time complexity of introsort?
a) O(n)
b) O(n log n)
c) O(n^2)
d) O(log n)

Answer: b
Explanation: Average time complexity of introsort remains to be O(n log n) as for most of the cases quick sort and heap sort are used which have O(n log n) time complexity for an average case.

10. What is the auxiliary space requirement of introsort?
a) O(n)
b) O(n log n)
c) O(n^2)
d) O(log n)

Answer: d
Explanation: Introsort is a hybrid of quick sort, heap sort and insertion sort. So like quick sort it may use O(log n) auxiliary space in the stack to store call statements.

11. Why is heap sort preferred over merge sort for introsort implementation?
a) Because heap sort is faster
b) Because heap sort requires less space
c) Because heap sort is easy to implement
d) Because heap sort is easy to understand

Answer: b
Explanation: Both heap sort and merge sort have the same time complexity. But heap sort is an in-place sorting algorithm whereas merge sort requires O(n) auxiliary space which makes heap sort a more preferable option.

12. Why is insertion sort preferred over other sorting algorithms (like selection sort, bubble sort etc.) for introsort implementation?
a) Because insertion sort is faster and adaptive
b) Because insertion sort requires less space
c) Because insertion sort is easy to implement
d) Because insertion sort is easy to understand

Answer: a
Explanation: When small arrays need to be sorted then insertion sort proves to be the best choice. Also it is adaptive so it performs better than others when the given array is fully/partially sorted.

13. What is the cut-off for switching from quick sort to insertion sort in the implementation of introsort?
a) 4
b) 8
c) 16
d) 32

Answer: c
Explanation: When small arrays needs to be sorted then insertion sort proves to be the best choice. So when the size of the partition is less than 16 introsort switches to insertion sort. This particular value has been deduced experimentally.

14. What is the cut-off for switching from quick sort to heap sort in the implementation of introsort?
a) 16
b) n^2
c) n log(n)
d) 2 log (n)

Answer: d
Explanation: Quicksort has a worst case time complexity of $O(n^2)$ which is not preferable. So in order to avoid worst case of quicksort, introsort switches to heap sort when the depth is greater than 2 log(n). This particular value has been deduced experimentally.

15. Which of the following sorting algorithm will be preferred when the size of partition is between 16 and 2 log(n) while implementing introsort?
a) quick sort
b) insertion sort
c) heap sort
d) merge sort

Answer: a
Explanation: Quicksort proves to be the best sorting algorithm for mid sized arrays as it has low space and time complexity. Thus quick sort is preferred when size of partition is between 16 and 2 log(n).

16. What will be the output of the given C++ code?

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
    int arr[] = {1,3,4,2,5};
    int n = sizeof(arr)/sizeof(arr[0]);
    sort(arr, arr+n, greater<int>());
    int a;
    for (a = 0; a < n; a++)
        cout << arr[a] << " ";
    return 0;
}
```

a) 1 2 3 4 5
b) 1 3 4 2 5

c) 5 4 3 2 1
d) error

Answer: c
Explanation: The given program sorts the input in descending order. It is due to the third parameter i.e. greater() which is passed to the function sort().

17. What will be the output of the given C++ code?

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
    int arr[] = {1, 3,4,2,5};
    int n = sizeof(arr)/sizeof(arr[0]);
    sort(arr, arr+n);
    int a;
    for ( a = 0; a< n; a++)
        cout << arr[a] << " ";
    return 0;
}
```

a) 1 2 3 4 5
b) 1 3 4 2 5
c) 5 4 3 2 1
d) error

Answer: a
Explanation: The given program sorts the input in ascending order. Function sort() uses two parameters in the form of address of the first and last element of the array to sort the array.

18. What will be the output of the given C++ code?

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
        int arr[] = {1, 3,4,2,5};
        int n = sizeof(arr)/sizeof(arr[0]);
        sort(arr+2, arr+n, greater<int>());
        int a;
        for (int a = 0; a < n; a++)
                cout << arr[a] << " ";
        return 0;
}
```

a) 1 2 3 4 5
b) 1 5 4 3 2
c) 5 4 3 2 1
d) 1 3 5 4 2

Answer: d
Explanation: As the first parameter to function sort() is arr+2 so the sorting begins from the third element i.e. 4. Also as there is a third argument greater () to the function sort() so the sorting will be done in descending order.

1. Which of the following is Python's standard sorting algorithm?
a) quick sort
b) introsort
c) merge sort
d) tim sort

Answer: d
Explanation: Tim sort has been python's standard sorting algorithm since its version 2.3. It is an example of hybrid

sorting algorithm which means it uses more than one sorting algorithm as a routine.

2. Which of the following sorting algorithm is a constituent of tim sort?
a) selection sort
b) quick sort
c) merge sort
d) heap sort

Answer: c
Explanation: Tim sort is a hybrid sorting algorithm which means it uses more than one sorting algorithm as a routine. It is derived from insertion sort and merge sort.

3. Tim sort begins sorting the given array by using which of the following sorting algorithm?
a) selection sort
b) quick sort
c) insertion sort
d) merge sort

Answer: c
Explanation: Tim sort begins sorting any given array by using insertion sort for each run. The array is divided into smaller parts for this purpose, each part having a size equal to value of run. Then these small parts called runs are merged in order to obtain sorted array.

4. Which of the following sorting algorithm is stable?
a) Tim sort
b) Introsort
c) Quick sort
d) Heap sort

Answer: a
Explanation: Out of the given options Tim sort is the only algorithm which is stable. As both constituents of Tim sort (I.e insertion sort and merge sort) are stable so Tim sort also becomes stable.

5. Which of the following sorting algorithm is not in-place?
a) insertion sort
b) tim sort
c) quick sort
d) intro sort

Answer: b
Explanation: Tim sort is not an in-place sorting algorithm as it requires auxiliary space. It is because it requires to merge sorted runs which requires a third array of the size equal to the sum of the two runs. Answer: a
Explanation: Merge sort and insertion sort are comparison based sorts. Thus overall Tim sort also becomes a comparison based sort.

7. What is the best case time complexity of Tim sort?
a) O(n)
b) O(n log n)
c) $O(n^2)$
d) O(log n)

Answer: a
Explanation: Best case time complexity of Tim sort occurs when the input array is already sorted. In such a case only one run will be required.

8. What is the worst case time complexity of Tim sort?
a) O(n)
b) O(n log n)

c) $O(n^2)$
d) O(log n)

Answer: b
Explanation: Worst case time complexity of Tim sort is O(n log n). It is because the worst complexity of merge sort is O(n log n) and insertion sort is only applied for small arrays.

9. What is the average time complexity of Tim sort?
a) O(n)
b) O(n log n)
c) $O(n^2)$
d) O(log n)

Answer: b
Explanation: Average time complexity of Tim sort remains to be O(n log n). It is the same as the average case complexity of merge sort.

10. What is the auxiliary space requirement of Tim sort?
a) O(n)
b) O(n log n)
c) $O(n^2)$
d) O(log n)

Answer: a
Explanation: Tim sort is a hybrid of merge sort and insertion sort. It requires to merge sorted runs which require a third array of the size equal to the sum of the two runs. So in worst case the auxiliary space requirement will be O(n).

11. Which of the following algorithm is implemented internally in java when we use function arrays.sort()?
a) intro sort
b) quick sort
c) tim sort
d) merge sort

Answer: c
Explanation: Java makes use of Tim sort internally for implementing arrays.sort(). It is mainly due to the fastness of this algorithm in comparison to other comparison based sorts.

12. Why is insertion sort preferred over other sorting algorithms (like selection sort, bubble sort etc.) for Tim sort implementation?
a) Because insertion sort is faster and adaptive
b) Because insertion sort requires less space
c) Because insertion sort is easy to implement
d) Because insertion sort is easy to understand

Answer: a
Explanation: When small arrays need to be sorted then insertion sort proves to be the best choice. Also, it is adaptive so it performs better than others when the given array is fully/partially sorted.

13. In which case will tim sort will work as an insertion sort?
a) when no. of elements are less than 64
b) when no. of elements are greater than 64
c) when no. of elements are less than size of run
d) when no. of elements are less than 32

Answer: c
Explanation: Tim sort uses a hybrid of insertion and merge sort. It reduces to insertion sort when the size of array is less than the size of run as insertion sort is efficient in sorting small arrays.

14. What is the usual size of a run in tim sort?
a) 32
b) less than 32
c) 32-64 depending on size of the array
d) 64

Answer: c
Explanation: Usually the size of the run is chosen somewhere between 32 and 64. The size of run is preferably chosen in powers of 2 in order to maintain balance while merging the sorted runs.

15. What will be the output of the given Java code?

```
import java.util.Arrays;
public class SortExample
{
        public static void main(String[] args)
        {
                // Our arr contains 8 elements
                int[] arr = {10,7,9,5,8,4};
                Arrays.sort(arr);
                System.out.printf(Arrays.toString(arr));
        }
}
```

a) [4,5,7,8,9,10]
b) [10,9,8,7,5,4]
c) 4,5,7,8,9,10
d) error

Answer: a
Explanation: The given program sorts the input in ascending order by using the function Arrays.sort(). It uses Tim sort internally.

16. What will be the output of the given Java code?

```
import java.util.Arrays;
public class SortExample
{
        public static void main(String[] args)
        {
                int[] arr = {10,7,9,5,8,4};
                Arrays.sort(arr, 1, 3);
                System.out.printf(Arrays.toString(arr));
        }
}
```

a) [4,5,7,8,9,10]
b) [10,9,8,7,5,4]
c) [10,5,7,8,9,4]
d) [10,7,9,5,8,4]

Answer: d
Explanation: The given program sorts only a portion of the input array. It is done by passing two extra arguments to the function Arrays.sort(). It sorts the elements between index 1 and 2.

1. Which of the following is an example of parallel sorting technique?
a) bogo sort
b) sleep sort
c) cube sort
d) merge sort

Answer: c

Explanation: Out of the given options only cube sort is a parallel sorting algorithm. It builds self balancing multi dimensional arrays from the input keys.

2. What is the worst case time complexity of cube sort?
a) O(n)
b) O(log n)
c) O(n log n)
d) O(n2)

Answer: c
Explanation: The worst case performance of cube sort is O(n log n). This is the fastest possible complexity with a comparison based sort.

3. What is the auxiliary space requirement of cube sort?
a) O(n)
b) O(1)
c) O(log n)
d) O(n log n)

Answer: a
Explanation: Cube sort requires an auxiliary space of O(n). This is the worst case of auxiliary space complexity.

4. What is the best case time complexity of cube sort?
a) $O(n^2)$
b) O(n)
c) O(n log n)
d) O(1)

Answer: b
Explanation: Best case time complexity of cube sort occurs when the input array is almost sorted. So in such a case only O(n) time is required for sorting.

5. What is the average case time complexity of cube sort?
a) $O(n^2)$
b) O(n log n)
c) O(log n)
d) O(n)

Answer: b
Explanation: The average case performance of cube sort is O(n log n). This is the fastest possible complexity with a comparison based sort.

6. Which of the following algorithm is stable?
a) heap sort
b) cube sort
c) quick sort
d) bogosort

Answer: d
Explanation: Out of the given algorithms only cube sort is stable. This implies that the relative position of equal valued elements in the input and sorted array remains the same.Answer: b
Explanation: Cube sort has an auxiliary space complexity of O(n). So it does not qualify to be an in-place sort.

8. Which of the following is a disadvantage of cube sort?
a) high memory overhead for small data
b) high memory overhead for any data
c) balancing is slow
d) Iteration is slow

Answer: a
Explanation: In a general case the memory overhead of cube sort is low. But when the data set is small then in that case the memory overhead becomes high.Answer: a
Explanation: Cube sort is a comparison based sorting algorithm. This is because it compares elements in order to sort them.

10. Which of the following sorting algorithm uses the method of insertion?
a) cube sort
b) bubble sort
c) quick sort
d) selection sort

Answer: a
Explanation: Cube sort is the only algorithm from the given ones that uses the method of insertion. Other than this, the insertion sort also uses the method of insertion for sorting.

1. Consider the original array 17 8 12 4 26. How many comparisons are needed to construct the BST on the original array?
a) 5
b) 4
c) 7
d) 10

Answer: d
Explanation: Original array is 17 8 12 4 26. The BST built on this array is shown in the figure below.

To built the BST, we travel down the tree until a leaf is reached. Therefore, for every element we compare the element with the internal nodes until we the leaves and then once again compare the element with its parent to decide whether it is right child or left child. So, for given array we need to perform 10 comparisons to build the BST.

2. In binary tree sort, we first construct the BST and then we perform _____ traversal to get the sorted order.
a) inorder
b) postorder
c) preorder
d) level order

Answer: a
Explanation: In binary tree sort is a sort algorithm where a binary search tree is built from the elements to be sorted, and then we perform inorder traversal on the BST to get the elements in sorted order.

3. What is the worst case time complexity of the binary tree sort?
a) O(n)
b) O(nlogn)
c) $O(n^2)$
d) O(logn)

Answer: c

Explanation: For the binary tree sort the worst case when the BST constructed is unbalanced. BST gets unbalanced when the elements are already sorted. So, in the worst case, $O(n^2)$ time is required to built the BST and O(n) time to traverse the tree. Therefore, the worst case time complexity is $O(n^2) + O(n) = O(n^2)$.

4. The insert() procedure, given below, builds the BST on the input elements, which is the first step of the binary tree sort. Choose the correct to fill the condition.

```
void insert(Tree* node, int newElement)
{
        if(node== NULL)
        {
                node = createNewNode();
                node-> value = newElement;
                node -> left = NULL;
                node -> right = NULL;
                return;
        }
        else if(____________________)
        {
                insert(node->left, newElement);
        }
        else
        {
                insert(node->right, newElement);
        }
}
```

a) newElement > node->value
b) newElement < node->value
c) newElement == root->value
d) newElement != root->value

Answer: b
Explanation: In binary tree sort, the BST is built on the input elements and the tree is traversed in in-order to get the sorted order. While building the BST, we travel down the tree until a leaf is reached. While traveling dawn the tree, we travel on left subtree if the new element is less than the node or to the right if the element is greater or equal to the node. So, correct option is newElement < node->value.

5. What is the best case time complexity of the binary tree sort?
a) O(n)
b) O(nlogn)
c) $O(n^2)$
d) O(logn)

Answer: b
Explanation: The best case occurs when the BST is balanced. So, when tree is balanced we require O(nlogn) time to build the tree and O(n) time to traverse the tree. So, the best case time complexity of the binary tree sort is O(nlogn).Answer: b
Explanation: In binary tree sort it is required to reserve one tree node for each array element. Its implementation requires two pointer variables for each node. So, it requires extra memory. The worst case space complexity of binary tree sort is $\Theta(n)$. Therefore, binary tree sort is not an in-place sorting algorithm.

7. Which of the following is false?
a) Binary tree sort and quick sort have same running time
b) Binary tree sort used BST as work area
c) As the number of elements to sort gets larger, binary tree sort gets more and more efficient
d) Both quick sort and binary tree are in place sorting algorithms

Answer: d
Explanation: Binary tree sort and quick sort have same running time i.e O(nlogn)
in average case and $O(n^2)$ in worst case. Binary tree is not in-place sorting algorithm.

8. Which of the following sorting algorithms can be considered as improvement to the binary tree sort?
a) Heap sort
b) Quick sort
c) Selection sort
d) Insertion sort

Answer: a
Explanation: Heap sort is basically improvement to the binary tree sort. Heap sort builds a heap on the input element by adjusting the position of the elements within the original array, rather than creating nodes as in binary tree sort.Answer: c
Explanation: Binary tree sort is dynamic sorting, that is it gets more efficient as more the elements are added. So, we can add elements gradually as they become available. Binary tree sort requires extra memory space, its worst case space complexity is Θ(n).

1. Which of the following is an example of an unstable sorting algorithm?
a) cycle sort
b) insertion sort
c) bubble sort
d) merge sort

Answer: a
Explanation: Out of the given options only cycle sort is an unstable sorting algorithm. This implies that the relative position of equal valued elements in the input and sorted array does not remain the same when we use cycle sort.

2. What is the worst case time complexity of cycle sort?
a) O(n)
b) O(log n)
c) O(n log n)
d) $O(n^2)$

Answer: d
Explanation: The worst case performance of cycle sort is $O(n^2)$. It is because it has to make n comparisons for each element of the array.

3. What is the auxiliary space requirement of cycle sort?
a) O(n)
b) O(1)
c) O(log n)
d) O(n log n)

Answer: b
Explanation: Cycle sort requires an auxiliary space of O(1). So this makes it an in-place sorting algorithm.

4. What is the best case time complexity of cycle sort?
a) $O(n^2)$
b) O(n)
c) O(n log n)
d) O(1)

Answer: a
Explanation: Best case time complexity of cycle sort is $O(n^2)$. This shows that cycle sort is not an adaptive sorting algorithm and thus makes it undesirable when the given array is already almost sorted.

5. What is the average case time complexity of cycle sort?
a) $O(n^2)$
b) O(n log n)
c) O(log n)

d) O(n)

Answer: a
Explanation: The average case performance of cycle sort is $O(n^2)$. It is because it has to make n comparisons for each element of the array.Answer: b
Explanation: The time complexity of cycle sort is $O(n^2)$ in any case. So cycle sort algorithm is not adaptive, as it will take the same time to sort an already sorted array and any other randomly arranged array.

7. Which of the following sorting algorithm is in-place?
a) Merge sort
b) Cycle sort
c) Counting sort
d) Radix sort

Answer: b
Explanation: Cycle sort has an auxiliary space complexity of O(1). So it qualifies to be an in-place sort. All other given options are not in place sorting algorithm.

8. Which of the following is an advantage of cycle sort?
a) it can sort large arrays efficiently
b) it has a low time complexity
c) it requires minimal write operations
d) it is an adaptive sorting algorithm

Answer: c
Explanation: Cycle sort is a slow sorting algorithm as compared to quick sort, merge sort etc. and is also not adaptive. But it offers an advantage that it performs minimal write operations which can be very useful in a situation where the write operation is expensive.Answer: a
Explanation: Cycle sort is a comparison based sorting algorithm. This is because it compares elements in order to sort them.

10. Which of the following sorting algorithm uses the method of insertion?
a) cycle sort
b) bubble sort
c) quick sort
d) selection sort

Answer: a
Explanation: Cycle sort is the only algorithm from the given ones that uses the method of insertion. Other than this, insertion sort also uses the method of insertion for sorting.

11. How many write operations will be required to sort the array arr={2,4,3,5,1} using cycle sort?
a) 4
b) 5
c) 6
d) 3

Answer: a
Explanation: Cycle sort requires a minimum number of write operations in order to sort a given array. So in this case, only 4 write operations will be required.

12. Which of the following algorithm is best suited for the case where swap operation is expensive?
a) bubble sort
b) cycle sort
c) cocktail sort
d) merge sort

Answer: b

Explanation: Cycle sort is a slow sorting algorithm but it requires a minimum number of write operations in order to sort a given array. So it is useful when the write/swap operation is expensive.

1. Which of the following is a disadvantage of library sort when compared to insertion sort?
a) library sort has greater time complexity
b) library sort has greater space complexity
c) library sort makes more comparisons
d) it has no significant disadvantage

Answer: b
Explanation: Library sort has a disadvantage that it takes up O(n) auxiliary space whereas insertion sort takes constant auxiliary space.Answer: a
Explanation: Library sort does not require the entire input data at the beginning itself in order to sort the array. It rather creates a partial solution in every step, so future elements are not required to be considered. Hence it is an online sorting algorithm like insertion sort.

3. Library sort is a modified version of which of the following sorting algorithm?
a) Bubble sort
b) selection sort
c) insertion sort
d) quick sort

Answer: c
Explanation: Library sort requires the use of Insertion sort and binary search in its code. So it is a modified version of insertion sort.

4. Which of the following sorting algorithm is stable?
a) Selection sort
b) Quick sort
c) Library sort
d) Heap sort

Answer: c
Explanation: Out of the given options library sort is the only algorithm which is stable. It is because the elements with identical values appear in the same order in the output array as they were in the input array.

5. Which of the following sorting algorithm requires the use of binary search in their implementation?
a) radix sort
b) library sort
c) odd-even sort
d) bead sort

Answer: b
Explanation: Library sort makes use of a binary search algorithm. It is used to find the correct index in the array where the element should be inserted. Then after the insertion of the element re-balancing of the array takes place.Answer: a
Explanation: In library sort, we need to compare elements in order to insert them at the correct index. So we can say that it uses comparisons in order to sort the array. Thus it qualifies as a comparison based sort.

7. What is the average case time complexity of library sort?
a) O(n)
b) O(n log n)
c) $O(n^2)$
d) O(log n)

Answer: b
Explanation: Library sort uses binary search in order to insert elements in the sorted segment of the array which reduces its time complexity. So the average time complexity of library sort is O(n log n).

8. What is the best case time complexity of library sort?
a) O(n)
b) O(n log n)
c) $O(n^2)$
d) O(log n)

Answer: a
Explanation: The best case time complexity of library sort is O(n). It occurs in the case when the input is already/almost sorted.

9. What is the worst case time complexity of library sort?
a) O(n)
b) O(n log n)
c) $O(n^2)$
d) O(log n)

Answer: c
Explanation: The worst case time complexity of library sort is the same as that of insertion sort. The worst case time complexity is $O(n^2)$.

10. Which of the following is an alternate name of library sort?
a) gapped insertion sort
b) binary insertion sort
c) recursive insertion sort
d) binary gap sort

Answer: a
Explanation: Library sort is also known as gapped insertion sort because it uses insertion sort with gaps in between successive elements. These gaps allow for fast insertion.

11. What is the advantage of library sort over insertion sort?
a) Library sort has a better average time complexity
b) Library sort has a better space complexity
c) Library sort has better best case time complexity
d) Library has better worst case time complexity

Answer: a
Explanation: Library sort has a better average time complexity as compared to insertion sort because it uses binary search for finding the index where the element has to be inserted in the sorted array. This makes the process faster.

12. What is the auxiliary space complexity of library sort?
a) O(n)
b) O(1)
c) O(n log n)
d) $O(n^2)$

Answer: a
Explanation: The auxiliary space required by library sort is O(n). This space is taken up by the gaps present in between successive elements.

13. Which of the following is an adaptive sorting algorithm?
a) library sort
b) merge sort
c) heap sort
d) selection sort

Answer: a

Explanation: Library sort is an adaptive algorithm. It is because the time complexity of the algorithm improves when the input array is almost sorted.

14. Which of the following sorting algorithm is not in place?
a) library sort
b) quick sort sort
c) heap sort
d) gnome sort

Answer: a
Explanation: Out of the given options library sort is the only algorithm which is not in place. It is because the auxiliary space required by library sort is O(n).

15. Which of the following is not true about library sort?
a) it uses binary search and insertion sort in its implementation
b) gaps are created between successive elements in order to ensure faster insertion
c) array needs to be re balanced after every insertion
d) it is an in place sorting algorithm

Answer: d
Explanation: Library sort is not an in place sorting algorithm as it requires O(n) auxiliary space. The array needs to be re balanced after every insertion so as to ensure that gaps are present between every successive element.

1. Which one of the following sorting algorithm requires recursion?
a) pigeonhole sort
b) strand sort
c) insertion sort
d) counting sort

Answer: b
Explanation: Strand sort requires the use of recursion for implementing its algorithm. On the other hand, the sorting algorithms given in the remaining options use iterative methods.

2. Strand sort is most efficient for data stored in?
a) linked list
b) arrays
c) trees
d) graphs

Answer: a
Explanation: Strand sort is most efficient when data is stored in a linked list as it involves many insertions and deletions which is performed quite efficiently with the help of a linked list. Using an array would increase time complexity significantly.

3. In which of the following case strand sort is most efficient?
a) when input array is already sorted
b) when input array is reverse sorted
c) when input array is large
d) when input array is has randomly spread elements

Answer: a
Explanation: The best case of strand sort occurs when the input array is already sorted. In this case, it has linear time complexity.

4. What is the auxiliary space complexity of strand sort?
a) O(n)
b) O(1)
c) O(log n)

d) O(n log n)

Answer: a
Explanation: The auxiliary space complexity of strand sort is O(n). It is because a sub-list of size n has to be maintained.

5. Which of the following sorting algorithm is not in place?
a) quick sort
b) strand sort
c) cycle sort
d) heap sort

Answer: b
Explanation: Strand sort has an auxiliary space complexity of O(n). So it is not an in place sorting algorithm.Answer: a
Explanation: Pigeonhole sort is an example of a comparison based sorting algorithm. This is because it compares the value of elements present in a list in order to sort them.Answer: a
Explanation: Strand sort is an example of a stable sorting algorithm. It is because the elements with identical values appear in the same order in the output array as they were in the input array.

8. What is the average time complexity of strand sort?
a) O(n)
b) O(n log n)
c) $O(n^2)$
d) $O(n^2 \log n)$

Answer: c
Explanation: Average case time complexity of strand sort is O(n2). So it is not as efficient as quick sort or merge sort.

9. What is the best case time complexity of strand sort?
a) O(n)
b) O(n log n)
c) $O(n^2)$
d) $O(n^2 \log n)$

Answer: a
Explanation: Best case time complexity of strand sort is O(n). It occurs in the case where the input array is already sorted.

10. What is the worst case time complexity of strand sort?
a) O(n)
b) O(n log n)
c) $O(n^2)$
d) $O(n^2 \log n)$

Answer: c
Explanation: Worst case time complexity of strand sort is $O(n^2)$. It occurs in the case where the input array is in reverse sorted order.

11. Strand sort algorithm used which of the following method for sorting a list?
a) merging
b) selection
c) insertion
d) partitioning

Answer: b
Explanation: Strand sort uses the method of selection for sorting any given list. This is because it selects the strands of elements from the input list which are sorted.

12. Which of the following is an adaptive sorting algorithm?
a) heap sort
b) strand sort
c) selection sort
d) merge sort

Answer: b
Explanation: Strand sort is an adaptive sorting algorithm. This is because it gives a better performance when the input list is almost sorted.

1. Cocktail sort is also known as _____
a) bidirectional sort
b) bubble sort
c) brick sort
d) ripple sort

Answer: d
Explanation: Cocktail sort is also known by the name of ripple sort. It is also known by other names like – bidirectional bubble sort, cocktail shaker sort, shuttle sort, shuffle sort etc.

2. Cocktail sort is a variation of _____
a) Bubble sort
b) Selection sort
c) Insertion sort
d) Gnome sort

Answer: a
Explanation: Cocktail sort is very similar to bubble sort. It works by traversing an array in both directions alternatively. It compares the adjacent elements in each iteration and swaps the ones which are out of order.

3. Auxiliary space requirement of cocktail sort is _____
a) O(n)
b) O(log n)
c) O(1)
d) $O(n^2)$

Answer: c
Explanation: In cocktail sort manipulation is done on the input array itself. So no extra space is required to perform sorting. Thus it requires constant auxiliary space.

4. Which of the following sorting algorithm is NOT stable?
a) Quick sort
b) Cocktail sort
c) Bubble sort
d) Merge sort

Answer: a
Explanation: Out of the given options quick sort is the only algorithm which is not stable. Cocktail sort like bubble sort is a stable sorting algorithm.

5. Which of the following sorting algorithm is in place?
a) cocktail sort
b) merge sort
c) counting sort
d) radix sort

Answer: a
Explanation: Cocktail sort is an in place sorting technique as it only requires constant auxiliary space for manipulating

the input array. Rest all other options are not in place.Answer: a
Explanation: Cocktail sort compares the value of different elements in the array for sorting. Thus, it is a comparison based sort.

7. Cocktail sort uses which of the following methods for sorting the input?
a) selection
b) partitioning
c) merging
d) exchanging

Answer: d
Explanation: Cocktail sort uses a method of exchanging as it swaps the elements which are out of order. This swapping is done in two phases i.e. forward phase and backward phase.

8. What is the worst case time complexity of cocktail sort?
a) O(n)
b) O(n log n)
c) O(n^2)
d) O(log n)

Answer: c
Explanation: Worst case complexity is observed when the input array is reverse sorted. This is the same as the worst case complexity of bubble sort.

9. What is the best case time complexity of cocktail sort?
a) O(n)
b) O(n log n)
c) O(n^2)
d) O(log n)

Answer: a
Explanation: Best case complexity is observed when the input array is already sorted. This is the same as the best case complexity of bubble sort.

10. What is the average case time complexity of odd-even sort?
a) O(n)
b) O(n log n)
c) O(n^2)
d) O(log n)

Answer: c
Explanation: Cocktail sort takes O(n^2) time on average as it keeps on applying bubble sort on the elements in two phases until they are sorted. This is the same as the average time complexity of bubble sort.

11. How many iterations are required to sort the array arr={2,3,4,5,1} using bubble sort and cocktail sort respectively?
a) 4,2
b) 5,3
c) 5,2
d) 4,3

Answer: a
Explanation: Cocktail sort applies bubble sort in two phases until the array gets sorted. So here bubble sort will take 4 iterations to sort the given array whereas cocktail sort takes only 2 iterations. This shows cocktail sort has a comparatively better performance.

12. The following function represents which sorting?

```
void Sorting(int a[], int n)
{
        bool swap = true;
        int first = 0;
        int last = n - 1;

        while (swap)
        {

                swap = false;

                for (int i = first; i < last;i++)
                {
                        if (a[i] > a[i + 1])
                        {
                                swap(a[i], a[i + 1]);
                                swap = true;
                        }
                }

                if (!swap)
                        break;

                swap = false;

                --last;


                for (int i = last - 1; i >= first; i--)
                {
                        if (a[i] > a[i + 1])
                        {
                                swap(a[i], a[i + 1]);
                                swap = true;
                        }
                }

                ++first;
        }
}
```

a) Bubble sort
b) Selection sort
c) Bidirectional bubble sort
d) Odd-even sort

Answer: c
Explanation: The given function represents bidirectional bubble sort also known as cocktail sort. In this sort, we apply bubble sort in two phases i.e. forward and backward phase until the array gets sorted.Answer: b
Explanation: Both bubble sort and cocktail sort has the same time complexities. But cocktail sort has a comparatively better performance.

1. Comb sort is an improved version of _____
a) Selection sort
b) Bubble sort
c) Insertion sort
d) Merge sort

Answer: b
Explanation: Comb sort compares two elements at a variable gap from each other in each iteration unlike bubble sort where the gap remains 1. This reduces the average time complexity of comb sort.

2. The gap between two elements being compared shrinks by a factor of _____ after every iteration.
a) 1.1
b) 1.2

c) 1.3
d) 1.4

Answer: c
Explanation: It has been found experimentally that the gap between the two elements should shrink by a factor of 1.3 after each iteration for the most efficient sorting.

3. The initial gap between two elements being compared _____
a) is equal to number of elements in the array
b) is equal to 1.3
c) depends on the number of iterations
d) depends on the compiler being used

Answer: a
Explanation: Initial gap is taken to be equal to the number of elements in the array and shrinks by a factor of 1.3 in each iteration, initial gap is independent of the number of iterations and compiler being used.

4. What is the worst case time complexity of comb sort?
a) $O(n^2)$
b) O(n log n)
c) O(n)
d) $O(n^2/2^a)$ (a=number of increment)

Answer: a
Explanation: Worst case complexity is observed when the input array is reverse sorted. This is same as the worst case complexity of bubble sort.

5. The gap value after 3 iterations in an array with 6 elements will be _____
a) 4
b) 3
c) 2
d) 1

Answer: c
Explanation: Initially the gap value will be 6. For the first iteration, it will be 6/1.3=4 then 4/1.3=3 for second iteration and 3/1.3=2 for the third iteration.

6. Auxiliary space used by comb sort is _____
a) O(1)
b) O(n)
c) O(log n)
d) O(n log n)

Answer: a
Explanation: Auxiliary space used by comb sort is O(1) as it does not use any extra space for manipulating the input.

7. The given array is arr={7,4,5,8,1,2}. The number of iterations required to sort the array using comb sort and bubble sort respectively will be _____
a) 7 and 8
b) 5 and 6
c) 5 and 5
d) 4 and 5

Answer: d
Explanation: Comb sort takes 1 iteration less as compared to bubble sort as it makes use of variable gap value whereas bubble sort uses a constant gap value of 1.Answer: b
Explanation: Comb sort is not a stable sorting algorithm as it does not sort the repeated elements in the same order as they appear in the input.

9. What is the best case time complexity of comb sort and bubble sort respectively?
a) $O(n^2)$ and O(n log n)
b) O(n log n) and O(n)
c) O(n) and $O(n^2)$
d) $O(n^2/2^a)$ (a=number of increment) and $O(n^2)$

Answer: b
Explanation: Best case complexity for comb sort and bubble sort is O(n log n) and O(n) respectively. It occurs when the input array is already sorted.

10. What is the advantage of comb sort over merge sort?
a) Comb sort is an in place sorting algorithm
b) Comb sort is a stable sorting algorithm
c) Comb sort is more efficient
d) It has no advantage

Answer: a
Explanation: Comb sort does not require auxiliary space for manipulating input so it is an in place sorting algorithm but merge sort does require O(n) of auxiliary space which makes comb sort better in terms of space complexity.

1. Gnome sort is also called _____
a) Smart sort
b) Stupid sort
c) Bogo sort
d) Special sort

Answer: b
Explanation: Gnome sort was originally named as stupid sort but later on it got renamed as gnome sort.

2. How many loops are required to implement gnome sorting algorithm?
a) Single loop
b) 2 nested loops
c) 3 nested loops
d) It does not require any loop

Answer: a
Explanation: In this sorting algorithm the variable representing the index number is not incremented in case the adjacent pair of elements are out of place. In such a case its value is decremented instead. Thus it is able to implement sorting using a single loop.

3. Which of the following pair of sorting algorithms are stable?
a) gnome sort and quick sort
b) merge sort and selection sort
c) gnome sort and merge sort
d) heap sort and merge sort

Answer: c
Explanation: Gnome sort and merge sort are stable sorting algorithms as the elements with identical values appear in the same order in the output array as they were in the input array when any of these sorting algorithms are implemented.

4. Auxiliary space used by gnome sort is _____
a) O(1)
b) O(n)
c) O(log n)
d) O(n log n)

Answer: a

Explanation: Auxiliary space used by gnome sort is O(1) as it does not use any extra space for manipulating the input. Thus it also qualifies as an in place sorting algorithm.

5. The given array is arr = {1,2,4,3,5}.The number of iterations required to sort the array using gnome sort will be _____

a) 5
b) 6
c) 7
d) 8

Answer: b
Explanation: 6 iterations will be required as one pair of elements i.e. {4,3} is out of place which causes the loop to take one step backward.

6. Gnome sort uses which of the following method to implement sorting?
a) Merging
b) Partitioning
c) Selection
d) Exchanging

Answer: d
Explanation: Gnome sort implements sorting by exchanging the adjacent elements which are out of order. Thus its method of sorting is called exchanging.

7. What is the best case time complexity of gnome sort?
a) O(n)
b) $O(n^2)$
c) O(n log n)
d) O(log n)

Answer: a
Explanation: When the input array is already sorted then in that case there will be no need to decrease the value of the index variable at any stage. So only O(n) time is required in this case as we keep on increasing its value after each iteration.

b)

```
while (index > n)
{
        if (index == 0)
            index++;
        if (arr[index] <= arr[index - 1])
            index++;
        else
        {
            swap(arr[index], arr[index - 1]);
            index--;
        }
}
```

c)

```
while (index < n)
{
        if (index == 0)
            index++;
        if (arr[index] <= arr[index - 1])
            index++;
        else
        {
            swap(arr[index], arr[index - 1]);
            index--;
```

```
        }
}
```

d)

```
while (index < n)
{
        if (index == 0)
            index++;
        if (arr[index] >= arr[index - 1])
            index++;
        else
        {
            swap(arr[index], arr[index - 1]);
            index--;
        }
}
```

9. What is the worst case time complexity of gnome sort?
a) O(n)
b) O(n^2)
c) O(n log n)
d) O(log n)

Answer: b
Explanation: Worst case occurs when the input array is reverse sorted as it will have the maximum number of decrements while sorting. This causes the algorithm to have a time complexity of O(n^2) in this case.

10. What is the average case time complexity of gnome sort?
a) O(n)
b) O(n^2)
c) O(n log n)
d) O(log n)

Answer: b
Explanation: In gnome sort the loop has to take one step backwards every time any adjacent pair of elements is out of place which causes it to have time complexity of O(n^2) on an average.

1. Which of the following is not an alternative name of bogosort?
a) stupid sort
b) permutation sort
c) donkey sort
d) monkey sort

Answer: c
Explanation: Bogosort is also known by names like stupid sort, monkey sort, permutation sort, slow sort and shotgun sort.These names are particularly chosen due to its inefficient algorithm.

2. Bogosort works by _____
a) generating random permutations of its input
b) partitioning the array
c) dividing the value of input elements
d) generating permutations according to the value of first element of array

Answer: a
Explanation: Bogosort algorithm successively generates permutations of its input. This process is repeated until the sorted version of the array is found.

3. What is the auxiliary space requirement of bogosort?
a) O(n)

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

Answer: b
Explanation: Bogosort algorithm do not require any extra space for sorting the input array. Thus its auxiliary space requirement is O(1).

4. What is the best case time complexity of bogosort?
a) $O(n^2)$
b) O(n)
c) O(n log n)
d) O(1)

Answer: b
Explanation: Best case time complexity of bogosort occurs when the input array is already sorted. So in such a case we only need to check whether all the elements are sorted which can be done in O(n) time.

5. What is the worst case time complexity of bogosort?
a) $O(n^2)$
b) O(n*n!)
c) O(infinity)
d) O(n log n)

Answer: c
Explanation: There is no upper bound to the worst case of this algorithm. It can go on to take very large amount of time if the array has many elements. So the worst case of this algorithm can be taken as O(infinity).

6. Which of the following sorting algorithm is not stable _____
a) insertion sort
b) bubble sort
c) merge sort
d) bogosort

Answer: d
Explanation: Out of the given algorithms only bogosort is not stable. This is because it creates permutations of the input array in order to obtain the sorted version. So there is no guarantee that the sorted version obtained by such a method gives a stable output.

7. Which of the following is an in-place sorting algorithm?
a) Merge sort
b) Bogosort
c) Radix sort
d) Counting sort

Answer: b
Explanation: Out of the given algorithms only bogosort is an in-place sorting algorithm. It is because bogosort algorithm do not require any extra space for sorting the input array. Answer: b
Explanation: If we sort an array using sleep sort then there is no guarantee that the output we get is correctly sorted. So even though sleep sort is better than bogosort in time complexity but it cannot be preferred due to its inaccuracy.

9. What is the average case time complexity of bogosort?
a) $O(n^2)$
b) O(n*n!)
c) O(infinity)
d) O(n log n)

Answer: b
*Explanation: For calculating the average we first need to calculate the number of possible permutations an array of size n can have. This will be equal to n!. As each permutation also needs to be checked whether it is sorted or not so this takes another n time. Thus overall time complexity becomes O(n*n!).*

b)

```
bool isSorted(int a[], int n)
{
    while ( --n > 1 )
        if (a[n] < a[n-1])
            return false;
    return true;
}
void shuffle(int a[], int n)
{
    for (int i=0; i < n; i++)
        swap(a[i], a[rand()%n]);
}
void bogosort(int a[], int n)
{

    while ( !isSorted(a, n) )
        shuffle(a, n);
}
```

c)

```
bool isSorted(int a[], int n)
{
    while ( --n > 1 )
        if (a[n] < a[n-1])
            return true;
    return false;
}
void shuffle(int a[], int n)
{
    for (int i=0; i < n; i++)
        swap(a[i], a[rand()%n]);
}
void bogosort(int a[], int n)
{

    while ( !isSorted(a, n) )
        shuffle(a, n);
}
```

d)

```
bool isSorted(int a[], int n)
{
    while ( --n > 1 )
        if (a[n] > a[n-1])
            return true;
    return false;
}
void shuffle(int a[], int n)
{
    for (int i=0; i < n; i++)
        swap(a[i], a[rand()%n]);
}
void bogosort(int a[], int n)
{

    while ( !isSorted(a, n) )
        shuffle(a, n);
}
```

1. Which of the following header file is a must to implement sleep sort algorithm?
a) string.h
b) math.hw
c) bios.h
d) windows.h

Answer: d
Explanation: To implement sleep sort algorithm we need functions like WaitForMultipleObjects(), _beginthread(). These are included in the header file windows.h.

2. Sleep sort does not work for _____
a) negative numbers
b) large numbers
c) small numbers
d) positive numbers

Answer: a
Explanation: Sleep sort algorithm does not work for negative numbers. It is because thread cannot sleep for negative amount of time.

3. In how many comparisons does the array arr={1,4,2,3,5} gets sorted if we use sleep sort?
a) 5
b) 3
c) 1
d) 0

Answer: d
Explanation: Sleep sort makes different elements of the array to sleep for an amount of time that is proportional to its magnitude. So it does not require to perform any comparison in order to sort the array.

4. Sleep sort works by _____
a) making elements to sleep for a time that is proportional to their magnitude
b) making elements to sleep for a time that is inversely proportional to their magnitude
c) partitioning the input array
d) dividing the value of input elements

Answer: a
Explanation: In sleep sort each element is made to sleep for a time that is proportional to its magnitude. Then the elements are printed in the order in which they wake up.

5. Sleep sort code cannot compile online because _____
a) it has very high time complexity
b) it has very high space complexity
c) it requires multithreading process
d) online compilers are not efficient

Answer: c
Explanation: Sleep sort requires multithreading process for making the elements to sleep. This process happens in the background at the core of the OS and so cannot be compiled on an online compiler.

6. Time complexity of sleep sort can be approximated to be _____
a) O(n + max(input))
b) $O(n^2)$
c) O(n log n + max(input))
d) O(n log n)

Answer: c
Explanation: As the sleep() function creates multiple threads by using priority queue which takes n log n time for

insertion. Also the output is obtained when all the elements wake up. This time is proportional to the max(input). So its time complexity is approximately O(n log n + max(input)).

7. Sleep sort can be preferred over which of the following sorting algorithms for large number of input elements?
a) Quick sort
b) Bubble sort
c) Selection sort
d) No sorting algorithm is preferred

Answer: d
Explanation: Sleep sort is not preferred over any of the given sorting algorithms as sleep sort does not guarantee a correct output every time. So sleep sort is not a reliable sorting technique.

8. Auxiliary space requirement of sleep sort is _____
a) O(n)
b) O(1)
c) O(max(input))
d) O(log n)

Answer: b
Explanation: All the major processes involved in sleep sort takes place internally in the OS. So it does not require any auxiliary space to sort the elements.

9. Sleep sort does gives a correct output when _____
a) any input element is negative
b) input array is reverse sorted
c) any input element is positive
d) when there is a very small number to the left of very large number

Answer: c
Explanation: Sleep sort gives a sorted output when the array elements are positive. But when any other case than this occur out of the above given cases then we may not see a correct output. This makes sleep sort very unreliable sorting technique.

10. Which of the following sorting algorithm is most closely related to the OS?
a) gnome sort
b) sleep sort
c) radix sort
d) bogo sort

Answer: b
Explanation: Sleep sort is most closely related to the operating system. It is because most of the major steps of this algorithm takes place at the core of OS.Answer: a
Explanation: Sleep sort is an in-place sorting technique as most of its major steps takes place in the background. So it does not require auxiliary space to sort the input.

1. How many comparisons will be made to sort the array arr={1,5,3,8,2} using pigeonhole sort?
a) 5
b) 7
c) 9
d) 0

Answer: d
Explanation: As pigeonhole sort is an example of a non-comparison sort so it is able to sort an array without making any comparison. So 0 comparisons are required.

2. Which of the following is a non-comparison sort?
a) heap sort

b) quick sort
c) merge sort
d) pigeonhole sort

Answer: d
Explanation: Heap sort, merge sort and quick sort are examples of comparison sort as it needs to compare array elements in order to sort an array. Whereas pigeonhole sort is a non-comparison based sort.

3. In which of the following case pigeonhole sort is most efficient?
a) when range of input is less than number of elements
b) when range of input is more than number of elements
c) when range of input is comparable to the number of elements
d) when the given array is almost sorted

Answer: c
Explanation: Pigeonhole sort is a non-comparison based sort. It is most efficient in the case where the number of elements are comparable to the input range.

4. What is the space complexity of pigeonhole sort (k=range of input)?
a) O(n*k)
b) O(n)
c) O(k)
d) O(n+k)

Answer: d
Explanation: Pigeonhole sort algorithm requires two arrays. The first one is required to store the input elements so its size is n. The second one is the pigeonhole array and has a size equal to range k. Overall space complexity becomes O(n+k).

5. The auxiliary array used in pigeonhole sorting is called _____
a) bucket
b) pigeon
c) hole
d) pigeonhole

Answer: d
Explanation: The auxiliary array used in pigeonhole sorting is called pigeonhole. It is used to store every element in its corresponding hole.Answer: a
Explanation: Pigeonhole sort is an example of a stable sorting algorithm. It is because the elements with identical values appear in the same order in the output array as they were in the input array. Answer: b
Explanation: Pigeonhole sort requires space of O(n+k). So it does not qualify to be an in place sorting algorithm.

8. What is the average time complexity of pigeonhole sort (k=range of input)?
a) O(n)
b) O(n+k)
c) $O(n^2)$
d) O(n*k)

Answer: b
Explanation: Time complexity of pigeonhole sort is O(n+k). It has two loops. One of the loops runs from 0 to range(k) and the other one runs from 0 to n so the time complexity becomes O(n+k).

9. The complexity of which of the following sorting algorithms remains to be the same in its best, average and worst case?
a) quick sort
b) insertion sort
c) pigeonhole sort

d) bubble sort

Answer: c
Explanation: The time complexity of pigeonhole remains unvaried in all three cases. It is given by O(n+k). But it is efficient only when the number of elements is comparable to the input range.

10. Choose the correct statement from the following.
a) pigeonhole sort is a comparison based sort
b) any comparison based sorting can be made stable
c) quick sort is not a comparison based sort
d) any comparison based sort requires at least $O(n^2)$ time

Answer: b
Explanation: Any comparison based sorting technique can be made stable by considering the position as criteria while making comparisons. Pigeonhole sort is a stable sort.

11. What is the advantage of pigeonhole sort over merge sort?
a) pigeonhole sort has lesser time complexity when range is comparable to number of input elements
b) pigeonhole sort has lesser space complexity
c) counting sort is not a comparison based sorting technique
d) pigeonhole sort is adaptive

Answer: a
Explanation: Pigeonhole sort is efficient in the cases where the range is comparable to a number of input elements as it performs sorting in linear time. Whereas merge sort takes O(n log n) in any case.

b)

```
void Sorting(int arr[], int n)
{

        int minimum = arr[0], maximum = arr[0];
        for (int i = 1; i < n; i++)
        {
                if (arr[i] < minimum)
                        minimum = arr[i];
                if (arr[i] > maximum)
                        maximum = arr[i];
        }
        int r = maximum - minimum + 1;
        vector<int> p_holes[r];
        for (int i = 0; i < n; i++)
                p_holes[arr[i]-minimum].push_back(arr[i]);
        int ind = 0;
        for (int i = 0; i < r; i++)
        {
            vector<int>::iterator it;
            for (it = p_holes[i].begin(); it != p_holes[i].end(); ++it)
                        arr[ind++] = *it;
        }
}
```

c)

```
void Sorting(int arr[], int n)
{

        int minimum = arr[0], maximum = arr[0];
        for (int i = 1; i < n; i++)
        {
                if (arr[i] < minimum)
                        minimum = arr[i];
                if (arr[i] > maximum)
                        maximum = arr[i];
```

```
        }
        int r = maximum - minimum + 1;
        vector<int> p_holes[n];
        for (int i = 0; i < n; i++)
                p_holes[arr[i]-minimum].push_back(arr[i]);
        int ind = 0;
        for (int i = 0; i < r; i++)
        {
            vector<int>::iterator it;
            for (it = p_holes[i].begin(); it != p_holes[i].end(); ++it)
                        arr[ind++] = *it;
        }
}
```

d)

```
void Sorting(int arr[], int n)
{

        int minimum = arr[0], maximum = arr[0];
        for (int i = 1; i < n; i++)
        {
                if (arr[i] < minimum)
                        minimum = arr[i];
                if (arr[i] > maximum)
                        maximum = arr[i];
        }
        int r = maximum - minimum + 1;
        vector<int> p_holes[r];
        for (int i = 0; i < n; i++)
                p_holes[arr[i]-minimum].push_back(arr[i]);
        int ind = 0;
        for (int i = 0; i < n; i++)
        {
            vector<int>::iterator it;
            for (it = p_holes[i].begin(); it != p_holes[i].end(); ++it)
                        arr[ind++] = *it;
        }
}
```

13. Which of the following algorithm takes linear time for sorting?
a) pigeonhole sort
b) heap sort
c) comb sort
d) cycle sort

Answer: a
Explanation: Pigeonhole sort has a linear time complexity of O(n+k). Whereas all other given options have a non linear time complexity. But it should be noted that pigeonhole sort may not be efficient in every case.

1. Which of the following is the distribution sort?
a) Heap sort
b) Smooth sort
c) Quick sort
d) LSD radix sort

Answer: d
Explanation: In Distribution sort the inputted values are distributed to multiple intermediate structures which are then combined and placed on the output. And LSD radix sort distributes values into buckets based on the digits within values, so it is a distribution sort.

2. What is the worst case time complexity of LSD radix sort?
a) O(nlogn)
b) O(wn)

c) O(n)
d) O(n + w)

Answer: b
Explanation: Time complexity of LSD radix sort depends upon the word size and the number on items. It runs in O(wn) time in worst case, where n is the number of inputted elements and w is the number of digits in the largest number.

3. LSD radix sort requires _____ passes to sort N elements.
a) (w/logR)
b) N(w/logR)
c) (w/log(RN))
d) (wN/log(N))

Answer: a
Explanation: LSD radix sort sorts the N elements in (w/logR) passes where w is the number of digits in largest number and R(radix) is extra space required for performing the sorting operation.

4. Which of the following is false?
a) LSD radix sort is an integer sorting algorithm
b) LSD radix sort is a comparison sorting algorithm
c) LSD radix sort is a distribution sort
d) LSD radix sort uses bucket sort

Answer: b
Explanation: LSD radix sort uses bucket sort for grouping the keys based on the digits at that value. And as it grouped the keys based on the digits at that values, it is integer sorting algorithm.

5. Which of the following sorting algorithm is stable?
a) Heap sort
b) Selection sort
c) In-place MSD radix sort
d) LSD radix sort

Answer: d
Explanation: In LSD radix sort after sorting the multiple elements with the same key will be in the same order as they were in the input array. So LSD radix sort is stable sort.Answer: b
Explanation: LSD radix sort is faster than comparison sorts when the word size is less than logn. But LSD radix sort runs slowly for elements with larger word size and smaller radix.

7. Which of the following should be used to sort a huge database on a fixed-length key field?
a) Insertion sort
b) Merge sort
c) LSD radix sort
d) Quick sort

Answer: c
Explanation: LSD radix requires only w passes to sort a fixed-length string, where w is a length of the strings. So, LSD radix sort is best suited to sort a huge database on a fixed-length key field.

8. Which of the following is a combination of LSD and MSD radix sorts?
a) Forward radix sort
b) 3-way radix quick sort
c) Trie base radix sort
d) Flash sort

Answer: a
Explanation: Forward radix sort combines the advantages of LSD and MSD radix sort. Forward radix sort inspects a complete horizontal strip at a time just like LSD radix sort.

9. Which of the following is true for the LSD radix sort?
a) works best for variable length strings
b) accesses memory randomly
c) inner loop has less instructions
d) sorts the keys in left-to-right order

Answer: b
Explanation: LSD radix sort sorts the keys in right-to-left order, working with Least Significant Digit first. The inner loop has a lot of instructions and LSD radix sort is used to sort fixed-length strings. Answer: a
Explanation: LSD radix is not an in-place sorting algorithm. It needs extra memory to store the elements in bucket and its worst case space complexity is O(w + R).

1. How many comparisons will be made to sort the array arr = {1, 5, 3, 8, 2} using MSD radix sort?
a) 5
b) 7
c) 9
d) 0

Answer: d
Explanation: As MSD radix sort is an example of non comparison sort so it is able to sort an array without making any comparison. So the answer should be 0.

2. What is the full form of MSD in MSD radix sort?
a) most significant digit
b) many significant digit
c) more significant digit
d) must significant digit

Answer: a
Explanation: MSD stands for Most Significant Digit. It is named so because in this algorithm the processing begins from the most significant digit.

3. Which of the following combines qualities of MSD radix sort and LSD radix sort?
a) in-place MSD radix sort
b) stable MSD radix sot
c) 3 way radix quick sort
d) forward radix sort

Answer: d
Explanation: Forward radix sort combines the qualities of MSD and LSD radix sort. The sorting is done by separating the strings into groups.

4. Which of the following is the most suitable definition of radix sort?
a) It is a non comparison based integer sort
b) It is a comparison based integer sort
c) It is a non comparison based non integer sort
d) It is a comparison based non integer sort

Answer: a
Explanation: Radix sort is a non-comparison based integer sort. It sorts the given data by grouping keys which share the same significant position value.

5. Which of the following is an alternate name of MSD radix sort?
a) bottom up radix sort
b) top down radix sort
c) forward radix sort
d) backward radix sort

Answer: b
Explanation: Top down radix sort is an alternate name of MSD radix sort. It is because in this algorithm the processing starts from the most significant digit and end at least significant digit.

6. Which of the following is not true about MSD radix sort?
a) its processing starts from the most significant digit
b) it is not a stable sort
c) it is an in place sorting algorithm
d) it is non comparison based sort

Answer: c
Explanation: MSD radix sort takes non constant time for sorting the input data. So it is not an in place sorting algorithm. Answer: b
Explanation: MSD radix sort is not a stable sort whereas LSD radix sort is stable. So when we require to preserve the original order then in that case we should prefer LSD radix sort.

8. What is the average time complexity of MSD radix sort (w= bits required to store each key)?
a) O(n + w)
b) O(n.w)
c) $O(n^2)$
d) O(n log n)

Answer: b
Explanation: Time complexity of radix sort is O(n.w). It performs better than quick sort when we have log n bits for every digit.Answer: b
Explanation: MSD radix sort takes non constant time for sorting the input data. So it is not an in place sorting algorithm.

10. Which of the following statement is not a stable sorting algorithm?
a) LSD radix sort
b) MSD radix sort
c) Counting sort
d) Pigeonhole sort

Answer: b
Explanation: MSD radix sort is not a stable sort. It is because the elements with identical values do not appear in the same order in the output array as they were in the input array.

11. Which of the following is not true about radix sort?
a) Radix sort performs better than quick sort when we have log n bits for every digit
b) Radix sort has better cache performance than quick sort
c) Radix sort has higher values of constant factor in asymptotic notation
d) Radix sort takes more space than quick sort

Answer: b
Explanation: Quick sort has a better cache performance than radix sort. Radix sort also takes more space as compared to quick sort.

12. What is the advantage of radix sort over quick sort?
a) radix sort performs better than quick sort when we have log n bits for every digit
b) radix sort has lesser space complexity
c) radix sort is not a comparison based sorting technique
d) radix sort has better cache performance than quick sort

Answer: a
Explanation: Radix sort performs better than quick sort when we have log n bits for every digit. But radix sort takes more space as compared to quick sort.

13. What will be the order of elements of the array arr = {23, 67, 143, 654, 43} after first iteration of MSD sort is complete?
a) 23, 43, 67, 143, 654
b) 23, 67, 43, 143, 654
c) 23, 67, 143, 654, 43
d) 23, 143, 43, 654, 67

Answer: b
Explanation: In the first iteration the array is sorted according to the most significant digit I.e. hundreds place value. So the order of elements will be 23, 67, 43, 143, 654.

1. How many comparisons will be made to sort the array arr={1,5,3,8,2} using counting sort?
a) 5
b) 7
c) 9
d) 0

Answer: d
Explanation: As counting sort is an example of non comparison sort so it is able to sort an array without making any comparison.

2. Which of the following is not an example of non comparison sort?
a) bubble sort
b) counting sort
c) radix sort
d) bucket sort

Answer: a
Explanation: Bubble sort is not an example of non comparison sort as it needs to compare array elements in order to sort an array.

3. Which of the following sorting techniques is most efficient if the range of input data is not significantly greater than a number of elements to be sorted?
a) selection sort
b) bubble sort
c) counting sort
d) insertion sort

Answer: c
Explanation: Time complexity of counting sort is given as O(n+k) where n is the number of input elements and k is the range of input. So if range of input is not significantly larger than number of elements in the array then it proves to be very efficient.

4. What is the auxiliary space requirement of counting sort?
a) O(1)
b) O(n)
c) O(log n)
d) O(n+k) k=range of input

Answer: d
Explanation: Counting sort uses two extra arrays to get the input array sorted. First array is required to store the count of all the elements which fall in the range of input data elements, so its size is k. The second array is required to store the input elements in sorted manner, so its size is n. Thus overall auxiliary space required becomes O(n+k).Answer: b
Explanation: It is possible to extend the counting sort algorithm for negative numbers as well. In such a case we store the minimum element at the 0th index.

6. Which of the following sorting techniques is stable?
a) quick sort

b) counting sort
c) heap sort
d) selection sort

Answer: b
Explanation: Counting sort is an example of stable sorting algorithm as the elements with identical values appear in the same order in the output array as they were in the input array.

7. Which of the following uses the largest amount of auxiliary space for sorting?
a) Bubble sort
b) Counting sort
c) Quick sort
d) Heap sort

Answer: b
Explanation: Counting sort requires auxiliary space of O(n+k) whereas quick sort, bubble sort and heap sort are in place sorting techniques. Thus counting sort requires most auxiliary space.

8. What is the average time complexity of counting sort?
a) O(n)
b) O(n+k) k=range of input
c) O(n^2)
d) O(n log n)

Answer: b
Explanation: Time complexity of counting sort is O(n+k) as counting the occurrence of each element in the input range takes k time and then finding the correct index value of each element in the sorted array takes n time.

9. The complexity of which of the following sorting algorithms remains to be the same in its best, average and worst case?
a) quick sort
b) insertion sort
c) counting sort
d) gnome sort

Answer: c
Explanation: The time complexity of counting sort remains unvaried in all the three cases. It is given by O(n+k).

10. Which of the following statement is true about comparison based sorting?
a) counting sort is a comparison based sort
b) any comparison based sorting can be made stable
c) bubble sort is not a comparison based sort
d) any comparison based sort requires at least O(n^2) time

Answer: b
Explanation: Any comparison based sorting technique can be made stable by considering a position as criteria while making comparisons.Answer: a
Explanation: Counting sort is used as a sub routine for radix sort as it is a stable and non comparison based sorting algorithm.

12. What is the advantage of counting sort over quick sort?
a) counting sort has lesser time complexity when range is comparable to number of input elements
b) counting sort has lesser space complexity
c) counting sort is not a comparison based sorting technique
d) it has no advantage

Answer: a
Explanation: Counting sort is very efficient in the cases where range is comparable to number of input elements as it

performs sorting in linear time.

b)

```
function countingSort(array, k) is
count ← new array of k zeros
for i = 1 to length(array) do
  count[array[i]] ← count[array[i]] + 1
for i = 2 to k do
  count[i] ← count[i] + count[i +1]
for i = length(array) downto 1 do
  output[count[array[i]]] ← array[i]
  count[array[i]] ← count[array[i]] - 1
return output
```

c)

```
function countingSort(array, k) is
count ← new array of k zeros
for i = 1 to length(array) do
  count[array[i]] ← count[array[i]] + 1
for i = 2 to k do
  count[i] ← count[i] + count[i - 1]
for i = length(array) downto 1 do
  output[count[array[i]]] ← array[i]
  count[array[i]] ← count[array[i]] - 1
return output
```

d)

```
function countingSort(array, k) is
count ← new array of k zeros
for i = 1 to length(array) do
  count[array[i]] ← count[array[i]] + 1
for i = 1 to k do
  count[i] ← count[i] + count[i - 1]
for i = length(array) downto 1 do
  output[count[array[i]]] ← array[i]
  count[array[i]] ← count[array[i]] - 1
return output
```

14. What is the disadvantage of counting sort?
a) counting sort has large time complexity
b) counting sort has large space complexity
c) counting sort is not a comparison based sorting technique
d) counting sort cannot be used for array with non integer elements

Answer: d
Explanation: Counting sort can only be used for arrays with integer elements because otherwise array of frequencies cannot be constructed.

15. Which of the following algorithm takes non linear time for sorting?
a) counting sort
b) quick sort
c) bucket sort
d) radix sort

Answer: b
Explanation: Quick sort requires O(n log n) time for sorting so it takes non linear time for sorting whereas counting sort, bucket sort and radix sort sorts in linear time.

1. How many comparisons will be made to sort the array arr={1, 5, 3, 8, 2} using bucket sort?
a) 5
b) 7

c) 9
d) 0

Answer: d
Explanation: As bucket sort is an example of a non-comparison sort so it is able to sort an array without making any comparison. So the answer should be 0.

2. What is the alternate name of bucket sort?
a) group sort
b) radix sort
c) bin sort
d) uniform sort

Answer: c
Explanation: Bucket sort is also known as bin sort. It is an example of a non-comparison sort.

3. Which of the following non-comparison sort can also be considered as a comparison based sort?
a) counting sort
b) MSD radix sot
c) bucket sort
d) pigeonhole sort

Answer: c
Explanation: Bucket sort can also be considered as a comparison based sort. It is because it can also be implemented with comparisons.

4. Which of the following is not true about bucket sort?
a) It is a non comparison based integer sort
b) It is a distribution sort
c) It can also be considered as comparison based sort
d) It is in place sorting algorithm

Answer: d
Explanation: Bucket sort is a non comparison based integer sort. It sorts the given data by distributing the array elements into a number of buckets. It is not an in place sorting technique.

5. Which of the following don't affect the time complexity of bucket sort?
a) algorithm implemented for sorting individual buckets
b) number of buckets used
c) distribution of input
d) input values

Answer: d
Explanation: Time complexity of bucket sort is affected by various factors. These include algorithm implemented for sorting individual buckets, number of buckets used and distribution of input. It doesnot depend on the input value. It can be either positive or negative or zero.

6. Bucket sort is most efficient in the case when _____
a) the input is non uniformly distributed
b) the input is uniformly distributed
c) the input is randomly distributed
d) the input range is large

Answer: b
Explanation: Bucket sort works by distributing the array elements into a number of buckets. So bucket sort is most efficient in the case when the input is uniformly distributed.

7. Bucket sort is a generalization of which of the following sort?

a) LSD radix sort
b) Pigeonhole sort
c) Counting sort
d) MSD radix sort

Answer: b
Explanation: Pigeonhole sort is a particular case of bucket sort. Bucket sort is also closely related to MSD radix sort.

8. What is the worst case time complexity of bucket sort (k = number of buckets)?
a) O(n + k)
b) O(n.k)
c) $O(n^2)$
d) O(n log n)

Answer: c
Explanation: Time complexity of bucket sort is $O(n^2)$ in the worst case. So if bucket sort does not get the inputs in the desired manner then it becomes very inefficient.

9. What is the best time complexity of bucket sort (k= number of buckets)?
a) O(n + k)
b) O(n.k)
c) $O(n^2)$
d) O(n log n)

Answer: a
Explanation: Time complexity of bucket sort is O(n+k). It performs better than any comparison based sort if there is a uniform input distribution.

10. Which of the following is not necessarily a stable sorting algorithm?
a) bucket sort
b) counting sort
c) merge sort
d) pigeonhole sort

Answer: a
Explanation: Bucket sort is not necessarily a stable sorting algorithm. This is because its stability depends on the stability of the algorithm that we have used to sort the individual buckets.Answer: b
Explanation: Bucket sort is not an in place sorting algorithm. It requires an auxiliary space of O(n k) in the worst case.

12. What is the worst space complexity of bucket sort (k = number of buckets)?
a) O(n + k)
b) O(n.k)
c) $O(n^2)$
d) O(n log n)

Answer: b
Explanation: Worst case space complexity of bucket sort is O(n.k). So it is not an in place sorting algorithm.

1. Bead sort is also known as _____
a) gravity sort
b) strand sort
c) abacus sort
d) counting sort

Answer: a
Explanation: Bead sort is also known as gravity sort. It is because this algorithm was designed by keeping the natural phenomenon of falling objects in mind.

2. Which of the following sorting algorithm was inspired by the natural phenomenon of falling objects?
a) bogo sort
b) heap sort
c) bead sort
d) strand sort

Answer: c
Explanation: The algorithm of bead sort was inspired by the natural phenomenon of falling objects. Thus, it is also known by the name of gravity sort.

3. Which of the following sorting algorithm is only applicable to positive integers?
a) quick sort
b) heap sort
c) bead sort
d) strand sort

Answer: c
Explanation: Bead sort algorithm is only applicable to positive integers. This is because it works by placing number of beads equal to key value, in each row.

4. What is the auxiliary space complexity of bead sort?
a) O(n)
b) O(1)
c) O(n^2)
d) O(n log n)

Answer: c
*Explanation: The auxiliary space complexity of bead sort is O(n^2). It is because an array of size maximum_element*n (maximum_element is the maximum element that is present in the array and n is the size of the array) has to be maintained.*

5. Which of the following sorting algorithm is not in place?
a) quick sort
b) bead sort
c) cycle sort
d) heap sort

Answer: b
Explanation: Bead sort has an auxiliary space complexity of O(n2). So it is not an in place sorting algorithm.Answer: b
Explanation: Bead sort is an example of non comparison based sorting algorithm. This is because it does not compare the value of elements present in a list in order to sort them.

7. How many comparisons will be required to sort the array arr={5, 4, 7, 1, 9} using bead sort?
a) 5
b) 4
c) 6
d) 0

Answer: d
Explanation: Bead sort is an example of a non-comparison based sorting algorithm. So no comparison is required to be performed in order to sort the array.

8. What is the average time complexity of bead sort (S = sum of input elements)?
a) O(n)
b) O(S)
c) O(n^2)
d) O(n log n)

Answer: b
Explanation: Average case time complexity of bead sort is O(S). It is because we drop each bead as a separate operation.

9. What is the best case time complexity of bead sort (S = sum of input elements)?
a) O(n)
b) O(S)
c) O(n^2)
d) O(n log n)

Answer: a
Explanation: Best case time complexity of bead sort is O(n). It is when a row of beads is dropped as a distinct operation and since the number of rows is equal to n.

10. What is the worst case time complexity of bead sort (S= sum of input elements)?
a) O(n)
b) O(S)
c) O(n^2)
d) O(n log n)

Answer: b
Explanation: Worst case time complexity of bead sort is O(S). It is because we drop each bead as a separate operation.

b)

```
for (int i = 0; i < n; i++)
{
         int j;
        for (j = 0; j < max; j++);
        //max is the maximum value element of given array a[]
        a[i] = j;
i}
```

c)

```
for (int i = 0; i < n; i++)
{
         int j;
        for (j = 0; j < max && beads[i * max + j]; j++);
        //max is the maximum value element of given array a[]
        a[i] = j;
}
```

d)

```
for (int i = 0; i < n; i++)
{
         int j;
        for (j = 0; j < beads[i * max + j]; j++);
       //max is the maximum value element of given array a[]
        a[j] = i;
}
```

Answer: a
Explanation: Bead sort algorithm is only applicable to positive integers. This is because it works by placing the number of beads equal to key value, in each row.

1. What is the time complexity for a given pancake sort given it undergoes "n" flip operations?
a) O(n)
b) O(n^2)
c) O(n^3)
d) O(2n)

Answer: b
Explanation: Most sorting algorithms try to sort making the least number of comparisons but in pancake sort we try to sort using as few reversals as possible. Because the total number of flip operations performed in a pancake sort is O(n), the overall time complexity is $O(n^2)$.

2. Which operation is most essential to the process of pancake sort?
a) Flip the given data
b) Find the largest of given data
c) Finding the least of given data
d) Inserting something into the given data

Answer: a
Explanation: When we use pancake sort, we sort the array to find the largest, and then flip the array at that point to bring that value to the bottom of the pancake stack. The size of the array that we are dealing with is then reduced and the process continues. Flip operation is the most important function in the pancake sort technique.

3. There is one small error in the following flip routine. Find out which line it is on.

```
1       void flip(int arr[], int i)
2       {
3             int t, init = 0;
4             while (init < i)
5             {
6                   t = arr[init];
7                   arr[i] = arr[init] ;
8                   arr[i] = t;
9                   init++;
10                  i--;
11            }
12      }
```

a) Line 3
b) Line 5
c) Line 7
d) Line 9

Answer: c
Explanation: After initialization of the array titled arr; for each while loop iteration of increasing init, we should make arr[init]=arr[i]. This makes sure that the changes will be made in order to flip the order of the array that was to be flipped. Here in line 7 it has been written in reverse and is incorrect.

4. How many flips does the simplest of pancake sorting techniques require?
a) 3n−3 flips
b) 2n-4 flips
c) 2n-3 flips
d) 3n-2 flips

Answer: c
Explanation: The minimum number of flips required to sort any stack of n pancakes has been shown to lie between 1.087n and 1.636n. using average of that 1.36n and extracting that for values of n>1. We have 1.36, 2.72, 4.08 etc. This matches best with 2n-3 which is equal to 1, 3, 5, 7, 9, etc. An upper bound of 2n-3 comes by iteratively using the next largest element in its correct place using two flips.

5. Pancake Sorting appears in which of the following?
a) Frequency Scaling
b) Storage Virtualization
c) Parallel Processing
d) Neural Networking

Answer: c

Explanation: Pancake Sorting finds application in educational use not to mention parallel processing networks by providing optimal routing algorithms between networks.

6. In addition to the pancake sorting problem, there is the case of the burnt pancake problem in which we are dealing with pancakes (discs) that are burnt on one side only. In this case it is taken that the burnt side must always end up _____
a) Faced down
b) Faced up
c) It doesn't matter
d) Both sides are burnt

Answer: a
Explanation: A varation of this pancake is with burnt pancakes. Here each pancake has a burnt side and all pancakes must, in addition, end up with the burnt side on bottom. It is a more difficult version of the regular pancake problem.

7. In a computational complexity theory, a problem with decision making is said to be NP-complete when it is both in NP and NP-hard. What does NP mean?
a) Non Polynomial time
b) Non-deterministic Probabilistic
c) Non-deterministic Polynomial time
d) Non Probabilistic time

Answer: c
Explanation: Although any given solution to an NP-complete problem can be validated quickly in polynomial time; there is no way to efficiently locate a solution to begin with. The unique characteristic of NP-complete problems is that no fast solution to them is known and hence NP-complete problems are said to be non-deterministic polynomial time.

8. When we realize a specific implementation of a pancake algorithm, every move when we find the greatest of the sized array and flipping can be modeled through _____
a) Combinations
b) Exponential functions
c) Logarithmic functions
d) Permutations

Answer: d
Explanation: Here when we flipping the array or stack, we have to take utmost priority to preserve the order of the list so that that sorting doesnt become invalid. Hence we use permutations, we are ensuring that order matters.

9. The Pancake Problems (1975, 1979, 1973) did NOT involve which of the following people?
a) Bill Gates
b) Jacob Goodman
c) Christos Papadimitriou
d) John Goodman

Answer: d
Explanation: (Jacob Goodman – 1975) What is the maximum number of flips needed to sort a permutation of [n] into ascending order?
(Bill Gates and Christos Papadimitriou – 1979) What is the maximum number of flips needed to sort a signed permutation of [n] into ascending order with all positive signs?

10. There is a one line error in the following routine. Find that line.

```
1.      int Max(int a[], int n)
2.      {
3.              int mi, i;
4.              for (mi = 0, i = 0; i < n; i++)
5.              if (a[i] > a[mi])
6.              mi = i;
7.              return mi;
```

```
8.      }
```

a) Line 2
b) Line 4
c) Line 6
d) Line 5

Answer: b
Explanation: The increment condition in the for loop declaration is incorrect. We should use ++i instead of i++.

1. Odd-even sort is also known as _____
a) stupid sort
b) smart sort
c) brick sort
d) bogo sort

Answer: c
Explanation: Odd-even sort is also known by the name of a brick sort. This algorithm was first proposed by Habermann in 1972 and was initially invented for parallel computation of local interconnection.

2. Odd-even sort is a variation of _____
a) Bubble sort
b) Selection sort
c) Insertion sort
d) Gnome sort

Answer: a
Explanation: Odd-even sort is very similar to bubble sort. It works by applying bubble sort in two phases I.e odd phase and even phase. In odd phase bubble sort is applied on odd indexed elements and in even phase bubble sort is applied on even indexed elements.

3. Auxiliary space requirement of odd-even sort is _____
a) O(n)
b) O(log n)
c) O(1)
d) $O(n^2)$

Answer: c
Explanation: In odd-even sort manipulation is done on the input array itself. So no extra space is required to perform sorting. Thus it requires constant auxiliary space.

4. Which of the following sorting algorithm is NOT stable?
a) Quick sort
b) Brick sort
c) Bubble sort
d) Merge sort

Answer: a
Explanation: Out of the given options quick sort is the only algorithm which is not stable. Brick sort like bubble sort is a stable sorting algorithm.

5. Which of the following sorting algorithm is in place?
a) brick sort
b) merge sort
C) counting sort
D) radix sort

Answer: a

Explanation: Brick sort is an in place sorting technique as it only requires constant auxiliary space for manipulating the input array.Answer: a
Explanation: Odd-even sort compares the value of different elements in the array for sorting. Thus, it is a comparison based sort.

7. Brick sort uses which of the following methods for sorting the input?
a) selection
b) partitioning
c) merging
d) exchanging

Answer: d
Explanation: Brick sort uses the method of exchanging as it swaps the elements which are out of order. This swapping is done in two phases i.e. odd phase and even phase.

8. What is the worst case time complexity of odd-even sort?
a) O(n)
b) O(n log n)
c) $O(n^2)$
d) O(log n)

Answer: c
Explanation: Worst case complexity is observed when the input array is reverse sorted. This is the same as the worst case complexity of bubble sort.

9. What is the best case time complexity of odd-even sort?
a) O(n)
b) O(n log n)
c) $O(n^2)$
d) O(log n)

Answer: a
Explanation: Best case complexity is observed when the input array is already sorted. This is the same as the best case complexity of bubble sort.

10. What is the average case time complexity of odd-even sort?
a) O(n)
b) O(n log n)
c) $O(n^2)$
d) O(log n)

Answer: c
Explanation: Odd-even sort takes $O(n^2)$ time on average as it keeps on applying bubble sort on the elements in two phases until they are sorted.

11. How many odd and even phases are required respectively to sort the given array using odd-even sort.arr= {3,2,3,8,5,6,2,1}.
a) 3,3
b) 4,4
c) 3,4
d) 4,3

Answer: b
Explanation: Odd-even sort applies bubble sort in two phases until the array gets sorted. So here 8 phases will be required in totality to sort the array. Out of these 4 phases will be odd phase and the other 4 will be even phase.

b)

```
void oddEvenSort(int arr[], int n)
{
        bool Sorted = false;
        while (!Sorted)
        {
                Sorted = true;
                for (int i=1; i<n-2; i=i+2)
                {
                        if (arr[i] > arr[i+1])
                        {
                                swap(arr[i], arr[i+1]);
                                Sorted = false;
                        }
                }
                for (int i=0; i<n-2; i=i+2)
                {
                        if (arr[i] > arr[i+1])
                        {
                                swap(arr[i], arr[i+1]);
                                Sorted = false;
                        }
                }
        }
        return;
}
```

c)

```
void oddEvenSort(int arr[], int n)
{
        bool Sorted = false;
        while (!Sorted)
        {
                Sorted = true;
                for (int i=1; i<n-1; i=i+2)
                {
                        if (arr[i] > arr[i+1])
                        {
                                swap(arr[i], arr[i+1]);
                                Sorted = false;
                        }
                }
                for (int i=0; i<n-1; i=i+2)
                {
                        if (arr[i] > arr[i+1])
                        {
                                swap(arr[i], arr[i+1]);
                                Sorted = false;
                        }
                }
        }
        return;
}
```

d)

```
void oddEvenSort(int arr[], int n)
{
        bool Sorted = false;
        while (!Sorted)
        {
                Sorted = true;
                for (int i=1; i<n-1; i=i+2)
                {
                        if (arr[i] < arr[i+1])
                        {
                                swap(arr[i], arr[i+1]);
                                Sorted = true;
                        }
```

```
                }
                for (int i=0; i<n-1; i=i+2)
                {
                        if (arr[i] < arr[i+1])
                        {
                                swap(arr[i], arr[i+1]);
                                Sorted = true;
                        }
                }
        }
        return;
}
```

1. Which one of the following sorting algorithm requires recursion?
a) odd even sort
b) stooge sort
c) selection sort
d) counting sort

Answer: b
Explanation: Stooge sort requires the use of recursion for implementing its algorithm. On the other hand, the sorting algorithms given in the remaining options use iterative methods.

2. What is the recurrence relation for stooge sort?
a) T(n) = 2T(2/3n) + O(n)
b) T(n) = 2T(2/3n) + O(1)
c) T(n) = 3T(2/3n) + O(n)
d) T(n) = 3T(2/3n) + O(1)

Answer: d
Explanation: In stooge sort recursion is applied to 2/3 part of the array 3 times. Rest of the portion of code has a constant time complexity. So the overall recurrence relation becomes T(n) = 3T(2/3n) + O(1).

3. In which of the following case stooge sort is most efficient (in terms of time complexity)?
a) when input array is already sorted
b) when input array is reverse sorted
c) when input array is large
d) it has the same time complexity in any case

Answer: d
Explanation: Stooge sort has the same time complexity under any case. It is given by the recurrence relation T(n) = 3T(2/3n) + O(1).

4. What is the space complexity of stooge sort?
a) O(n)
b) O(1)
c) O(log n)
d) O(n log n)

Answer: a
Explanation: The space complexity of the stooge sort is O(n). It is used to store the input array.

5. What is the first step in the algorithm of stooge sort(after base case)?
a) apply stooge sort on first 2/3 elements of array
b) apply stooge sort on last 2/3 elements of array
c) apply stooge sort on first 1/3 elements of array
d) compare first and last element of the array

Answer: d
Explanation: The first step in the algorithm of stooge sort is to compare the first and last element of the array and switch

them if found out of order. In the second step stooge sort is applied on the first 2/3 elements of the array.Answer: a
Explanation: Stooge sort is an example of a comparison based sorting algorithm. This is because it compares the value of elements present in a list in order to sort them.Answer: b
Explanation: Stooge sort is not a stable sorting algorithm. It is because the elements with identical values do not appear in the same order in the output array as they were in the input array.

8. What is the average time complexity of stooge sort?
a) $O(n^2)$
b) $O(n^3)$
c) $O(n^{2.6})$
d) $O(n^{2.7})$

Answer: d
Explanation: The recurrence relation of stooge sort is given as $T(n) = 3T(2/3n) + O(1)$. It is found to be equal to $O(n^{2.7})$ using the master's theorem.

9. How many recursive statements are used in the algorithm of stooge sort?
a) 0
b) 1
c) 2
d) 3

Answer: d
Explanation: The algorithm of stooge sort uses 3 recursive statements in its algorithm. The first and third recursive statement applies stooge sort to the first 2/3 elements of the array and the second recursive statement applies stooge sort to last 2/3 elements of the array.

10. Which of the following sorting algorithm has the same time complexity in every case?
a) stooge sort
b) strand sort
c) quick sort
d) bubble sort

Answer: a
Explanation: Stooge sort has the same time complexity of $O(n^{2.7})$ in any case. This also shows that it is not an adaptive sorting algorithm.

11. Which of the following sorting algorithm is worst in terms of time complexity?
a) bubble sort
b) selection sort
c) insertion sort
d) stooge sort

Answer: d
Explanation: Stooge sort has a time complexity of $O(n^{2.7})$ which is the worst out of the given options. This shows that stooge sort is even less efficient than bubble sort which is itself considered to be a very inefficient sort.

12. Which of the following is not an adaptive sorting algorithm?
a) insertion sort
b) strand sort
c) stooge sort
d) bubble sort

Answer: c
Explanation: Stooge sort is not an adaptive sorting algorithm. This is because it does not perform better in the case when the array is already/almost sorted.

b)

```
void stooge_sort(int arr[], int l, int r)
{
        if (l >= r)
                return;
        if (arr[l] > arr[r])
                swap(arr[l], arr[h]);
        if (r - l + 1 > =3)
        {
                int p = (r - l + 1) / 3;
                stooge_sort(arr, l, r - p);
                stooge_sort(arr, l + p, r);
                stooge_sort(arr, l, r - p);
        }
}
```

c)

```
void stooge_sort(int arr[], int l, int r)
{
        if (l >= r)
                return;
        if (arr[l] < arr[r])
                swap(arr[l], arr[h]);
        if (r - l + 1 >=3)
        {
                int p = (r - l + 1) / 3;
                stooge_sort(arr, l, r - p);
                stooge_sort(arr, l + p, r);
                stooge_sort(arr, l, r - p);
        }
}
```

d)

```
void stooge_sort(int arr[], int l, int r)
{
        if (l >= r)
                return;
        if (arr[l] > arr[r])
                swap(arr[l], arr[h]);
        if (r - l + 1 > =3)
        {
                int p = (r - l + 1) / 3;
                stooge_sort(arr, l, r - p);
                stooge_sort(arr, l, r - p);
                stooge_sort(arr, l + p, r);
        }
}
```

1. Which of the following is not an alternative name of permutation sort?
a) stupid sort
b) bogo sort
c) donkey sort
d) monkey sort

Answer: c
Explanation: Permutation sort is also known by names like stupid sort, monkey sort, bogo sort, slow sort and shotgun sort. These names are particularly chosen due to its inefficient algorithm.

2. Permutation sort works by _____
a) generating random permutations of its input
b) partitioning the array
c) dividing the value of input elements
d) generating permutations according to the value of first element of array

Answer: a
Explanation: Permutation sort algorithm successively generates permutations of its input. This process is repeated until the sorted version of the array is found.

3. What is the auxiliary space requirement of permutation sort?
a) O(n)
b) O(1)
c) O(log n)
d) O(n log n)

Answer: b
Explanation: Permutation sort algorithm does not require any extra space for sorting the input array. Thus its auxiliary space requirement is O(1).

4. What is the best case time complexity of permutation sort?
a) $O(n^2)$
b) O(n)
c) O(n log n)
d) O(1)

Answer: b
Explanation: Best case time complexity of permutation sort occurs when the input array is already sorted. So in such a case we only need to check whether all the elements are sorted which can be done in O(n) time.

5. What is the worst case time complexity of permutation sort?
a) $O(n^2)$
b) O(n*n!)
c) O(infinity)
d) O(n log n)

Answer: c
Explanation: There is no upper bound to the worst case of this algorithm. It can go on to take a very large amount of time if the array has many elements. So the worst case of this algorithm can be taken as O(infinity).

6. Which of the following sorting algorithm is not stable _____
a) insertion sort
b) bubble sort
c) merge sort
d) bogosort

Answer: d
Explanation: Out of the given algorithms only bogosort is not stable. This is because it creates permutations of the input array in order to obtain the sorted version. So there is no guarantee that the sorted version obtained by such a method gives a stable output.

7. Which of the following is an in-place sorting algorithm?
a) Merge sort
b) Permutation sort
c) Radix sort
d) Counting sort

Answer: b
Explanation: Out of the given algorithms only permutation sort is an in-place sorting algorithm. It is because the permutation sort algorithm does not require any extra space for sorting the input array. Answer: b
Explanation: If we sort an array using sleep sort then there is no guarantee that the output we get is correctly sorted. So even though sleep sort is better than bogosort in time complexity but it cannot be preferred due to its inaccuracy.

9. What is the average case time complexity of permutation sort?

a) $O(n^2)$
b) O(n*n!)
c) O(infinity)
d) O(n log n)

Answer: b
*Explanation: For calculating the average we first need to calculate the number of possible permutations an array of size n can have. This will be equal to n!. As each permutation also needs to be checked whether it is sorted or not so this takes another n time. Thus overall time complexity becomes O(n*n!).*

b)

```
bool isSorted(int a[], int n)
{
    while ( --n > 1 )
        if (a[n] < a[n-1])
            return false;
    return true;
}

void shuffle(int a[], int n)
{
    for (int i=0; i < n; i++)
        swap(a[i], a[rand()%n]);
}

void bogosort(int a[], int n)
{
    while ( !isSorted(a, n) )
        shuffle(a, n);
}
```

c)

```
bool isSorted(int a[], int n)
{
    while ( --n > 1 )
        if (a[n] < a[n-1])
            return true;
    return false;
}

void shuffle(int a[], int n)
{
    for (int i=0; i < n; i++)
        swap(a[i], a[rand()%n]);
}

void bogosort(int a[], int n)
{
    while ( !isSorted(a, n) )
        shuffle(a, n);
}
```

d)

```
bool isSorted(int a[], int n)
{
    while ( --n > 1 )
        if (a[n] > a[n-1])
            return true;
    return false;
}

void shuffle(int a[], int n)
{
    for (int i=0; i < n; i++)
```

```
        swap(a[i], a[rand()%n]);
}

void bogosort(int a[], int n)
{
    while ( !isSorted(a, n) )
        shuffle(a, n);
}
```

1. Which of the following is an advantage of recursive bubble sort over its iterative version?
a) it has better time complexity
b) it has better space complexity
c) it is easy to implement
d) it has no significant advantage

Answer: d
Explanation: Recursive bubble sort has no significant advantage over iterative bubble sort. It is just a different way to implement the same.

2. Bubble sort is also known as _____
a) stupid sort
b) ship sort
c) sinking sort
d) shell sort

Answer: c
Explanation: Bubble sort is also referred to as sinking sort. It continuously compares the value of adjacent elements as it traverses through an array and swaps the elements which are out of order.

3. What will be the recurrence relation of the code of recursive bubble sort?
a) T(n) = 2T(n/2) + n
b) T(n) = 2T(n/2) + c
c) T(n) = T(n-1) + n
d) T(n) = T(n-1) + c

Answer: c
Explanation: The recurrence relation of the code of recursive bubble sort is T(n) = T(n-1) + n. It can be solved by the method of substitution and is found to be equal to n^2.

4. Which of the following sorting algorithm is stable?
a) Selection sort
b) Quick sort
c) Bubble sort
d) Heap sort

Answer: c
Explanation: Out of the given options bubble sort is the only algorithm which is stable. It is because the elements with identical values appear in the same order in the output array as they were in the input array.

5. Which of the following is a variation of bubble sort?
a) selection sort
b) odd even sort
c) cocktail sort
d) stupid sort

Answer: b
Explanation: Odd even sort is a variation of bubble sort. But unlike bubble sort, odd even sort traverses the array in two phases- odd phase and even phase.Answer: a
Explanation: In bubble sort, we need to compare elements in order to find the minimum element in each iteration. So we

can say that it uses comparisons in order to sort the array. Thus it qualifies as a comparison based sort.

7. What is the average case time complexity of recursive bubble sort?
a) O(n)
b) O(n log n)
c) $O(n^2)$
d) O(log n)

Answer: c
Explanation: The overall recurrence relation of recursive bubble sort is given by T(n) = T(n-1) + n. It is found to be equal to $O(n^2)$.

8. What is the best case time complexity of recursive bubble sort?
a) O(n)
b) O(n log n)
c) $O(n^2)$
d) O(log n)

Answer: a
Explanation: The best case time complexity of recursive bubble sort is O(n). It occurs in the case when the input is already/almost sorted.

9. What is the worst case time complexity of recursive bubble sort?
a) O(n)
b) O(n log n)
c) $O(n^2)$
d) O(log n)

Answer: c
Explanation: The overall recurrence relation of recursive bubble sort is given by T(n) = T(n-1) + n. It is found to be equal to $O(n^2)$.

10. What are the number of swaps required to sort the array arr={1, 2, 4, 3, 7, 5, 6} using recursive bubble sort?
a) 0
b) 1
c) 2
d) 3

Answer: d
Explanation: The first swap takes place between 4 and 3 then the second swap takes place between 7 and 5 and then finally 6 and 7 are swapped which sorts our array.

b)

```
if(n < 1)
return;
```

c)

```
if(n == 0)
return;
```

d)

```
if(n == 1)
return;
```

12. What is the auxiliary space complexity of recursive bubble sort?
a) O(n)

b) O(1)
c) O(n log n)
d) O(n^2)

Answer: b
Explanation: The auxiliary space required by recursive bubble sort is O(1). So it qualifies as an in-place sorting algorithm.Answer: a
Explanation: Bubble sort is an adaptive algorithm. It is because the time complexity of the algorithm improves when the input array is almost sorted.

14. Which of the following sorting algorithm is in place?
a) recursive bubble sort
b) merge sort
c) radix sort
d) counting sort

Answer: a
Explanation: Out of the given options recursive bubble sort is the only algorithm which is in place. It is because the auxiliary space required by recursive bubble sort is O(1).

b)

```
If(n == 2)
return;
```

c)

```
void rec_bubbleSort(int arr[], int n)
{
        if (n == 1)
                return;
        for (int i=0; i<n-1; i++)
                if (arr[i] > arr[i+1])
        {
                        swap(arr[i], arr[i+1]);
        }
        rec_bubbleSort(arr, n-1);
}
```

d)

```
void rec_bubbleSort(int arr[], int n)
{
        if (n == 2)
                return;
        for (int i=0; i<n-1; i++)
                if (arr[i] > arr[i+1])
        {
                        swap(arr[i], arr[i+1]);
        }
        rec_bubbleSort(arr, n-1);
}
```

1. Which of the following is an advantage of binary insertion sort over its standard version?
a) it has better time complexity
b) it has better space complexity
c) it makes less number of comparisons
d) it has no significant advantage

Answer: c
Explanation: Binary insertion sort makes less number of comparisons as compared to the standard version of insertion sort. Binary insertion sort makes log n comparisons in the worst case as compared to n comparisons made in the

standard version.Answer: a
Explanation: Binary Insertion sort does not require the entire input data at the beginning itself in order to sort the array. It rather creates a partial solution in every step, so future elements are not required to be considered. Hence it is an online sorting algorithm.

3. How many comparisons will be made in the worst case when an array of size n will be sorted by using a binary insertion sort algorithm?
a) n
b) 1
c) log n
d) n log n

Answer: c
Explanation: Binary insertion sort makes log n comparisons in the worst case. Whereas the standard version makes n comparisons in the worst case.

4. Which of the following sorting algorithm is stable?
a) Selection sort
b) Quick sort
c) Binary insertion sort
d) Heap sort

Answer: c
Explanation: Out of the given options binary insertion sort is the only algorithm which is stable. It is because the elements with identical values appear in the same order in the output array as they were in the input array.

5. Which of the following sorting algorithm uses a binary search?
a) radix sort
b) binary insertion sort
c) odd-even sort
d) bead sort

Answer: b
Explanation: Binary insertion sort makes use of a binary search algorithm. It is used to find the correct index in the array where the element should be inserted.Answer: a
Explanation: In insertion sort, we need to compare elements in order to find the minimum element in each iteration. So we can say that it uses comparisons in order to sort the array. Thus it qualifies as a comparison based sort.

7. What is the average case time complexity of binary insertion sort?
a) O(n)
b) O(n log n)
c) O(n^2)
d) O(log n)

Answer: c
Explanation: The time complexity does not change when we use binary insertion sort in place of standard insertion sort. So the average case time complexity is O(n^2).

8. What is the best case time complexity of binary insertion sort?
a) O(n)
b) O(n log n)
c) O(n^2)
d) O(log n)

Answer: a
Explanation: The best case time complexity of binary insertion sort is O(n). It occurs in the case when the input is already/almost sorted.

9. What is the worst case time complexity of binary insertion sort?
a) O(n)
b) O(n log n)
c) O(n^2)
d) O(log n)

Answer: c
Explanation: The time complexity does not change when we use binary insertion sort in place of standard insertion sort. So the worst case time complexity is O(n^2).

10. Choose the correct statement regarding binary insertion sort?
a) It has a better time complexity as compared to the standard version
b) It has a better space complexity as compared to the standard version
c) it takes less number of comparisons in the best case as compared to the standard version
d) it takes less number of comparisons in the worst case as compared to the standard version

Answer: d
Explanation: Binary insertion sort has the advantage that it takes less number of comparisons in the worst case as compared to the standard version. In the best case, both standard insertion sort and binary insertion sort makes only 1 comparison.

b)

```
If(high<=low)
{
    If(Item>a[low])
    return low+1;
    return low;
}
```

c)

```
If(high>=low)
{
    If(Item<a[low])
    return low+1;
    return low;
}
```

d)

```
If(high<=low)
{
    If(Item<a[low])
    return low;
    return low+1;
}
```

12. What is the auxiliary space complexity of binary insertion sort?
a) O(n)
b) O(1)
c) O(n log n)
d) O(n^2)

Answer: b
Explanation: The auxiliary space required by a binary insertion sort is O(1). So it qualifies as an in place sorting algorithm.

13. Which of the following is an adaptive sorting algorithm?
a) binary insertion sort
b) merge sort

c) heap sort
d) selection sort

Answer: a
Explanation: Binary insertion sort is an adaptive algorithm. It is because the time complexity of the algorithm improves when the input array is almost sorted.

14. Which of the following sorting algorithm is in place?
a) binary insertion sort
b) merge sort
c) radix sort
d) counting sort

Answer: a
Explanation: Out of the given options binary insertion sort is the only algorithm which is in place. It is because the auxiliary space required by recursive bubble sort is O(1).

b)

```
If(high<=low)
{
    If(Item>a[low])
    return low;
    return low+1;
}
```

c)

```
void BinaryinsertionSort(int a[], int n)
{
    int key;

    for (int i = 1; i <=n-1; ++i)
    {
        int j = i - 1;
        key = a[i];
       //function binarySearch() returns the index
        int index = binarySearch(a, key, 0, j);
       //where the key should be inserted
        while (j >= index)
        {
            a[j+1] = a[j];
            j--;
        }
        a[j+1] = key;
    }
}
```

d)

```
void BinaryinsertionSort(int a[], int n)
{
    int key;

    for (int i = 1; i < =n-1; ++i)
    {
        int j = i - 1;
        key = a[i];
        //function binarySearch() returns the index
        int index = binarySearch(a, key, j,0);
        //where the key should be inserted
        while (j >= index)
        {
            a[j+1] = a[j];
            J++;
        }
```

```
        a[j+1] = key;
    }
}
```

1. Which of the following is an advantage of recursive insertion sort over its iterative version?
a) it has better time complexity
b) it has better space complexity
c) it is easy to implement
d) it has no significant advantage

Answer: d
Explanation: Recursive insertion sort has no significant advantage over iterative insertion sort. It is just a different way to implement the same.Answer: a
Explanation: Insertion sort does not require the entire input data in the beginning itself in order to sort the array. It rather creates a partial solution in every step, so future elements are not required to be considered. Hence it is an online sorting algorithm.

3. What will be the recurrence relation of the code of recursive insertion sort?
a) T(n) = 2T(n/2) + n
b) T(n) = 2T(n/2) + c
c) T(n) = T(n-1) + n
d) T(n) = T(n-1) + c

Answer: c
Explanation: The recurrence relation of the code of recursive insertion sort is T(n) = T(n-1) + n. It can be solved by the method of substitution and is found to be equal to n^2.

4. Which of the following sorting algorithm is stable?
a) Selection sort
b) Quick sort
c) Insertion sort
d) Heap sort

Answer: c
Explanation: Out of the given options insertion sort is the only algorithm which is stable. It is because the elements with identical values appear in the same order in the output array as they were in the input array.

5. Which of the following is a variant of insertion sort?
a) selection sort
b) shell sort
c) odd-even sort
d) stupid sort

Answer: b
Explanation: Shell sort is a variation of insertion sort. It has a better running time in practical applications. Answer: a
Explanation: In insertion sort, we need to compare elements in order to find the minimum element in each iteration. So we can say that it uses comparisons in order to sort the array. Thus it qualifies as a comparison based sort.

7. What is the average case time complexity of recursive insertion sort?
a) O(n)
b) O(n log n)
c) $O(n^2)$
d) O(log n)

Answer: c
Explanation: The overall recurrence relation of recursive insertion sort is given by T(n) = T(n-1) + n. It is found to be equal to $O(n^2)$.

8. What is the best case time complexity of recursive insertion sort?
a) O(n)
b) O(n log n)
c) $O(n^2)$
d) O(log n)

Answer: a
Explanation: The best case time complexity of recursive insertion sort is O(n). It occurs in the case when the input is already/almost sorted.

9. What is the worst case time complexity of recursive insertion sort?
a) O(n)
b) O(n log n)
c) $O(n^2)$
d) O(log n)

Answer: c
Explanation: The overall recurrence relation of recursive insertion sort is given by T(n) = T(n-1) + n. It is found to be equal to $O(n^2)$.

10. How many swaps will be required in the worst case to sort an array having n elements using binary insertion sort?
a) n
b) 1
c) n * log n
d) log n

Answer: d
Explanation: In a normal insertion sort at most n comparisons are required to sort the array. But if we also implement the concept of a binary sort in insertion sort then we can sort by having log n comparisons only.

12. What is the auxiliary space complexity of recursive insertion sort?
a) O(n)
b) O(1)
c) O(n log n)
d) $O(n^2)$

Answer: b
Explanation: The auxiliary space required by recursive insertion sort is O(1). So it qualifies as an in place sorting algorithm.

13. Which of the following is an adaptive sorting algorithm?
a) recursive insertion sort
b) merge sort
c) heap sort
d) selection sort

Answer: a
Explanation: Insertion sort is an adaptive algorithm. It is because the time complexity of the algorithm improves when the input array is almost sorted.

14. Which of the following sorting algorithm is in place?
a) recursive insertion sort
b) merge sort
c) radix sort
d) counting sort

Answer: a

Explanation: Out of the given options recursive insertion sort is the only algorithm which is in place. It is because the auxiliary space required by recursive bubble sort is O(1).

b)

```
void RecInsertionSort(int arr[], int n)
{
        if (n <= 1)
                return;
        RecInsertionSort( arr, n-1 );
        int key = arr[n-1];
        int j = n-2;
        while (j >= 0 && arr[j] > key)
        {
                arr[j+1] = arr[j];
                j--;
        }
        arr[j+1] = key;
}
```

c)

```
void RecInsertionSort(int arr[], int n)
{

        if (n < 1)
                return;

        RecInsertionSort( arr, n-1 );

        int key = arr[n-1];
        int j = n-2;

        while (j >= 0 || arr[j] > key)
        {
                arr[j+1] = arr[j];
                j--;
        }
        arr[j] = key;
}
```

d)

```
void RecInsertionSort(int arr[], int n)
{

        if (n <1)
                return;
        RecInsertionSort( arr, n-1 );

        int key = arr[n-1];
        int j = n-2;

        while (j >= 0 && arr[j] > key)
        {
                arr[j+1] = arr[j];
                j--;
        }
        arr[j] = key;
}
```

1. Which of the following data structure is required for the implementation of tree sort?
a) any ordinary tree
b) balanced tree
c) binary search tree
d) unbalanced tree

Answer: c
Explanation: Tree sort uses a binary search tree for sorting the given elements. Tree sort can also be performed by using an unbalanced binary search tree.Answer: a
Explanation: Tree sort does not require the entire input data at the beginning itself in order to sort the array. It rather creates a partial solution in every step, so future elements are not required to be considered. Hence it is an online sorting algorithm.

3. Which of the following traversal in a binary search tree results in a sorted output?
a) in order traversal
b) pre order traversal
c) post order traversal
d) breadth first traversal

Answer: a
Explanation: Tree sort uses a binary search tree for sorting the given elements. First a BST is formed by using the input data elements and then the BST is traversed in an in order fashion which gives a sorted output.

4. Which of the following sorting algorithm is stable?
a) Selection sort
b) Quick sort
c) Tree sort
d) Heap sort

Answer: c
Explanation: Out of the given options Tree sort is the only algorithm which is stable. It is because the elements with identical values appear in the same order in the output array as they were in the input array.

5. Which of the following sorting algorithm uses a binary search tree?
a) radix sort
b) tree sort
c) odd-even sort
d) bead sort

Answer: b
Explanation: Tree sort makes use of a binary search tree. It is because every time when a BST is traversed in an in order fashion it gives a sorted output.

6. Which of the following is a comparison based sort?
a) tree sort
b) radix sort
c) counting sort
d) pigeonhole sort

Answer: a
Explanation: In tree sort, we need to compare elements as we insert them in the binary search tree. Thus it qualifies as a comparison based sort.

7. What is the average case time complexity of tree sort?
a) O(n)
b) O(n log n)
c) $O(n^2)$
d) O(log n)

Answer: b
Explanation: As on an average every element takes log n time for insertion in a binary search tree so for n elements O(n log n) time will be required on an average.

8. What is the best case time complexity of tree sort?

a) O(n)
b) O(n log n)
c) O(n^2)
d) O(log n)

Answer: b
Explanation: The best case time complexity of tree sort is the same as its average case complexity. So best case time complexity is O(n log n).

9. What is the worst case time complexity of tree sort (when implemented with a balanced tree)?
a) O(n)
b) O(n log n)
c) O(n^2)
d) O(log n)

Answer: b
Explanation: The worst case time complexity of tree sort depends on whether the tree used in the implementation is balanced or not. If the tree is balanced then the worst case complexity is O(n log n).

10. What is the worst case time complexity of tree sort (when implemented with an unbalanced tree)?
a) O(n)
b) O(n log n)
c) O(n^2)
d) O(log n)

Answer: c
Explanation: The worst case time complexity of tree sort depends on whether the tree used in the implementation is balanced or not. If the tree is unbalanced then the worst case complexity is O(n^2).

11. What is the auxiliary space complexity of tree sort?
a) O(1)
b) O(n)
c) O(log n)
d) O(n log n)

Answer: b
Explanation: Tree sort requires auxiliary space for maintaining a binary search tree. So the auxiliary space complexity of tree sort is O(n).

12. In which of the following case does a tree sort become adaptive?
a) when implemented with an unbalanced tree
b) when implemented with a balanced tree
c) when implemented with a splay tree as BST
d) when implemented with AVL tree as BST

Answer: c
Explanation: Tree sort becomes an adaptive sort when it is implemented with a splay tree as a BST. In such a case the best case time complexity is better than (n log n).

13. Which of the following is not true about tree sort?
a) it is not an in place sorting algorithm
b) its every implementation is adaptive
c) it requires in order traversal of BST for sorting input elements
d) it is a stable sort

Answer: b
Explanation: Every implementation of tree sort is not adaptive. It becomes adaptive only when implemented with a splay

tree as BST.

14. Which of the following sorting algorithm is not in place?
a) insertion sort
b) quick sort
c) tree sort
d) gnome sort

Answer: c
Explanation: Out of the given options tree sort is the only algorithm which is not in place. It is because the auxiliary space required by tree sort is O(n). Answer: b
Explanation: The time complexity of tree sort is affected when implemented with an unbalanced tree or a balanced tree. In case of a balanced tree it is O(n log n) and in case of unbalanced tree it is $O(n^2)$.

16. Which of the following is not an advantage of tree sort?
a) it has a low space complexity
b) it has good time complexity for balanced BST
c) it is an online sorting algorithm
d) it is stable sorting algorithm

Answer: a
Explanation: Tree sort has a linear time complexity O(n) which makes it inefficient. This the main reason why sorting algorithms like quick sort, heap sort etc. are preferred over it.

17. Which of the following version of tree sort will have the highest worst case time complexity?
a) using AVL tree as BST
b) using red black tree as BST
c) using splay tree as BST
d) using ordinary BST

Answer: d
Explanation: Out of the given options tree sort has the highest worst case time complexity with ordinary BST as it may not be balanced in every case. Whereas all other options have self balancing BST.

1. What is a Rabin and Karp Algorithm?
a) String Matching Algorithm
b) Shortest Path Algorithm
c) Minimum spanning tree Algorithm
d) Approximation Algorithm

Answer: a
Explanation: The string matching algorithm which was proposed by Rabin and Karp, generalizes to other algorithms and for two-dimensional pattern matching problems.

2. What is the pre-processing time of Rabin and Karp Algorithm?
a) Theta(m^2)
b) Theta(mlogn)
c) Theta(m)
d) Big-Oh(n)

Answer: c
Explanation: The for loop in the pre-processing algorithm runs for m(length of the pattern) times. Hence the pre-processing time is Theta(m).Answer: a
Explanation: Rabin Karp Algorithm makes use of elementary theoretic number notions such as the equivalence of two numbers modulo a third number.

4. What is the basic formula applied in Rabin Karp Algorithm to get the computation time as Theta(m)?
a) Halving rule

b) Horner's rule
c) Summation lemma
d) Cancellation lemma

Answer: b
Explanation: The pattern can be evaluated in time Theta(m) using Horner's rule:
p = P[m] + 10(P[m-1] + 10(P[m-2] +…+ 10(P[2]+10P[1])…)).

5. What is the worst case running time of Rabin Karp Algorithm?
a) Theta(n)
b) Theta(n-m)
c) Theta((n-m+1)m)
d) Theta(nlogm)

Answer: c
Explanation: The worst case running time of Rabin Karp Algorithm is Theta(n-m+1)m). We write Theta(n-m+1) instead of Theta(n-m) because there are n-m+1 different values that the given text takes on.Answer: a
Explanation: Since Rabin-Karp algorithm is a pattern detecting algorithm in a text or string, it can be used for detecting plagiarism in a sentence.

b)

```
for i=1 to n
do t0=(dt0 + P[i])mod q
p=(dp+T[i])mod q
```

c)

```
for i=1 to n
do p=(dp + P[i])mod q
t0=(dt0+T[i])mod q
```

d)

```
for i=1 to m
do t0=(dp + P[i])mod q
p=(dt0+T[i])mod q
```

b)

```
for i=1 to m
do p=(dp + P[i])mod q
t0=(dt0+T[i])mod q
```

c)

```
for s=0 to n
                do if p=t0
                        then if P[1..m]=T[s+1..s+m]
                                then print "Pattern occurs with shift" s
```

d)

```
for s=0 to n-m
                do if p=ts
                        then if P[1..m]=T[s+1..s+m]
                                then print "Pattern occurs with shift" s
```

9. What happens when the modulo value(q) is taken large?
a) Complexity increases
b) Spurious hits occur frequently
c) Cost of extra checking is low
d) Matching time increases

Answer: c
Explanation: If the modulo value(q) is large enough then the spurious hits occur infrequently enough that the cost of extra checking is low.

10. Given a pattern of length-5 window, find the suitable modulo value.

```
for s=0 to m
               do if p=ts
                       then if P[1..m]=T[s+1..s+m]
                               then print "Pattern occurs with shift" s
```

a) 13
b) 14
c) 12
d) 11

Answer: a
Explanation: The modulus q is typically chosen as a prime number that is large enough to reduce the complexity when p is very large.

11. Given a pattern of length- 5 window, find the valid match in the given text.

```
for s=0 to n-m
               do if p!=ts
                       then if P[1..m]=T[s+1..s+m]
                               then print "Pattern occurs with shift" s
```

a) 11-16
b) 3-8
c) 13-18
d) 15-20

Answer: c
Explanation: The pattern 2 1 9 3 6 occurs in the text starting from position 13 to 18. In the given pattern value is computed as 12 by having the modulus as 21. The same text string values are computed for each possible position of a 5 length window.

12. Given a pattern of length- 5 window, find the spurious hit in the given text string.

```
                                        4 3 2 5 0
```

a) 6-10
b) 12-16
c) 3-7
d) 13-17

Answer: d
Explanation: The sub string in the range 13-17, 6 7 3 9 9 produces the same value 7 as the given pattern. But the pattern numbers don't match with sub string identified, hence it is a spurious hit.

13. If the expected number of valid shifts is small and modulus is larger than the length of pattern what is the matching time of Rabin Karp Algorithm?
a) Theta(m)
b) Big-Oh(n+m)
c) Theta(n-m)
d) Big-Oh(n)

Answer: b
Explanation: When the number of valid shifts(v) is Big-Oh(1) and q>=m then the matching time is given by O(n)+O(m(v+n/q)) is simplified as O(n+m).

14. What is the basic principle in Rabin Karp algorithm?
a) Hashing
b) Sorting
c) Augmenting
d) Dynamic Programming

Answer: a
Explanation: The basic principle employed in Rabin Karp algorithm is hashing. In the given text every substring is converted to a hash value and compared with the hash value of the pattern.

15. Who created the Rabin Karp Algorithm?
a) Joseph Rabin and Michael Karp
b) Michael Rabin and Joseph Karp
c) Richard Karp and Michael Rabin
d) Michael Karp and Richard Rabin

Answer: c
Explanation: Rabin Karp algorithm was invented by Richard Karp and Michael Rabin in the year 1987 for searching a pattern in the given string.

1. Which of the following is the fastest algorithm in string matching field?
a) Boyer-Moore's algorithm
b) String matching algorithm
c) Quick search algorithm
d) Linear search algorithm

Answer: c
Explanation: Quick search algorithm is the fastest algorithm in string matching field whereas Linear search algorithm searches for an element in an array of elements.

2. Which of the following algorithms formed the basis for the Quick search algorithm?
a) Boyer-Moore's algorithm
b) Parallel string matching algorithm
c) Binary Search algorithm
d) Linear Search algorithm

Answer: a
Explanation: Quick search algorithm was originally formed to overcome the drawbacks of Boyer-Moore's algorithm and also for increased speed and efficiency.

3. What is the time complexity of the Quick search algorithm?
a) O(n)
b) O(log n)
c) O(m+n)
d) O(mn)

Answer: c
Explanation: The time complexity of the Quick search algorithm was found to be O(m+n) and is proved to be faster than Boyer-Moore's algorithm.

4. What character shift tables does quick search algorithm use?
a) good-character shift tables
b) bad-character shift tables
c) next-character shift tables
d) both good and bad character shift tables

Answer: b
Explanation: Quick search algorithm uses only bad character shift tables and it is one of the reasons for its increased

speed than Boyer-Moore's algorithm.

5. What is the space complexity of quick search algorithm?
a) O(n)
b) O(log n)
c) O(m+n)
d) O(mn)

Answer: a
Explanation: The space complexity of quick search algorithm is mathematically found to be O(n) where n represents the input size.Answer: b
Explanation: Quick search algorithm starts searching from the left most character to the right and it uses only bad character shift tables.

7. What character shift tables does Boyer-Moore's search algorithm use?
a) good-character shift tables
b) bad-character shift tables
c) next-character shift tables
d) both good and bad character shift tables

Answer: d
Explanation: Boyer-Moore's search algorithm uses both good and bad character shift tables whereas quick search algorithm uses only bad character shift tables.

8. What is the worst case running time in searching phase of Boyer-Moore's algorithm?
a) O(n)
b) O(log n)
c) O(m+n)
d) O(mn)

Answer: d
Explanation: If the pattern occurs in the text, the worst case running time of Boyer-Moore's algorithm is found to be O(mn).Answer: a
Explanation: During the searching phase, the comparison between pattern and text characters can be done in any order. It has a quadratic worst case behaviour and good practical behaviour.

10. Given input string = "ABCDABCATRYCARCABCSRT" and pattern string = "CAT". Find the first index of the pattern match using quick search algorithm.
a) 2
b) 6
c) 11
d) 14

Answer: b
Explanation: By using quick search algorithm, the given input text string is preprocessed and starts its search from the left most character and finds the first occurrence of the pattern at index=2.

1. Euclid's algorithm is used for finding _____
a) GCD of two numbers
b) GCD of more than three numbers
c) LCM of two numbers
d) LCM of more than two numbers

Answer: a
Explanation: Euclid's algorithm is basically used to find the GCD of two numbers. It cannot be directly applied to three or more numbers at a time.

2. Who invented Euclid's algorithm?

a) Sieve
b) Euclid
c) Euclid-Sieve
d) Gabriel lame

Answer: b
Explanation: Euclid invented Euclid's algorithm. Sieve provided an algorithm for finding prime numbers. Gabriel lame proved a theorem in Euclid's algorithm.

3. If 4 is the GCD of 16 and 12, What is the GCD of 12 and 4?
a) 12
b) 6
c) 4
d) 2

Answer: c
Explanation: Euclid's algorithm states that the GCD of two numbers does not change even if the bigger number is replaced by a difference of two numbers. So, GCD of 16 and 12 and 12 and (16-12)=4 is the same.

4. Which of the following is not an application of Euclid's algorithm?
a) Simplification of fractions
b) Performing divisions in modular arithmetic
c) Solving quadratic equations
d) Solving diophantine equations

Answer: c
Explanation: Solving quadratic equations is not an application of Euclid's algorithm whereas the rest of the options are mathematical applications of Euclid's algorithm.Answer: a
Explanation: The Euclid's algorithm runs efficiently if the remainder of two numbers is divided by the minimum of two numbers until the remainder is zero. This improvement in efficiency was put forth by Gabriel Lame.

6. According to Gabriel lame, how many steps does Euclid's algorithm require to solve a problem?
a) Less than five times the number of digits
b) More than five times the number of digits
c) Less than two times the number of digits
d) More than two times the number of digits

Answer: a
Explanation: The Euclid's algorithm requires less than five times the number of digits. It runs by dividing two numbers. It stops when a remainder zero is reached.

7. Which of the following is the correct mathematical application of Euclid's algorithm?
a) Determination of prime numbers
b) Lagrange's four square theorem
c) Cauchy-Euler theorem
d) Residue theorem

Answer: b
Explanation: Lagrange's four square theorem is one of the mathematical applications of Euclid's algorithm and it is the basic tool for proving theorems in number theory. It can be generalized into other types of numbers like the Gaussian integers.

8. If GCD of two numbers is 1, then the two numbers are said to be _____
a) Co-prime numbers
b) Prime numbers
c) Composite numbers
d) Rational numbers

Answer: a
Explanation: If GCD of two numbers is 1, they are called as co-prime or relatively prime numbers. It does not mean that they are prime numbers. They don't have any prime factors in common.

9. What is the total running time of Euclid's algorithm?
a) O(N)
b) O(N log M)
c) O(N log N)
d) O(log N +1)

Answer: a
Explanation: The total running time of Euclid's algorithm according to Lame's analysis is found to be O(N).Answer: a
Explanation: Euclid's algorithm does not require the calculation of prime factors. We derive the answer straight away using formula. And also, factorization is complex.

11. What is the formula for Euclidean algorithm?
a) GCD (m,n) = GCD (n, m mod n)
b) LCM(m,n)=LCM(n, m mod n)
c) GCD(m,n,o,p) = GCD (m, m mod n, o, p mod o)
d) LCM (m,n,o,p) = LCM (m, m mod n, o, p mod o)

Answer: a
Explanation: The formula for computing GCD of two numbers using Euclidean algorithm is given as GCD (m,n)= GCD (n, m mod n). It is used recursively until zero is obtained as a remainder.

12. What is the total running time of the binary GCD algorithm?
a) O(N)
b) $O(N^2)$
c) O(log N)
d) O(N log N)

Answer: b
Explanation: Binary GCD algorithm is a sub division of Euclidean algorithm with more faster operations. Its running time is given by $O(N^2)$.

13. What is the GCD of 20 and 12 using Euclid's algorithm?
a) 8
b) 2
c) 4
d) 6

Answer: c
Explanation: GCD(m,n)=GCD(n, m mod n)
GCD(20,12)=GCD(12,8)
= GCD(8,4)
= GCD(4,0) = 4.

1. Strassen's algorithm is a/an_____ algorithm.
a) Non- recursive
b) Recursive
c) Approximation
d) Accurate

Answer: b
Explanation: Strassen's Algorithm for matrix multiplication is a recursive algorithm since the present output depends on previous outputs and inputs.

2. What is the running time of Strassen's algorithm for matrix multiplication?

a) $O(n^{2.81})$
b) $O(n^3)$
c) $O(n^{1.8})$
d) $O(n^2)$

Answer: a
Explanation: Strassen's matrix algorithm requires only 7 recursive multiplications of n/2 x n/2 matrix and Theta(n^2) scalar additions and subtractions yielding the running time as $O(n^{2.81})$.

3. What is the running time of naïve matrix multiplication algorithm?
a) $O(n^{2.81})$
b) $O(n^4)$
c) O(n)
d) $O(n^3)$

Answer: d
Explanation: The traditional matrix multiplication algorithm takes $O(n^3)$ time. The number of recursive multiplications involved in this algorithm is 8.

4. Strassen's matrix multiplication algorithm follows _____ technique.
a) Greedy technique
b) Dynamic Programming
c) Divide and Conquer
d) Backtracking

Answer: c
Explanation: Strassen's matrix multiplication algorithm follows divide and conquer technique. In this algorithm the input matrices are divided into n/2 x n/2 sub matrices and then the recurrence relation is applied.

5. The number of scalar additions and subtractions used in Strassen's matrix multiplication algorithm is _____
a) $O(n^{2.81})$
b) Theta(n^2)
c) Theta(n)
d) $O(n^3)$

Answer: b
Explanation: Using Theta(n^2) scalar additions and subtractions, 14 matrices are computed each of which is n/2 x n/2. Then seven matrix products are computed recursively.Answer: a
Explanation: Strassen's Algorithm requires only 7 recursive multiplications when compared with the naïve Theta(n^3) method which reuires 9 recursive multiplications to compute the product.

7. Given the program of naïve method.

```
for i=1 to n do
   for j=1 to n do
       Z[i][j]=0;
       for k=1 to n do
            _____________________________
```

Fill in the blanks with appropriate formula
a) Z[i][j] = Z[i][j] + X[i][k]*Y[k][j]
b) Z[i][j] = Z[i][j] + X[i][k] + Y[k][j]
c) Z[i][j] = Z[i][j] * X[i][k]*Y[k][j]
d) Z[i][j] = Z[i][j] * X[i][k] + Y[k][j]

Answer: a
Explanation: In the naïve method of matrix multiplication the number of iterating statements involved are 3, because of

the presence of rows and columns. The element in each row of one matrix is multiplied with each element in the column of the second matrix. The computed value is placed in the new matrix Z[i][j].

8. Who demonstrated the difference in numerical stability?
a) Strassen
b) Bailey
c) Lederman
d) Higham

Answer: d
Explanation: The difference in the numerical stability was demonstrated by Higham. He overemphasized that Strassen's algorithm is numericaly unstable for some applications.

9. What is the recurrence relation used in Strassen's algorithm?
a) 7T(n/2) + Theta(n^2)
b) 8T(n/2) + Theta(n^2)
c) 7T(n/2) + O(n^2)
d) 8T(n/2) + O(n^2)

Answer: a
Explanation: The recurrence relation used in Strassen's algorithm is 7T(n/2) + Theta(n^2) since there are only 7 recursive multiplications and Theta(n^2) scalar additions and subtractions involved for computing the product.

10. Who discussed techniques for reducing the memory requirements for Strassen's algorithm?
a) Strassen
b) Lederman
c) Bailey
d) Higham

Answer: c
Explanation: The submatrices formed at the levels of recursion consume space. Hence in order to overcome that Bailey discussed techniques for reducing the memory required.

11. What is the formula to calculate the element present in second row, first column of the product matrix?
a) M1+M7
b) M1+M3
c) M2+M4 – M5 + M7
d) M2+M4

Answer: d
Explanation: The element at second row, first column can be found by the formula M2 + M4, where M2 and M4 can be calculated by
M2= (A(2,1) + A(2,2)) B(1,1)
M4=A(2,2)(B(1,2) – B(1,1)).

12. Strassen's Matrix Algorithm was proposed by _____
a) Volker Strassen
b) Andrew Strassen
c) Victor Jan
d) Virginia Williams

Answer: a
Explanation: Strassen's matrix multiplication algorithm was first published by Volker Strassen in the year 1969 and proved that the n^3 general matrix multiplication algorithm wasn't optimal.

13. How many iterating statements are involved in the naïve method of matrix multiplication?
a) 1

b) 2
c) 3
d) 4

Answer: c
Explanation: In the naïve method of matrix multiplication the number of iterating statements involved are 3, because of the presence of rows and columns in a matrix. The element in each row of the first matrix is multiplied with each element in the column of the second matrix. Answer: b
Explanation: Strassen's algorithm is too numerically unstable for some applications. The computed result C=AB satisfies the inequality with a unit roundoff error which corresponds to strong stability inequality(obtained by replacing matrix norms with absolute values of the matrix elements).Answer: d
Explanation: The solution can be obtained by
*C11=1*8 + 3*6 =8+18=26*
*C12=1*4 + 3*2 =4+6=10*
*C21=5*8 + 7*6 =40+42=82*
*C22= 5*4 + 7*2=20+14=34.*

1. What is pseudo random number generator?
a) an algorithm that generates random numbers with help of mathematical formula
b) an algorithm that generates random numbers according to user activity
c) an algorithm that generates random numbers according to time
d) an algorithm that generates random numbers with help of user input

Answer: a
Explanation: A pseudo random number generator generates random numbers with the help of a mathematical formula. We can seed the random number generator with a different value everytime if we want unique random numbers to be generated.

2. What should be the return type of rand() function?
a) int
b) float
c) long
d) double

Answer: a
Explanation: The return type of rand () is int. It can generate random numbers from 0 to RAND_MAX.

3. What is the purpose of using function srand()?
a) to set the seed of rand() function
b) to generate random numbers
c) to enable rand() function
d) to improve efficiency of rand()

Answer: a
Explanation: The function srand() sets the seed of rand(). It can be used to generate a unique set of random numbers every time.

4. What is the range of rand()?
a) 0 to RAND_MAX
b) 0 to infinity
c) 0 to 2147483647
d) 0 to 32767

Answer: a
Explanation: The function rand() generates random numbers in the range 0 to RAND_MAX. The value of RAND_MAX is minimum 32,767.

5.What is the correct formula for generating random numbers in the range (lower,upper) using rand()?

a) rand() % (upper – lower)
b) rand() + lower
c) (rand()%(upper-lower)) + lower
d) (rand()%(upper-lower+1)) + lower

Answer: d
Explanation: The correct formula for generating random numbers in the range (lower,upper) using rand() is (rand()% (upper-lower+1)) + lower. The function rand() generates random numbers in the range 0 to RAND_MAX.

6. Which of the following will generate random numbers in the range 1-100 (both inclusive)?
a) rand() % 100
b) rand() % 101
c) (rand() % (101)) + 1
d) (rand() % (100)) + 1

Answer: d
Explanation: Formula for generating random numbers in the range (lower,upper) using rand() is (rand()%(upper-lower+1)) + lower. So the correct answer will be (rand() % (100)) + 1.

7. What is the minimum value of RAND_MAX possible in any implementation?
a) 0
b) 32767
c) 2147483647
d) 128

Answer: b
Explanation: The value of RAND_MAX varies according to implementation. But it is guaranteed to be at least 32767.Answer: b
Explanation: Function rand() does not generate unique random numbers every time. For achieving this we have to use srand() along with rand().

9. What is the default value of seed if function rand() is called before srand()?
a) srand(0)
b) srand(1)
c) srand(time(null))
d) srand(-1)

Answer: b
Explanation: If srand() is not called before the call to the function rand() then the value of seed is taken as srand(1) by default. We should use srand() before rand() in order to use our own seed.Answer: b
Explanation: Pseudo random number generators cannot be used for data encryption as the random numbers generated by them are predictable. For data encryption the numbers should be absolutely unpredictable.

11. Predict the output of the following code.

```
#include <stdlib.h>
int main()
{
     srand(0);
     printf("%d\n", rand());
     return 0;
}
```

a) compilation error
b) random number between 0 to RAND_MAX
c) cannot be predicted
d) 0

Answer: b

Explanation: The function rand() generates random numbers in the range 0 to RAND_MAX. The function srand() seeds rand(). We get unique set of random numbers if rand() is seeded with a different value every time.

12. Which header file contains the function rand() in C language?
a) stdlib
b) iostream
c) stdio
d) time

Answer: a
Explanation: In C language the header file stdlib contains the function rand(). It generates pseudo random numbers.

13. Predict the output of the following code.

```
#include <stdlib.h>
int main()
{
     srand(0);
     printf("%d\n", rand()%50);
     return 0;
}
```

a) compilation error
b) random number between 0 to 50 (both inclusive)
c) random number between 0 to 51 (both inclusive)
d) random number between 0 to 49 (both inclusive)

Answer: d
Explanation: Formula for generating random numbers in the range (lower,upper) using rand() is (rand()%(upper-lower+1)) + lower. So the given code will generate random numbers between 0 to 49.

b)

```
#include <stdlib.h>
int main()
{
     srand(0);
     printf("%d\n", rand()%50);
     return 0;
}
```

c)

```
#include <stdlib.h>
int main()
{
     srand(time(0));
     printf("%d\n", rand()%50);
     return 0;
}
```

d)

```
#include <stdlib.h>
int main()
{
    srand(1);
    printf("%d\n", rand()%50);
    return 0;
}
```

b)

```
#include <stdlib.h>
```

```
int main()
{
     printf("%d\n", rand()%50);
     return 0;
}
```

c)

```
#include <stdlib.h>
int main()
{
     printf("%d\n", srand());
     return 0;
}
```

d)

```
#include <stdlib.h>
int main()
{
     srand(time(0));
     printf("%d\n", rand()%50);
     return 0;
}
```

1. The dictionary ordering of elements is known as?
a) Lexicographical order
b) Colexicographical order
c) Well order
d) Sorting

Answer: a
Explanation: Lexicographical order is also known as dictionary order. It is a generalized method of the way words are alphabetically ordered in a dictionary.

2. How many permutations will be formed from the array arr={1,2,3}?
a) 2
b) 4
c) 6
d) 8

Answer: c
Explanation: No.of permutations for an array of size n will be given by the formula nPn. So for the given problem, we have 3P3=6 or 3!=6.

3. What will be the lexicographical order of permutations formed from the array arr={1,2,3}?
a) {{2,1,3},{3,2,1},{3,1,2},{2,3,1},{1,2,3},{1,3,2}}
b) {{1,2,3},{1,3,2},{2,3,1},{2,1,3},{3,2,1},{3,1,2}}
c) {{1,2,3},{1,3,2},{2,1,3},{2,3,1},{3,1,2},{3,2,1}}
d) {{2,1,3},{3,1,2},{3,2,1},{2,3,1},{1,2,3},{1,3,2}}

Answer: c
Explanation: The number of permutations for the problem will be 6 according to the formula 3P3. When ordered in lexicographical manner these will be {{1,2,3},{1,3,2},{2,1,3},{2,3,1},{3,1,2},{3,2,1}}.

4. What is the name given to the algorithm depicted in the pseudo code below?

```
procedure generate(n : integer, Arr : array):
    if n = 1 then
          output(Arr)
    else
        for i = 0; i <= n - 2; i ++ do
            generate(n - 1, Arr)
```

```
            if n is even then
                swap(Arr[i], Arr[n-1])
            else
                swap(Arr[0], Arr[n-1])
            end if
        end for
        generate(n - 1, Arr )
    end if
```

a) bubble sort
b) heap sort
c) heap's algorithm
d) prim's algorithm

Answer: c
Explanation: The given algorithm is called Heap's algorithm. It is used for generating permutations of a given list.Answer: b
Explanation: Heap's algorithm does not require any extra array for generating permutations. Thus it is able to keep its space requirement to a very low level. This makes it preferable algorithm for generating permutations.

6. What is the time complexity of Heap's algorithm?
a) O(n log n)
b) O(n^2)
c) O(n*n!)
d) O(n!)

Answer: d
*Explanation: The recurrence relation for the heap's algorithm is given by the expression T(n)=n * T(n-1). It is calculated by using substitution method. It is found to be equal to O(n!).*

7. What will be the output for following code?

```
#include <stdio.h>
#include <string.h>
#include<iostream>
using namespace std;
void swap(char *x, char *y)
{
        char temp;
        temp = *x;
        *x = *y;
        *y = temp;
}

void func(char *a, int l, int r)
{
int i;
if (l == r)
        cout<<a<<" ,";
else
{
        for (i = l; i <= r; i++)
        {
                swap((a+l), (a+i));
                func(a, l+1, r);
                swap((a+l), (a+i));
        }
}
}

int main()
{
        char str[] = "AB";
        int n = strlen(str);
        func(str, 0, n-1);
```

```
        return 0;
}
```

a) AB,BA,
b) BA,AB,
c) AB,BA
d) BA,AB,

Answer: a
Explanation: The given code prints the permutations of the an array. So there will be 2 permutations for an array of 2 elements. Note that the comma is printed even for the last permutation.

8. What will be the time complexity of the given code?

```
#include <stdio.h>
#include <string.h>
#include<iostream>
using namespace std;
void swap(char *x, char *y)
{
        char temp;
        temp = *x;
        *x = *y;
        *y = temp;
}

void func(char *a, int l, int r)
{
int i;
if (l == r)
        cout<<a<<" ,";
else
{
        for (i = l; i <= r; i++)
        {
                swap((a+l), (a+i));
                func(a, l+1, r);
                swap((a+l), (a+i));
        }
}
}

int main()
{
        char str[] = "AB";
        int n = strlen(str);
        func(str, 0, n-1);
        return 0;
}
```

a) O(n^2)
b) O(n * n!)
c) O(n!)
d) O(n log n)

Answer: b
*Explanation: The recurrence relation for the heap's algorithm is given by the expression T(n)=n * T(n-1). But as each permutations takes n time to be printed so the overall complexity will be O(n*n!).*

9. What will be the output of the following code?

```
#include <iostream>
#include<string>
using namespace std;
void func1(string input,string output)
```

```
{
    if(input.length()==0)
    {
        cout<<output<<",";
        return;
    }
    for(int i=0;i<=output.length();i++)
    func1(input.substr(1),output.substr(0,i) + input[0] + output.substr(i));
}

int main()
{
    char str[] = "AB";

        func1(str, "");
    return 0;
}
```

a) AA,BA,
b) AB,BA,
c) BA,AB,
d) AB,AB,

Answer: c
Explanation: The given code prints the permutations of the an array. So there will be 2 permutations for an array of 2 elements. In this code the difference is that BA is printed before AB. Note that the comma is printed even for the last permutation.

10. What will be the output of the following code?

```
#include <stdio.h>
#include <string.h>
#include<iostream>
using namespace std;
void swap(char *x, char *y)
{
        char temp;
        temp = *x;
        *x = *y;
        *y = temp;
}

void func(char *a, int l, int r)
{
int i;
if (l == r)
        cout<<a<<" ,";
else
{
        for (i = l; i <= r; i++)
        {
                swap((a+l), (a+i));
                func(a, l+1, r);
                swap((a+l), (a+i));
        }
}
}

int main()
{
        char str[] = "AA";
        int n = strlen(str);
        func(str, 0, n-1);
        return 0;
}
```

a) AA,

b) AA,AA
c) A,A
d) AA,AA,

Answer: d
Explanation: The given code prints the permutations of the given array but it is not able to handle duplicates due to which it prints 2P2 permutations even in this case. So the output becomes AA,AA,.

11. What will be the output of the code that generates permutations and also has the ability to handle duplicates, for the input str[]="AA"?
a) AA
b) AA,AA
c) A,A
d) A

Answer: a
Explanation: If a code is able to handle duplicates then the two A's are not considered to be different elements due to which only one permutation will be formed. So the output will be AA only.

1. What is meant by the term lexicographical order?
a) dictionary ordering of elements
b) reverse dictionary ordering of elements
c) to sort according to value of first element
d) to sort according to value of last element

Answer: a
Explanation: Lexicographical order is also known as dictionary order. It is a generalized method of the way words are alphabetically ordered in a dictionary.

2. How many combinations of 2 elements will be formed from the array arr={1,2,3}?
a) 1
b) 2
c) 3
d) 4

Answer: c
Explanation: No.of combinations of r elements for an array of size n will be given by the formula nCr. So for the given problem, we have 3C2=3.

3. What will be the lexicographical order of combinations of 2 elements each formed from the array arr={1,2,3}?
a) {{2,1},{3,2},{3,1}}
b) {{1,2},{2,3},{1,3}}
c) {{1,2},{1,3},{2,3}}
d) {{2,1},{3,1},{3,2}}

Answer: c
Explanation: The number of combinations for the problem will be 3 according to the formula 3C2. When ordered in lexicographical manner these will be {{1,2},{1,3},{2,3}}.

4. What will be the auxiliary space requirement (excluding call stack) of the program to print combinations of r elements each from array of size n?
a) O(n*r)
b) O(n/r)
c) O(n)
d) O(r)

Answer: d
Explanation: Code to print combinations will require an auxiliary space of O(r) other than call stack memory. This

memory is required by an array that is used for storing combinations formed.Answer: b
Explanation: Code to print combinations will require an auxiliary space of O(r) other than call stack memory. So it is not an in-place algorithm.

6. What will be the time complexity of the code to print combinations?
a) O(n)
b) O(n^2)
c) O(n log n)
d) O(2^n)

Answer: d
Explanation: The recurrence relation of the code to print combinations will be T(n)=2T(n-1)+k. It is found to be equal to O(2^n).

7. What will be the output for the following code?

```
#include<stdio.h>
void combination(int arr[],int n,int r,int index,int aux[],int i);
void print(int arr[], int n, int r)
{
        int aux[r];
        combination(arr, n, r, 0, aux, 0);
}
void combination(int arr[], int n, int r, int index, int aux[], int i)
{
        if (index == r)
        {
                for (int j=0; j<r; j++)
                        printf("%d ",aux[j]);
                printf(", ");
                return;
        }
        if (i >= n)
                return;
        aux[index] = arr[i];
        combination(arr, n, r, index+1, aux, i+1);
        combination(arr, n, r, index, aux, i+1);
}
int main()
{
        int arr[] = {1, 2,2};
        int r = 2;
        int n = sizeof(arr)/sizeof(arr[0]);
        print(arr, n, r);
        return 0;
}
```

a) 1 2, 1 2, 2 2
b) 1 2, 1 2, 2 2 ,
c) 1 2, 2 1, 2 2 ,
d) 1 2, 2 1, 2 2

Answer: b
Explanation: The given code prints the combinations of the given array in lexicographical order.The given code considers the duplicate 2s as different entities.Note that the comma is printed even for the last combination.

b)

```
#include <stdio.h>
void combination(int arr[], int aux[], int start, int end, int index, int r);
void print(int arr[], int n, int r)
{
        int aux[r];
        combination(arr, aux, 0, n-1, 0, r);
```

```
}
void combination(int arr[], int aux[], int start, int end,
                                        int index, int r)
{
        if (index <= r)
        {
                for (int j=0; j<r; j++)
                        printf("%d ", aux[j]);
                printf(", ");
                return;
        }
        for (int i=start; i<=end && end-i+1 >= r-index; i++)
        {
                aux[index] = arr[i];
                combination(arr, aux, i+1, end, index+1, r);
        }
}
int main()
{
        int arr[] = {1, 2, 3};
        int r = 2;
        int n = sizeof(arr)/sizeof(arr[0]);
        print(arr, n, r);
}
```

c)

```
#include <stdio.h>
void combination(int arr[], int aux[], int start, int end,
                                        int index, int r);
void print(int arr[], int n, int r)
{
        int aux[r];
        combination(arr, aux, 0, n-1, 0, r);
}
void combination(int arr[], int aux[], int start, int end,
                                        int index, int r)
{
        if (index == r)
        {
                for (int j=0; j<r; j++)
                        printf("%d ", aux[j]);
                printf(", ");
                return;
        }
        for (int i=start; i<=end && end-i+1 >= r-index; i++)
        {
                aux[index] = arr[i];
                combination(arr, aux, i+1, end, index+1, r);
        }
}

int main()
{
        int arr[] = {1, 2, 3};
        int r = 2;
        int n = sizeof(arr)/sizeof(arr[0]);
        print(arr, n, r);
}
```

d)

```
#include <stdio.h>
void combination(int arr[], int aux[], int start, int end,
                                        int index, int r);
void print(int arr[], int n, int r)
{
        int aux[r];
        combination(arr, aux, 0, n-1, 0, r);
```

```
}
void combination(int arr[], int aux[], int start, int end,
                                        int index, int r)
{
        if (index == r)
        {
                for (int j=0; j<r; j++)
                        printf("%d ", aux[j]);
                printf(", ");
                return;
        }
        for (int i=start; i<=end ; i++)
        {
                aux[index] = arr[i];
                combination(arr, aux, i+1, end, index+1, r);
        }
}

int main()
{
        int arr[] = {1, 2, 3};
        int r = 2;
        int n = sizeof(arr)/sizeof(arr[0]);
        print(arr, n, r);
}
```

b)

```
#include <stdio.h>
void combination(int arr[], int aux[], int start, int end,
                                        int index, int r);
void print(int arr[], int n, int r)
{
        int aux[r];
        combination(arr, aux, 0, n-1, 0, r);
}
void combination(int arr[], int aux[], int start, int end,
                                        int index, int r)
{
        if (index <= r)
        {
                for (int j=0; j<r; j++)
                        printf("%d ", aux[j]);
                printf(", ");
                return;
        }
        for (int i=start; i<=end; i++)
        {
                aux[index] = arr[i];
                combination(arr, aux, i+1, end, index+1, r);
        }
}

int main()
{
        int arr[] = {1, 2, 3};
        int r = 2;
        int n = sizeof(arr)/sizeof(arr[0]);
        print(arr, n, r);
}
```

c)

```
#include<stdio.h>
void combination(int arr[],int n,int r,int index,int aux[],int i);
void print(int arr[], int n, int r)
{
        int aux[r];
        combination(arr, n, r, 0, aux, 0);
```

```
}
void combination(int arr[], int n, int r, int index, int aux[], int i)
{
        if (index == r)
        {
                for (int j=0; j<r; j++)
                        printf("%d ",aux[j]);
                printf(", ");
                return;
        }
        if (i >= n)
                return;
        aux[index] = arr[i];
        combination(arr, n, r, index+1, aux, i+1);
        combination(arr, n, r, index, aux, i+1);
}

int main()
{
        int arr[] = {1, 2,2};
        int r = 2;
        int n = sizeof(arr)/sizeof(arr[0]);
        print(arr, n, r);
        return 0;
}
```

d)

```
#include<stdio.h>
void combination(int arr[],int n,int r,int index,int aux[],int i);
void print(int arr[], int n, int r)
{
        int aux[r];
        combination(arr, n, r, 0, aux, 0);
}
void combination(int arr[], int n, int r, int index, int aux[], int i)
{
        if (index == r)
        {
                for (int j=0; j<r; j++)
                        printf("%d ",aux[j]);
                printf(", ");
                return;
        }
        if (i >= n)
                return;
        aux[index] = arr[i];
        combination(arr, n, r, index+1, aux, i+1);
        combination(arr, n, r, index+1, aux, i+1);
}
int main()
{
        int arr[] = {1, 2,2};
        int r = 2;
        int n = sizeof(arr)/sizeof(arr[0]);
        print(arr, n, r);
        return 0;
}
```

10. What will be the output for the following code?

```
#include<stdio.h>
void combination(int arr[],int n,int r,int index,int aux[],int i);
void print(int arr[], int n, int r)
{
        int aux[r];
        combination(arr, n, r, 0, aux, 0);
}
void combination(int arr[], int n, int r, int index, int aux[], int i)
```

```
{
        if (index == r)
        {
                for (int j=0; j<r; j++)
                        printf("%d ",aux[j]);
                printf(", ");
                return;
        }
        if (i >= n)
                return;
        aux[index] = arr[i];
        combination(arr, n, r, index-1, aux, i+1);
        combination(arr, n, r, index, aux, i+1);
}
int main()
{
        int arr[] = {1, 2,2};
        int r = 2;
        int n = sizeof(arr)/sizeof(arr[0]);
        print(arr, n, r);
        return 0;
}
```

a) 1 2, 1 3, 2 3
b) 1 3,1 2,2 3,
c) 1 2, 1 3, 2 3,
d) 1 2,1 3,2 3,

Answer: c
Explanation: The given code prints the combinations of the given array in lexicographical order.Note that the comma is printed even for the last combination.

11. What will be the output for the following code?

```
#include<stdio.h>
void combination(int arr[],int n,int r,int index,int aux[],int i);
void print(int arr[], int n, int r)
{
        int aux[r];
        combination(arr, n, r, 0, aux, 0);
}
void combination(int arr[], int n, int r, int index, int aux[], int i)
{
        if (index == r)
        {
                for (int j=0; j<r; j++)
                        printf("%d ",aux[j]);
                printf(", ");
                return;
        }
        if (i >= n)
                return;
        aux[index] = arr[i];
        combination(arr, n, r, index, aux, i+1);
        combination(arr, n, r, index+1, aux, i+1);
}
int main()
{
        int arr[] = {1, 2,2};
        int r = 2;
        int n = sizeof(arr)/sizeof(arr[0]);
        print(arr, n, r);
        return 0;
}
```

a) 1 2, 1 2,2 2,
b) 1 2, 2 1, 2 2,

c) 1 2, 2 2,
d) 1 2, 2 2, 2 1,

Answer: c
Explanation:The given code prints combination of the elements of a given array.But the difference here is that this code also handles duplicate elements due to which we get only 2 combinations instead of 3.

1. What is meant by integer partition?
a) representing an integer as sum of positive and negative real numbers
b) representing an integer as sum of positive and negative integers
c) representing an integer as sum of positive integers
d) representing an integer as sum of positive real numbers

Answer: c
Explanation: Integer partition is the way of representing an integer as sum of positive integers. Partitions differing only in their order are considered to be same.

2. How many partitions will be formed for the integer 3?
a) 2
b) 3
c) 4
d) 8

Answer: b
Explanation: We need to find the combinations of positive integers which give 3 as their sum. These will be {3}, {2,1}, {1,1,1}. Thus the correct answer is 3.

3. What is meant by number theory?
a) study of integers
b) study of complex numbers
c) numerology
d) theory of origination of mathematics

Answer: a
Explanation: Number theory is a branch of mathematics that deals with the study of integers. Partitioning of a number comes under the study of number theory.

4. Which of the following is true according to Ramanujan's congruence?
a) No. of partitions are divisible by 5 for a number 3 more than a multiple of 5
b) No. of partitions are divisible by 5 for a number 4 more than a multiple of 5
c) No. of partitions are divisible by 5 for a number 2 more than a multiple of 5
d) No. of partitions are divisible by 5 for a number 1 more than a multiple of 5

Answer: b
Explanation: Ramanujan's congruence are some relations found for the no. of partitions of an integer. According to it, the number of partitions of an integer is divisible by 5 if that integer is 4 more than a multiple of 5.

5. The no. of partitions of which of the following integer will be divisible by 5?
a) 3
b) 5
c) 9
d) 6

Answer: c
Explanation: According to Ramanujan's congruence number of partitions of an integer is divisible by 5 if that integer is 4 more than a multiple of 5. So out of the given options 9 is the only integer which satisfies this condition.

6. What is the output of the following code?

```
#include<iostream>
using namespace std;
void printArray(int p[], int n)
{
        for (int i = 0; i <= n-1; i++)
        cout << p[i] << " ";
        cout << endl;
}
void func1(int n)
{
        int p[n];
        int k = 0;
        p[k] = n;
        while (true)
        {
                printArray(p, k+1);
                int rem_val = 0;
                while (k >= 0 && p[k] == 1)
                {
                        rem_val += p[k];
                        k--;
                }
                if (k < 0) return;
                p[k]--;
                rem_val++;
                while (rem_val > p[k])
                {
                        p[k+1] = p[k];
                        rem_val = rem_val - p[k];
                        k++;
                }
                p[k+1] = rem_val;
                k++;
        }
}
int main()
{
int n=3;
        func1(n);
        return 0;
}
```

Answer: b
Explanation: Partitions differing in order are considered to be same. Thus 2 1 and 1 2 are considered to be same when we generate partitions of integer 3.

8. What is the approach implemented in the following code?

```
#include<iostream>
using namespace std;
void printArray(int p[], int n)
{
        for (int i = 0; i <= n-1; i++)
        cout << p[i] << " ";
        cout << endl;
}
void func1(int n)
{
        int p[n];
        int k = 0;
        p[k] = n;
        while (true)
        {
                printArray(p, k+1);
                int rem_val = 0;
                while (k >= 0 && p[k] == 1)
                {
                        rem_val += p[k];
                        k--;
```

```
                }
                if (k < 0) return;
                p[k]--;
                rem_val++;
                while (rem_val > p[k])
                {
                        p[k+1] = p[k];
                        rem_val = rem_val - p[k];
                        k++;
                }
                p[k+1] = rem_val;
                k++;
        }
}

int main()
{
      int n;
      cin>>n;
      func1(n);
      return 0;
}
```

a) greedy approach
b) dynamic programming
c) recursion(divide and conquer)
d) backtracking

Answer: b
Explanation: The given code prints the partition of a number in decreasing order. This code uses the approach of dynamic programming. We can also use recursion for fulfilling the same purpose.

9. What will be the output of the following code?

```
#include<iostream>
using namespace std;
int list[200];
void func(int n, int m = 0)
{
    int i;
    if(n == 0)
    {
         for(i = 0; i < m; ++i)
         printf("%d ", list[i]);
         printf("\n");
         return;
    }
    for(i = n; i > 0; --i)
    {
         if(m == 0 || i <= list[m - 1])
         {
             list[m] = i;
             func(n - i, m + 1);
         }
    }

}
int main()
{
        int n=3;
        func(n,0);
        return 0;
}
```

b)

```
#include<iostream>
```

```
using namespace std;
int print[200];
void partitn(int n,int k,int idx)
{

        if(n==0)
        {
                for(int i=0;i<idx;i++)
                        cout<<print[i]<<" ";
                cout<<endl;
                return ;
        }
        for(int i=k;i>0;i--)
        {
                if(i>n)continue;
                print[idx]=i;
                partitn(n-i,i,idx+1);
        }
}
int main()
{
        int n;
        cin>>n;
        partitn(n,n,0);
        return 0;
}
```

c)

```
#include<iostream>
using namespace std;
int print[200];
void partitn(int n,int k,int idx)
{
        if(n==0)
        {
                for(int i=0;i<idx;i++)
                        cout<<print[i]<<" ";
                cout<<endl;
                return ;
        }
        for(int i=k;i>0;i--)
        {
                if(i>n)continue;
                print[idx]=i;
                partitn(n-i,i,idx+1);
        }
}
int main()
{
        int n;
        cin>>n;
        partitn(n,0,0);
        return 0;
}
```

d)

```
#include<iostream>
using namespace std;
int print[200];
void partitn(int n,int k,int idx)
{
        if(n==0)
        {
                for(int i=0;i<idx;i++)
                        cout<<print[i]<<" ";
                cout<<endl;
                return ;
```

```
        }
        for(int i=k;i>0;i--)
        {
                print[idx]=i;
                partitn(n-i,i,idx+1);
        }
}
int main()
{
        int n;
        cin>>n;
        partitn(n,0,0);
        return 0;
}
```

1. What is meant by the power set of a set?
a) subset of all sets
b) set of all subsets
c) set of particular subsets
d) empty set

Answer: b
Explanation: Power set of a set is defined as the set of all subsets. Ex- S={1,2} then P={{},{1},{2}{1,2}}.

2. Number of elements in the power set of set S={1,2,3} will be?
a) 2
b) 4
c) 6
d) 8

Answer: d
Explanation: Power set of a set is defined as the set of all subsets. Number of elements in the power set of a set having n elements is given as 2^n. Thus, here number of elements will be $2^3=8$.

3. Number of elements in the power set of set S={1,2,2} will be?
a) 2
b) 4
c) 6
d) 8

Answer: c
Explanation: For finding the number of elements in the power set of the given set we need to remove duplicates. So we will be left with 6 unique elements which will be P={{},{1},{2},{1,2},{2,2},{1,2,2}}.

4. Choose the correct statement for the following code segment?

```
bool check (int N)
{
    if( N & (1 << i) )
        return true;
    else
        return false;
}
```

a) function returns true if N is odd
b) function returns true if N is even
c) function returns true if ith bit of N is set
d) function returns false if ith bit of N is set

Answer: c
Explanation: As the value of 1 << i is 2^i so the given function checks whether the i^{th} bit of N is set or not. If it is set then the function returns true.

5. What will be the output for the following code?

```
#include <stdio.h>
#include <math.h>
void PowerSet(char *set, int set_size)
{
        unsigned int pow_size = pow(2, set_size);
        int count, j;
        for(count = 0; count < pow_size; count++)
        {
             for(j = 0; j < set_size; j++)
             {

                if(count & (1<<j))
                        printf("%c", set[j]);
             }
             printf(",");
        }
}
int main()
{
        char strset[] = {'a','b','c'};
        PowerSet(strset, 3);
        return 0;
}
```

a) a,b,ab,c,ac,bc,abc,
b) a,b,ab,c,ac,bc,abc
c) ,a,b,ab,c,ac,bc,abc,
d) ,abc,bc,ac,c,ab,b,a,

Answer: c
Explanation: The given code prints the elements of power set of the given set strset[]. It uses binary counter of appropriate length in order to print corresponding subsets of the given set.

6. What will be the time complexity of the following code?

```
#include <stdio.h>
#include <math.h>
void PowerSet(char *set, int set_size)
{
        unsigned int pow_size = pow(2, set_size);
        int count, j;
        for(count = 0; count < pow_size; count++)
        {
             for(j = 0; j < set_size; j++)
             {

                if(count & (1<<j))
                        printf("%c", set[j]);
             }
             printf(",");
        }
}
int main()
{
        char strset[] = {'a','b','c'};
        PowerSet(strset, 3);
        return 0;
}
```

a) O(n 2^n)
b) O(n^2)
c) O(n log n)
d) O(2^n) (n is the size of set)

Answer: a
Explanation: In the given code we have a nested for loop. In this loop the outer loop runs 2^n times and the inner loop runs n times. So the overall time complexity becomes $n2^n$.

7. What will be the auxiliary space requirement of the following code?

```
#include <stdio.h>
#include <math.h>
void PowerSet(char *set, int set_size)
{
        unsigned int pow_size = pow(2, set_size);
        int count, j;
        for(count = 0; count < pow_size; count++)
        {
        for(j = 0; j < set_size; j++)
        {
                if(count & (1<<j))
                        printf("%c", set[j]);
        }
        printf(",");
        }
}
int main()
{
        char strset[] = {'a','b','c'};
        PowerSet(strset, 3);
        return 0;
}
```

a) O(n)
b) O(1)
c) O(n log n)
d) $O(2^n)$ (n is the size of set)

Answer: b
Explanation: The given code prints the elements of power set of the given set strset[]. As this code does not require any extra space for generating the output so its auxiliary space requirement will be O(1).

b)

```
#include<iostream>
using namespace std;
void Set(string str, int index = 0,
                        string curr = "")
{
        int n = str.size();
        if (index == n)
        {
                cout << curr << endl;
                return;
        }
        Set(str, index , curr + str[index]);
        Set(str, index + 1, curr);
}
int main()
{
        string str = "ab";
    Set(str);
        return 0;
}
```

c)

```
#include<iostream>
using namespace std;
void Set(string str, int index = 0,
```

```
                        string curr = "")
{
        int n = str.size();
        if (index == n)
        {
                cout << curr << endl;
                return;
        }
        Set(str, index + 1, curr + str[index]);
        Set(str, index , curr);
}
int main()
{
        string str = "ab";
    Set(str);
        return 0;
}
```

d)

```
#include<iostream>
using namespace std;
void Set(string str, int index = 0,
                        string curr = "")
{
        int n = str.size();
        if (index == n)
        {
                cout << curr << endl;
                return;
        }
        Set(str, index + 1, curr + str[index]);
        Set(str, index + 1, curr);
}
int main()
{
        string str = "ab";
    Set(str);
        return 0;
}
```

Answer: b
Explanation: In case when duplicates are present in the set then the number of elements in the power set decreases. It is because we remove subsets with identical elements.

b)

```
#include<iostream>
using namespace std;
void Set(string str, int index = 0,
                        string curr = "")
{
        int n = str.size();
        if (index == n) {
                cout << curr << endl;
                return;
        }
        Set(str, index , curr+str);
        Set(str, index + 1, curr);
}
int main()
{
        string str = "ab";
    Set(str);
        return 0;
}
```

c)

```
#include <bits/stdc++.h>
using namespace std;
void Set(string str, int ind = -1,
                        string curr = "")
{
        int n = str.size();

        if (ind == n)
                return;
        cout << curr << ",";
        for (int i = ind+ 1; i < n; i++)
        {
                curr += str[i];
            Set(str, i, curr);
                curr.erase(curr.size() - 1);
        }
        return;
}
int main()
{
        string str = "abc";
    Set(str);
        return 0;
}
```

d)

```
#include <bits/stdc++.h>
using namespace std;
void Set(string str, int ind = -1,
                        string curr = "")
{
        int n = str.size();
        if (ind == 0)
                return;
        cout << curr << ",";
        for (int i = ind+ 1; i < n; i++)
        {
                curr += str[i];
            Set(str, i, curr);
                curr.erase(curr.size() - 1);
        }
        return;
}
int main()
{
        string str = "abc";
    Set(str);
        return 0;
}
```

1. Which one of the following problem types does inclusion-exclusion principle belong to?
a) Numerical problems
b) Graph problems
c) String processing problems
d) Combinatorial problems

Answer: d
Explanation: Inclusion-Exclusion principle is a kind of combinatorial problem. It is a counting technique to obtain the number of elements present in sets(two, three , etc.,).

2. Which of the following is a correct representation of inclusion exclusion principle (|A,B| represents intersection of sets A,B)?
a) |A U B|=|A|+|B|-|A,B|
b) |A,B|=|A|+|B|-|A U B|
c) |A U B|=|A|+|B|+|A,B|

d) |A,B|=|A|+|B|+|A U B|

Answer: a
Explanation: The formula for computing the union of two sets according to inclusion-exclusion principle is |A U B|=|A|+|B|-|A,B| where |A,B| represents the intersection of the sets A and B.

3. _____ is one of the most useful principles of enumeration in combinationatorics and discrete probability.
a) Inclusion-exclusion principle
b) Quick search algorithm
c) Euclid's algorithm
d) Set theory

Answer: a
Explanation: Inclusion-exclusion principle serves as one of the most useful principles of enumeration in combinationatorics and discrete probability because it provides simple formula for generalizing results.

4. Which of the following is not an application of inclusion-exclusion principle?
a) Counting intersections
b) Graph coloring
c) Matching of bipartite graphs
d) Maximum flow problem

Answer: d
Explanation: Counting intersections, Graph coloring and Matching of bipartite graphs are all examples of inclusion-exclusion principle whereas maximum flow problem is solved using Ford-Fulkerson algorithm.

5. Who invented the concept of inclusion-exclusion principle?
a) Abraham de Moivre
b) Daniel Silva
c) J.J. Sylvester
d) Sieve

Answer: a
Explanation: The concept of inclusion- exclusion principle was initially invented by Abraham de Moivre in 1718 but it was published first by Daniel Silva in his paper in 1854.Answer: b
Explanation: According to inclusion-exclusion principle, a n-tuple wise intersection is included if n is odd and excluded if n is even.Answer: a
Explanation: The formula for computing the union of three sets using inclusion-exclusion principle is|A U B U C|=|A|+|B|+|C|-|A,B|-|A,C|-|B,C|+|A, B,C| where |A,B|, |B,C|, |A,C|, |A,B,C| represents the intersection of the sets A and B, B and C, A and C, A, B and C respectively.

8. Which of the following statement is incorrect with respect to generalizing the solution using the inclusion-exclusion principle?
a) including cardinalities of sets
b) excluding cardinalities of pairwise intersections
c) excluding cardinalities of triple-wise intersections
d) excluding cardinalities of quadraple-wise intersections

Answer: c
Explanation: According to inclusion-exclusion principle, an intersection is included if the intersecting elements are odd and excluded, if the intersecting elements are even. Hence triple-wise intersections should be included.Answer: a
Explanation: The application of counting intersections can be fulfiled if and only if it is combined with De Morgan laws to count the cardinality of intersection of sets.

10. Using the inclusion-exclusion principle, find the number of integers from a set of 1-100 that are not divisible by 2, 3 and 5.
a) 22
b) 25

c) 26
d) 33

Answer: c
Explanation: Consider sample space S={1,...100}. Consider three subsets A, B, C that have elements that are divisible by 2, 3, 5 respectively. Find integers that are divisible by A and B, B and C, A and C. Then find the integers that are divisible by A, B, C. Applying the inclusion-exclusion principle, 100 − (50 + 33 + 20) + (16 + 10 + 6) − 3 = 26.

11. _____ is an arithmetic function that calculates the total number of positive integers less than or equal to some number n, that are relatively prime to n.
a) Euler's phi function
b) Euler's omega function
c) Cauchy's totient function
d) Legrange's function

Answer: a
Explanation: Euler's phi function is an arithmetic function that calculates the total number of positive integers less than or equal to some number n, that are relatively prime to n. The inclusion-exclusion principle is used to obtain a formula for Euler's phi function.

12. Let A={1,2,3} B={2,3,4} C={1,3,5} D={2,3}. Find the cardinality of sum of all the sets.
a) 6
b) 5
c) 4
d) 7

Answer: b
Explanation: First, include the cardinalities of all the sets. Then, exclude the cardinalities of even intersections. Then include the cardinalities of odd intersections. Hence, 3+3+3+2-2-2-2-1-2-1+1+2+1+1-1=5.

1. The shortest distance between a line and a point is achieved when?
a) a line is drawn at 90 degrees to the given line from the given point
b) a line is drawn at 180 degrees to the given line from the given point
c) a line is drawn at 60 degrees to the given line from the given point
d) a line is drawn at 270 degrees to the given line from the given point

Answer: a
Explanation: The shortest distance between a line and a point is achieved when a line is drawn at 90 degrees to the given line from the given point.

2. What is the shortest distance between the line given by ax + by + c = 0 and the point (x1,y1)?

a)

b)

c)
d) ax1+by1+c

Answer: a
Explanation: The shortest distance between a line and a point is given by the formula $(ax1+by1+c)/(\sqrt{a^2+b^2})$. *This formula can be derived using the formula of area of a triangle.*

3. What is the shortest distance between the line given by -2x + 3y + 4 = 0 and the point (5,6)?
a) 4.5 units

b) 5.4 units
c) 4.3 units
d) 3.3 units

Answer: d
Explanation: The shortest distance between a line and a point is given by the formula $(ax1+by1+c)/(\sqrt{a^2+b^2})$. Using this formula we get the answer 3.3 units.

4. What is the general formula for finding the shortest distance between two parallel lines given by ax+by+c1=0 and ax+by+c2=0?

a)

b)

c)
d) c1+c2

Answer: b
Explanation: The general formula for finding the shortest distance between two parallel lines given by ax+by+c1 and ax+by+c2 is $(c1-c2)/(\sqrt{a^2+b^2})$. We can find this by considering the distance of any one point on one of the line to the other line.

5. What is the distance between the lines 3x-4y+7=0 and 3x-4y+5=0?
a) 1 unit
b) 0.5 unit
c) 0.8 unit
d) 0.4 unit

Answer: d
Explanation: As the given lines are parallel so the distance between them can be calculated by using the formula $(c1-c2)/(\sqrt{a^2+b^2})$. So we get the distance as 0.4 unit.

6. What will be the slope of the line given by ax + by + c = 0?
a) -a/b
b) -b/a
c) -c/a
d) a/c

Answer: a
Explanation: The slope of a line given by the equation ax + by + c=0 has the slope of -a/b. So two lines having the same ratio of the coefficient of x and y will be parallel to each other.

7. What will be the slope of the line given by 10x + 5y + 8=0?
a) -5
b) -2
c) -1.25
d) 5

Answer: b
Explanation: The slope of a line given by the equation ax + by + c=0 has the slope of -a/b. So the slope of the given line will be -2.

8. What will be the co-ordinates of foot of perpendicular line drawn from the point (-1,3) to the line 3x-4y-16=0?
a) (1/5,2/5)

b) (2/25,5/25)
c) (68/25,-49/25)
d) (-49/25,68/25)

Answer: c
Explanation: The foot of perpendicular can be found by equating the distance between the two points and the distance between point and line. This is found to be (68/25,-49/25).

9. Which of the following is used to find the absolute value of the argument in C++?
a) abs()
b) fabs()
c) mod()
d) ab()

Answer: b
Explanation: In C++ the absolute value of an argument can be found by using the function fabs(). It is available under the header file math.h.

10. What will be the slope of the line perpendicular to the line 6x-3y-16=0?
a) 1/2
b) -1/2
c) 2
d) -2

Answer: b
Explanation: For two lines to be perpendicular the product of their slopes should be equal to -1. So as the slope of given line is 2 so the slope of line perpendicular to it will be -1/2.

11. Find the output of the following code.

```
#include<math.h>
#include<iostream>
using namespace std;
void distance(float x, float y, float a, float b, float c)
{
        float d = fabs((a * x + b * y + c)) / (sqrt(a * a + b * b));
        cout<<d;
        return;
}
int main()
{
        float x = -2;
        float y = -3;
        float a = 5;
        float b = -2;
        float c = - 4;
        distance(x, y, a, b, c);
        return 0;
}
```

a) 2.8
b) 1.8
c) 1.4
d) 2.4

Answer: c
Explanation: The given code calculates the shortest distance between line and a point. So the output will be 1.4.

1. Which of the following areas do closest pair problem arise?
a) computational geometry
b) graph colouring problems

c) numerical problems
d) string matching

Answer: a
Explanation: Closest pair problem arises in two most important areas- computational geometry and operational research.

2. Which approach is based on computing the distance between each pair of distinct points and finding a pair with the smallest distance?
a) Brute force
b) Exhaustive search
c) Divide and conquer
d) Branch and bound

Answer: a
Explanation: Brute force is a straight forward approach that solves closest pair problem using that algorithm.

3. What is the runtime efficiency of using brute force technique for the closest pair problem?
a) O(N)
b) O(N log N)
c) $O(N^2)$
d) $O(N^3 \log N)$

Answer: c
Explanation: The efficiency of closest pair algorithm by brute force technique is mathematically found to be $O(N^2)$.

4. The most important condition for which closest pair is calculated for the points (p_i, p_j) is?
a) i>j
b) i!=j
c) i=j
d) i<j

Answer: d
Explanation: To avoid computing the distance between the same pair of points twice, we consider only the pair of points (p_i, p_j) for which $i<j$.

5. What is the basic operation of closest pair algorithm using brute force technique?
a) Euclidean distance
b) Radius
c) Area
d) Manhattan distance

Answer: a
Explanation: The basic operation of closest pair algorithm is Euclidean distance and its formula is given by $d=\sqrt{(x_i-x_j)^2+(y_i-y_j)^2}$.

6. Which of the following is similar to Euclidean distance?
a) Manhattan distance
b) Pythagoras metric
c) Chebyshev distance
d) Heuristic distance

Answer: b
Explanation: In older times, Euclidean distance metric is also called a Pythagoras metric which is the length of the line segment connecting two points.Answer: b
Explanation: Brute force is a straight forward approach to solve critical problems. Here, we use brute force technique to

find the closest distance between p1 and p2.Answer: a
Explanation: Manhattan distance is an alternative way to calculate distance. It is the distance between two points measured along axes at right angles.

9. What is the optimal time required for solving the closest pair problem using divide and conquer approach?
a) O(N)
b) O(log N)
c) O(N log N)
d) O(N^2)

Answer: c
Explanation: The optimal time for solving using a divide and conquer approach is mathematically found to be O(N log N).

10. In divide and conquer, the time is taken for merging the subproblems is?
a) O(N)
b) O(N log N)
c) O(N^2)
d) O(log N)

Answer: b
Explanation: The time taken for merging the smaller subproblems in a divide and conquer approach is mathematically found to be O(N log N).Answer: a
Explanation: The optimal time obtained through divide and conquer approach is the best class efficiency and it is given by Ω(N log N).Answer: b
Explanation: The above diagram depicts the implementation of divide and conquer. The problem is divided into sub problems and are separated by a line.Answer: c
Explanation: From symmetry, we determine that the closest pair is p2 and p3. But the exact calculations have to be done using Euclid's algorithm.

1. Cross product is a mathematical operation performed between _____
a) 2 scalar numbers
b) a scalar and a vector
c) 2 vectors
d) any 2 numbers

Answer: c
Explanation: Cross product is a mathematical operation that is performed on 2 vectors in a 3D plane. It has many applications in computer programming and physics.

2. Cross product is also known as?
a) scalar product
b) vector product
c) dot product
d) multiplication

Answer: b
Explanation: Cross product is also known as a vector product. It is a mathematical operation that is performed on 2 vectors in 3D plane.

3. What is the magnitude of resultant of cross product of two parallel vectors a and b?
a) |a|.|b|
b) |a|.|b| cos(180)
c) |a|.|b| sin(180)
d) 1

Answer: c

Explanation: The resultant of cross product of 2 parallel vectors is always 0 as the angle between them is 0 or 180 degrees. So the answer is |a|.|b| sin(180).

4. What is the general formula for finding the magnitude of the cross product of two vectors a and b with angle θ between them?
a) |a|.|b|
b) |a|.|b| cos(θ)
c) |a|.|b| sin(θ)
d) |a|.|b| tan(θ)

Answer: c
Explanation: The general formula for finding the magnitude of cross product of two vectors is |a|.|b| sin(θ). Its direction is perpendicular to the plane containing a and b.Answer: a
Explanation: The concept of cross product find its application in the field of computer graphics. It can be used to find the winding of polygon about a point.

6. Which of the following equals the a x b (a and b are two vectors)?
a) – (a x b)
b) a.b
c) b x a
d) – (b x a)

Answer: d
Explanation: The vector product a x b is equal to – (b x a). The minus sign shows that these vectors have opposite directions.

7. Cross product of two vectors can be used to find?
a) area of rectangle
b) area of square
c) area of parallelogram
d) perimeter of rectangle

Answer: c
Explanation: Cross product of two vectors can be used to find the area of parallelogram. For this, we need to consider the vectors as the adjacent sides of the parallelogram.

8. The resultant vector from the cross product of two vectors is _____
a) perpendicular to any one of the two vectors involved in cross product
b) perpendicular to the plane containing both vectors
c) parallel to to any one of the two vectors involved in cross product
d) parallel to the plane containing both vectors

Answer: b
Explanation: The resultant vector from the cross product of two vectors is perpendicular to the plane containing both vectors. In other words, it should be perpendicular to both the vectors involved in the cross product.

9. What will be the cross product of the vectors 2i + 3j + k and 3i + 2j + k?
a) i + 2j + k
b) 2i + 3j + k
c) i + j – 5k
d) 2i – j – 5k

Answer: c
Explanation: We can find the cross product of the given vectors by solving the determinant.

10. What will be the cross product of the vectors 2i + 3j + k and 6i + 9j + 3k?
a) i + 2j + k
b) i – j – 5k
c) 0
d) 2i – j – 5k

Answer: c
Explanation: The given vectors are parallel to each other. The cross product of parallel vectors is 0 because sin(0) is 0.

11. Find the output of the following code.

```
#include <bits/stdc++.h>
using namespace std;
void crossP(int A[], int B[], int cross[])
{
        cross[0] = A[1] * B[2] - A[2] * B[1];
        cross[1] = A[0] * B[2] - A[2] * B[0];
        cross[2] = A[0] * B[1] - A[1] * B[0];
}
int main()
{
        int A[] = { 1, 2, 4 };
        int B[] = { 2, 3, 2 };
        int cross[3];
        crossP(A, B, cross);
        for (int i = 0; i < 3; i++)
                cout << cross[i] << " ";
        return 0;
}
```

a) 1 2 5
b) -1 -5 -3
c) -6 -8 -1
d) -8 -6 -1

Answer: d
Explanation: The given code calculates the cross product of the vectors stored in arrays A and B respectively. So the output will be -8 -6 -1.

12. Which of the following operation will give a vector that is perpendicular to both vectors a and b?
a) a x b
b) a.b
c) b x a
d) both a x b and b x a

Answer: d
Explanation: The resultant vector from the cross product of two vectors is perpendicular to the plane containing both vectors. So both a x b and b x a will give a vector that is perpendicular to both vectors a and b.

1. _____ is a method of constructing a smallest polygon out of n given points.
a) closest pair problem
b) quick hull problem
c) path compression
d) union-by-rank

Answer: b

Explanation: Quick hull is a method of constructing a smallest convex polygon out of n given points in a plane.

2. What is the other name for quick hull problem?
a) convex hull
b) concave hull
c) closest pair
d) path compression

Answer: a
Explanation: The other name for quick hull problem is convex hull problem whereas the closest pair problem is the problem of finding the closest distance between two points.

3. How many approaches can be applied to solve quick hull problem?
a) 1
b) 2
c) 3
d) 4

Answer: b
Explanation: Most commonly, two approaches are adopted to solve quick hull problem- brute force approach and divide and conquer approach.

4. What is the average case complexity of a quick hull algorithm?
a) O(N)
b) O(N log N)
c) $O(N^2)$
d) O(log N)

Answer: b
Explanation: The average case complexity of quickhull algorithm using divide and conquer approach is mathematically found to be O(N log N).

5. What is the worst case complexity of quick hull?
a) O(N)
b) O(N log N)
c) $O(N^2)$
d) O(log N)

Answer: c
Explanation: The worst case complexity of quickhull algorithm using divide and conquer approach is mathematically found to be $O(N^2)$.Answer: b
Explanation: The above diagram is a depiction of convex hull, also known as quick hull, since it encloses n points into a convex polygon.

7. Which of the following statement is not related to quickhull algorithm?
a) finding points with minimum and maximum coordinates
b) dividing the subset of points by a line
c) eliminating points within a formed triangle
d) finding the shortest distance between two points

Answer: d
Explanation: Finding the shortest distance between two points belongs to closest pair algorithm while the rest is quickhull.Answer: a
Explanation: It is proved that the quick hull algorithm runs faster if the input uses non-extreme points and also, if it uses less memory.

9. To which type of problems does quick hull belong to?

a) numerical problems
b) computational geometry
c) graph problems
d) string problems

Answer: b
Explanation: Quick hull problem and closest pair algorithms are some of the examples of computational problems.

10. Which of the following algorithms is similar to a quickhull algorithm?
a) merge sort
b) shell sort
c) selection sort
d) quick sort

Answer: d
Explanation: Quickhull algorithm is similar to a quick sort algorithm with respect to the run time average case and worst case efficiencies.

11. Who formulated quick hull algorithm?
a) Eddy
b) Andrew
c) Chan
d) Graham

Answer: a
Explanation: Eddy formulated quick hull algorithm. Graham invented graham scan. Andrew formulated Andrew's algorithm and Chan invented Chan's algorithm.

12. The time is taken to find the 'n' points that lie in a convex quadrilateral is?
a) O(N)
b) O(N log N)
c) O(N^2)
d) O(log N)

Answer: a
Explanation: The time taken to find the 'n' points that lie in a convex quadrilateral is mathematically found to be O(N).

1. Chan's algorithm is used for computing _____
a) Closest distance between two points
b) Convex hull
c) Area of a polygon
d) Shortest path between two points

Answer: b
Explanation: Chan's algorithm is an output-sensitive algorithm used to compute the convex hull set of n points in a 2D or 3D space. Closest pair algorithm is used to compute the closest distance between two points.

2. What is the running time of Chan's algorithm?
a) O(log n)
b) O(n log n)
c) O(n log h)
d) O(log h)

Answer: c
Explanation: The running time of Chan's algorithm is calculated to be O(n log h) where h is the number of vertices of the convex hull.

3. Who formulated Chan's algorithm?

a) Timothy
b) Kirkpatrick
c) Frank Nielsen
d) Seidel

Answer: a
Explanation: Chan's algorithm was formulated by Timothy Chan. Kirkpatrick and Seidel formulated the Kirkpatrick-Seidel algorithm. Frank Nielsen developed a paradigm relating to Chan's algorithm.Answer: a
Explanation: The O(n log h) running time of Chan's algorithm is obtained by combining the running time of Graham's scan [O(n log n)] and Jarvis match [O(nh)].

5. Which of the following is called the "ultimate planar convex hull algorithm"?
a) Chan's algorithm
b) Kirkpatrick-Seidel algorithm
c) Gift wrapping algorithm
d) Jarvis algorithm

Answer: b
Explanation: Kirkpatrick-Seidel algorithm is called as the ultimate planar convex hull algorithm. Its running time is the same as that of Chan's algorithm (i.e.) O(n log h).

6. Which of the following algorithms is the simplest?
a) Chan's algorithm
b) Kirkpatrick-Seidel algorithm
c) Gift wrapping algorithm
d) Jarvis algorithm

Answer: a
Explanation: Chan's algorithm is very practical for moderate sized problems whereas Kirkpatrick-Seidel algorithm is not. Although, they both have the same running time. Gift wrapping algorithm is a non-output sensitive algorithm and has a longer running time.

7. What is the running time of Hershberger algorithm?
a) O(log n)
b) O(n log n)
c) O(n log h)
d) O(log h)

Answer: b
Explanation: Hershberger's algorithm is an output sensitive algorithm whose running time was originally O(n log n). He used Chan's algorithm to speed up to O(n log h) where h is the number of edges.

8. Which of the following statements is not a part of Chan's algorithm?
a) eliminate points not in the hull
b) recompute convex hull from scratch
c) merge previously calculated convex hull
d) reuse convex hull from the previous iteration

Answer: b
Explanation: Chan's algorithm implies that the convex hulls of larger points can be arrived at by merging previously calculated convex hulls. It makes the algorithm simpler instead of recomputing every time from scratch.

9. Which of the following factors account more to the cost of Chan's algorithm?
a) computing a single convex hull
b) locating points that constitute a hull
c) computing convex hull in groups
d) merging convex hulls

Answer: c
Explanation: The majority of the cost of the algorithm lies in the pre-processing (i.e.) computing convex hull in groups. To reduce cost, we reuse convex hulls from previous iterations.Answer: a
Explanation: An extension of Chan's algorithm can be used for proving solutions to complex problems like computing the lower envelope L(S) where S is a set of 'n' line segments in a trapezoid.

1. Depth First Search is equivalent to which of the traversal in the Binary Trees?
a) Pre-order Traversal
b) Post-order Traversal
c) Level-order Traversal
d) In-order Traversal

Answer: a
Explanation: In Depth First Search, we explore all the nodes aggressively to one path and then backtrack to the node. Hence, it is equivalent to the pre-order traversal of a Binary Tree.

2. Time Complexity of DFS is? (V – number of vertices, E – number of edges)
a) O(V + E)
b) O(V)
c) O(E)
d) O(V*E)

Answer: a
Explanation: The Depth First Search explores every node once and every edge once (in worst case), so it's time complexity is O(V + E).

3. The Data structure used in standard implementation of Breadth First Search is?
a) Stack
b) Queue
c) Linked List
d) Tree

Answer: a
Explanation: The Depth First Search is implemented using recursion. So, stack can be used as data structure to implement depth first search.

4. The Depth First Search traversal of a graph will result into?
a) Linked List
b) Tree
c) Graph with back edges
d) Array

Answer: b
Explanation: The Depth First Search will make a graph which don't have back edges (a tree) which is known as Depth First Tree.

5. A person wants to visit some places. He starts from a vertex and then wants to visit every vertex till it finishes from one vertex, backtracks and then explore other vertex from same vertex. What algorithm he should use?
a) Depth First Search
b) Breadth First Search
c) Trim's algorithm
d) Kruskal's Algorithm

Answer: a
Explanation: This is the definition of the Depth First Search. Exploring a node, then aggressively finding nodes till it is not able to find any node.

6. Which of the following is not an application of Depth First Search?

a) For generating topological sort of a graph
b) For generating Strongly Connected Components of a directed graph
c) Detecting cycles in the graph
d) Peer to Peer Networks

Answer: d
Explanation: Depth First Search is used in the Generation of topological sorting, Strongly Connected Components of a directed graph and to detect cycles in the graph. Breadth First Search is used in peer to peer networks to find all neighbourhood nodes.

7. When the Depth First Search of a graph is unique?
a) When the graph is a Binary Tree
b) When the graph is a Linked List
c) When the graph is a n-ary Tree
d) When the graph is a ternary Tree

Answer: b
Explanation: When Every node will have one successor then the Depth First Search is unique. In all other cases, when it will have more than one successor, it can choose any of them in arbitrary order.

8. Regarding implementation of Depth First Search using stacks, what is the maximum distance between two nodes present in the stack? (considering each edge length 1)
a) Can be anything
b) 0
c) At most 1
d) Insufficient Information

Answer: a
Explanation: In the stack, at a time, there can be nodes which can differ in many levels. So, it can be the maximum distance between two nodes in the graph.

9. In Depth First Search, how many times a node is visited?
a) Once
b) Twice
c) Equivalent to number of indegree of the node
d) Thrice

Answer: c
Explanation: In Depth First Search, we have to see whether the node is visited or not by it's ancestor. If it is visited, we won't let it enter it in the stack.

1. Which of the following data structure is used to implement DFS?
a) linked list
b) tree
c) stack
d) queue

Answer: c
Explanation: Stack is used in the standard implementation of depth first search. It is used to store the elements which are to be explored.

2. Which of the following traversal in a binary tree is similar to depth first traversal?
a) level order
b) post order
c) pre order
d) in order

Answer: c

Explanation: In DFS we keep on exploring as far as possible along each branch before backtracking. It terminates when all nodes are visited. So it is similar to pre order traversal in binary tree.Answer: b
Explanation: Depth first search is similar to pre order traversal in a tree. So here we will get the same result as for the pre order traversal (root,left right).Answer: a
Explanation: As 1 is the source element so it will be considered first. Then we start exploring the vertices which are connected to 1. So there will be two possible results-1 2 3 4 5 and 1 4 5 2 3.

b)

```
procedure DFS-non_recursive(G,v):
  //let St be a stack
  St.push(v)
  while St is not empty
    v = St.pop()
    if v is not discovered:
      label v as discovered
      for all adjacent vertices of v do
        St.push(a) //a being the adjacent vertex
```

c)

```
procedure DFS-non_recursive(G,v):
  //let St be a stack
  St.pop()
  while St is not empty
    v = St.push(v)
    if v is not discovered:
      label v as discovered
      for all adjacent vertices of v do
        St.push(a) //a being the adjacent vertex
```

d)

```
procedure DFS-non_recursive(G,v):
  //let St be a stack
  St.push(v)
  while St is not empty
    v = St.pop()
    if v is not discovered:
      label v as discovered
      for all adjacent vertices of v do
        St.push(v)
```

6. What will be the time complexity of the iterative depth first traversal code(V=no. of vertices E=no.of edges)?
a) O(V+E)
b) O(V)
c) O(E)
d) O(V*E)

Answer: a
Explanation: As the time required to traverse a full graph is V+E so its worst case time complexity becomes O(V+E). The time complexity of iterative and recursive DFS are same.

b)

```
procedure DFS-non_recursive(G,v):
  //let St be a stack
  St.pop(v)
  while St is not empty
    v = St.pop()
    if v is not discovered:
      label v as discovered
      for all adjacent vertices of v do
        St.push(a) //a being the adjacent vertex
```

c)

```
void DFS(int s)
{
    vector<bool> discovered(V, true);
    stack<int> st;
    st.push(s);
    while (!st.empty())
    {
        s = st.top();
        st.pop();
        if (!discovered[s])
        {
            cout << s << " ";
            discovered[s] = true;
        }
        for (auto i = adjacent[s].begin(); i != adjacent[s].end(); ++i)
            if (!discovered[*i])
                st.push(*i);
    }
}
```

d)

```
void DFS(int s)
{
    vector<bool> discovered(V, false);
    stack<int> st;
    st.push(s);
    while (!st.empty())
    {
        s = st.top();
        st.pop();
        if (!discovered[s])
        {
            cout << s << " ";
            discovered[s] = true;
        }
        for (auto i = adjacent[s].begin(); i != adjacent[s].end(); ++i)
            if (!discovered[*i])
                st.push(*i);
    }
}
```

8. What is the space complexity of standard DFS(V: no. of vertices E: no. of edges)?
a) O(V+E)
b) O(V)
c) O(E)
d) O(V*E)

Answer: b
Explanation: In the worst case the space complexity of DFS will be O(V) in the case when all the vertices are stored in stack. This space complexity is excluding the space required to store the graph.

9. Which of the following data structure is used to implement BFS?
a) linked list
b) tree
c) stack
d) queue

Answer: d
Explanation: Queue is used in the standard implementation of breadth first search. It is used to store the vertices according to the code algorithm.

10. Choose the incorrect statement about DFS and BFS from the following?

a) BFS is equivalent to level order traversal in trees
b) DFS is equivalent to post order traversal in trees
c) DFS and BFS code has the same time complexity
d) BFS is implemented using queue

Answer: b
Explanation: DFS is equivalent to pre order traversal in trees, not post order traversal. It is so because in DFS we keep on exploring as far as possible along each branch before backtracking. So it should be equivalent to pre order traversal.

1. Breadth First Search is equivalent to which of the traversal in the Binary Trees?
a) Pre-order Traversal
b) Post-order Traversal
c) Level-order Traversal
d) In-order Traversal

Answer: c
Explanation: The Breadth First Search Algorithm searches the nodes on the basis of level. It takes a node (level 0), explores it's neighbors (level 1) and so on.

2. Time Complexity of Breadth First Search is? (V – number of vertices, E – number of edges)
a) O(V + E)
b) O(V)
c) O(E)
d) O(V*E)

Answer: a
Explanation: The Breadth First Search explores every node once and every edge once (in worst case), so it's time complexity is O(V + E).

3. The Data structure used in standard implementation of Breadth First Search is?
a) Stack
b) Queue
c) Linked List
d) Tree

Answer: b
Explanation: The Breadth First Search explores every node once and put that node in queue and then it takes out nodes from the queue and explores it's neighbors.

4. The Breadth First Search traversal of a graph will result into?
a) Linked List
b) Tree
c) Graph with back edges
d) Arrays

Answer: b
Explanation: The Breadth First Search will make a graph which don't have back edges (a tree) which is known as Breadth First Tree.

5. A person wants to visit some places. He starts from a vertex and then wants to visit every place connected to this vertex and so on. What algorithm he should use?
a) Depth First Search
b) Breadth First Search
c) Trim's algorithm
d) Kruskal's algorithm

Answer: b
Explanation: This is the definition of the Breadth First Search. Exploring a node, then it's neighbors and so on.

6. Which of the following is not an application of Breadth First Search?
a) Finding shortest path between two nodes
b) Finding bipartiteness of a graph
c) GPS navigation system
d) Path Finding

Answer: d
Explanation: Breadth First Search can be applied to Bipartite a graph, to find the shortest path between two nodes, in GPS Navigation. In Path finding, Depth First Search is used.

7. When the Breadth First Search of a graph is unique?
a) When the graph is a Binary Tree
b) When the graph is a Linked List
c) When the graph is a n-ary Tree
d) When the graph is a Ternary Tree

Answer: b
Explanation: When Every node will have one successor then the Breadth First Search is unique. In all other cases, when it will have more than one successor, it can choose any of them in arbitrary order.

8. Regarding implementation of Breadth First Search using queues, what is the maximum distance between two nodes present in the queue? (considering each edge length 1)
a) Can be anything
b) 0
c) At most 1
d) Insufficient Information

Answer: c
Explanation: In the queue, at a time, only those nodes will be there whose difference among levels is 1. Same as level order traversal of the tree.

9. In BFS, how many times a node is visited?
a) Once
b) Twice
c) Equivalent to number of indegree of the node
d) Thrice

Answer: c
Explanation: In Breadth First Search, we have to see whether the node is visited or not by it's ancestor. If it is visited, we won't let it enter it in the queue.Answer: a
Explanation: Best First Search is a searching algorithm used in graphs. It explores it by choosing a node by heuristic evaluation rule. It is used in solving searching for related problems.

2. Who described this Best First Search algorithm using heuristic evaluation rule?
a) Judea Pearl
b) Max Bezzel
c) Franz Nauck
d) Alan Turing

Answer: a
Explanation: The best first search algorithm using heuristic evaluation rule or function was proposed by an Israeli – American computer scientist and philosopher Judea Pearl.

3. Which type of best first search algorithm was used to predict the closeness of the end of path and its solution?
a) Greedy BFS
b) Divide and Conquer
c) Heuristic BFS
d) Combinatorial

Answer: a
Explanation: The greedy best first search algorithm was used to predict the closeness of the end of the path and its solution by some of the computer scientists.

4. What is the other name of the greedy best first search?
a) Heuristic Search
b) Pure Heuristic Search
c) Combinatorial Search
d) Divide and Conquer Search

Answer: b
Explanation: The greedy best first search algorithm was used to predict the closeness of the end of the path and its solution by some of the computer scientists. It is also known as Pure Heuristic Search.

5. Which algorithm is used in graph traversal and path finding?
a) A*
b) C*
c) D*
d) E*

Answer: a
Explanation: In computer science A algorithm is used in graph traversal and path finding. It is a process of node finding in between a path. It is an example of the best first search.*

6. Which algorithm is used to find the least cost path from source node to destination node?
a) A* BFS
b) C* BFS
c) D* BFS
d) B* BFS

Answer: d
Explanation: In computer science, B algorithm is used to find the least cost path between the source node and the destination node. It is an example of the best first search.*

7. Which of the following is an example of Best First Search algorithm?
a) A*
b) B*
c) C*
d) Both A* and B*

Answer: d
Explanation: In computer science, A algorithm is used in graph traversal and path finding. It is a process of node finding in between a path. B* algorithm is used to find the least cost path between the source node and the destination node.*

8. Which of the following is the greedy best first search?
a) Pure Heuristic Search
b) A*
c) B*
d) Both A* and B*

Answer: a
Explanation: Pure Heuristic Search is also called greedy best first search while A and B* search algorithms are not greedy best first search.*

9. Which of the following scientists didn't publish A* algorithm?
a) Peter Hart
b) Nils Nilsson

c) Bertram Raphael
d) Hans Berliner

Answer: d
Explanation: Peter Hart Nils Nilsson Bertram Raphael are the three scientists of SRI International who first published the A search algorithm which uses heuristics for better performance. Hans Berliner published B* algorithm.*

10. Who published the B* search algorithm?
a) Peter Hart
b) Nils Nilsson
c) Bertram Raphael
d) Hans Berliner

Answer: d
Explanation: Hans Berliner was a Computer Science professor who first published the B search algorithm in 1979. While Peter Hart Nils Nilsson Bertram Raphael are the three scientists of SRI International who first published the A* search algorithm which uses heuristics for better performance.*

1. Branch and bound is a _____
a) problem solving technique
b) data structure
c) sorting algorithm
d) type of tree

Answer: a
Explanation: Branch and bound is a problem solving technique generally used for solving combinatorial optimization problems. Branch and bound helps in solving them faster.

2. Which of the following is not a branch and bound strategy to generate branches?
a) LIFO branch and bound
b) FIFO branch and bound
c) Lowest cost branch and bound
d) Highest cost branch and bound

Answer: d
Explanation: LIFO, FIFO and Lowest cost branch and bound are different strategies to generate branches. Lowest cost branch and bound helps us find the lowest cost path.

3. Which data structure is used for implementing a LIFO branch and bound strategy?
a) stack
b) queue
c) array
d) linked list

Answer: a
Explanation: Stack is the data structure is used for implementing LIFO branch and bound strategy. This leads to depth first search as every branch is explored until a leaf node is discovered.

4. Which data structure is used for implementing a FIFO branch and bound strategy?
a) stack
b) queue
c) array
d) linked list

Answer: b
Explanation: Queue is the data structure is used for implementing FIFO branch and bound strategy. This leads to breadth first search as every branch at depth is explored first before moving to the nodes at greater depth.

5. Which data structure is most suitable for implementing best first branch and bound strategy?
a) stack
b) queue
c) priority queue
d) linked list

Answer: c
Explanation: Priority Queue is the data structure is used for implementing best first branch and bound strategy. Dijkstra's algorithm is an example of best first search algorithm.

6. Which of the following branch and bound strategy leads to breadth first search?
a) LIFO branch and bound
b) FIFO branch and bound
c) Lowest cost branch and bound
d) Highest cost branch and bound

Answer: b
Explanation: LIFO, FIFO and Lowest cost branch and bound are different strategies to generate branches. FIFO branch and bound leads to breadth first search.

7. Which of the following branch and bound strategy leads to depth first search?
a) LIFO branch and bound
b) FIFO branch and bound
c) Lowest cost branch and bound
d) Highest cost branch and bound

Answer: a
Explanation: LIFO, FIFO and Lowest cost branch and bound are different strategies to generate branches. LIFO branch and bound leads to depth first search.Answer: b
Explanation: FIFO branch and bound leads to breadth first search. Whereas backtracking leads to depth first search.Answer: a
Explanation: Both backtrackings as well as branch and bound are problem solving algorithms. Both LIFO branch and bound strategy and backtracking leads to depth first search.

10. Choose the correct statement from the following.
a) branch and bound is more efficient than backtracking
b) branch and bound is not suitable where a greedy algorithm is not applicable
c) branch and bound divides a problem into at least 2 new restricted sub problems
d) backtracking divides a problem into at least 2 new restricted sub problems

Answer: c
Explanation: Both backtracking as well as branch and bound are problem solving algorithms. Branch and bound is less efficient than backtracking. Branch and bound divides a problem into at least 2 new restricted sub problems.

11. Which of the following can traverse the state space tree only in DFS manner?
a) branch and bound
b) dynamic programming
c) greedy algorithm
d) backtracking

Answer: d
Explanation: Both backtracking as well as branch and bound are problem solving algorithms. Branch and bound can traverse in DFS as well as BFS manner whereas backtracking only traverses in DFS manner.

1. Which of the following is false in the case of a spanning tree of a graph G?
a) It is tree that spans G
b) It is a subgraph of the G
c) It includes every vertex of the G

d) It can be either cyclic or acyclic

Answer: d
Explanation: A graph can have many spanning trees. Each spanning tree of a graph G is a subgraph of the graph G, and spanning trees include every vertex of the gram. Spanning trees are always acyclic.Answer: b
Explanation: Minimum spanning tree is a spanning tree with the lowest cost among all the spacing trees. Sum of all of the edges in the spanning tree is the cost of the spanning tree. There can be many minimum spanning trees for a given graph.

3. Consider a complete graph G with 4 vertices. The graph G has ____ spanning trees.
a) 15
b) 8
c) 16
d) 13

Answer: c
Explanation: A graph can have many spanning trees. And a complete graph with n vertices has $n^{(n-2)}$ spanning trees. So, the complete graph with 4 vertices has $4^{(4-2)} = 16$ spanning trees.

4. The travelling salesman problem can be solved using _____
a) A spanning tree
b) A minimum spanning tree
c) Bellman – Ford algorithm
d) DFS traversal

Answer: b
Explanation: In the travelling salesman problem we have to find the shortest possible route that visits every city exactly once and returns to the starting point for the given a set of cities. So, travelling salesman problem can be solved by contracting the minimum spanning tree.Answer: c
Explanation: Here all non-diagonal elements in the adjacency matrix are 1. So, every vertex is connected every other vertex of the graph. And, so graph M has 3 distinct minimum spanning trees.

6. Consider a undirected graph G with vertices { A, B, C, D, E}. In graph G, every edge has distinct weight. Edge CD is edge with minimum weight and edge AB is edge with maximum weight. Then, which of the following is false?
a) Every minimum spanning tree of G must contain CD
b) If AB is in a minimum spanning tree, then its removal must disconnect G
c) No minimum spanning tree contains AB
d) G has a unique minimum spanning tree

Answer: c
Explanation: Every MST will contain CD as it is smallest edge. So, Every minimum spanning tree of G must contain CD is true. And G has a unique minimum spanning tree is also true because the graph has edges with distinct weights. So, no minimum spanning tree contains AB is false.Answer: a
Explanation: A subgraph is a graph formed from a subset of the vertices and edges of the original graph. And the subset of vertices includes all endpoints of the subset of the edges. So, we can say MST of a graph is a subgraph when all weights in the original graph are positive.Answer: c
Explanation: The minimum spanning tree of the given graph is shown below. It has cost 56.

9. Which of the following is not the algorithm to find the minimum spanning tree of the given graph?
a) Boruvka's algorithm
b) Prim's algorithm
c) Kruskal's algorithm
d) Bellman–Ford algorithm

Answer: d
Explanation: The Boruvka's algorithm, Prim's algorithm and Kruskal's algorithm are the algorithms that can be used to find the minimum spanning tree of the given graph.
The Bellman-Ford algorithm is used to find the shortest path from the single source to all other vertices.

10. Which of the following is false?
a) The spanning trees do not have any cycles
b) MST have n – 1 edges if the graph has n edges
c) Edge e belonging to a cut of the graph if has the weight smaller than any other edge in the same cut, then the edge e is present in all the MSTs of the graph
d) Removing one edge from the spanning tree will not make the graph disconnected

Answer: d
Explanation: Every spanning tree has n – 1 edges if the graph has n edges and has no cycles. The MST follows the cut property, Edge e belonging to a cut of the graph if has the weight smaller than any other edge in the same cut, then the edge e is present in all the MSTs of the graph.

1. Kruskal's algorithm is used to _____
a) find minimum spanning tree
b) find single source shortest path
c) find all pair shortest path algorithm
d) traverse the graph

Answer: a
Explanation: The Kruskal's algorithm is used to find the minimum spanning tree of the connected graph. It construct the MST by finding the edge having the least possible weight that connects two trees in the forest.

2. Kruskal's algorithm is a _____
a) divide and conquer algorithm
b) dynamic programming algorithm
c) greedy algorithm
d) approximation algorithm

Answer: c
Explanation: Kruskal's algorithm uses a greedy algorithm approach to find the MST of the connected weighted graph. In the greedy method, we attempt to find an optimal solution in stages.Answer: d
Explanation: Kruskal's algorithm constructs the minimum spanning tree by constructing by adding the edges to spanning tree one-one by one. The MST for the given graph is,

So, the weight of the MST is 19.

4. What is the time complexity of Kruskal's algorithm?
a) O(log V)
b) O(E log V)
c) O(E^2)
d) O(V log E)

Answer: b
Explanation: Kruskal's algorithm involves sorting of the edges, which takes O(E logE) time, where E is a number of edges in graph and V is the number of vertices. After sorting, all edges are iterated and union-find algorithm is applied. union-find algorithm requires O(logV) time. So, overall Kruskal's algorithm requires O(E log V) time.Answer: c
Explanation: In Krsuskal's algorithm the edges are selected and added to the spanning tree in increasing order of their weights. Therefore, the first edge selected will be the minimal one. So, correct option is BE.Answer: a
Explanation: Using Krushkal's algorithm on the given graph, the generated minimum spanning tree is shown below.

So, the edges in the MST are, (B-E)(G-E)(E-F)(D-F).

7. Which of the following is true?
a) Prim's algorithm can also be used for disconnected graphs
b) Kruskal's algorithm can also run on the disconnected graphs
c) Prim's algorithm is simpler than Kruskal's algorithm
d) In Kruskal's sort edges are added to MST in decreasing order of their weights

Answer: b
Explanation: Prim's algorithm iterates from one node to another, so it can not be applied for disconnected graph. Kruskal's algorithm can be applied to the disconnected graphs to construct the minimum cost forest. Kruskal's algorithm is comparatively easier and simpler than prim's algorithm.

8. Which of the following is false about the Kruskal's algorithm?
a) It is a greedy algorithm
b) It constructs MST by selecting edges in increasing order of their weights
c) It can accept cycles in the MST
d) It uses union-find data structure

Answer: c
Explanation: Kruskal's algorithm is a greedy algorithm to construct the MST of the given graph. It constructs the MST by selecting edges in increasing order of their weights and rejects an edge if it may form the cycle. So, using Kruskal's

algorithm is never formed.Answer: b
Explanation: Prim's algorithm outperforms the Kruskal's algorithm in case of the dense graphs. It is significantly faster if graph has more edges than the Kruskal's algorithm.Answer: d
Explanation: In Kruskal's algorithm, the disjoint-set data structure efficiently identifies the components containing a vertex and adds the new edges. And Kruskal's algorithm always finds the MST for the connected graph.

1. Which of the following is true?
a) Prim's algorithm initialises with a vertex
b) Prim's algorithm initialises with a edge
c) Prim's algorithm initialises with a vertex which has smallest edge
d) Prim's algorithm initialises with a forest

Answer: a
Explanation: Steps in Prim's algorithm: (I) Select any vertex of given graph and add it to MST (II) Add the edge of minimum weight from a vertex not in MST to the vertex in MST; (III) It MST is complete the stop, otherwise go to step (II).Answer: c
Explanation: In Prim's algorithm, we select a vertex and add it to the MST. Then we add the minimum edge from the vertex in MST to vertex not in MST. From, figure shown below weight of MST = 27.

3. Worst case is the worst case time complexity of Prim's algorithm if adjacency matrix is used?
a) O(log V)
b) $O(V^2)$
c) $O(E^2)$
d) O(V log E)

Answer: b
Explanation: Use of adjacency matrix provides the simple implementation of the Prim's algorithm. In Prim's algorithm, we need to search for the edge with a minimum for that vertex. So, worst case time complexity will be $O(V^2)$, where V is the number of vertices.

4. Prim's algorithm is a _____
a) Divide and conquer algorithm
b) Greedy algorithm
c) Dynamic Programming
d) Approximation algorithm

Answer: b
Explanation: Prim's algorithm uses a greedy algorithm approach to find the MST of the connected weighted graph. In greedy method, we attempt to find an optimal solution in stages.Answer: a
Explanation: In Prim's algorithm, the MST is constructed starting from a single vertex and adding in new edges to the MST that link the partial tree to a new vertex outside of the MST. And Dijkstra's algorithm also rely on the similar approach of finding the next closest vertex. So, Prim's algorithm resembles Dijkstra's algorithm.Answer: a
Explanation: Prim's algorithm and Kruskal's algorithm perform equally in case of the sparse graphs. But Kruskal's algorithm is simpler and easy to work with. So, it is best suited for sparse graphs. Answer: d
Explanation: The MST for the given graph using Prim's algorithm starting from vertex 4 is,

So, the MST contains edges (4-3)(3-2)(2-1)(1-5).

8. Prim's algorithm is also known as _____
a) Dijkstra–Scholten algorithm
b) Borůvka's algorithm
c) Floyd–Warshall algorithm
d) DJP Algorithm

Answer: d
Explanation: The Prim's algorithm was developed by Vojtěch Jarník and it was latter discovered by the duo Prim and Dijkstra. Therefore, Prim's algorithm is also known as DJP Algorithm.

9. Prim's algorithm can be efficiently implemented using _____ for graphs with greater density.
a) d-ary heap
b) linear search
c) fibonacci heap
d) binary search

Answer: a
Explanation: In Prim's algorithm, we add the minimum weight edge for the chosen vertex which requires searching on the array of weights. This searching can be efficiently implemented using binary heap for dense graphs. And for graphs with greater density, Prim's algorithm can be made to run in linear time using d-ary heap(generalization of binary heap).

10. Which of the following is false about Prim's algorithm?
a) It is a greedy algorithm
b) It constructs MST by selecting edges in increasing order of their weights
c) It never accepts cycles in the MST
d) It can be implemented using the Fibonacci heap

Answer: b
Explanation: Prim's algorithm can be implemented using Fibonacci heap and it never accepts cycles. And Prim's algorithm follows greedy approach. Prim's algorithms span from one vertex to another.

1. Dijkstra's Algorithm is used to solve _____ problems.
a) All pair shortest path
b) Single source shortest path
c) Network flow
d) Sorting

Answer: b
Explanation: Dijkstra's Algorithm is used for solving single source shortest path problems. In this algorithm, a single node is fixed as a source node and shortest paths from this node to all other nodes in graph is found.

2. Which of the following is the most commonly used data structure for implementing Dijkstra's Algorithm?
a) Max priority queue
b) Stack
c) Circular queue
d) Min priority queue

Answer: d
Explanation: Minimum priority queue is the most commonly used data structure for implementing Dijkstra's Algorithm because the required operations to be performed in Dijkstra's Algorithm match with specialty of a minimum priority queue.

3. What is the time complexity of Dijikstra's algorithm?
a) O(N)
b) $O(N^3)$
c) $O(N^2)$
d) O(logN)

Answer: c
Explanation: Time complexity of Dijkstra's algorithm is $O(N^2)$ because of the use of doubly nested for loops. It depends on how the table is manipulated.

4. Dijkstra's Algorithm cannot be applied on _____
a) Directed and weighted graphs
b) Graphs having negative weight function
c) Unweighted graphs
d) Undirected and unweighted graphs

Answer: b
Explanation: Dijkstra's Algorithm cannot be applied on graphs having negative weight function because calculation of cost to reach a destination node from the source node becomes complex.

b)

```
if(!T[w].Known)
        if(T[v].Dist + C(v,w) < T[w].Dist)  {
                 Decrease(T[w].Dist to T[v].Dist +C(v,w));
                T[w].path=v; }
```

c)

```
if(T[w].Known)
        if(T[v].Dist + C(v,w) < T[w].Dist)  {
                 Increase (T[w].Dist to T[v].Dist +C(v,w));
                T[w].path=v; }
```

d)

```
if(!T[w].Known)
        if(T[v].Dist + C(v,w) > T[w].Dist)  {
                 Decrease(T[w].Dist to T[v].Dist +C(v,w);
                  T[w].path=v; }
```

6. How many priority queue operations are involved in Dijkstra's Algorithm?
a) 1
b) 3
c) 2
d) 4

Answer: b
Explanation: The number of priority queue operations involved is 3. They are insert, extract-min and decrease key.

7. How many times the insert and extract min operations are invoked per vertex?
a) 1
b) 2
c) 3
d) 0

Answer: a
Explanation: Insert and extract min operations are invoked only once per vertex because each vertex is added only once to the set and each edge in the adjacency list is examined only once during the course of algorithm.

8. The maximum number of times the decrease key operation performed in Dijkstra's algorithm will be equal to _____
a) Total number of vertices
b) Total number of edges
c) Number of vertices – 1
d) Number of edges – 1

Answer: b
Explanation: If the total number of edges in all adjacency list is E, then there will be a total of E number of iterations, hence there will be a total of at most E decrease key operations.

9. What is running time of Dijkstra's algorithm using Binary min- heap method?
a) O(V)
b) O(VlogV)
c) O(E)
d) O(ElogV)

Answer: d
Explanation: Time required to build a binary min heap is O(V). Each decrease key operation takes O(logV) and there are still at most E such operations. Hence total running time is O(ElogV).Answer: b
Explanation: The number of iterations involved in Bellmann Ford Algorithm is more than that of Dijkstra's Algorithm.Answer: a
Explanation: The equality d[u]=delta(s,u) holds good when vertex u is added to set S and this equality is maintained thereafter by the upper bound property.

12. Given pseudo code of Dijkstra's Algorithm.

```
if(T[w].Known)
        if(T[v].Dist + C(v,w) < T[w].Dist)  {
                 Increase(T[w].Dist to T[v].Dist);
                T[w].path=v; }
```

What happens when "While Q != 0" is changed to "while Q>1"?
a) While loop gets executed for v times
b) While loop gets executed for v-1 times
c) While loop gets executed only once
d) While loop does not get executed

Answer: b
Explanation: In the normal execution of Dijkstra's Algorithm, the while loop gets executed V times. The change in the while loop statement causes it to execute only V – 1 times.Answer: d
Explanation: The minimum cost to reach f vertex from b vertex is 6 by having vertices g and e as intermediates.
b to g, cost is 1
g to e, cost is 4
e to f, cost is 1
hence total cost 1+4+1=6.Answer: b
Explanation: The minimum cost required to travel from vertex A to E is via vertex C
A to C, cost= 3
C to E, cost= 2
Hence the total cost is 5.

15. Dijkstra's Algorithm is the prime example for _____
a) Greedy algorithm
b) Branch and bound

c) Back tracking
d) Dynamic programming

Answer: a
Explanation: Dijkstra's Algorithm is the prime example for greedy algorithms because greedy algorithms generally solve a problem in stages by doing what appears to be the best thing at each stage.

1. The Bellmann Ford algorithm returns _____ value.
a) Boolean
b) Integer
c) String
d) Double

Answer: a
Explanation: The Bellmann Ford algorithm returns Boolean value whether there is a negative weight cycle that is reachable from the source.

2. Bellmann ford algorithm provides solution for _____ problems.
a) All pair shortest path
b) Sorting
c) Network flow
d) Single source shortest path

Answer: d
Explanation: Bellmann ford algorithm is used for finding solutions for single source shortest path problems. If the graph has no negative cycles that are reachable from the source then the algorithm produces the shortest paths and their weights.Answer: a
Explanation: Bellmann Ford algorithm returns true if the graph does not have any negative weight cycles and returns false when the graph has negative weight cycles.

4. How many solution/solutions are available for a graph having negative weight cycle?
a) One solution
b) Two solutions
c) No solution
d) Infinite solutions

Answer: c
Explanation: If the graph has any negative weight cycle then the algorithm indicates that no solution exists for that graph.

5. What is the running time of Bellmann Ford Algorithm?
a) O(V)
b) $O(V^2)$
c) O(ElogV)
d) O(VE)

Answer: d
Explanation: Bellmann Ford algorithm runs in time O(VE), since the initialization takes O(V) for each of V-1 passes and the for loop in the algorithm takes O(E) time. Hence the total time taken by the algorithm is O(VE).

6. How many times the for loop in the Bellmann Ford Algorithm gets executed?
a) V times
b) V-1
c) E
d) E-1

Answer: b
Explanation: The for loop in the Bellmann Ford Algorithm gets executed for V-1 times. After making V-1 passes, the

algorithm checks for a negative weight cycle and returns appropriate boolean value.Answer: a
Explanation: The running time of Bellmann Ford Algorithm is O(VE) whereas Dijikstra's Algorithm has running time of only $O(V^2)$.

b)

```
for i=1 to V[g]-1
        do for each edge (u,v) in E[g]
                do Relax(u,v,w)
   for each edge (u,v) in E[g]
        do if d[v]>d[u]+w(u,v)
                then return False
   return True
```

c)

```
for i=1 to V[g]-1
       for each edge (u,v) in E[g]
        do if d[v]>d[u]+w(u,v)
                then return False
    return True
```

d)

```
for i=1 to V[g]-1
        do for each edge (u,v) in E[g]
                do Relax(u,v,w)
   for each edge (u,v) in E[g]
        do if d[v]<d[u]+w(u,v)
                then return true
   return True
```

9. What is the basic principle behind Bellmann Ford Algorithm?
a) Interpolation
b) Extrapolation
c) Regression
d) Relaxation

Answer: d
Explanation: Relaxation methods which are also called as iterative methods in which an approximation to the correct distance is replaced progressively by more accurate values till an optimum solution is found.

10. Bellmann Ford Algorithm can be applied for _____
a) Undirected and weighted graphs
b) Undirected and unweighted graphs
c) Directed and weighted graphs
d) All directed graphs

Answer: c
Explanation: Bellmann Ford Algorithm can be applied for all directed and weighted graphs. The weight function in the graph may either be positive or negative.

11. Bellmann Ford algorithm was first proposed by _____
a) Richard Bellmann
b) Alfonso Shimbe
c) Lester Ford Jr
d) Edward F. Moore

Answer: b
Explanation: Alfonso Shimbe proposed Bellmann Ford algorithm in the year 1955. Later it was published by Richard Bellmann in 1957 and Lester Ford Jr in the year 1956. Hence it is called Bellmann Ford Algorithm.Answer: b
Explanation: The minimum cost to travel from node A to node C is 2.

A-D, cost=1
D-B, cost=-2
B-C, cost=3
Hence the total cost is 2.Answer: d
Explanation: The minimum cost taken by the path a-d-b-c-e-f is 4.
a-d, cost=2
d-b, cost=-2
b-c, cost=1
c-e, cost= 2
e-f, cost=1
Hence the total cost is 4.

14. Bellmann Ford Algorithm is an example for _____
a) Dynamic Programming
b) Greedy Algorithms
c) Linear Programming
d) Branch and Bound

Answer: a
Explanation: In Bellmann Ford Algorithm the shortest paths are calculated in bottom up manner which is similar to other dynamic programming problems.

15. A graph is said to have a negative weight cycle when?
a) The graph has 1 negative weighted edge
b) The graph has a cycle
c) The total weight of the graph is negative
d) The graph has 1 or more negative weighted edges

Answer: c
Explanation: When the total weight of the graph sums up to a negative number then the graph is said to have a negative weight cycle. Bellmann Ford Algorithm provides no solution for such graphs.

1. Floyd Warshall's Algorithm is used for solving _____
a) All pair shortest path problems
b) Single Source shortest path problems
c) Network flow problems
d) Sorting problems

Answer: a
Explanation: Floyd Warshall's Algorithm is used for solving all pair shortest path problems. It means the algorithm is used for finding the shortest paths between all pairs of vertices in a graph.

2. Floyd Warshall's Algorithm can be applied on _____
a) Undirected and unweighted graphs
b) Undirected graphs
c) Directed graphs
d) Acyclic graphs

Answer: c
Explanation: Floyd Warshall Algorithm can be applied in directed graphs. From a given directed graph, an adjacency matrix is framed and then all pair shortest path is computed by the Floyd Warshall Algorithm.

3. What is the running time of the Floyd Warshall Algorithm?
a) Big-oh(V)
b) Theta(V^2)
c) Big-Oh(VE)
d) Theta(V^3)

Answer: d
Explanation: The running time of the Floyd Warshall algorithm is determined by the triply nested for loops. Since each execution of the for loop takes O(1) time, the algorithm runs in time Theta(V^3).

4. What approach is being followed in Floyd Warshall Algorithm?
a) Greedy technique
b) Dynamic Programming
c) Linear Programming
d) Backtracking

Answer: b
Explanation: Floyd Warshall Algorithm follows dynamic programming approach because the all pair shortest paths are computed in bottom up manner.

5. Floyd Warshall Algorithm can be used for finding _____
a) Single source shortest path
b) Topological sort
c) Minimum spanning tree
d) Transitive closure

Answer: d
Explanation: One of the ways to compute the transitive closure of a graph in Theta(N^3) time is to assign a weight of 1 to each edge of E and then run the Floyd Warshall Algorithm.

6. What procedure is being followed in Floyd Warshall Algorithm?
a) Top down
b) Bottom up
c) Big bang
d) Sandwich

Answer: b
Explanation: Bottom up procedure is being used to compute the values of the matrix elements dij(k). The input is an n x n matrix. The procedure returns the matrix D(n) of the shortest path weights.

7. Floyd- Warshall algorithm was proposed by _____
a) Robert Floyd and Stephen Warshall
b) Stephen Floyd and Robert Warshall
c) Bernad Floyd and Robert Warshall
d) Robert Floyd and Bernad Warshall

Answer: a
Explanation: Floyd- Warshall Algorithm was proposed by Robert Floyd in the year 1962. The same algorithm was proposed by Stephen Warshall during the same year for finding the transitive closure of the graph.

8. Who proposed the modern formulation of Floyd-Warshall Algorithm as three nested loops?
a) Robert Floyd
b) Stephen Warshall
c) Bernard Roy
d) Peter Ingerman

Answer: d
Explanation: The modern formulation of Floyd-Warshall Algorithm as three nested for-loops was proposed by Peter Ingerman in the year 1962.

9. Complete the program.

```
n=rows[W]
D(0)=W
```

```
for k=1 to n
     do for i=1 to n
          do for j=1 to n
                do _____________________________
return D(n)
```

a) dij(k)=min(dij(k-1), dik(k-1) – dkj(k-1))
b) dij(k)=max(dij(k-1), dik(k-1) – dkj(k-1))
c) dij(k)=min(dij(k-1), dik(k-1) + dkj(k-1))
d) dij(k)=max(dij(k-1), dik(k-1) + dkj(k-1))

Answer: c
Explanation: In order to compute the shortest path from vertex i to vertex j, we need to find the minimum of 2 values which are dij(k-1) and sum of dik(k-1) and dkj(k-1).

10. What happens when the value of k is 0 in the Floyd Warshall Algorithm?
a) 1 intermediate vertex
b) 0 intermediate vertex
c) N intermediate vertices
d) N-1 intermediate vertices

Answer: b
Explanation: When k=0, a path from vertex i to vertex j has no intermediate vertices at all. Such a path has at most one edge and hence dij(0) = wij. Answer: a
Explanation: In computers, logical operations on single bit values execute faster than arithmetic operations on integer words of data.Answer: b
Explanation: The time taken to compute the transitive closure of a graph is Theta(n^3). Transitive closure can be computed by assigning weight of 1 to each edge and by running Floyd Warshall Algorithm.Answer: d
Explanation: The minimum cost required to travel from node 1 to node 5 is -3.
1-5, cost is -4
5-4, cost is 6
4-3, cost is -5
Hence total cost is -4 + 6 + -5 = -3.Answer: c
Explanation: The minimum cost to travel from node a to node e is 1 by passing via nodes b and c.
a-b, cost 5
b-c, cost 3
c-e, cost -7
Hence the total cost is 1.

15. What is the formula to compute the transitive closure of a graph?
a) tij(k) = tij(k-1) AND (tik(k-1) OR tkj(k-1))
b) tij(k) = tij(k-1) OR (tik(k-1) AND tkj(k-1))
c) tij(k) = tij(k-1) AND (tik(k-1) AND tkj(k-1))
d) tij(k) = tij(k-1) OR (tik(k-1) OR tkj(k-1))

Answer: b
Explanation: Transitive closure of a graph can be computed by using Floyd Warshall algorithm. This method involves substitution of logical operations (logical OR and logical AND) for arithmetic operations min and + in Floyd Warshall Algorithm.
Floyd Warshall Algorithm: dij(k)=min(dij(k-1), dik(k-1) + dkj(k-1))
Transitive closure: tij(k)= tij(k-1) OR (tik(k-1) AND tkj(k-1)).

1. What does Maximum flow problem involve?
a) finding a flow between source and sink that is maximum
b) finding a flow between source and sink that is minimum
c) finding the shortest path between source and sink
d) computing a minimum spanning tree

Answer: a
Explanation: The maximum flow problem involves finding a feasible flow between a source and a sink in a network that is maximum and not minimum.Answer: b
Explanation: A network can have only one source and one sink inorder to find the feasible flow in a weighted connected graph.

3. What is the source?
a) Vertex with no incoming edges
b) Vertex with no leaving edges
c) Centre vertex
d) Vertex with the least weight

Answer: a
Explanation: Vertex with no incoming edges is called as a source. Vertex with no leaving edges is called as a sink.

4. Which algorithm is used to solve a maximum flow problem?
a) Prim's algorithm
b) Kruskal's algorithm
c) Dijkstra's algorithm
d) Ford-Fulkerson algorithm

Answer: d
Explanation: Ford-fulkerson algorithm is used to compute the maximum feasible flow between a source and a sink in a network.

5. Does Ford- Fulkerson algorithm use the idea of?
a) Naïve greedy algorithm approach
b) Residual graphs
c) Minimum cut
d) Minimum spanning tree

Answer: b
Explanation: Ford-Fulkerson algorithm uses the idea of residual graphs which is an extension of naïve greedy approach allowing undo operations.

6. The first step in the naïve greedy algorithm is?
a) analysing the zero flow
b) calculating the maximum flow using trial and error
c) adding flows with higher values
d) reversing flow if required

Answer: a
Explanation: The first step in the naïve greedy algorithm is to start with the zero flow followed by adding edges with higher values.

7. Under what condition can a vertex combine and distribute flow in any manner?
a) It may violate edge capacities
b) It should maintain flow conservation
c) The vertex should be a source vertex
d) The vertex should be a sink vertex

Answer: b
Explanation: A vertex can combine and distribute flow in any manner but it should not violate edge capacities and it should maintain flow conservation.Answer: c
Explanation: Initially, zero flow is computed. Then, computing flow= 7+1+5+2=15. Hence, maximum flow=15.

9. A simple acyclic path between source and sink which pass through only positive weighted edges is called?
a) augmenting path

b) critical path
c) residual path
d) maximum path

Answer: a
Explanation: Augmenting path between source and sink is a simple path without cycles. Path consisting of zero slack edges is called critical path.

10. In what time can an augmented path be found?
a) O(|E| log |V|)
b) O(|E|)
c) O($|E|^2$)
d) O($|E|^2$ log |V|)

Answer: b
Explanation: An augmenting path can be found in O(|E|) mathematically by an unweighted shortest path algorithm.Answer: a
Explanation: Dinic's algorithm includes construction of level graphs and resLidual graphs and finding of augmenting paths along with blocking flow and is faster than the Ford-Fulkerson algorithm.

12. What is the running time of an unweighted shortest path algorithm whose augmenting path is the path with the least number of edges?
a) O(|E|)
b) O(|E||V|)
c) O($|E|^2$|V|)
d) O(|E| log |V|)

Answer: c
Explanation: Each augmenting step takes O(|E|) using an unweighted shortest path algorithm yielding a O(|E|2|V|) bound on the running time.

13. Who is the formulator of Maximum flow problem?
a) Lester R. Ford and Delbert R. Fulkerson
b) T.E. Harris and F.S. Ross
c) Y.A. Dinitz
d) Kruskal

Answer: b
Explanation: The first ever people to formulate Maximum flow problem were T.E. Harris and F.S. Ross. Lester R. Ford and Delbert R. Fulkerson formulated Ford- Fulkerson algorithm.

14. What is the running time of Dinic's blocking flow algorithm?
a) O(V^2E)
b) O(VE^2)
c) O(V^3)
d) O(E max |f|)

Answer: a
Explanation: The running time of Dinic's blocking flow algorithm is O(V^2E). The running of Ford-Fulkerson algorithm is O(E max |f|).

15. How many constraints does flow have?
a) one
b) three
c) two
d) four

Answer: c
Explanation: A flow is a mapping which follows two constraints- conservation of flows and capacity constraints.

1. Stable marriage problem is an example of?
a) Branch and bound algorithm
b) Backtracking algorithm
c) Greedy algorithm
d) Divide and conquer algorithm

Answer: b
Explanation: Stable marriage problem is an example for recursive algorithm because it recursively uses backtracking algorithm to find an optimal solution.

2. Which of the following algorithms does Stable marriage problem uses?
a) Gale-Shapley algorithm
b) Dijkstra's algorithm
c) Ford-Fulkerson algorithm
d) Prim's algorithm

Answer: a
Explanation: Stable marriage problem uses Gale-Shapley algorithm. Maximum flow problem uses Ford-Fulkerson algorithm. Prim's algorithm involves minimum spanning tree.

3. An optimal solution satisfying men's preferences is said to be?
a) Man optimal
b) Woman optimal
c) Pair optimal
d) Best optimal

Answer: a
Explanation: An optimal solution satisfying men's preferences are said to be man optimal. An optimal solution satisfying woman's preferences are said to be woman optimal.

4. When a free man proposes to an available woman, which of the following happens?
a) She will think and decide
b) She will reject
c) She will replace her current mate
d) She will accept

Answer: d
Explanation: When a man proposes to an available woman, she will accept his proposal irrespective of his position on his preference list.Answer: a
Explanation: If there are n couples such that a man and a woman are not married, and if they prefer each other to their actual partners, the assignment is unstable.

6. How many 2*2 matrices are used in this problem?
a) 1
b) 2
c) 3
d) 4

Answer: b
*Explanation: Two 2*2 matrices are used. One for men representing corresponding woman and ranking and the other for women.*

7. What happens when a free man approaches a married woman?
a) She simply rejects him
b) She simply replaces her mate with him

c) She goes through her preference list and accordingly, she replaces her current mate with him
d) She accepts his proposal

Answer: c
Explanation: If the preference of the man is greater, she replaces her current mate with him, leaving her current mate free.

8. In case of stability, how many symmetric possibilities of trouble can occur?
a) 1
b) 2
c) 4
d) 3

Answer: b
Explanation: Possibilities- There might be a woman pw, preferred to w by m, who herself prefers m to be her husband and the same applies to man as well.Answer: a
Explanation: W2 is married to M1. But the preference of W2 has M2 before M1. Hence, W2 replaces M1 with M2.Answer: c
Explanation: M3 will approach W3 first. Since W3 is married and since her preference list has her current mate before M3, she rejects his proposal.

11. Who formulated a straight forward backtracking scheme for stable marriage problem?
a) McVitie and Wilson
b) Gale
c) Ford and Fulkerson
d) Dinitz

Answer: a
Explanation: McVitie and Wilson formulated a much faster straight forward backtracking scheme for stable marriage problem. Ford and Fulkerson formulated Maximum flow problem.Answer: b
Explanation: Stable marriage problem can be solved using branch and bound approach because branch and bound follows backtracking scheme with a limitation factor.

13. What is the prime task of the stable marriage problem?
a) To provide man optimal solution
b) To provide woman optimal solution
c) To determine stability of marriage
d) To use backtracking approach

Answer: c
Explanation: The prime task of stable marriage problem is to determine stability of marriage (i.e) finding a man and a woman who prefer each other to others.

14. Which of the following problems is related to stable marriage problem?
a) Choice of school by students
b) N-queen problem
c) Arranging data in a database
d) Knapsack problem

Answer: a
Explanation: Choice of school by students is the most related example in the given set of options since both school and students will have a preference list.

15. What is the efficiency of Gale-Shapley algorithm used in stable marriage problem?
a) O(N)
b) O(N log N)
c) $O(N^2)$

d) O(log N)

Answer: c
Explanation: The time efficiency of Gale-Shapley algorithm is mathematically found to be $O(N^2)$ where N denotes stable marriage problem.

1. _____ is a matching with the largest number of edges.
a) Maximum bipartite matching
b) Non-bipartite matching
c) Stable marriage
d) Simplex

Answer: a
Explanation: Maximum bipartite matching matches two elements with a property that no two edges share a vertex.Answer: a
Explanation: Maximum matching is also called as maximum cardinality matching (i.e.) matching with the largest number of edges.

3. How many colours are used in a bipartite graph?
a) 1
b) 2
c) 3
d) 4

Answer: b
Explanation: A bipartite graph is said to be two-colourable so that every edge has its vertices coloured in different colours.

4. What is the simplest method to prove that a graph is bipartite?
a) It has a cycle of an odd length
b) It does not have cycles
c) It does not have a cycle of an odd length
d) Both odd and even cycles are formed

Answer: c
Explanation: It is not difficult to prove that a graph is bipartite if and only if it does not have a cycle of an odd length.

5. A matching that matches all the vertices of a graph is called?
a) Perfect matching
b) Cardinality matching
c) Good matching
d) Simplex matching

Answer: a
Explanation: A matching that matches all the vertices of a graph is called perfect matching.

6. What is the length of an augmenting path?
a) Even
b) Odd
c) Depends on graph
d) 1

Answer: b
Explanation: The length of an augmenting path in a bipartite graph is always said to be always odd.

7. In a bipartite graph G=(V,U,E), the matching of a free vertex in V to a free vertex in U is called?
a) Bipartite matching
b) Cardinality matching

c) Augmenting
d) Weight matching

Answer: c
Explanation: A simple path from a free vertex in V to a free vertex in U whose edges alternate between edges not in M and edges in M is called a augmenting path.Answer: a
Explanation: According to the theorem discovered by the French mathematician Claude Berge, it means that the current matching is maximal if there is no augmenting path.

9. Which one of the following is an application for matching?
a) Proposal of marriage
b) Pairing boys and girls for a dance
c) Arranging elements in a set
d) Finding the shortest traversal path

Answer: b
Explanation: Pairing boys and girls for a dance is a traditional example for matching. Proposal of marriage is an application of stable marriage problem.

10. Which is the correct technique for finding a maximum matching in a graph?
a) DFS traversal
b) BFS traversal
c) Shortest path traversal
d) Heap order traversal

Answer: b
Explanation: The correct technique for finding a maximum matching in a bipartite graph is by using a Breadth First Search(BFS).

11. The problem of maximizing the sum of weights on edges connecting matched pairs of vertices is?
a) Maximum- mass matching
b) Maximum bipartite matching
c) Maximum weight matching
d) Maximum node matching

Answer: c
Explanation: The problem is called as maximum weight matching which is similar to a bipartite matching. It is also called as assignment problem.

12. What is the total number of iterations used in a maximum- matching algorithm?
a) [n/2]
b) [n/3]
c) [n/2]+n
d) [n/2]+1

Answer: d
Explanation: The total number of iterations cannot exceed [n/2]+1 where n=|V|+|U| denoting the number of vertices in the graph.

13. What is the efficiency of algorithm designed by Hopcroft and Karp?
a) O(n+m)
b) O(n(n+m)
c) O(√n(n+m))
d) O(n+2)

Answer: c
Explanation: The efficiency of algorithm designed by Hopcroft and Karp is mathematically found to be O(√n(n+m)).

14. Who was the first person to solve the maximum matching problem?
a) Jack Edmonds
b) Hopcroft
c) Karp
d) Claude Berge

Answer: a
Explanation: Jack Edmonds was the first person to solve the maximum matching problem in 1965.Answer: a
Explanation: One of the solutions of the matching problem is given by a-w,b-v,c-x,d-y,e-z. Hence the answer is 5.

1. Which algorithm is used to solve a minimum cut algorithm?
a) Gale-Shapley algorithm
b) Ford-Fulkerson algorithm
c) Stoer-Wagner algorithm
d) Prim's algorithm

Answer: c
Explanation: Minimum cut algorithm is solved using Stoer-Wagner algorithm. Maximum flow problem is solved using Ford-Fulkerson algorithm. Stable marriage problem is solved using Gale-Shapley algorithm.

2. _____ is a partition of the vertices of a graph in two disjoint subsets that are joined by atleast one edge.
a) Minimum cut
b) Maximum flow
c) Maximum cut
d) Graph cut

Answer: a
Explanation: Minimum cut is a partition of the vertices in a graph 4. in two disjoint subsets joined by one edge. It is a cut that is minimal in some sense.Answer: a
Explanation: Minimum cut algorithm is considered to be an extension of the maximum flow problem. Minimum cut is finding a cut that is minimal.Answer: a
Explanation: The given figure is a depiction of min cut problem since the graph is partitioned to find the minimum cut.

5. _____ separates a particular pair of vertices in a graph.
a) line
b) arc
c) cut
d) flow

Answer: c
Explanation: A cut separates a particular pair of vertices in a weighted undirected graph and has minimum possible weight.

6. What is the minimum number of cuts that a graph with 'n' vertices can have?
a) n+1
b) n(n-1)
c) n(n+1)/2
d) n(n-1)/2

Answer: c
Explanation: The mathematical formula for a graph with 'n' vertices can at the most have n(n-1)/2 distinct vertices.

7. What is the running time of Karger's algorithm to find the minimum cut in a graph?
a) O(E)
b) $O(|V|^2)$
c) O(V)
d) O(|E|)

Answer: b
Explanation: The running time of Karger's algorithm to find the minimum cut is mathematically found to be $O(|V|^2)$.

8. _____ is a family of combinatorial optimization problems in which a graph is partitioned into two or more parts with constraints.
a) numerical problems
b) graph partition
c) network problems
d) combinatorial problems

Answer: b
Explanation: Graph partition is a problem in which the graph is partitioned into two or more parts with additional conditions.Answer: b
Explanation: According to max-flow min-cut theorem, the weight of the cut is equal to the maximum flow that is sent from source to sink.

10. _____ is a data structure used to collect a system of cuts for solving min-cut problem.
a) Gomory-Hu tree
b) Gomory-Hu graph
c) Dancing tree
d) AA tree

Answer: a
Explanation: Gomory-Hu tree is a weighted tree that contains the minimum cuts for all pairs in a graph. It is constructed in |V|-1 max-flow computations.

11. In how many ways can a Gomory-Hu tree be implemented?
a) 1
b) 2
c) 3
d) 4

Answer: b
Explanation: Gomory-Hu tree can be implemented in two ways- sequential and parallel.

12. The running time of implementing naïve solution to min-cut problem is?
a) O(N)
b) O(N log N)
c) O(log N)
d) $O(N^2)$

Answer: d
Explanation: The running time of min-cut algorithm using naïve implementation is mathematically found to be $O(N^2)$.

13. What is the running time of implementing a min-cut algorithm using bidirected edges in a graph?
a) O(N)
b) O(N log N)
c) $O(N^4)$
d) $O(N^2)$

Answer: c
Explanation: The running time of a min-cut algorithm using Ford-Fulkerson method of making edges birected in a graph is mathematically found to be $O(N^4)$.

14. Which one of the following is not an application of max-flow min-cut algorithm?
a) network reliability
b) closest pair

c) network connectivity
d) bipartite matching

Answer: b
Explanation: Network reliability, connectivity and bipartite matching are all applications of min-cut algorithm whereas closest pair is a different kind of problem.Answer: a
Explanation: The minimum cut of the given graph network is found to be ({1,3},{4,3},{4,5}) and its capacity is 23.

1. Which of the following is not a type of graph in computer science?
a) undirected graph
b) bar graph
c) directed graph
d) weighted graph

Answer: b
Explanation: According to the graph theory a graph is the collection of dots and lines. A bar graph is not a type of graph in computer science.

2. What is vertex coloring of a graph?
a) A condition where any two vertices having a common edge should not have same color
b) A condition where any two vertices having a common edge should always have same color
c) A condition where all vertices should have a different color
d) A condition where all vertices should have same color

Answer: a
Explanation: The condition for vertex coloring of a graph is that two vertices which share a common edge should not have the same color. If it uses k colors in the process then it is called k coloring of graph.

3. How many edges will a tree consisting of N nodes have?
a) Log(N)
b) N
c) N – 1
d) N + 1

Answer: c
Explanation: In order to have a fully connected tree it must have N-1 edges. So the correct answer will be N-1.

4. Minimum number of unique colors required for vertex coloring of a graph is called?
a) vertex matching
b) chromatic index
c) chromatic number
d) color number

Answer: c
Explanation: The minimum number of colors required for proper vertex coloring of graph is called chromatic number whereas the minimum number of colors required for proper edge coloring of graph is called chromatic index of a graph.

5. How many unique colors will be required for proper vertex coloring of an empty graph having n vertices?
a) 0
b) 1
c) 2
d) n

Answer: b
Explanation: An empty graph is a graph without any edges. So the number of unique colors required for proper coloring of the graph will be 1.

6. How many unique colors will be required for proper vertex coloring of a bipartite graph having n vertices?

a) 0
b) 1
c) 2
d) n

Answer: c
Explanation: A bipartite graph is a graph such that no two vertices of the same set are adjacent to each other. So the number of unique colors required for proper coloring of the graph will be 2.

7. Which of the following is an NP complete problem?
a) Hamiltonian cycle
b) Travelling salesman problem
c) Calculating chromatic number of graph
d) Finding maximum element in an array

Answer: c
Explanation: Calculating the chromatic number of a graph is an NP complete problem as a chromatic number of an arbitrary graph cannot be determined by using any convenient method.

8. How many unique colors will be required for proper vertex coloring of a line graph having n vertices?
a) 0
b) 1
c) 2
d) n

Answer: d
Explanation: A line graph of a simple graph is obtained by connecting two vertices with an edge. So the number of unique colors required for proper coloring of the graph will be n.

9. How many unique colors will be required for proper vertex coloring of a complete graph having n vertices?
a) 0
b) 1
c) n
d) n!

Answer: c
Explanation: A complete graph is the one in which each vertex is directly connected with all other vertices with an edge. So the number of unique colors required for proper coloring of the graph will be n.

10. Minimum number of colors required for proper edge coloring of a graph is called?
a) Chromatic color
b) Chromatic index
c) Edge matching
d) Color number

Answer: b
Explanation: The minimum number of colors required for proper edge coloring of graph is called chromatic index. It is also known as edge chromatic number.Answer: b
Explanation: The given graph will only require 2 unique colors so that no two vertices connected by a common edge will have the same color. All nodes will have the same color except the central node.Answer: c
Explanation: The given graph will require 4 unique colors so that no two vertices connected by a common edge will have the same color. So its chromatic number will be 4.Answer: b
Explanation: The given graph will require 3 unique colors because two diagonal vertex are connected to each other directly. So its chromatic number will be 3.Answer: b
Explanation: Vertex coloring of a graph is an assignment of colors to the vertices of a graph such that no two adjacent vertices have the same color. Whereas chromatic number refers to the minimum number of unique colors required for vertex coloring of the graph.

1. What is the definition of graph according to graph theory?
a) visual representation of data
b) collection of dots and lines
c) collection of edges
d) collection of vertices

Answer: b
Explanation: According to the graph theory a graph is the collection of dots and lines. Vertices are also called dots and lines are also called edges.

2. What is the condition for proper coloring of a graph?
a) two vertices having a common edge should not have same color
b) two vertices having a common edge should always have same color
c) all vertices should have a different color
d) all vertices should have same color

Answer: a
Explanation: The condition for proper coloring of graph is that two vertices which share a common edge should not have the same color. If it uses k colors in the process then it is called k coloring of graph.

3. The number of colors used by a proper coloring graph is called?
a) k coloring graph
b) x coloring graph
c) m coloring graph
d) n coloring graph

Answer: a
Explanation: A proper graph ensures that two vertices which share a common edge should not have the same color. If it uses k colors in the process then it is called k coloring of graph.

4. What is a chromatic number?
a) The maximum number of colors required for proper edge coloring of graph
b) The maximum number of colors required for proper vertex coloring of graph
c) The minimum number of colors required for proper vertex coloring of graph
d) The minimum number of colors required for proper edge coloring of graph

Answer: c
Explanation: The minimum number of colors required for proper vertex coloring of graph is called chromatic number whereas the minimum number of colors required for proper edge coloring of graph is called chromatic index of a graph.

5. What will be the chromatic number for an empty graph having n vertices?
a) 0
b) 1
c) 2
d) n

Answer: b
Explanation: An empty graph is a graph without any edges. So the chromatic number for such a graph will be 1.

6. What will be the chromatic number for an bipartite graph having n vertices?
a) 0
b) 1
c) 2
d) n

Answer: c
Explanation: A bipartite graph is graph such that no two vertices of the same set are adjacent to each other. So the chromatic number for such a graph will be 2.

7. Calculating the chromatic number of a graph is a
a) P problem
b) NP hard problem
c) NP complete problem
d) cannot be identified as any of the given problem types

Answer: c
Explanation: Chromatic number of an arbitrary graph cannot be determined by using any convenient method. So calculating the chromatic number of a graph is an NP complete problem.

8. What will be the chromatic number for a line graph having n vertices?
a) 0
b) 1
c) 2
d) n

Answer: d
Explanation: A line graph of a simple graph is obtained by connecting two vertices with an edge. A line graph has a chromatic number of n.

9. What will be the chromatic number for a complete graph having n vertices?
a) 0
b) 1
c) n
d) n!

Answer: c
Explanation: A complete graph is the one in which each vertex is directly connected with all other vertices with an edge. So in such a case each vertex should have a unique color. Thus the chromatic number will be n.

10. What will be the chromatic number for a tree having more than 1 vertex?
a) 0
b) 1
c) 2
d) Varies with the structure and number of vertices of the tree

Answer: c
Explanation: The minimum number of colors required for proper vertex coloring of graph is called chromatic number. So every tree having more than 1 vertex is 2 chromatic.

11. A graph with chromatic number less than or equal to k is called?
a) K chromatic
b) K colorable
c) K chromatic colorable
d) K colorable chromatic

Answer: b
Explanation: Any graph that has a chromatic number less than or equal to k is called k colorable. Whereas a graph with chromatic number k is called k chromatic.Answer: b
Explanation: The given graph will only require 2 unique colors so that no two vertices connected by a common edge will have the same color. So its chromatic number will be 2.Answer: b
Explanation: The given graph will require 3 unique colors so that no two vertices connected by a common edge will have the same color. So its chromatic number will be 3.Answer: b
Explanation: The given graph will require 3 unique colors so that no two vertices connected by a common edge will have the same color. So its chromatic number will be 3.Answer: b
Explanation: The chromatic number of a star graph is always 2 (for more than 1 vertex) whereas the chromatic number of complete graph with 3 vertices will be 3. So chromatic number of complete graph will be greater.Answer: b

Explanation: The chromatic number of a star graph and a tree is always 2 (for more than 1 vertex). So both have the same chromatic number.

1. In graph theory collection of dots and lines is called
a) vertex
b) edge
c) graph
d) map

Answer: c
Explanation: According to the graph theory a graph is the collection of dots and lines. Vertices are also called dots and lines are also called edges.

2. What is the condition for proper edge coloring of a graph?
a) Two vertices having a common edge should not have same color
b) Two vertices having a common edge should always have same color
c) No two incident edges should have the same color
d) No two incident edges should have different color

Answer: c
Explanation: The condition for proper edge coloring of graph is that no two incident edges should have the same color. If it uses k colors in the process then it is called k edge coloring of graph.

3. The number of colors used by a proper edge coloring graph is called?
a) k edge coloring graph
b) x edge coloring graph
c) m edge coloring graph
d) n edge coloring graph

Answer: a
Explanation: A proper edge coloring graph ensures that no two incident edges has the same color. If it uses k colors in the process then it is called k coloring of graph.

4. What is a chromatic index?
a) The maximum number of colors required for proper edge coloring of graph
b) The maximum number of colors required for proper vertex coloring of graph
c) The minimum number of colors required for proper vertex coloring of graph
d) The minimum number of colors required for proper edge coloring of graph

Answer: d
Explanation: The minimum number of colors required for proper edge coloring of graph is called chromatic index whereas the minimum number of colors required for proper vertex coloring of graph is called chromatic number of a graph.

5. What will be the chromatic index for an empty graph having n vertices?
a) 0
b) 1
c) 2
d) n

Answer: a
Explanation: An empty graph is a graph without any edges. So the chromatic index for such a graph will be 0.

6. If chromatic number of a line graph is 4 then the chromatic index of the graph will be?
a) 0
b) 1
c) 4
d) information insufficient

Answer: c
Explanation: The chromatic index of a graph is always equal to the chromatic number of its line graph. So the chromatic index of the graph will be 4.

7. Calculating the chromatic index of a graph is a _____
a) P problem
b) NP hard problem
c) NP complete problem
d) Cannot be identified as any of the given problem types

Answer: c
Explanation: Chromatic index of an arbitrary graph cannot be determined by using any convenient method. So calculating the chromatic index of a graph is an NP complete problem. Answer: a
Explanation: The chromatic index of a graph is always equal to the chromatic number of its line graph. So we can calculate the chromatic index of a graph by calculating the chromatic number of its line graph.Answer: a
Explanation: A bipartite graph has an edge chromatic number equal to Δ. So bipartite graphs belongs to class 1 graphs. Answer: b
Explanation: The given graph will require 2 unique colors so that no two incident edges have the same color. So its chromatic index will be 2.Answer: b
Explanation: The given graph will require 3 unique colors so that no two incident edges have the same color. So its chromatic index will be 3.Answer: c
Explanation: The given graph will require 2 unique colors so that no two incident edges have the same color. So its chromatic index will be 2.

13. What will be the chromatic index for a complete graph having n vertices (consider n to be an odd number)?
a) n
b) n + 1
c) n – 1
d) 2n + 1

Answer: a
Explanation: A complete graph is the one in which each vertex is directly connected with all other vertices with an edge. The chromatic index for an odd number of vertices will be n.

14. What will be the chromatic index for a complete graph having n vertices (consider n to be an even number)?
a) n
b) n + 1
c) n – 1
d) 2n + 1

Answer: c
Explanation: A complete graph is the one in which each vertex is directly connected with all other vertices with an edge. The chromatic index for even number of vertices will be n-1.

1. Given G is a bipartite graph and the bipartitions of this graphs are U and V respectively. What is the relation between them?
a) Number of vertices in U = Number of vertices in V
b) Sum of degrees of vertices in U = Sum of degrees of vertices in V
c) Number of vertices in U > Number of vertices in V
d) Nothing can be said

Answer: b
Explanation: We can prove this by induction. By adding one edge, the degree of vertices in U is equal to 1 as well as in V. Let us assume that this is true for n-1 edges and add one more edge. Since the given edge adds exactly once to both U and V we can tell that this statement is true for all n vertices.

2. A k-regular bipartite graph is the one in which degree of each vertices is k for all the vertices in the graph. Given

that the bipartitions of this graph are U and V respectively. What is the relation between them?
a) Number of vertices in U=Number of vertices in V
b) Number of vertices in U not equal to number of vertices in V
c) Number of vertices in U always greater than the number of vertices in V
d) Nothing can be said

Answer: a
Explanation: We know that in a bipartite graph sum of degrees of vertices in U=sum of degrees of vertices in V. Given that the graph is a k-regular bipartite graph, we have k(number of vertices in U)=k*(number of vertices in V).*

3. There are four students in a class namely A, B, C and D. A tells that a triangle is a bipartite graph. B tells pentagon is a bipartite graph. C tells square is a bipartite graph. D tells heptagon is a bipartite graph. Who among the following is correct?
a) A
b) B
c) C
d) D

Answer: c
Explanation: We can prove it in this following way. Let '1' be a vertex in bipartite set X and let '2' be a vertex in the bipartite set Y. Therefore the bipartite set X contains all odd numbers and the bipartite set Y contains all even numbers. Now let us consider a graph of odd cycle (a triangle). There exists an edge from '1' to '2', '2' to '3' and '3' to '1'. The latter case ('3' to '1') makes an edge to exist in a bipartite set X itself. Therefore telling us that graphs with odd cycles are not bipartite.

4. A complete bipartite graph is a one in which each vertex in set X has an edge with set Y. Let n be the total number of vertices. For maximum number of edges, the total number of vertices hat should be present on set X is?
a) n
b) n/2
c) n/4
d) data insufficient

Answer: b
Explanation: We can prove this by calculus. Let x be the total number of vertices on set X. Therefore set Y will have n-x. We have to maximize x(n-x). This is true when x=n/2.*

5. When is a graph said to be bipartite?
a) If it can be divided into two independent sets A and B such that each edge connects a vertex from to A to B
b) If the graph is connected and it has odd number of vertices
c) If the graph is disconnected
d) If the graph has at least n/2 vertices whose degree is greater than n/2

Answer: a
Explanation: A graph is said to be bipartite if it can be divided into two independent sets A and B such that each edge connects a vertex from A to B.

6. Are trees bipartite?
a) Yes
b) No
c) Yes if it has even number of vertices
d) No if it has odd number of vertices

Answer: a
Explanation: Condition needed is that there should not be an odd cycle. But in a tree there are no cycles at all. Hence it is bipartite.

7. A graph has 20 vertices. The maximum number of edges it can have is? (Given it is bipartite)
a) 100

b) 140
c) 80
d) 20

Answer: a
Explanation: Let the given bipartition X have x vertices, then Y will have 20-x vertices. We need to maximize x(20-x). This will be maxed when x=10.Answer: a*
Explanation: It is required that the graph is connected also. If it is not then it cannot be called a bipartite graph.

9. Can there exist a graph which is both eulerian and is bipartite?
a) Yes
b) No
c) Yes if it has even number of edges
d) Nothing can be said

Answer: a
Explanation: If a graph is such that there exists a path which visits every edge atleast once, then it is said to be Eulerian. Taking an example of a square, the given question evaluates to yes.

10. A graph is found to be 2 colorable. What can be said about that graph?
a) The given graph is eulerian
b) The given graph is bipartite
c) The given graph is hamiltonian
d) The given graph is planar

Answer: b
Explanation: A graph is said to be colorable if two vertices connected by an edge are never of the same color. 2 colorable mean that this can be achieved with just 2 colors.

1. Which type of graph has no odd cycle in it?
a) Bipartite
b) Histogram
c) Cartesian
d) Pie

Answer: a
Explanation: The graph is known as Bipartite if the graph does not contain any odd length cycle in it. Odd length cycle means a cycle with the odd number of vertices in it.

2. What type of graph has chromatic number less than or equal to 2?
a) Histogram
b) Bipartite
c) Cartesian
d) Tree

Answer: b
Explanation: A graph is known as bipartite graph if and only if it has the total chromatic number less than or equal to 2. The smallest number of graphs needed to color the graph is chromatic number.

3. Which of the following is the correct type of spectrum of the bipartite graph?
a) Symmetric
b) Anti – Symmetric
c) Circular
d) Exponential

Answer: a
Explanation: The spectrum of the bipartite graph is symmetric in nature. The spectrum is the property of graph that are related to polynomial, Eigen values, Eigen vectors of the matrix related to graph.

4. Which of the following is not a property of the bipartite graph?
a) No Odd Cycle
b) Symmetric spectrum
c) Chromatic Number Is Less Than or Equal to 2
d) Asymmetric spectrum

Answer: d
Explanation: A graph is known to be bipartite if it has odd length cycle number. It also has symmetric spectrum and the bipartite graph contains the total chromatic number less than or equal to 2.

5. Which one of the following is the chromatic number of bipartite graph?
a) 1
b) 4
c) 3
d) 5

Answer: a
Explanation: A graph is known as bipartite graph if and only if it has the total chromatic number less than or equal to 2. The smallest number of graphs needed to color the graph is the chromatic number.

6. Which graph has a size of minimum vertex cover equal to maximum matching?
a) Cartesian
b) Tree
c) Heap
d) Bipartite

Answer: d
Explanation: The Konig's theorem given the equivalence relation between the minimum vertex cover and the maximum matching in graph theory. Bipartite graph has a size of minimum vertex cover equal to maximum matching.

7. Which theorem gives the relation between the minimum vertex cover and maximum matching?
a) Konig's Theorem
b) Kirchhoff's Theorem
c) Kuratowski's Theorem
d) Kelmans Theorem

Answer: a
Explanation: The Konig's theorem given the equivalence relation between the minimum vertex cover and the maximum matching in graph theory. Bipartite graph has a size of minimum vertex cover equal to maximum matching.

8. Which of the following is not a property of perfect graph?
a) Compliment of Line Graph of Bipartite Graph
b) Compliment of Bipartite Graph
c) Line Graph of Bipartite Graph
d) Line Graph

Answer: d
Explanation: TThe Compliment of Line Graph of Bipartite Graph, Compliment of Bipartite Graph, Line Graph of Bipartite Graph and every Bipartite Graph is known as a perfect graph in graph theory. Normal line graph is not a perfect graph whereas line perfect graph is a graph whose line graph is a perfect graph.

9. Which of the following graphs don't have chromatin number less than or equal to 2?
a) Compliment of Line Graph of Bipartite Graph
b) Compliment of Bipartite Graph
c) Line Graph of Bipartite Graph
d) Wheel graph

Answer: d

Explanation: The perfect bipartite graph has chromatic number 2. Also, the Compliment of Line Graph of Bipartite Graph, Compliment of Bipartite Graph, Line Graph of Bipartite Graph and every Bipartite Graph is known as perfect graph in graph theory. Wheel graph Wn has chromatin number 3 if n is odd and 4 if n is even.

10. Which of the following has maximum clique size 2?
a) Perfect graph
b) Tree
c) Histogram
d) Cartesian

Answer: a
Explanation: The perfect bipartite graph has clique size 2. Also, the clique size of Compliment of Line Graph of Bipartite Graph, Compliment of Bipartite Graph, Line Graph of Bipartite Graph and every Bipartite Graph is 2.

11. What is the chromatic number of compliment of line graph of bipartite graph?
a) 0
b) 1
c) 2
d) 3

Answer: c
Explanation: The perfect bipartite graph has chromatic number 2. So the Compliment of Line Graph of Bipartite Graph, Compliment of Bipartite Graph, Line Graph of Bipartite Graph and every Bipartite Graph has chromatic number 2.

12. What is the clique size of the line graph of bipartite graph?
a) 0
b) 1
c) 2
d) 3

Answer: c
Explanation: The perfect bipartite graph has clique size 2. So the clique size of Compliment of Line Graph of Bipartite Graph, Compliment of Bipartite Graph, Line Graph of Bipartite Graph and every Bipartite Graph is 2.Answer: b
Explanation: A graph is known as bipartite graph if and only if it has the total chromatic number less than or equal to 2. The smallest number of graphs needed to color the graph is the chromatic number. But the chromatic number cannot be negative.Answer: a
Explanation: Berge theorem proves the forbidden graph characterization of every perfect graphs. Because of that reason every bipartite graph is perfect graph.

15. Which structure can be modelled by using Bipartite graph?
a) Hypergraph
b) Perfect Graph
c) Hetero Graph
d) Directed Graph

Answer: a
Explanation: A combinatorial structure such as Hypergraph can be made using the bipartite graphs. A hypergraph in graph theory is a type of graph in which edge can join any number of vertices.

1. Which type of graph has all the vertex of the first set connected to all the vertex of the second set?
a) Bipartite
b) Complete Bipartite
c) Cartesian
d) Pie

Answer: b
Explanation: The graph is known as Bipartite if the graph does not contain any odd length cycle in it. The complete bipartite graph has all the vertex of first set connected to all the vertex of second set.

2. Which graph is also known as biclique?
a) Histogram
b) Complete Bipartite
c) Cartesian
d) Tree

Answer: b
Explanation: A graph is known as complete bipartite graph if and only if it has all the vertex of first set connected to all the vertex of second set. Complete Bipartite graph is also known as Biclique.

3. Which term defines all the complete bipartite graph that are trees?
a) Symmetric
b) Anti – Symmetric
c) Circular
d) Stars

Answer: d
Explanation: Star is a complete bipartite graph with one internal node and k leaves. Therefore, all complete bipartite graph which is trees are known as stars in graph theory.

4. How many edges does a n vertex triangle free graph contains?
a) n^2
b) $n^2 + 2$
c) $n^2 / 4$
d) n^3

Answer: c
Explanation: A n vertex triangle free graph contains a total of $n^2 / 4$ number of edges. This is stated by Mantel's Theorem which is a special case in Turan's theorem for r=2.

5. Which graph is used to define the claw free graph?
a) Bipartite Graph
b) Claw Graph
c) Star Graph
d) Cartesian Graph

Answer: b
Explanation: Star is a complete bipartite graph with one internal node and k leaves. Star with three edges is called a claw. Hence this graph is used to define claw free graph.

6. What is testing of a complete bipartite subgraph in a bipartite graph problem called?
a) P Problem
b) P-Complete Problem
c) NP Problem
d) NP-Complete Problem

Answer: d
Explanation: NP stands for nondeterministic polynomial time. In a bipartite graph, the testing of a complete bipartite subgraph in a bipartite graph is an NP-Complete Problem.

7. Which graph cannot contain K3, 3 as a minor of graph?
a) Planar Graph
b) Outer Planar Graph
c) Non Planar Graph
d) Inner Planar Graph

Answer: a

Explanation: Minor graph is formed by deleting certain number of edges from a graph or by deleting certain number off vertices from a graph. Hence Planar graph cannot contain K3, 3 as a minor graph.

8. Which of the following is not an Eigen value of the adjacency matrix of the complete bipartite graph?
a) $(nm)^{1/2}$
b) $(-nm)^{1/2}$
c) 0
d) nm

Answer: d
Explanation: The adjacency matrix is a square matrix that is used to represent a finite graph. Therefore, the Eigen values for the complete bipartite graph is found to be $(nm)^{1/2}$, $(-nm)^{1/2}$, 0.

9. Which complete graph is not present in minor of Outer Planar Graph?
a) K3, 3
b) K3, 1
c) K3, 2
d) K1, 1

Answer: c
Explanation: Minor graph is formed by deleting certain number of edges from a graph or by deleting certain number off vertices from a graph. Hence Outer Planar graph cannot contain K3, 2 as a minor graph.Answer: a
Explanation: In graph theory, Moore graph is defined as a regular graph that has a degree d and diameter k. therefore, every complete bipartite graph is a Moore Graph.

11. What is the multiplicity for the adjacency matrix of complete bipartite graph for 0 Eigen value?
a) 1
b) n + m – 2
c) 0
d) 2

Answer: b
Explanation: The adjacency matrix is a square matrix that is used to represent a finite graph. The multiplicity of the adjacency matrix off complete bipartite graph with Eigen Value 0 is n + m – 2.

12. Which of the following is not an Eigen value of the Laplacian matrix of the complete bipartite graph?
a) n + m
b) n
c) 0
d) n*m

Answer: d
Explanation: The laplacian matrix is used to represent a finite graph in the mathematical field of Graph Theory. Therefore, the Eigen values for the complete bipartite graph is found to be n + m, n, m, 0.

13. What is the multiplicity for the laplacian matrix of the complete bipartite graph for n Eigen value?
a) 1
b) m-1
c) n-1
d) 0

Answer: b
Explanation: The laplacian matrix is used to represent a finite graph in the mathematical field of Graph Theory. The multiplicity of the laplacian matrix of complete bipartite graph with Eigen Value n is m-1.Answer: a
Explanation: Yes, the modular graph in graph theory is defined as an undirected graph in which all three vertices have at least one median vertex. So all complete bipartite graph is called modular graph.

15. How many spanning trees does a complete bipartite graph contain?
a) n^m
b) $m^{n-1} * n^{n-1}$
c) 1
d) 0

Answer: b
Explanation: Spanning tree of a given graph is defined as the subgraph or the tree with all the given vertices but having minimum number of edges. So, there are a total of $m^{n-1} * n^{n-1}$ spanning trees for a complete bipartite graph.

1. Recursion is a method in which the solution of a problem depends on _____
a) Larger instances of different problems
b) Larger instances of the same problem
c) Smaller instances of the same problem
d) Smaller instances of different problems

Answer: c
Explanation: In recursion, the solution of a problem depends on the solution of smaller instances of the same problem.

2. Which of the following problems can't be solved using recursion?
a) Factorial of a number
b) Nth fibonacci number
c) Length of a string
d) Problems without base case

Answer: d
Explanation: Problems without base case leads to infinite recursion call. In general, we will assume a base case to avoid infinite recursion call. Problems like finding Factorial of a number, Nth Fibonacci number and Length of a string can be solved using recursion.

3. Recursion is similar to which of the following?
a) Switch Case
b) Loop
c) If-else
d) if elif else

Answer: b
Explanation: Recursion is similar to a loop.

4. In recursion, the condition for which the function will stop calling itself is _____
a) Best case
b) Worst case
c) Base case
d) There is no such condition

Answer: c
Explanation: For recursion to end at some point, there always has to be a condition for which the function will not call itself. This condition is known as base case.

5. What will happen when the below code snippet is executed?

```
void my_recursive_function()
{
   my_recursive_function();
}
int main()
{
   my_recursive_function();
   return 0;
```

```
}
```

a) The code will be executed successfully and no output will be generated
b) The code will be executed successfully and random output will be generated
c) The code will show a compile time error
d) The code will run for some time and stop when the stack overflows

Answer: d
Explanation: Every function call is stored in the stack memory. In this case, there is no terminating condition(base case). So, my_recursive_function() will be called continuously till the stack overflows and there is no more space to store the function calls. At this point of time, the program will stop abruptly.

6. What is the output of the following code?

```
void my_recursive_function(int n)
{
    if(n == 0)
    return;
    printf("%d ",n);
    my_recursive_function(n-1);
}
int main()
{
    my_recursive_function(10);
    return 0;
}
```

a) 10
b) 1
c) 10 9 8 … 1 0
d) 10 9 8 … 1

Answer: d
Explanation: The program prints the numbers from 10 to 1.

7. What is the base case for the following code?

```
void my_recursive_function(int n)
{
     if(n == 0)
     return;
     printf("%d ",n);
     my_recursive_function(n-1);
}
int main()
{
     my_recursive_function(10);
     return 0;
}
```

a) return
b) printf(“%d “, n)
c) if(n == 0)
d) my_recursive_function(n-1)

Answer: c
Explanation: For the base case, the recursive function is not called. So, “if(n == 0)” is the base case.

8. How many times is the recursive function called, when the following code is executed?

```
void my_recursive_function(int n)
{
     if(n == 0)
     return;
```

```
     printf("%d ",n);
     my_recursive_function(n-1);
}
int main()
{
     my_recursive_function(10);
     return 0;
}
```

a) 9
b) 10
c) 11
d) 12

Answer: c
Explanation: The recursive function is called 11 times.

9. What does the following recursive code do?

```
void my_recursive_function(int n)
{
     if(n == 0)
     return;
     my_recursive_function(n-1);
     printf("%d ",n);
}
int main()
{
     my_recursive_function(10);
     return 0;
}
```

a) Prints the numbers from 10 to 1
b) Prints the numbers from 10 to 0
c) Prints the numbers from 1 to 10
d) Prints the numbers from 0 to 10

Answer: c
Explanation: The above code prints the numbers from 1 to 10.

10. Which of the following statements is true?
a) Recursion is always better than iteration
b) Recursion uses more memory compared to iteration
c) Recursion uses less memory compared to iteration
d) Iteration is always better and simpler than recursion

Answer: b
Explanation: Recursion uses more memory compared to iteration because every time the recursive function is called, the function call is stored in stack.

11. What will be the output of the following code?

```
int cnt=0;
void my_recursive_function(int n)
{
     if(n == 0)
     return;
     cnt++;
     my_recursive_function(n/10);
}
int main()
{
     my_recursive_function(123456789);
     printf("%d",cnt);
```

```
    return 0;
}
```

a) 123456789
b) 10
c) 0
d) 9

Answer: d
Explanation: The program prints the number of digits in the number 123456789, which is 9.

12. What will be the output of the following code?

```
void my_recursive_function(int n)
{
    if(n == 0)
    {
         printf("False");
           return;
    }
    if(n == 1)
    {
         printf("True");
         return;
    }
    if(n%2==0)
    my_recursive_function(n/2);
    else
    {
         printf("False");
         return;
    }

}
int main()
{
     my_recursive_function(100);
     return 0;
}
```

Answer: b
Explanation: The function checks if a number is a power of 2. Since 100 is not a power of 2, it prints false.

13. What is the output of the following code?

```
int cnt = 0;
void my_recursive_function(char *s, int i)
{
     if(s[i] == '\0')
        return;
     if(s[i] == 'a' || s[i] == 'e' || s[i] == 'i' || s[i] == 'o' || s[i] == 'u')
     cnt++;
     my_recursive_function(s,i+1);
}
int main()
{
     my_recursive_function("thisisrecursion",0);
     printf("%d",cnt);
     return 0;
}
```

a) 6
b) 9
c) 5
d) 10

Answer: a
Explanation: The function counts the number of vowels in a string. In this case the number is vowels is 6.

14. What is the output of the following code?

```
void my_recursive_function(int *arr, int val, int idx, int len)
{
    if(idx == len)
    {
         printf("-1");
         return ;
    }
    if(arr[idx] == val)
    {
         printf("%d",idx);
         return;
    }
    my_recursive_function(arr,val,idx+1,len);
}
int main()
{
     int array[10] = {7, 6, 4, 3, 2, 1, 9, 5, 0, 8};
     int value = 2;
     int len = 10;
     my_recursive_function(array, value, 0, len);
     return 0;
}
```

a) 3
b) 4
c) 5
d) 6

Answer: b
Explanation: The program searches for a value in the given array and prints the index at which the value is found. In this case, the program searches for value = 2. Since, the index of 2 is 4(0 based indexing), the program prints 4.

1. In general, which of the following methods isn't used to find the factorial of a number?
a) Recursion
b) Iteration
c) Dynamic programming
d) Non iterative / recursive

Answer: d
Explanation: In general we use recursion, iteration and dynamic programming to find the factorial of a number. We can also implement without using iterative / recursive method by using tgammal() method. Most of us never use it generally.

2. Which of the following recursive formula can be used to find the factorial of a number?
a) fact(n) = n * fact(n)
b) fact(n) = n * fact(n+1)
c) fact(n) = n * fact(n-1)
d) fact(n) = n * fact(1)

Answer: c
*Explanation: fact(n) = n * fact(n – 1) can be used to find the factorial of a number.*

3. Consider the following iterative implementation to find the factorial of a number. Which of the lines should be inserted to complete the below code?

```
int main()
{
    int n = 6, i;
    int fact = 1;
```

```
    for(i=1;i<=n;i++)
      _________;
    printf("%d",fact);
    return 0;
}
```

a) fact = fact + i
b) fact = fact * i
c) i = i * fact
d) i = i + fact

Answer: b
*Explanation: The line "fact = fact * i" should be inserted to complete the above code.*

4. Consider the following recursive implementation to find the factorial of a number. Which of the lines should be inserted to complete the below code?

```
int fact(int n)
{
     if(_________)
        return 1;
     return n * fact(n - 1);
}
int main()
{
      int n = 5;
      int ans = fact(n);
      printf("%d",ans);
      return 0;
}
```

a) n = 0
b) n != 0
c) n == 0
d) n == 1

Answer: c
Explanation: The line "n == 0" should be inserted to complete the above code.
Note: "n == 1" cannot be used because it does not take care of the case when n = 0, i.e when we want to find the factorial of 0.

5. The time complexity of the following recursive implementation to find the factorial of a number is _____

```
int fact(int n)
{
     if(_________)
        return 1;
     return n * fact(n - 1);
}
int main()
{
      int n = 5;
      int ans = fact(n);
      printf("%d",ans);
      return 0;
}
```

a) O(1)
b) O(n)
c) $O(n^2)$
d) $O(n^3)$

Answer: b
Explanation: The time complexity of the above recursive implementation to find the factorial of a number is O(n).

6. What is the space complexity of the following recursive implementation to find the factorial of a number?

```
int fact(int n)
{
     if(_________)
        return 1;
     return n * fact(n - 1);
}
int main()
{
      int n = 5;
      int ans = fact(n);
      printf("%d",ans);
      return 0;
}
```

a) O(1)
b) O(n)
c) O(n^2)
d) O(n^3)

Answer: a
Explanation: The space complexity of the above recursive implementation to find the factorial of a number is O(1).

7. Consider the following recursive implementation to find the factorial of a number. Which of the lines is the base case?

```
int fact(int n)
{
     if(n == 0)
        return 1;
     return n * fact(n - 1);
}
int main()
{
      int n = 5;
      int ans = fact(n);
      printf("%d",ans);
      return 0;
}
```

a) return 1
b) return n * fact(n-1)
c) if(n == 0)
d) if(n == 1)

Answer: c
Explanation: The line "if(n == 0)" is the base case.

8. What is the output of the following code?

```
int fact(int n)
{
      if(n == 0)
        return 1;
      return n * fact(n - 1);
}
int main()
{
      int n = 0;
      int ans = fact(n);
      printf("%d",ans);
      return 0;
}
```

a) 0
b) 1
c) 2
d) 3

Answer: b
Explanation: The program prints 0!, which is 1.

9. What is the output of the following code?

```
int fact(int n)
{
      if(n == 0)
        return 1;
      return n * fact(n - 1);
}
int main()
{
      int n = 1;
      int ans = fact(n);
      printf("%d",ans);
      return 0;
}
```

a) 0
b) 1
c) 2
d) 3

Answer: b
Explanation: The program prints 1!, which is 1.

10. How many times will the function fact() be called when the following code is executed?

```
int fact(int n)
{
      if(n == 0)
        return 1;
      return n * fact(n - 1);
}
int main()
{
      int n = 5;
      int ans = fact(n);
      printf("%d",ans);
      return 0;
}
```

a) 4
b) 5
c) 6
d) 7

Answer: c
Explanation: The fact() function will be called 6 times with the following arguments:
fact(5), fact(4), fact(3), fact(2), fact(1), fact(0).

11. What is the output of the following code?

```
int fact(int n)
{
      if(n == 0)
        return 1;
      return n * fact(n - 1);
```

```
}
int main()
{
      int n = 5;
      int ans = fact(n);
      printf("%d",ans);
      return 0;
}
```

a) 24
b) 120
c) 720
d) 1

Answer: b
Explanation: The function prints 5!, which is 120.

1. Suppose the first fibonnaci number is 0 and the second is 1. What is the sixth fibonnaci number?
a) 5
b) 6
c) 7
d) 8

Answer: a
Explanation: The sixth fibonnaci number is 5.

2. Which of the following is not a fibonnaci number?
a) 8
b) 21
c) 55
d) 14

Answer: d
Explanation: 14 is not a fibonnaci number.

3. Which of the following option is wrong?
a) Fibonacci number can be calculated by using Dynamic programming
b) Fibonacci number can be calculated by using Recursion method
c) Fibonacci number can be calculated by using Iteration method
d) No method is defined to calculate Fibonacci number

Answer: d
Explanation: Fibonacci number can be calculated by using Dynamic Programming, Recursion method, Iteration Method.

4. Consider the following iterative implementation to find the nth fibonacci number?

```
int main()
{
    int  n = 10,i;
    if(n == 1)
        printf("0");
    else if(n == 2)
        printf("1");
    else
    {
        int a = 0, b = 1, c;
        for(i = 3; i <= n; i++)
        {
            c = a + b;
            ________;
                        ________;
        }
        printf("%d",c);
```

```
    }
   return 0;
}
```

5. Which of the following recurrence relations can be used to find the nth fibonacci number?
a) F(n) = F(n) + F(n – 1)
b) F(n) = F(n) + F(n + 1)
c) F(n) = F(n – 1)
d) F(n) = F(n – 1) + F(n – 2)

Answer: d
Explanation: The relation F(n) = F(n – 1) + F(n – 2) can be used to find the nth fibonacci number.

6. Consider the following recursive implementation to find the nth fibonacci number:

```
int fibo(int n)
{
     if(n == 1)
        return 0;
     else if(n == 2)
        return 1;
     return ________;
}
int main()
{
     int n = 5;
     int ans = fibo(n);
     printf("%d",ans);
     return 0;
}
```

Which of the following lines should be inserted to complete the above code?
a) fibo(n – 1)
b) fibo(n – 1) + fibo(n – 2)
c) fibo(n) + fibo(n – 1)
d) fibo(n – 2) + fibo(n – 1)

Answer: b
Explanation: The line fibo(n – 1) + fibo(n – 2) should be inserted to complete the above code.

7. Consider the following recursive implementation to find the nth fibonnaci number:

```
int fibo(int n)
{
     if(n == 1)
        return 0;
     else if(n == 2)
        return 1;
     return fibo(n - 1) + fibo(n - 2);
}
int main()
{
     int n = 5;
     int ans = fibo(n);
     printf("%d",ans);
     return 0;
}
```

Which of the following is the base case?
a) if(n == 1)
b) else if(n == 2)
c) return fibo(n – 1) + fibo(n – 2)
d) both if(n == 1) and else if(n == 2)

Answer: d
Explanation: Both if(n == 1) and else if(n == 2) are the base cases.

8. What is the time complexity of the following recursive implementation to find the nth fibonacci number?

```
int fibo(int n)
{
     if(n == 1)
        return 0;
     else if(n == 2)
        return 1;
     return fibo(n - 1) + fibo(n - 2);
}
int main()
{
     int n = 5;
     int ans = fibo(n);
     printf("%d",ans);
     return 0;
}
```

a) O(1)
b) O(2*n)
c) O(n^2)
d) O(2^n)

Answer: d
Explanation: The time complexity of the above recursive implementation to find the nth fibonacci number is O(2^n).

9. What is the space complexity of the following recursive implementation to find the nth fibonacci number?

```
int fibo(int n)
{
     if(n == 1)
        return 0;
     else if(n == 2)
        return 1;
     return fibo(n - 1) + fibo(n - 2);
}
int main()
{
     int n = 5;
     int ans = fibo(n);
     printf("%d",ans);
     return 0;
}
```

a) O(1)
b) O(2*n)
c) O(n^2)
d) O(2^n)

Answer: a
Explanation: The space complexity of the above recursive implementation to find the nth fibonacci number is O(1).

10. What is the output of the following code?

```
int fibo(int n)
{
      if(n == 1)
        return 0;
      else if(n == 2)
        return 1;
      return fibo(n - 1) + fibo(n - 2);
```

```
}
int main()
{
     int n = -1;
     int ans = fibo(n);
     printf("%d",ans);
     return 0;
}
```

a) 0
b) 1
c) Compile time error
d) Runtime error

Answer: d
Explanation: Since negative numbers are not handled by the program, the function fibo() will be called infinite times and the program will produce a runtime error when the stack overflows.

11. What is the output of the following code?

```
int fibo(int n)
{
      if(n == 1)
        return 0;
      else if(n == 2)
        return 1;
      return fibo(n - 1) + fibo(n - 2);
}
int main()
{
     int n = 5;
     int ans = fibo(n);
     printf("%d",ans);
     return 0;
}
```

a) 1
b) 2
c) 3
d) 5

Answer: c
Explanation: The program prints the 5th fibonacci number, which is 3.

12. How many times will the function fibo() be called when the following code is executed?

```
int fibo(int n)
{
      if(n == 1)
         return 0;
      else if(n == 2)
        return 1;
      return fibo(n - 1) + fibo(n - 2);
}
int main()
{
     int n = 5;
     int ans = fibo(n);
     printf("%d",ans);
     return 0;
}
```

a) 5
b) 6
c) 8

d) 9

Answer: d
Explanation: The function fibo() will be called 9 times, when the above code is executed.

13. What is the output of the following code?

```
int fibo(int n)
{
      if(n == 1)
        return 0;
      else if(n == 2)
        return 1;
      return fibo(n - 1) + fibo(n - 2);
}
int main()
{
     int n = 10;
     int ans = fibo(n);
     printf("%d",ans);
     return 0;
}
```

a) 21
b) 34
c) 55
d) 13

Answer: b
Explanation: The program prints the 10th fibonacci number, which is 34.

1. Which of the following option is wrong about natural numbers?
a) Sum of first n natural numbers can be calculated by using Iteration method
b) Sum of first n natural numbers can be calculated by using Recursion method
c) Sum of first n natural numbers can be calculated by using Binomial coefficient method
d) No method is prescribed to calculate sum of first n natural number

Answer: d
Explanation: All of the above mentioned methods can be used to find the sum of first n natural numbers.

2. Which of the following gives the sum of the first n natural numbers?
a) nC2
b) (n-1)C2
c) (n+1)C2
d) (n+2)C2

Answer: c
Explanation: The sum of first n natural numbers is given by n(n+1)/2, which is equal to (n+1)C2.*

3. Consider the following iterative solution to find the sum of first n natural numbers:

```
#include<stdio.h>
int get_sum(int n)
{
      int sm = 0, i;
      for(i = 1; i <= n; i++)
        ________;
      return sm;
}
int main()
{
    int n = 10;
    int ans = get_sum(n);
```

```
    printf("%d",ans);
    return 0;
}
```

Which of the following lines completes the above code?
a) sm = i
b) sm += i
c) i = sm
d) i += sm

Answer: b
Explanation: The line "sm += i" completes the above code.

4. What is the output of the following code?

```
#include<stdio.h>
int get_sum(int n)
{
      int sm, i;
      for(i = 1; i <= n; i++)
        sm += i;
      return sm;
}
int main()
{
    int n = 10;
    int ans = get_sum(n);
    printf("%d",ans);
    return 0;
}
```

a) 55
b) 45
c) 35
d) Depends on compiler

Answer: d
Explanation: Since the variable "sm" is not initialized to 0, it will produce a garbage value. Some compiler will automatically initialises variables to 0 if not initialised. In that case the value is 55. Hence the value depends on the compiler.

5. What is the time complexity of the following iterative method used to find the sum of the first n natural numbers?

```
#include<stdio.h>
int get_sum(int n)
{
      int sm, i;
      for(i = 1; i <= n; i++)
        sm += i;
      return sm;
}
int main()
{
    int n = 10;
    int ans = get_sum(n);
    printf("%d",ans);
    return 0;
}
```

a) O(1)
b) O(n)
c) $O(n^2)$
d) $O(n^3)$

Answer: b
Explanation: The time complexity of the above iterative method used to find the sum of first n natural numbers is O(n).

6. Consider the following code:

```
#include<stdio.h>
int recursive_sum(int n)
{
      if(n == 0)
        return 0;
      return ________;
}
int main()
{
    int n = 5;
    int ans = recursive_sum(n);
    printf("%d",ans);
    return 0;
}
```

Which of the following lines is the recurrence relation for the above code?
a) (n – 1) +recursive_sum(n)
b) n + recursive_sum(n)
c) n + recursive_sum(n – 1)
d) (n – 1) + recursive_sum(n – 1)

Answer: c
Explanation: The recurrence relation for the above code is: n + recursive_sum(n – 1).

7. Consider the following code:

```
#include<stdio.h>
int recursive_sum(int n)
{
      if(n == 0)
        return 0;
      return n + recursive_sum(n - 1);
}
int main()
{
     int n = 5;
     int ans = recursive_sum(n);
     printf("%d",ans);
     return 0;
}
```

Which of the following is the base case for the above recursive code?
a) if(n == 0)
b) return 0
c) return n + recursive_sum(n – 1)
d) if(n == 1)

Answer: a
Explanation: "if(n == 0)" is the base case for the above recursive code.

8. What is the time complexity of the following recursive implementation used to find the sum of the first n natural numbers?

```
#include<stdio.h>
int recursive_sum(int n)
{
      if(n == 0)
        return 0;
      return n + recursive_sum(n - 1);
}
```

```
int main()
{
     int n = 5;
     int ans = recursive_sum(n);
     printf("%d",ans);
     return 0;
}
```

a) O(1)
b) O(n)
c) O(n^2)
d) O(n^3)

Answer: b
Explanation: The time complexity of the above recursive implementation used to find the sum of first n natural numbers is O(n).

9. Which of the following methods used to find the sum of first n natural numbers has the least time complexity?
a) Recursion
b) Iteration
c) Binomial coefficient
d) All have equal time complexity

Answer: c
Explanation: Recursion and iteration take O(n) time to find the sum of first n natural numbers while binomial coefficient takes O(1) time.

10. What is the output of the following code?

```
#include<stdio.h>
int recursive_sum(int n)
{
      if(n == 0)
        return 0;
      return n + recursive_sum(n - 1);
}
int main()
{
      int n = 5;
      int ans = recursive_sum(n);
      printf("%d",ans);
      return 0;
}
```

a) 10
b) 15
c) 21
d) 14

Answer: b
Explanation: The above code prints the sum of first 5 natural numbers, which is 15.

11. How many times is the function recursive_sum() called when the following code is executed?

```
#include<stdio.h>
int recursive_sum(int n)
{
      if(n == 0)
        return 0;
      return n + recursive_sum(n - 1);
}
int main()
{
     int n = 5;
```

```
    int ans = recursive_sum(n);
    printf("%d",ans);
    return 0;
}
```

a) 4
b) 5
c) 6
d) 7

Answer: c
Explanation: The function recursive_sum is called 6 times when the following code is executed.

12. What is the output of the following code?

```
#include<stdio.h>
int recursive_sum(int n)
{
      if(n == 0)
        return 0;
      return n + recursive_sum(n - 1);
}
int main()
{
     int n = 0;
     int ans = recursive_sum(n);
     printf("%d",ans);
     return 0;
}
```

a) -1
b) 0
c) 1
d) runtime error

Answer: b
Explanation: The program prints the sum of first 0 natural numbers, which is 0.

13. What is the output of the following code?

```
#include<stdio.h>
int recursive_sum(int n)
{
      if(n == 0)
        return 0;
      return n + recursive_sum(n - 1);
}
int main()
{
     int n = -4;
     int ans = recursive_sum(n);
     printf("%d",ans);
     return 0;
}
```

a) 0
b) -10
c) 1
d) runtime error

Answer: d
Explanation: The above code doesn't handle the case of negative numbers and so the function recursive_sum() will be called again and again till the stack overflows and the program produces a runtime error.

1. Which of the following is not another name for GCD(Greatest Common Divisor)?
a) LCM
b) GCM
c) GCF
d) HCF

Answer: a
Explanation: : LCM (Least Common Multiple) and GCD are not same. GCM (Greatest Common Measure), GCF (Greatest Common Factor), HCF (Highest Common Factor) are other names for GCD.

2. What is the GCD of 8 and 12?
a) 8
b) 12
c) 2
d) 4

Answer: d
Explanation: GCD is largest positive integer that divides each of the integer. So the GCD of 8 and 12 is 4.

3. If GCD of two number is 8 and LCM is 144, then what is the second number if first number is 72?
a) 24
b) 2
c) 3
d) 16

Answer: d
*Explanation: As A * B = GCD (A, B) * LCM (A, B). So B = (144 * 8)/72 = 16.*

4. Which of the following is also known as GCD?
a) Highest Common Divisor
b) Highest Common Multiple
c) Highest Common Measure
d) Lowest Common Multiple

Answer: a
Explanation: GCM (Greatest Common Measure), GCF (Greatest Common Factor), HCF (Highest Common Factor) and HCF (Highest Common Divisor) are also known as GCD.

5. Which of the following is coprime number?
a) 54 and 24
b) 4 and 8
c) 6 and 12
d) 9 and 28

Answer: d
Explanation: Coprime numbers have GCD 1. So 9 and 28 are coprime numbers. While 54 and 24 have GCD 6, 4 and 8 have GCD 4, 6 and 12 have GCD 6.

6. In terms of Venn Diagram, which of the following expression gives GCD (Given A ∩ B ≠ Ø)?
a) Multiplication of A U B terms
b) Multiplication of A ∩ B terms
c) Multiplication of A*B terms
d) Multiplication of A-B terms

Answer: b
Explanation: In terms of Venn Diagram, the GCD is given by the intersection of two sets. So A ∩ B gives the GCD. While A U B gives the LCM.Answer: c
Explanation: In terms of Venn Diagram, the GCD is given by the intersection of two sets. So A ∩ B gives the GCD. While

A U B gives the LCM. So here A ∩ B is 5.

8. What is the GCD of a and b?
a) a + b
b) gcd (a-b, b) if a>b
c) gcd (a+b, a-b)
d) a – b

Answer: b
Explanation: As per Euclid's Algorithm, gcd (a, b) = gcd (a-b, b) if a > b or gcd (a, b) = gcd (a, b-a) if b > a.

9. What is the GCD of 48, 18, 0?
a) 24
b) 2
c) 3
d) 6

Answer: d
Explanation: GCD is largest positive integer that divides each of the integer. So the GCD of 48, 18, 0 is 6.Answer: a
Explanation: Coprime numbers have GCD 1. So 9 and 28 are coprime numbers.

11. If gcd (a, b) is defined by the expression, d=a*p + b*q where d, p, q are positive integers and a, b is both not zero, then what is the expression called?
a) Bezout's Identity
b) Multiplicative Identity
c) Sum of Product
d) Product of Sum

Answer: a
*Explanation: If gcd (a, b) is defined by the expression, d=a*p + b*q where d, p, q are positive integers and a, b is both not zero, then the expression is called Bezout's Identity and p, q can be calculated by extended form of Euclidean algorithm.Answer: a*
Explanation: The gcd function is an associative function as gcd (a, gcd (b, c)) = gcd (gcd (a, b), c).

13. Which is the correct term of the given relation, gcd (a, b) * lcm (a, b) =?
a) |a*b|
b) a + b
c) a – b
d) a / b

Answer: a
*Explanation: The gcd is closely related to lcm as gcd (a, b) * lcm (a, b) = |a*b|.*

14. Who gave the expression for the probability and expected value of gcd?
a) James E. Nymann
b) Riemann
c) Thomae
d) Euler

Answer: a
Explanation: In the year 1972, James E. Nymann showed some result to show the probability and expected value of gcd.

15. What is the computational complexity of Binary GCD algorithm where a and b are integers?
a) O (log a + log b)^2)
b) O (log (a + b))
c) O (log ab)
d) O (log a-b)

Answer: a
Explanation: From the Binary GCD algorithm, it is found that the computational complexity is O (log a + log b)^2) as the total number of steps in the execution is at most the total sum of number of bits of a and b.

1. LCM is also called as _____
a) GCD
b) SCM
c) GCF
d) HCF

Answer: b
Explanation: GCD (Greatest Common Divisor), GCF (Greatest Common Factor), HCF (Highest Common Factor) is not an alias for LCM. LCM is also called as Smallest Common Multiple(SCM).

2. What is the LCM of 8 and 13?
a) 8
b) 12
c) 20
d) 104

Answer: d
Explanation: 104 is the smallest positive integer that is divisible by both 8 and 12.

3. Which is the smallest number of 3 digits that is divisible by 2, 4, 8?
a) 100
b) 102
c) 116
d) 104

Answer: d
Explanation: LCM of 2, 4, 8 is 8. So check for the number that is divisible by 8. So 104 is the smallest number that is divisible by 2, 4, 8.

4. Which of the following is also known as LCM?
a) Lowest Common Divisor
b) Least Common Multiple
c) Lowest Common Measure
d) Highest Common Multiple

Answer: a
Explanation: Least Common Multiple is also known as LCM or Lowest Common Multiple.

5. What is the LCM of two coprime numbers?
a) 1
b) 0
c) Addition of two coprime numbers
d) Multiplication of two coprime numbers

Answer: d
Explanation: Coprime numbers have GCD 1. While LCM of coprime numbers is the product of those two coprime numbers.

6. In terms of Venn Diagram, which of the following expression gives LCM (Given A ∩ B ≠ Ø)?
a) Multiplication of A U B terms
b) Multiplication of A ∩ B terms
c) Multiplication of A*B terms
d) Multiplication of A-B terms

Answer: a
Explanation: In terms of Venn Diagram, the LCM is given by the Union of two sets. So A U B gives the LCM. While A ∩ B gives the GCD.Answer: c
Explanation: In terms of Venn Diagram, the LCM is given by the Union of two sets. So A U B gives the LCM. So product of all the terms is 180.

8. What is the lcm (a, b)?
a) a + b
b) gcd (a-b, b) if a>b
c) lcm (b, a)
d) a – b

Answer: c
Explanation: Since the LCM function is commutative, so lcm (a, b) = lcm (b, a).

9. What is the LCM of 48, 18, 6?
a) 12^2
b) 12*2
c) 3
d) 6

Answer: a
Explanation: The LCM of 48, 18, 6 is 144 and 12^2 is 144.Answer: a
Explanation: Coprime numbers have GCD 1 and LCM is the product of the two given terms. So 9 and 28 are coprime numbers.

11. What is the following expression, lcm (a, lcm (b, c) equal to?
a) lcm (a, b, c)
b) a*b*c
c) a + b + c
d) lcm (lcm (a, b), c)

Answer: d
Explanation: Since LCM function follows associativity, hence lcm (a, lcm (b, c) is equal to lcm (lcm (a, b), c).Answer: a
Explanation: The lcm function is an associative function as lcm (a, lcm (b, c) is equal to lcm (lcm (a, b), c).

13. Which is the correct term of the given relation, lcm (a, b) * gcd (a, b) =?
a) |a*b|
b) a + b
c) a – b
d) a / b

Answer: a
*Explanation: The lcm is closely related to gcd as lcm (a, b) * gcd (a, b) = |a*b|.*

14. What is the following expression, lcm (a, gcd (a, b)) equal to?
a) a
b) b
c) a*b
d) a + b

Answer: a
Explanation: Since the lcm function follows absorption laws so lcm (a, gcd (a, b)) equal to a.

15. Which algorithm is the most efficient numerical algorithm to obtain lcm?
a) Euler's Algorithm
b) Euclid's Algorithm
c) Chebyshev Function

d) Partial Division Algorithm

Answer: b
Explanation: The most efficient way of calculating the lcm of a given number is using Euclid's algorithm which computes the lcm in much lesser time compared to other algorithms.

1. Which of the following methods can be used to find the sum of digits of a number?
a) Recursion
b) Iteration
c) Greedy algorithm
d) Both recursion and iteration

Answer: d
Explanation: Both recursion and iteration can be used to find the sum of digits of a number.

2. What can be the maximum sum of digits for a 4 digit number?
a) 1
b) 16
c) 36
d) 26

Answer: c
Explanation: The sum of digits will be maximum when all the digits are 9. Thus, the sum will be maximum for the number 9999, which is 36.

3. What can be the minimum sum of digits for a 4 digit number?
a) 0
b) 1
c) 16
d) 36

Answer: b
Explanation: The sum of digits will be minimum for the number 1000 and the sum is 1.

4. Consider the following iterative implementation to find the sum of digits of a number:

```
#include<stdio.h>
int sum_of_digits(int n)
{
      int sm = 0;
      while(n != 0)
      {
          _________;
          n /= 10;
      }
      return sm;
}
int main()
{
      int n = 1234;
      int ans = sum_of_digits(n);
      printf("%d",ans);
      return 0;
}
```

Which of the following lines should be inserted to complete the above code?
a) sm += n
b) sm += n%10
c) sm += n-10
d) sm += n/10

Answer: b
Explanation: The line "sm += n % 10" adds the last digit(LSB) of the number to the current sum. Thus, the line "sm += n%10" should be added to complete the above code.

5. What is the output of the following code?

```
#include<stdio.h>
int sum_of_digits(int n)
{
      int sm = 0;
      while(n != 0)
      {
          sm += n%10;
          n /= 10;
      }
      return sm;
}
int main()
{
      int n = 1234;
      int ans = sum_of_digits(n);
      printf("%d",ans);
      return 0;
}
```

a) 1
b) 3
c) 7
d) 10

Answer: d
Explanation: The above code prints the sum of digits of the number 1234, which is 10.

6. Consider the following recursive implementation to find the sum of digits of number:

```
#include<stdio.h>
int recursive_sum_of_digits(int n)
{
      if(n == 0)
        return 0;
      return _________;
}
int main()
{
      int n = 1201;
      int ans = recursive_sum_of_digits(n);
      printf("%d",ans);
      return 0;
}
```

Which of the following lines should be inserted to complete the above code?
a) (n / 10) + recursive_sum_of_digits(n % 10)
b) (n) + recursive_sum_of_digits(n % 10)
c) (n % 10) + recursive_sum_of_digits(n / 10)
d) (n % 10) + recursive_sum_of_digits(n % 10)

Answer: c
Explanation: The line "(n % 10) + recursive_sum_of_digits(n / 10)" should be inserted to complete the above code.

7. What is the time complexity of the following recursive implementation to find the sum of digits of a number n?

```
#include<stdio.h>
int recursive_sum_of_digits(int n)
{
      if(n == 0)
```

```
        return 0;
      return _________;
}
int main()
{
      int n = 1201;
      int ans = recursive_sum_of_digits(n);
      printf("%d",ans);
      return 0;
}
```

a) O(n)
b) O(1)
c) O(len(n)), where len(n) is the number of digits in n
d) O(1/2)

Answer: c
Explanation: The time complexity of the above recursive implementation to find the sum of digits of a number is O(len(n)).

8. What is the output of the following code?

```
#include<stdio.h>
int recursive_sum_of_digits(int n)
{
      if(n == 0)
        return 0;
      return n % 10 + recursive_sum_of_digits(n/10);
}
int main()
{
      int n = 1234321;
      int ans = recursive_sum_of_digits(n);
      printf("%d",ans);
      return 0;
}
```

a) 10
b) 16
c) 15
d) 14

Answer: b
Explanation: The above code prints the sum of digits of the number 1234321, which is 16.

9. How many times is the function recursive_sum_of_digits() called when the following code is executed?

```
#include<stdio.h>
int recursive_sum_of_digits(int n)
{
      if(n == 0)
        return 0;
      return n % 10 + recursive_sum_of_digits(n/10);
}
int main()
{
      int n = 1201;
      int ans = recursive_sum_of_digits(n);
      printf("%d",ans);
      return 0;
}
```

a) 6
b) 7
c) 5

d) 9

Answer: c
Explanation: The function recursive_sum_of_digits() is called 8 times, when the following code is executed.

10. You have to find the sum of digits of a number given that the number is always greater than 0. Which of the following base cases can replace the base case for the below code?

```
#include<stdio.h>
int recursive_sum_of_digits(int n)
{
      if(n == 0)
        return 0;
      return n % 10 + recursive_sum_of_digits(n/10);
}
int main()
{
      int n = 1201;
      int ans = recursive_sum_of_digits(n);
      printf("%d",ans);
      return 0;
}
```

a) if(n == 0) return 1
b) if(n == 1) return 0
c) if(n == 1) return 1
d) no need to modify the base case

Answer: d
Explanation: None of the above mentioned base cases can replace the base case if(n == 0) return 0.

11. What is the output of the following code?

```
#include<stdio.h>
int recursive_sum_of_digits(int n)
{
      if(n == 0)
        return 0;
      return n % 10 + recursive_sum_of_digits(n/10);
}
int main()
{
      int n = 10000;
      int ans = recursive_sum_of_digits(n);
      printf("%d",ans);
      return 0;
}
```

a) 0
b) 1
c) runtime error
d) -1

Answer: b
Explanation: The program prints the sum of digits of the number 10000, which is 1.

12. What is the output of the following code?

```
#include<stdio.h>
int cnt =0;
int my_function(int n, int sm)
{
      int i, tmp_sm;
      for(i=1;i<=n;i++)
      {
```

```
            tmp_sm = recursive_sum_of_digits(i);
            if(tmp_sm == sm)
              cnt++;
      }
      return cnt;
}
int recursive_sum_of_digits(int n)
{
      if(n == 0)
        return 0;
      return n % 10 + recursive_sum_of_digits(n/10);
}
int main()
{
      int n = 20, sum = 3;
      int ans = my_function(n,sum);
      printf("%d",ans);
      return 0;
}
```

a) 0
b) 1
c) 2
d) 3

Answer: c
Explanation: The code prints the count of numbers between 1 and 20 such that the sum of their digits is 3. There are only two such numbers: 3 and 12.

1. Consider the following iterative implementation used to reverse a string:

```
#include<stdio.h>
#include<string.h>
void reverse_string(char *s)
{
     int len = strlen(s);
     int i,j;
     i=0;
     j=len-1;
     while(______)
     {
         char tmp = s[i];
         s[i] = s[j];
         s[j] = tmp;
         i++;
         j--;
     }
}
```

Which of the following lines should be inserted to complete the above code?
a) i > j
b) i < len
c) j > 0
d) i < j

Answer: d
Explanation: The line "i < j" should be inserted to complete the above code.

2. What is the output of the following code?

```
#include<stdio.h>
#include<string.h>
void reverse_string(char *s)
{
     int len = strlen(s);
     int i,j;
```

```
    i=0;
    j=len-1;
    while(i < j)
    {
        char tmp = s[i];
        s[i] = s[j];
        s[j] = tmp;
        i++;
        j--;
    }
}
int main()
{
     char s[100] = "reverse";
     reverse_string(s);
     printf("%s",s);
     return 0;
}
```

a) ersevre
b) esrever
c) eserver
d) eresevr

Answer: b
Explanation: The program reverses the string "reverse" and prints "esrever".

3. What is the time complexity of the above code used to reverse a string?
a) O(1)
b) O(n)
c) O(n^2)
d) O(n^3)

Answer: b
Explanation: The time complexity of the above code used to reverse a string is O(n).

4. What does the following code do?

```
#include<stdio.h>
#include<string.h>
void reverse_string(char *s)
{
    int len = strlen(s);
    int i,j;
    i=0;
    j=len-1;
    while(i < j)
    {
        char tmp = s[i];
        s[i] = s[j];
        s[j] = tmp;
        i++;
        j--;
    }
}
int main()
{
     char s[100] = "abcdefg";
     char t[100];
     strcpy(t,s);
     reverse_string(s);
     if(strcmp(t,s) == 0)
       printf("Yes");
     else
       printf("No");
     return 0;
```

```
}
```

a) Copies a string to another string
b) Compares two strings
c) Reverses a string
d) Checks if a string is a palindrome

Answer: d
Explanation: The main purpose of the above code is to check if a string is a palindrome.

5. What is the output of the following code?

```
#include<stdio.h>
#include<string.h>
void reverse_string(char *s)
{
     int len = strlen(s);
     int i,j;
     i=0;
     j=len-1;
     while(i < j)
     {
         char tmp = s[i];
         s[i] = s[j];
         s[j] = tmp;
         i++;
         j--;
     }
}
int main()
{
      char s[100] = "rotator";
      char t[100];
      strcpy(t,s);
      reverse_string(s);
      if(strcmp(t,s) == 0)
        printf("Yes");
      else
        printf("No");
      return 0;
}
```

a) Yes
b) No
c) Runtime error
d) Compile time error

Answer: a
Explanation: The program checks if a string is a palindrome. Since the string rotator is a palindrome, it prints “yes”.

6. Consider the following recursive implementation used to reverse a string:

```
void recursive_reverse_string(char *s, int left, int right)
{
     if(left < right)
     {
         char tmp = s[left];
         s[left] = s[right];
         s[right] = tmp;
         _________;
     }
}
```

Which of the following lines should be inserted to complete the above code?
a) recursive_reverse_string(s, left+1, right+1)

b) recursive_reverse_string(s, left-1, right-1)
c) recursive_reverse_string(s, left+1, right-1)
d) recursive_reverse_string(s, left-1, right+1)

Answer: c
Explanation: The line "recursive_reverse_string(s, left+1, right-1)" should be inserted to complete the above code.

7. What is the output of the following code?

```
#include<stdio.h>
#include<string.h>
void recursive_reverse_string(char *s, int left, int right)
{
     if(left < right)
     {
         char tmp = s[left];
         s[left] = s[right];
         s[right] = tmp;
         recursive_reverse_string(s, left+1, right-1);
     }
}
int main()
{
     char s[100] = "recursion";
     int len = strlen(s);
     recursive_reverse_string(s,0,len-1);
     printf("%s",s);
     return 0;
}
```

a) recursion
b) nsoirucer
c) noisrcuer
d) noisrucer

Answer: d
Explanation: The program prints the reversed string of "recursion", which is "noisrucer".

8. How many times is the function recursive_reverse_string() called when the following code is executed?

```
#include<stdio.h>
#include<string.h>
void recursive_reverse_string(char *s, int left, int right)
{
     if(left < right)
     {
         char tmp = s[left];
         s[left] = s[right];
         s[right] = tmp;
         recursive_reverse_string(s, left+1, right-1);
     }
}
int main()
{
     char s[100] = "madam";
     int len = strlen(s);
     recursive_reverse_string(s,0,len-1);
     printf("%s",s);
     return 0;
}
```

a) 3
b) 4
c) 5
d) 6

Answer: a
Explanation: The function recursive_reverse_string() is called 3 times when the above code is executed.

9. What is the time complexity of the above recursive implementation used to reverse a string?
a) O(1)
b) O(n)
c) O(n^2)
d) O(n^3)

Answer: b
Explanation: The time complexity of the above recursive implementation used to reverse a string is O(n).

10. In which of the following cases is the reversal of a string not equal to the original string?
a) Palindromic strings
b) Strings of length 1
c) Empty String
d) Strings of length 2

Answer: d
Explanation: String "ab" is a string of length 2 whose reversal is not the same as the given one. Palindromic Strings, String of length 1 and Empty Strings case – the reversal is the same as the one given.

1. Which of the following is the binary representation of 100?
a) 1010010
b) 1110000
c) 1100100
d) 1010101

Answer: c
Explanation: 100 = 64 + 32 + 4 = $2^6 + 2^5 + 2^2$ = 1100100.

2. Consider the following iterative code used to convert a decimal number to its equivalent binary:

```
#include<stdio.h>
void dec_to_bin(int n)
{
      int arr[31],len = 0,i;
      if(n == 0)
      {
          arr[0] = 0;
          len = 1;
      }
      while(n != 0)
      {
          arr[len++] = n % 2;
          _______;
      }
      for(i=len-1; i>=0; i--)
        printf("%d",arr[i]);
}
int main()
{
     int n = 10;
     dec_to_bin(n);
     return 0;
}
```

Which of the following lines should be inserted to complete the above code?
a) n–
b) n /= 2
c) n /= 10
d) n++

Answer: b
Explanation: The line "n /= 2" should be inserted to complete the above code.

3. What is the output of the following code?

```
#include<stdio.h>
void dec_to_bin(int n)
{
    int arr[31],len = 0,i;
    if(n == 0)
    {
        arr[0] = 0;
        len = 1;
    }
    while(n != 0)
    {
        arr[len++] = n % 2;
        n /= 2;
    }
    for(i=len-1; i>=0; i--)
        printf("%d",arr[i]);
}
int main()
{
    int n = 63;
    dec_to_bin(n);
    return 0;
}
```

a) 111111
b) 111011
c) 101101
d) 101010

Answer: a
Explanation: The program prints the binary equivalent of 63, which is 111111.

4. What is the output of the following code?

```
#include<stdio.h>
void dec_to_bin(int n)
{
      int arr[31],len = 0,i;
      if(n == 0)
      {
          arr[0] = 0;
          len = 1;
      }
      while(n != 0)
      {
          arr[len++] = n % 2;
          n /= 2;
      }
      for(i=len-1; i>=0; i--)
        printf("%d",arr[i]);
}
int main()
{
     int n = 0;
     dec_to_bin(n);
     return 0;
}
```

a) 0
b) 1
c) Runtime error

d) Garbage value

Answer: a
Explanation: The program prints the binary equivalent of 0, which is 0.

5. What is the time complexity of the following code used to convert a decimal number to its binary equivalent?

```
#include<stdio.h>
void dec_to_bin(int n)
{
      int arr[31],len = 0,i;
      if(n == 0)
      {
          arr[0] = 0;
          len = 1;
      }
      while(n != 0)
      {
          arr[len++] = n % 2;
          n /= 2;
      }
      for(i=len-1; i>=0; i--)
        printf("%d",arr[i]);
}
int main()
{
     int n = 0;
     dec_to_bin(n);
     return 0;
}
```

a) O(1)
b) O(n)
c) O(n^2)
d) O(logn)

Answer: d
Explanation: The time complexity of the above code used to convert a decimal number to its binary equivalent is O(logn).

6. Consider the following recursive implementation used to convert a decimal number to its binary equivalent:

```
#include<stdio.h>
int arr[31], len = 0;
void recursive_dec_to_bin(int n)
{
      if(n == 0 && len == 0)
      {
          arr[len++] = 0;
          return;
      }
      if(n == 0)
          return;
        __________;
      recursive_dec_to_bin(n/2);
}
int main()
{
     int n = 100,i;
     recursive_dec_to_bin(n);
     for(i=len-1; i>=0; i--)
     printf("%d",arr[i]);
     return 0;
}
```

Which of the following lines should be inserted to complete the above code?

a) arr[len] = n
b) arr[len] = n % 2
c) arr[len++] = n % 2
d) arr[len++] = n

Answer: c
Explanation: The line "arr[len++] = n % 2" should be inserted to complete the above code.

7. Consider the following code:

```
#include<stdio.h>
int arr[31], len = 0;
void recursive_dec_to_bin(int n)
{
      if(n == 0 && len == 0)
      {
          arr[len++] = 0;
          return;
      }
      if(n == 0)
         return;
      arr[len++] = n % 2;
      recursive_dec_to_bin(n/2);
}
```

Which of the following lines is the base case for the above code?
a) if(n ==0 && len == 0)
b) if(n == 0)
c) if(n ==0 && len == 0) and if(n == 0)
d) if(n == 1)

Answer: c
Explanation: Both of the above mentioned lines are the base cases for the above code.

8. What is the output of the following code?

```
#include<stdio.h>
int arr[31], len = 0;
void recursive_dec_to_bin(int n)
{
      if(n == 0 && len == 0)
      {
          arr[len++] = 0;
          return;
      }
      if(n == 0)
         return;
      arr[len++] = n % 2;
      recursive_dec_to_bin(n/2);
}
int main()
{
     int n = -100,i;
     recursive_dec_to_bin(n);
     for(i=len-1; i>=0; i--)
     printf("%d",arr[i]);
     return 0;
}
```

a) -1100100
b) 1100100
c) 2's complement of 1100100
d) Garbage value

Answer: d

Explanation: The program doesn't handle negative inputs and so produces a garbage value.

9. What is the time complexity of the recursive implementation used to convert a decimal number to its binary equivalent?

```
#include<stdio.h>
int arr[31], len = 0;
void recursive_dec_to_bin(int n)
{
      if(n == 0 && len == 0)
      {
          arr[len++] = 0;
          return;
      }
      if(n == 0)
         return;
      arr[len++] = n % 2;
      recursive_dec_to_bin(n/2);
}
```

a) O(1)
b) O(n)
c) O(n^2)
d) O(logn)

Answer: d
Explanation: The time complexity of the recursive implementation used to convert a decimal number to its binary equivalent is O(logn).

10. What is the space complexity of the recursive implementation used to convert a decimal number to its binary equivalent?

```
#include<stdio.h>
int arr[31], len = 0;
void recursive_dec_to_bin(int n)
{
      if(n == 0 && len == 0)
      {
          arr[len++] = 0;
          return;
      }
      if(n == 0)
         return;
      arr[len++] = n % 2;
      recursive_dec_to_bin(n/2);
}
```

a) O(1)
b) O(n)
c) O(n^2)
d) O(logn)

Answer: d
Explanation: The space complexity of the recursive implementation used to convert a decimal number to its binary equivalent is O(logn).

11. What is the output of the following code?

```
#include<stdio.h>
int arr[31], len = 0;
void recursive_dec_to_bin(int n)
{
      if(n == 0 && len == 0)
      {
```

```
          arr[len++] = 0;
          return;
      }
      if(n == 0)
         return;
      arr[len++] = n % 2;
      recursive_dec_to_bin(n/2);
}
int main()
{
     int n = 111,i;
     recursive_dec_to_bin(n);
     for(i=len-1; i>=0; i--)
     printf("%d",arr[i]);
     return 0;
}
```

a) 1110111
b) 1001111
c) 1101111
d) 1010111

Answer: c
Explanation: The program prints the binary equivalent of 111, which is 1101111.

12. How many times is the function recursive_dec_to_bin() called when the following code is executed?

```
#include<stdio.h>
int arr[31], len = 0;
void recursive_dec_to_bin(int n)
{
      if(n == 0 && len == 0)
      {
          arr[len++] = 0;
          return;
      }
      if(n == 0)
         return;
      arr[len++] = n % 2;
      recursive_dec_to_bin(n/2);
}
int main()
{
    int n = 111,i;
    recursive_dec_to_bin(n);
    for(i=len-1; i>=0; i--)
    printf("%d",arr[i]);
    return 0;
}
```

a) 7
b) 8
c) 9
d) 10

Answer: b
Explanation: The function recursive_dec_to_bin() is called 8 times when the above code is executed.

1. Consider the following iterative implementation used to find the length of a linked list:

```
struct Node
{
      int val;
      struct Node *next;
}*head;
int get_len()
```

```
{
      struct Node *temp = head->next;
      int len = 0;
      while(_____)
      {
          len++;
          temp = temp->next;
      }
      return len;
}
```

Which of the following conditions should be checked to complete the above code?
a) temp->next != 0
b) temp == 0
c) temp != 0
d) temp->next == 0

Answer: c
Explanation: The condition "temp != 0" should be checked to complete the above code.

2. What is the output of the following code?

```
#include<stdio.h>
#include<stdlib.h>
struct Node
{
      int val;
      struct Node *next;
}*head;
int get_len()
{
      struct Node *temp = head->next;
      int len = 0;
      while(temp != 0)
      {
          len++;
          temp = temp->next;
      }
      return len;
}
int main()
{
      int arr[10] = {1,2,3,4,5}, n = 5, i;
      struct Node *temp, *newNode;
      head = (struct Node*)malloc(sizeof(struct Node));
      head->next = 0;
      temp = head;
      for(i=0; i<n; i++)
      {
          newNode = (struct Node*)malloc(sizeof(struct Node));
          newNode->val = arr[i];
          newNode->next = 0;
          temp->next = newNode;
          temp = temp->next;
      }
      int len = get_len();
      printf("%d",len);
      return 0;
}
```

a) 4
b) 5
c) 6
d) 7

Answer: b

Explanation: The program prints the length of the linked list, which is 5.

3. What is the time complexity of the following iterative implementation used to find the length of a linked list?

```
#include<stdio.h>
#include<stdlib.h>
struct Node
{
      int val;
      struct Node *next;
}*head;
int get_len()
{
      struct Node *temp = head->next;
      int len = 0;
      while(temp != 0)
      {
          len++;
          temp = temp->next;
      }
      return len;
}
int main()
{
      int arr[10] = {1,2,3,4,5}, n = 5, i;
      struct Node *temp, *newNode;
      head = (struct Node*)malloc(sizeof(struct Node));
      head->next = 0;
      temp = head;
      for(i=0; i<n; i++)
      {
          newNode = (struct Node*)malloc(sizeof(struct Node));
          newNode->val = arr[i];
          newNode->next = 0;
          temp->next = newNode;
          temp = temp->next;
      }
      int len = get_len();
      printf("%d",len);
      return 0;
}
```

a) O(1)
b) O(n)
c) O(n^2)
d) O(logn)

Answer: b
Explanation: The time complexity of the above iterative implementation used to find the length of a linked list is O(n).

4. What is the output of the following code?

```
#include<stdio.h>
#include<stdlib.h>
struct Node
{
      int val;
      struct Node *next;
}*head;
int get_len()
{
      struct Node *temp = head->next;
      int len = 0;
      while(temp != 0)
      {
          len++;
          temp = temp->next;
```

```
      }
      return len;
}
int main()
{
      int arr[10] = {1,2,3,4,5}, n = 5, i;
      struct Node *temp, *newNode;
      head = (struct Node*)malloc(sizeof(struct Node));
      head->next = 0;
      int len = get_len();
      printf("%d",len);
      return 0;
}
```

a) 0
b) Garbage value
c) Compile time error
d) Runtime error

Answer: a
Explanation: The program prints the length of the linked list, which is 0.

5. Which of the following can be the base case for the recursive implementation used to find the length of linked list?

```
#include<stdio.h>
#include<stdlib.h>
struct Node
{
      int val;
      struct Node *next;
}*head;
int get_len()
{
      struct Node *temp = head->next;
      int len = 0;
      while(temp != 0)
      {
          len++;
          temp = temp->next;
      }
      return len;
}
int main()
{
      int arr[10] = {1,2,3,4,5}, n = 5, i;
      struct Node *temp, *newNode;
      head = (struct Node*)malloc(sizeof(struct Node));
      head->next = 0;
      int len = get_len();
      printf("%d",len);
      return 0;
}
```

a) if(current_node == 0) return 1
b) if(current_node->next == 0) return 1
c) if(current_node->next == 0) return 0
d) if(current_node == 0) return 0

Answer: d
Explanation: The line "if(current_node == 0) return 0" can be used as a base case in the recursive implementation used to find the length of a linked list. Note: The line "if(current_node->next == 0) return 1" cannot be used because it won't work when the length of the linked list is zero.

6. Which of the following lines should be inserted to complete the following recursive implementation used to find the length of a linked list?

```
#include<stdio.h>
#include<stdlib.h>
struct Node
{
      int val;
      struct Node *next;
}*head;
int recursive_get_len(struct Node *current_node)
{
      if(current_node == 0)
        return 0;
      return _____;
}
int main()
{
      int arr[10] = {1,2,3,4,5}, n = 5, i;
      struct Node *temp, *newNode;
      head = (struct Node*)malloc(sizeof(struct Node));
      head->next = 0;
      temp = head;
      for(i=0; i<n; i++)
      {
          newNode = (struct Node*)malloc(sizeof(struct Node));
          newNode->val = arr[i];
          newNode->next = 0;
          temp->next = newNode;
          temp = temp->next;
      }
      int len = recursive_get_len(head->next);
      printf("%d",len);
      return 0;
}
```

a) recursive_get_len(current_node)
b) 1 + recursive_get_len(current_node)
c) recursive_get_len(current_node->next)
d) 1 + recursive_get_len(current_node->next)

Answer: d
Explanation: The line "1 + recursive_get_len(current_node->next)" should be inserted to complete the above code.

7. What is the output of the following code?

```
#include<stdio.h>
#include<stdlib.h>
struct Node
{
      int val;
      struct Node *next;
}*head;
int recursive_get_len(struct Node *current_node)
{
      if(current_node == 0)
        return 0;
      return 1 + recursive_get_len(current_node->next);
}
int main()
{
      int arr[10] = {-1,2,3,-3,4,5,0}, n = 7, i;
      struct Node *temp, *newNode;
      head = (struct Node*)malloc(sizeof(struct Node));
      head->next = 0;
      temp = head;
      for(i=0; i<n; i++)
      {
          newNode = (struct Node*)malloc(sizeof(struct Node));
          newNode->val = arr[i];
          newNode->next = 0;
```

```
            temp->next = newNode;
            temp = temp->next;
        }
        int len = recursive_get_len(head->next);
        printf("%d",len);
        return 0;
}
```

a) 6
b) 7
c) 8
d) 9

Answer: b
Explanation: The program prints the length of the linked list, which is 7.

8. What is the time complexity of the following code used to find the length of a linked list?

```
#include<stdio.h>
#include<stdlib.h>
struct Node
{
        int val;
        struct Node *next;
}*head;
int recursive_get_len(struct Node *current_node)
{
        if(current_node == 0)
          return 0;
        return 1 + recursive_get_len(current_node->next);
}
int main()
{
        int arr[10] = {-1,2,3,-3,4,5,0}, n = 7, i;
        struct Node *temp, *newNode;
        head = (struct Node*)malloc(sizeof(struct Node));
        head->next = 0;
        temp = head;
        for(i=0; i<n; i++)
        {
            newNode = (struct Node*)malloc(sizeof(struct Node));
            newNode->val = arr[i];
            newNode->next = 0;
            temp->next = newNode;
            temp = temp->next;
        }
        int len = recursive_get_len(head->next);
        printf("%d",len);
        return 0;
}
```

a) O(1)
b) O(n)
c) $O(n^2)$
d) $O(n^3)$

Answer: b
Explanation: To find the length of the linked list, the program iterates over the linked list once. So, the time complexity of the above code is O(n).

9. What is the output of the following code?

```
#include<stdio.h>
#include<stdlib.h>
struct Node
{
```

```
      int val;
      struct Node *next;
}*head;
int recursive_get_len(struct Node *current_node)
{
      if(current_node == 0)
        return 0;
      return 1 + recursive_get_len(current_node->next);
}
int main()
{
      int arr[10] = {-1,2,3,-3,4,5}, n = 6, i;
      struct Node *temp, *newNode;
      head = (struct Node*)malloc(sizeof(struct Node));
      head->next = 0;
      temp = head;
      for(i=0; i<n; i++)
      {
          newNode = (struct Node*)malloc(sizeof(struct Node));
          newNode->val = arr[i];
          newNode->next = 0;
          temp->next = newNode;
          temp = temp->next;
      }
      int len = recursive_get_len(head->next);
      printf("%d",len);
      return 0;
}
```

a) 5
b) 6
c) 7
d) 8

Answer: b
Explanation: The program prints the length of the linked list, which is 6.

10. How many times is the function recursive_get_len() called when the following code is executed?

```
#include<stdio.h>
#include<stdlib.h>
struct Node
{
      int val;
      struct Node *next;
}*head;
int recursive_get_len(struct Node *current_node)
{
      if(current_node == 0)
        return 0;
      return 1 + recursive_get_len(current_node->next);
}
int main()
{
      int arr[10] = {-1,2,3,-3,4,5}, n = 6, i;
      struct Node *temp, *newNode;
      head = (struct Node*)malloc(sizeof(struct Node));
      head->next = 0;
      temp = head;
      for(i=0; i<n; i++)
      {
          newNode = (struct Node*)malloc(sizeof(struct Node));
          newNode->val = arr[i];
          newNode->next = 0;
          temp->next = newNode;
          temp = temp->next;
      }
      int len = recursive_get_len(head->next);
```

```
      printf("%d",len);
      return 0;
}
```

a) 5
b) 6
c) 7
d) 8

Answer: c
Explanation: The function is called "len + 1" times. Since the length of the linked list in the above code is 6, the function is called 6 + 1 = 7 times.

1. Consider the following iterative implementation to find the length of the string:

```
#include<stdio.h>
int get_len(char *s)
{
      int len = 0;
      while(________)
        len++;
      return len;
}
int main()
{
      char *s = "harsh";
      int len = get_len(s);
      printf("%d",len);
      return 0;
}
```

Which of the following lines should be inserted to complete the above code?
a) s[len-1] != 0
b) s[len+1] != 0
c) s[len] != '\0'
d) s[len] == '\0'

Answer: c
Explanation: The line "s[len] != '\0'" should be inserted to complete the above code.

2. What is the output of the following code?

```
#include<stdio.h>
int get_len(char *s)
{
      int len = 0;
      while(s[len] != '\0')
        len++;
      return len;
}
int main()
{
      char *s = "lengthofstring";
      int len = get_len(s);
      printf("%d",len);
      return 0;
}
```

a) 14
b) 0
c) Compile time error
d) Runtime error

Answer: a

Explanation: The program prints the length of the string "lengthofstring", which is 14.

3. What is the time complexity of the following code used to find the length of the string?

```
#include<stdio.h>
int get_len(char *s)
{
      int len = 0;
      while(s[len] != '\0')
        len++;
      return len;
}
int main()
{
      char *s = "lengthofstring";
      int len = get_len(s);
      printf("%d",len);
      return 0;
}
```

a) O(1)
b) O(n)
c) O(n^2)
d) O(logn)

Answer: b
Explanation: The time complexity of the code used to find the length of the string is O(n).

4. What is the output of the following code?

```
#include<stdio.h>
int get_len(char *s)
{
      int len = 0;
      while(s[len] != '\0')
        len++;
      return len;
}
int main()
{
      char *s = "";
      int len = get_len(s);
      printf("%d",len);
      return 0;
}
```

a) 0
b) 1
c) Runtime error
d) Garbage value

Answer: a
Explanation: The program prints the length of an empty string, which is 0.

5. Which of the following can be the base case for the recursive implementation used to find the length of a string?

```
#include<stdio.h>
int get_len(char *s)
{
      int len = 0;
      while(s[len] != '\0')
        len++;
      return len;
}
int main()
```

```
{
      char *s = "";
      int len = get_len(s);
      printf("%d",len);
      return 0;
}
```

a) if(string[len] == 1) return 1
b) if(string[len+1] == 1) return 1
c) if(string[len] == '\0') return 0
d) if(string[len] == '\0') return 1

Answer: c
Explanation: "if(string[len] == '\0') return 0" can be used as base case in the recursive implementation used to find the length of the string.

6. Consider the following recursive implementation used to find the length of a string:

```
#include<stdio.h>
int recursive_get_len(char *s, int len)
{
      if(s[len] == 0)
        return 0;
      return ________;
}
int main()
{
      char *s = "abcdef";
      int len = recursive_get_len(s,0);
      printf("%d",len);
      return 0;
}
```

Which of the following lines should be inserted to complete the above code?
a) 1
b) len
c) recursive_get_len(s, len+1)
d) 1 + recursive_get_len(s, len+1)

Answer: d
Explanation: The line "1 + recursive_get_len(s, len+1)" should be inserted to complete the code.

7. What is the output of the following code?

```
#include<stdio.h>
int recursive_get_len(char *s, int len)
{
      if(s[len] == 0)
        return 0;
      return 1 + recursive_get_len(s, len+1);
}
int main()
{
      char *s = "abcdef";
      int len = recursive_get_len(s,0);
      printf("%d",len);
      return 0;
}
```

a) 5
b) 6
c) 7
d) 8

Answer: b
Explanation: The above code prints the length of the string "abcdef", which is 6.

8. What is the time complexity of the following recursive implementation used to find the length of the string?

```
#include<stdio.h>
int recursive_get_len(char *s, int len)
{
      if(s[len] == 0)
        return 0;
      return 1 + recursive_get_len(s, len+1);
}
int main()
{
      char *s = "abcdef";
      int len = recursive_get_len(s,0);
      printf("%d",len);
      return 0;
}
```

a) O(1)
b) O(n)
c) $O(n^2)$
d) $O(n^3)$

Answer: b
Explanation: The time complexity of the above recursive implementation used to find the length of the string is O(n).

9. How many times is the function recursive_get_len() called when the following code is executed?

```
#include<stdio.h>
int recursive_get_len(char *s, int len)
{
      if(s[len] == 0)
        return 0;
      return 1 + recursive_get_len(s, len+1);
}
int main()
{
      char *s = "adghjkl";
      int len = recursive_get_len(s,0);
      printf("%d",len);
      return 0;
}
```

a) 6
b) 7
c) 8
d) 9

Answer: c
Explanation: The function recursive_get_len() is called 8 times when the above code is executed.

10. What is the output of the following code?

```
#include<stdio.h>
int recursive_get_len(char *s, int len)
{
      if(s[len] == 0)
        return 0;
      return 1 + recursive_get_len(s, len+1);
}
int main()
{
      char *s = "123-1-2-3";
```

```
    int len = recursive_get_len(s,0);
    printf("%d",len);
    return 0;
}
```

a) 3
b) 6
c) 9
d) 10

Answer: c
Explanation: The above program prints the length of the string "123-1-2-3", which is 9.

1. If Matrix A is of order X*Y and Matrix B is of order M*N, then what is the order of the Matrix A*B given that Y=M?
a) Y*N
b) X*M
c) X*N
d) Y*M

Answer: c
*Explanation: The Matrix A*B is of order X*N as it is given that Y=M i.e. number of columns in Matrix A is equal to total number of rows in matrix B. So the Matrix A*B must have X number of rows and N number of columns.*

2. How many recursive calls are there in Recursive matrix multiplication through Simple Divide and Conquer Method?
a) 2
b) 6
c) 9
d) 8

Answer: d
Explanation: For the multiplication two square matrix recursively using Simple Divide and Conquer Method, there are 8 recursive calls performed for high time complexity.

3. What is the time complexity of matrix multiplied recursively by Divide and Conquer Method?
a) O(n)
b) $O(n^2)$
c) $O(n^3)$
d) O(n!)

Answer: c
Explanation: The time complexity of recursive multiplication of two square matrices by the Divide and Conquer method is found to be $O(n^3)$ since there are total of 8 recursive calls.

4. What is the time complexity of matrix multiplied recursively by Strassen's Method?
a) $O(n^{log7})$
b) $O(n^2)$
c) $O(n^3)$
d) O(n!)

Answer: a
Explanation: The time complexity of recursive multiplication of two square matrices by Strassen's Method is found to be $O(n^{log7})$ since there are total 7 recursive calls.

5. How many recursive calls are there in Recursive matrix multiplication by Strassen's Method?
a) 5
b) 7

c) 8
d) 4

Answer: b
Explanation: For the multiplication two square matrix recursively using Strassen's Method, there are 7 recursive calls performed for high time complexity.

6. Matrix A is of order 3*4 and Matrix B is of order 4*5. How many elements will be there in a matrix A*B multiplied recursively.
a) 12
b) 15
c) 16
d) 20

Answer: b
*Explanation: The resultant matrix will be of order 3*5 when multiplied recursively and therefore the matrix will have 3*5=15 elements.Answer: a*
*Explanation: Given that B=C, so the two matrix can be recursively multiplied. Therefore, the order of the Matrix X*Y is A*D.*

8. What is the time complexity of the fastest known matrix multiplication algorithm?
a) $O(n^{log7})$
b) $O(n^{2.37})$
c) $O(n^3)$
d) O(n!)

Answer: b
Explanation: The Coppersmith-Winograd algorithm multiplies the matrices in $O(n^{2.37})$ time. Several improvements have been made in the algorithm since 2010.Answer: a
Explanation: Since The Coppersmith-Winograd algorithm multiplies the matrices in $O(n^{2.37})$ time. The time complexity of recursive multiplication of two square matrices by Strassen's Method is found to be $O(n^{2.80})$. Therefore, Coppersmith-Winograd algorithm better than Strassen's algorithm in terms of time complexity.

1. Which of the following statement is true about stack?
a) Pop operation removes the top most element
b) Pop operation removes the bottom most element
c) Push operation adds new element at the bottom
d) Push operation removes the top most element

Answer: a
Explanation: As stack is based on LIFO(Last In First Out) principle so the deletion takes place from the topmost element. Thus pop operator removes topmost element.

2. What is the space complexity of program to reverse stack recursively?
a) O(1)
b) O(log n)
c) O(n)
d) O(n log n)

Answer: c
Explanation: The recursive program to reverse stack uses memory of the order n to store function call stack.

3. Stack can be reversed without using extra space by _____
a) using recursion
b) using linked list to implement stack
c) using an extra stack
d) it is not possible

Answer: b
Explanation: If linked list is used for implementing stack then it can be reversed without using any extra space.

4. Which of the following is considered as the top of the stack in the linked list implementation of the stack?
a) Last node
b) First node
c) Random node
d) Middle node

Answer: b
Explanation: First node is considered as the top element when stack is implemented using linked list.

5. What is the time complexity of the program to reverse stack when linked list is used for its implementation?
a) O(n)
b) O(n log n)
c) O(n^2)
d) O(log n)

Answer: a
Explanation: As a linked list takes O(n) time for getting reversed thus linked list version of stack will also take the same time.

6. Which of the following takes O(n) time in worst case in array implementation of stack?
a) pop
b) push
c) isEmpty
d) pop, push and isEmpty takes constant time

Answer: d
Explanation: Functions pop, push and isEmpty all are implemented in constant time in worst case.

7. What will be the time complexity of the code to reverse stack recursively?
a) O(n)
b) O(n log n)
c) O(log n)
d) O(n^2)

Answer: d
Explanation: The recurrence relation for the recursive code to reverse stack will be given by-T(n)=T(n-1)+n.This is calculated to be equal to O(n^2).

b)

```
int reverse()
{
    if(s.size()<0)
    {
        int x = s.top();
        s.pop();
        reverse();
        BottomInsert(x);
    }
}
```

c)

```
int reverse()
{
    if(s.size()>=0)
    {
        int x = s.top();
```

```
        s.pop();
        reverse();
        BottomInsert(x);
    }
}
```

d)

```
int reverse()
{
    if(s.size()>0)
    {
        int x = s.top();
        s.pop();
        reverse();
        BottomInsert(x);
    }
}
```

b)

```
int reverse()
{
    if(s.size()>0)
    {
        int x = s.top();
        BottomInsert(x);
        s.pop();
        reverse();
    }
}
```

c)

```
 int BottomInsert(int x)
 {
        if(s.size()!=0) s.push(x);
        else
        {
                int a = s.top();
                s.pop();
                BottomInsert(x);
                s.push(a);
        }
}
```

d)

```
int BottomInsert(int x)
{
        if(s.size()==0) s.push(x);
        else
        {
                int a = s.top();
                s.pop();
                s.push(a);
                BottomInsert(x);
        }
}
```

b)

```
int BottomInsert(int x)
{
        if(s.size()==0) s.push(x);
        else
        {
                int a = s.top();
```

```
                s.pop();
                BottomInsert(x);
                s.push(a);
        }
}
```

c)

```
int BottomInsert(int x)
{
        if(s.size()==0) s.push(x);
        else
        {
                s.pop();
                int a = s.top();
                BottomInsert(x);
                s.push(a);
        }
}
```

d)

```
 #include <stack>
void reverseStack(stack<int> &input, stack<int> &extra)
{
        while(input.size()!=0)
        {
                extra.push(input.top());
                input.pop();
    }
    input.swap(extra);
}
```

1. Which of the following sorting algorithm has best case time complexity of $O(n^2)$?
a) bubble sort
b) selection sort
c) insertion sort
d) stupid sort

Answer: b
Explanation: Selection sort is not an adaptive sorting algorithm. It finds the index of minimum element in each iteration even if the given array is already sorted. Thus its best case time complexity becomes $O(n^2)$.

2. Which of the following is the biggest advantage of selection sort?
a) its has low time complexity
b) it has low space complexity
c) it is easy to implement
d) it requires only n swaps under any condition

Answer: d
Explanation: Selection sort works by obtaining least value element in each iteration and then swapping it with the current index. So it will take n swaps under any condition which will be useful when memory write operation is expensive.

3. What will be the recurrence relation of the code of recursive selection sort?
a) T(n) = 2T(n/2) + n
b) T(n) = 2T(n/2) + c
c) T(n) = T(n-1) + n
d) T(n) = T(n-1) + c

Answer: c
Explanation: Function to find the minimum element index takes n time.The recursive call is made to one less element than in the previous call so the overall recurrence relation becomes T(n) = T(n-1) + n.

4. Which of the following sorting algorithm is NOT stable?
a) Selection sort
b) Brick sort
c) Bubble sort
d) Merge sort

Answer: a
Explanation: Out of the given options selection sort is the only algorithm which is not stable. It is because the order of identical elements in sorted output may be different from input array.

5. What will be the best case time complexity of recursive selection sort?
a) O(n)
b) $O(n^2)$
c) O(log n)
d) O(n log n)

Answer: b
Explanation: Selection sort's algorithm is such that it finds the index of minimum element in each iteration even if the given array is already sorted. Thus its best case time complexity becomes $O(n^2)$.Answer: a
Explanation: In selection sort we need to compare elements in order to find the minimum element in each iteration. So we can say that it uses comparisons in order to sort the array. Thus it qualifies as a comparison based sort.

7. What is the average case time complexity of recursive selection sort?
a) O(n)
b) O(n log n)
c) $O(n^2)$
d) O(log n)

Answer: c
Explanation: The overall recurrence relation of recursive selection sort is given by T(n) = T(n-1) + n. It is found to be equal to $O(n^2)$. It is unvaried throughout the three cases.

8. What is the bidirectional variant of selection sort?
a) cocktail sort
b) bogo sort
c) gnome sort
d) bubble sort

Answer: a
Explanation: A bidirectional variant of selection sort is called cocktail sort. It's an algorithm which finds both the minimum and maximum values in the array in every pass. This reduces the number of scans of the array by a factor of 2.

b)

```
#include <iostream>
using namespace std;
int minIndex(int a[], int i, int j)
{
        if (i == 0)
                return i;
        int k = minIndex(a, i + 1, j);
        return (a[i] < a[k])? i : k;
}
void recursiveSelectionSort(int a[], int n, int index = 0)
{

        if (index == n)
        return;
        int x = minIndex(a, index, n-1);
```

```
        if (x == index)
        {
                swap(a[x], a[index]);
        }
        recursiveSelectionSort(a, n, index + 1);
}
int main()
{
        int arr[] = {5,3,2,4,1};
        int n = sizeof(arr)/sizeof(arr[0]);
        recursiveSelectionSort(arr, n);
        return 0;
}
```

c)

```
#include <iostream>
using namespace std;
int minIndex(int a[], int i, int j)
{
        if (i == j)
                return i;
        int k = minIndex(a, i + 1, j);
        return (a[i] < a[k])? i : k;
}
void recursiveSelectionSort(int a[], int n, int index = 0)
{

        if (index == n)
        return;
        int x = minIndex(a, index, n-1);
        if (x != index)
        {
                swap(a[x], a[index]);
        }
        recursiveSelectionSort(a, n, index + 1);
}
int main()
{
        int arr[] = {5,3,2,4,1};
        int n = sizeof(arr)/sizeof(arr[0]);
        recursiveSelectionSort(arr, n);
        return 0;
}
```

d)

```
#include <iostream>
using namespace std;
int minIndex(int a[], int i, int j)
{
        if (i == j)
                return i;
        int k = minIndex(a, i + 1, j);
        return (a[i] > a[k])? i : k;
}
void recursiveSelectionSort(int a[], int n, int index = 0)
{
        if (index == n)
        return;
        int x = minIndex(a, index, n-1);
        if (x != index)
        {
                swap(a[x], a[index]);
        }
        recursiveSelectionSort(a, n, index + 1);
}
int main()
{
```

```
        int arr[] = {5,3,2,4,1};
        int n = sizeof(arr)/sizeof(arr[0]);
        recursiveSelectionSort(arr, n);
        return 0;
}
```

10. What is the number of swaps required to sort the array arr={5,3,2,4,1} using recursive selection sort?
a) 0
b) 1
c) 2
d) 3

Answer: c
Explanation: The first swap takes place between 1 and 5. The second swap takes place between 3 and 2 which sorts our array.

1. Which of the following methods can be used to find the largest and smallest element in an array?
a) Recursion
b) Iteration
c) Both recursion and iteration
d) No method is suitable

Answer: c
Explanation: Both recursion and iteration can be used to find the largest and smallest element in an array.

2. Consider the following iterative code snippet to find the largest element:

```
int get_max_element(int *arr,int n)
{
      int i, max_element = arr[0];
      for(i = 1; i < n; i++)
          if(________)
          max_element = arr[i];
      return max_element;
}
```

Which of the following lines should be inserted to complete the above code?
a) arr[i] > max_element
b) arr[i] < max_element
c) arr[i] == max_element
d) arr[i] != max_element

Answer: a
Explanation: The line "arr[i] > max_element" should be inserted to complete the above code snippet.

3. Consider the following code snippet to find the smallest element in an array:

```
int get_min_element(int *arr, int n)
{
      int i, min_element = arr[0];
      for(i = 1; i < n; i++)
        if(_______)
          min_element = arr[i];
      return min_element;
}
```

Which of the following lines should be inserted to complete the above code?
a) arr[i] > min_element
b) arr[i] < min_element
c) arr[i] == min_element
d) arr[i] != min_element

Answer: b
Explanation: The line "arr[i] < min_element" should be inserted to complete the above code.

4. What is the output of the following code?

```
#include<stdio.h>
int get_max_element(int *arr,int n)
{
      int i, max_element = arr[0];
      for(i = 1; i < n; i++)
        if(arr[i] > max_element)
          max_element = arr[i];
      return max_element;
}
int get_min_element(int *arr, int n)
{
      int i, min_element = arr[0];
      for(i = 1; i < n; i++)
        if(arr[i] < min_element)
          min_element = arr[i];
      return min_element;
}
int main()
{
     int n = 7, arr[7] = {5,2,4,7,8,1,3};
     int max_element = get_max_element(arr,n);
     int min_element = get_min_element(arr,n);
     printf("%d %d",max_element,min_element);
     return 0;
}
```

a) 5 3
b) 3 5
c) 8 1
d) 1 8

Answer: c
Explanation: The program prints the values of the largest and the smallest elements in the array, which are 8 and 1 respectively.

5. What is the output of the following code?

```
#include<stdio.h>
int get_max_element(int *arr,int n)
{
      int i, max_element = arr[0];
      for(i = 1; i < n; i++)
        if(arr[i] > max_element)
          max_element = arr[i];
      return max_element;
}
int get_min_element(int *arr, int n)
{
      int i, min_element;
      for(i = 1; i < n; i++)
        if(arr[i] < min_element)
          min_element = arr[i];
      return min_element;
}
int main()
{
     int n = 7, arr[7] = {1,1,1,0,-1,-1,-1};
     int max_element = get_max_element(arr,n);
     int min_element = get_min_element(arr,n);
     printf("%d %d",max_element,min_element);
     return 0;
}
```

a) 1 -1
b) -1 1
c) 1 Garbage value
d) Depends on the compiler

Answer: c
Explanation: Since the min_element variable is not initialised, some compilers will auto initialise as 0 which produces -1 as output whereas some compilers won't initialise automatically. In that case, the result will be a garbage value hence the output depends on the compiler.

6. What is the time complexity of the following iterative implementation used to find the largest and smallest element in an array?

```
#include<stdio.h>
int get_max_element(int *arr,int n)
{
      int i, max_element = arr[0];
      for(i = 1; i < n; i++)
        if(arr[i] > max_element)
          max_element = arr[i];
      return max_element;
}
int get_min_element(int *arr, int n)
{
      int i, min_element;
      for(i = 1; i < n; i++)
        if(arr[i] < min_element)
          min_element = arr[i];
      return min_element;
}
int main()
{
     int n = 7, arr[7] = {1,1,1,0,-1,-1,-1};
     int max_element = get_max_element(arr,n);
     int min_element = get_min_element(arr,n);
     printf("%d %d",max_element,min_element);
     return 0;
}
```

a) O(1)
b) O(n)
c) O(n^2)
d) O(n/2)

Answer: b
Explanation: The time complexity of the above iterative implementation used to find the largest and the smallest element in an array is O(n).

7. Consider the following recursive implementation to find the largest element in an array.

```
int max_of_two(int a, int b)
{
      if(a > b)
        return a;
      return b;
}
int recursive_max_element(int *arr, int len, int idx)
{
      if(idx == len - 1)
      return arr[idx];
      return _______;
}
```

Which of the following lines should be inserted to complete the above code?

a) max_of_two(arr[idx], recursive_max_element(arr, len, idx))
b) recursive_max_element(arr, len, idx)
c) max_of_two(arr[idx], recursive_max_element(arr, len, idx + 1))
d) recursive_max_element(arr, len, idx + 1)

Answer: c
Explanation: The line "max_of_two(arr[idx], recursive_max_element(arr, len, idx + 1))" should be inserted to complete the above code.

8. What is the output of the following code?

```
#include<stdio.h>
int max_of_two(int a, int b)
{
      if(a > b)
        return a;
      return b;
}
int min_of_two(int a, int b)
{
      if(a < b)
        return a;
      return b;
}
int recursive_max_element(int *arr, int len, int idx)
{
      if(idx == len - 1)
      return arr[idx];
      return max_of_two(arr[idx], recursive_max_element(arr, len, idx + 1));
}
int recursive_min_element(int *arr, int len, int idx)
{
      if(idx == len - 1)
      return arr[idx];
      return min_of_two(arr[idx], recursive_min_element(arr, len, idx + 1));
}
int main()
{
    int n = 10, idx = 0, arr[] = {5,2,6,7,8,9,3,-1,1,10};
    int max_element = recursive_max_element(arr,n,idx);
    int min_element = recursive_min_element(arr,n,idx);
    printf("%d %d",max_element,min_element);
    return 0;
}
```

a) -1 10
b) 10 -1
c) 1 10
d) 10 1

Answer: b
Explanation: The program prints the values of the largest and the smallest element in the array, which are 10 and -1 respectively.

9. What is the time complexity of the following recursive implementation used to find the largest and the smallest element in an array?

```
#include<stdio.h>
int max_of_two(int a, int b)
{
      if(a > b)
        return a;
      return b;
}
int min_of_two(int a, int b)
```

```
{
      if(a < b)
        return a;
      return b;
}
int recursive_max_element(int *arr, int len, int idx)
{
      if(idx == len - 1)
      return arr[idx];
      return max_of_two(arr[idx], recursive_max_element(arr, len, idx + 1));
}
int recursive_min_element(int *arr, int len, int idx)
{
      if(idx == len - 1)
      return arr[idx];
      return min_of_two(arr[idx], recursive_min_element(arr, len, idx + 1));
}
int main()
{
    int n = 10, idx = 0, arr[] = {5,2,6,7,8,9,3,-1,1,10};
    int max_element = recursive_max_element(arr,n,idx);
    int min_element = recursive_min_element(arr,n,idx);
    printf("%d %d",max_element,min_element);
    return 0;
}
```

a) O(1)
b) O(n)
c) O(n^2)
d) O(n^3)

Answer: b
Explanation: The time complexity of the above recursive implementation used to find the largest and smallest element in an array is O(n).

10. How many times is the function recursive_min_element() called when the following code is executed?

```
int min_of_two(int a, int b)
{
      if(a < b)
        return a;
      return b;
}
int recursive_min_element(int *arr, int len, int idx)
{
      if(idx == len - 1)
      return arr[idx];
      return min_of_two(arr[idx], recursive_min_element(arr, len, idx + 1));
}
int main()
{
     int n = 10, idx = 0, arr[] = {5,2,6,7,8,9,3,-1,1,10};
     int min_element = recursive_min_element(arr,n,idx);
     printf("%d",min_element);
     return 0;
}
```

a) 9
b) 10
c) 11
d) 12

Answer: b
Explanation: The function recursive_min_element() is called 10 times when the above code is executed.

11. What is the output of the following code?

```
#include<stdio.h>
int max_of_two(int a, int b)
{
      if(a > b)
        return a;
      return b;
}
int min_of_two(int a, int b)
{
      if(a < b)
        return a;
      return b;
}
int recursive_max_element(int *arr, int len, int idx)
{
      if(idx == len - 1)
      return arr[idx];
      return max_of_two(arr[idx], recursive_max_element(arr, len, idx + 1));
}
int recursive_min_element(int *arr, int len, int idx)
{
      if(idx == len - 1)
      return arr[idx];
      return min_of_two(arr[idx], recursive_min_element(arr, len, idx + 1));
}
int main()
{
     int n = 5, idx = 0, arr[] = {1,1,1,1,1};
     int max_element = recursive_max_element(arr,n,idx);
     int min_element = recursive_min_element(arr,n,idx);
     printf("%d %d",max_element,min_element);
     return 0;
}
```

a) 1 1
b) 0 0
c) compile time error
d) runtime error

Answer: a
Explanation: The program prints the values of the largest and the smallest element in the array, which are 1 and 1.

1. Which of the following methods can be used to find the largest and smallest number in a linked list?
a) Recursion
b) Iteration
c) Both Recursion and iteration
d) Impossible to find the largest and smallest numbers

Answer: c
Explanation: Both recursion and iteration can be used to find the largest and smallest number in a linked list.

2. Consider the following code snippet to find the largest element in a linked list:

```
struct Node{
   int val;
   struct Node *next;
}*head;
int get_max()
{
      struct Node* temp = head->next;
          int max_num = temp->val;
          while(______)
          {
                if(temp->val > max_num)
                    max_num = temp->val;
```

```
                temp = temp->next;
        }
        return max_num;
}
```

Which of the following lines should be inserted to complete the above code?
a) temp->next != 0
b) temp != 0
c) head->next != 0
d) head != 0

Answer: b
Explanation: The line "temp != 0" should be inserted to complete the above code.

3. Consider the following code snippet to find the smallest element in a linked list:

```
struct Node
{
    int val;
    struct Node* next;
}*head;
int get_min()
{
     struct Node* temp = head->next;
        int min_num = temp->val;
        while(temp != 0)
        {
            if(_________)
                min_num = temp->val;
             temp = temp->next;
        }
        return min_num;
}
```

Which of the following lines should be inserted to complete the above code?
a) temp > min_num
b) val > min_min
c) temp->val < min_num
d) temp->val > min_num

Answer: c
Explanation: The line "temp->val = min_num" should be inserted to complete the above code.

4. What is the output of the following code:

```
#include<stdio.h>
#include<stdlib.h>
struct Node
{
    int val;
    struct Node* next;
}*head;
int get_max()
{
     struct Node* temp = head->next;
        int max_num = temp->val;
        while(temp != 0)
        {
            if(temp->val > max_num)
                max_num = temp->val;
            temp = head->next;
        }
        return max_num;
}
int main()
{
```

```
      int n = 9, arr[9] ={5,1,3,4,5,2,3,3,1},i;
      struct Node *temp, *newNode;
      head = (struct Node*)malloc(sizeof(struct Node));
      head -> next =0;
      temp = head;
      for(i=0;i<n;i++)
      {
          newNode =(struct Node*)malloc(sizeof(struct Node));
          newNode->next = 0;
          newNode->val = arr[i];
          temp->next =newNode;
          temp = temp->next;
      }
      int max_num = get_max();
      printf("%d %d",max_num);
      return 0;
}
```

a) 5
b) 1
c) runtime error
d) garbage value

Answer: c
Explanation: The variable temp will always point to the first element in the linked list due to the line "temp = head->next" in the while loop. So, it will be an infinite while loop and the program will produce a runtime error.

5. What is the output of the following code?

```
#include<stdio.h>
#include<stdlib.h>
struct Node
{
     int val;
     struct Node* next;
}*head;
int get_max()
{
      struct Node* temp = head->next;
          int max_num = temp->val;
          while(temp != 0)
          {
                if(temp->val > max_num)
                    max_num = temp->val;
                temp = temp->next;
          }
          return max_num;
}
int get_min()
{
      struct Node* temp = head->next;
          int min_num = temp->val;
          while(temp != 0)
          {
                if(temp->val < min_num)
                    min_num = temp->val;
                temp = temp->next;
          }
          return min_num;
}
int main()
{
      int i, n = 9, arr[9] ={8,3,3,4,5,2,5,6,7};
      struct Node *temp, *newNode;
      head = (struct Node*)malloc(sizeof(struct Node));
      head -> next =0;
      temp = head;
      for(i=0;i<n;i++)
```

```
      {
          newNode =(struct Node*)malloc(sizeof(struct Node));
          newNode->next = 0;
          newNode->val = arr[i];
          temp->next =newNode;
          temp = temp->next;
      }
      int max_num = get_max();
      int min_num = get_min();
      printf("%d %d",max_num,min_num);
      return 0;
}
```

a) 2 2
b) 8 8
c) 2 8
d) 8 2

Answer: d
Explanation: The program prints the largest and smallest elements in the linked list, which are 8 and 2 respectively.

6. What is the time complexity of the following iterative code used to find the smallest and largest element in a linked list?

```
#include<stdio.h>
#include<stdlib.h>
struct Node
{
     int val;
     struct Node* next;
}*head;
int get_max()
{
      struct Node* temp = head->next;
          int max_num = temp->val;
          while(temp != 0)
          {
                if(temp->val > max_num)
                    max_num = temp->val;
                temp = temp->next;
          }
          return max_num;
}
int get_min()
{
      struct Node* temp = head->next;
          int min_num = temp->val;
          while(temp != 0)
          {
                if(temp->val < min_num)
                    min_num = temp->val;
                temp = temp->next;
          }
          return min_num;
}
int main()
{
      int i, n = 9, arr[9] ={8,3,3,4,5,2,5,6,7};
      struct Node *temp, *newNode;
      head = (struct Node*)malloc(sizeof(struct Node));
      head -> next =0;
      temp = head;
      for(i=0;i<n;i++)
      {
          newNode =(struct Node*)malloc(sizeof(struct Node));
          newNode->next = 0;
          newNode->val = arr[i];
          temp->next =newNode;
```

```
            temp = temp->next;
      }
      int max_num = get_max();
      int min_num = get_min();
      printf("%d %d",max_num,min_num);
      return 0;
}
```

a) O(1)
b) O(n)
c) O(n^2)
d) O(n^3)

Answer: b
Explanation: The time complexity of the above iterative code used to find the largest and smallest element in a linked list is O(n).

7. Consider the following recursive implementation to find the largest element in a linked list:

```
struct Node
{
     int val;
     struct Node* next;
}*head;
int max_of_two(int a, int b)
{
      if(a > b)
        return a;
      return b;
}
int recursive_get_max(struct Node* temp)
{
      if(temp->next == 0)
        return  temp->val;
      return max_of_two(______, _______);
}
```

Which of the following arguments should be passed to the function max_of two() to complete the above code?
a) temp->val,recursive_get_max(temp->next)
b) temp, temp->next
c) temp->val, temp->next->val
d) temp->next->val, temp

Answer: a
Explanation: The arguments {temp->val,recursive_get_max(temp->next)} should be passed to the function max_of_two() to complete the above code.

8. What is the output of the following code?

```
#include<stdio.h>
#include<stdlib.h>
struct Node
{
     int val;
     struct Node* next;
}*head;
int max_of_two(int a, int b)
{
      if(a > b)
        return a;
      return b;
}
int recursive_get_max(struct Node* temp)
{
      if(temp->next == 0)
```

```
        return  temp->val;
      return max_of_two(temp->val,recursive_get_max(temp->next));
}
int min_of_two(int a, int b)
{
      if(a < b)
        return a;
      return b;
}
int recursive_get_min(struct Node* temp)
{
      if(temp->next == 0)
        return  temp->val;
      return min_of_two(temp->val,recursive_get_min(temp->next));
}
int main()
{
     int n = 9, arr[9] ={1,3,2,4,5,0,5,6,7},i;
     struct Node *temp, *newNode;
     head = (struct Node*)malloc(sizeof(struct Node));
     head -> next =0;
     temp = head;
     for(i=0;i<n;i++)
     {
           newNode =(struct Node*)malloc(sizeof(struct Node));
           newNode->next = 0;
           newNode->val = arr[i];
           temp->next =newNode;
           temp = temp->next;
     }
     int max_num = recursive_get_max(head->next);
     int min_num = recursive_get_min(head->next);
     printf("%d %d",max_num,min_num);
     return 0;
}
```

a) 7 1
b) 0 7
c) 7 0
d) 1 1

Answer: c
Explanation: The program prints the largest and the smallest elements in the linked list, which are 7 and 0 respectively.

9. What is the time complexity of the recursive implementation used to find the largest and smallest element in a linked list?

```
#include<stdio.h>
#include<stdlib.h>
struct Node
{
     int val;
     struct Node* next;
}*head;
int max_of_two(int a, int b)
{
      if(a > b)
        return a;
      return b;
}
int recursive_get_max(struct Node* temp)
{
      if(temp->next == 0)
        return  temp->val;
      return max_of_two(temp->val,recursive_get_max(temp->next));
}
int min_of_two(int a, int b)
{
```

```
      if(a < b)
        return a;
      return b;
}
int recursive_get_min(struct Node* temp)
{
      if(temp->next == 0)
        return  temp->val;
      return min_of_two(temp->val,recursive_get_min(temp->next));
}
int main()
{
     int n = 9, arr[9] ={1,3,2,4,5,0,5,6,7},i;
     struct Node *temp, *newNode;
     head = (struct Node*)malloc(sizeof(struct Node));
     head -> next =0;
     temp = head;
     for(i=0;i<n;i++)
     {
           newNode =(struct Node*)malloc(sizeof(struct Node));
           newNode->next = 0;
           newNode->val = arr[i];
           temp->next =newNode;
           temp = temp->next;
     }
     int max_num = recursive_get_max(head->next);
     int min_num = recursive_get_min(head->next);
     printf("%d %d",max_num,min_num);
     return 0;
}
```

a) O(1)
b) O(n)
c) O(n^2)
d) O(n^3)

Answer: b
Explanation: The time complexity of the above recursive implementation used to find the largest and smallest element in linked list is O(n).

10. What is the output of the following code?

```
#include<stdio.h>
#include<stdlib.h>
struct Node
{
     int val;
     struct Node* next;
}*head;
int min_of_two(int a, int b)
{
      if(a < b)
        return a;
      return b;
}
int recursive_get_min(struct Node* temp)
{
      if(temp->next == 0)
        return  temp->val;
      return min_of_two(temp->val,recursive_get_min(temp->next));
}
int main()
{
     int n = 5, arr[5] ={1,1,1,1,1},i;
     struct Node *temp, *newNode;
     head = (struct Node*)malloc(sizeof(struct Node));
     head -> next =0;
```

```
     temp = head;
     for(i=0;i<n;i++)
     {
           newNode =(struct Node*)malloc(sizeof(struct Node));
           newNode->next = 0;
           newNode->val = arr[i];
           temp->next =newNode;
           temp = temp->next;
     }
     int min_num = recursive_get_min(head->next);
     printf("%d",min_num);
     return 0;
}
```

a) 1
b) 0
c) compile time error
d) runtime error

Answer: a
Explanation: The program prints the smallest element in the linked list, which is 1.

11. How many times will the function recursive_get_min() be called when the following code is executed?

```
#include<stdio.h>
#include<stdlib.h>
struct Node
{
     int val;
     struct Node* next;
}*head;
int min_of_two(int a, int b)
{
      if(a < b)
        return a;
      return b;
}
int recursive_get_min(struct Node* temp)
{
      if(temp->next == 0)
        return  temp->val;
      return min_of_two(temp->val,recursive_get_min(temp->next));
}
int main()
{
     int n = 5, arr[5] ={1,1,1,1,1},i;
     struct Node *temp, *newNode;
     head = (struct Node*)malloc(sizeof(struct Node));
     head -> next =0;
     temp = head;
     for(i=0;i<n;i++)
     {
           newNode =(struct Node*)malloc(sizeof(struct Node));
           newNode->next = 0;
           newNode->val = arr[i];
           temp->next =newNode;
           temp = temp->next;
     }
     int min_num = recursive_get_min(head->next);
     printf("%d",min_num);
     return 0;
}
```

a) 4
b) 5
c) 6
d) 7

Answer: b
Explanation: The function recursive_get_min() will be called 5 times when the above code is executed.

1. Which of the following techniques can be used to search an element in an unsorted array?
a) Iterative linear search
b) Recursive binary search
c) Iterative binary search
d) Normal binary search

Answer: a
Explanation: Iterative linear search can be used to search an element in an unsorted array.
Note: Binary search can be used only when the array is sorted.

2. Consider the array {1,1,1,1,1}. Select the wrong option?
a) Iterative linear search can be used to search for the elements in the given array
b) Recursive linear search can be used to search for the elements in the given array
c) Recursive binary search can be used to search for the elements in the given array
d) No method is defined to search for an element in the given array

Answer: d
Explanation: Iterative linear search, Recursive linear search and Recursive binary search can be applied to search for an element in the above given array.

3. What does the following code do?

```
#include<stdio.h>
int search_num(int *arr, int num, int len)
{
     int i;
     for(i = 0; i < len; i++)
     if(arr[i] == num)
      return i;
     return -1;
}
int main()
{
      int arr[5] ={1,2,3,4,5},num=3,len = 5;
      int indx = search_num(arr,num,len);
      printf("Index of %d is %d",num,indx);
      return 0;
}
```

a) Search and returns the index of all the occurrences of the number that is searched
b) Search and returns the index of the first occurrence of the number that is searched
c) Search and returns of the last occurrence of the number that is searched
d) Returns the searched element from the given array

Answer: b
Explanation: The code finds the index of the first occurrence of the number that is searched.

4. What is the output of the following code?

```
#include<stdio.h>
int search_num(int *arr, int num, int len)
{
     int i;
     for(i = 0; i < len; i++)
     if(arr[i] == num)
        return i;
     return -1;
}
int main()
{
```

```
      int arr[5] ={1,3,3,3,5},num=3,len = 5;
      int indx = search_num(arr,num,len);
      printf("Index of %d is %d",num,indx);
      return 0;
}
```

a) Index of 3 is 0
b) Index of 3 is 1
c) Index of 3 is 2
d) Index of 3 is 3

Answer: b
Explanation: The program prints the index of the first occurrence of 3, which is 1.

5. What is the time complexity of the following code used to search an element in an array?

```
#include<stdio.h>
int search_num(int *arr, int num, int len)
{
     int i;
     for(i = 0; i < len; i++)
     if(arr[i] == num)
        return i;
     return -1;
}
int main()
{
      int arr[5] ={1,3,3,3,5},num=3,len = 5;
      int indx = search_num(arr,num,len);
      printf("Index of %d is %d",num,indx);
      return 0;
}
```

a) O(1)
b) O(n)
c) O(n^2)
d) O(n^3)

Answer: b
Explanation: The time complexity of the above code used to search an element in an array is O(n).

6. Consider the following recursive implementation of linear search:

```
#include<stdio.h>
int recursive_search_num(int *arr, int num, int idx, int len)
{
     if(idx == len)
     return -1;
     if(arr[idx] == num)
       return idx;
     return __________;
}
int main()
{
      int arr[5] ={1,3,3,3,5},num=2,len = 5;
      int indx = recursive_search_num(arr,num,0,len);
      printf("Index of %d is %d",num,indx);
      return 0;
}
```

Which of the following recursive calls should be added to complete the above code?
a) recursive_search_num(arr, num+1, idx, len);
b) recursive_search_num(arr, num, idx, len);
c) recursive_search_num(arr, num, idx+1, len);
d) recursive_search_num(arr, num+1, idx+1, len);

Answer: c
Explanation: The recursive call "recursive_search_num(arr, num, idx+1, len)" should be added to complete the above code.

7. What is the output of the following code?

```
#include<stdio.h>
int recursive_search_num(int *arr, int num, int idx, int len)
{
     if(idx == len)
     return -1;
     if(arr[idx] == num)
       return idx;
     return recursive_search_num(arr, num, idx+1, len);
}
int main()
{
      int arr[8] ={1,2,3,3,3,5,6,7},num=5,len = 8;
      int indx = recursive_search_num(arr,num,0,len);
      printf("Index of %d is %d",num,indx);
      return 0;
}
```

a) Index of 5 is 5
b) Index of 5 is 6
c) Index of 5 is 7
d) Index of 5 is 8

Answer: a
Explanation: The program prints the index of 5, which is 5.

8. How many times is the function recursive_search_num() called when the following code is executed?

```
#include<stdio.h>
int recursive_search_num(int *arr, int num, int idx, int len)
{
     if(idx == len)
     return -1;
     if(arr[idx] == num)
     return idx;
     return recursive_search_num(arr, num, idx+1, len);
}
int main()
{
      int arr[8] ={1,2,3,3,3,5,6,7},num=5,len = 8;
      int indx = recursive_search_num(arr,num,0,len);
      printf("Index of %d is %d",num,indx);
      return 0;
}
```

a) 5
b) 6
c) 7
d) 8

Answer: b
Explanation: The function recursive_search_num() is called 6 times when the above code is executed.

9. What is the time complexity of the following recursive implementation of linear search?

```
#include<stdio.h>
int recursive_search_num(int *arr, int num, int idx, int len)
{
     if(idx == len)
```

```
     return -1;
     if(arr[idx] == num)
     return idx;
     return recursive_search_num(arr, num, idx+1, len);
}
int main()
{
      int arr[8] ={1,2,3,3,3,5,6,7},num=5,len = 8;
      int indx = recursive_search_num(arr,num,0,len);
      printf("Index of %d is %d",num,indx);
      return 0;
}
```

a) O(1)
b) O(n)
c) O(n^2)
d) O(n^3)

Answer: b
Explanation: The time complexity of the above recursive implementation of linear search is O(n).

1. What is the output of the following code?

```
#include<stdio.h>
int recursive_search_num(int *arr, int num, int idx, int len)
{
     if(idx == len)
      return -1;
     if(arr[idx] == num)
      return idx;
     return recursive_search_num(arr, num, idx+1, len);
}
int main()
{
      int arr[8] ={-11,2,-3,0,3,5,-6,7},num = -2,len = 8;
      int indx = recursive_search_num(arr,num,0,len);
      printf("Index of %d is %d",num,indx);
      return 0;
}
```

a) Index of -2 is 1
b) Index of -2 is 0
c) Index of -2 is -1
d) None of the mentioned

Answer: c
Explanation: The program prints the index of the first occurrence of -2. Since, -2 doesn't exist in the array, the program prints -1.

2. What does the following code do?

```
#include<stdio.h>
int cnt = 0;
int recursive_search_num(int *arr, int num, int idx, int len)
{
      int cnt = 0;
      if(idx == len)
      return cnt;
      if(arr[idx] == num)
       cnt++;
      return cnt + recursive_search_num(arr, num, idx+1, len);
}
int main()
{
      int arr[8] ={0,0,0,0,3,5,-6,7},num = 0,len = 8;
      int ans = recursive_search_num(arr,num,0,len);
```

```
      printf("%d",ans);
      return 0;
}
```

a) Adds all the indexes of the number 0
b) Finds the first last occurrence of the number 0
c) Counts the number of occurrences of the number 0
d) None of the mentioned

Answer: c
Explanation: The above code counts the number of occurrences of the number 0.

3. Consider the following recursive implementation of the binary search:

```
#include<stdio.h>
int recursive_binary_search(int *arr, int num, int lo, int hi)
{
      if(lo > hi)
       return -1;
      int mid = (lo + hi)/2;
      if(arr[mid] == num)
      return mid;
      else if(arr[mid] < num)
          __________;
      else
          hi = mid - 1;
     return recursive_binary_search(arr, num, lo, hi);
}
int main()
{
      int arr[8] ={0,0,0,0,3,5,6,7},num = 7,len = 8;
      int indx = recursive_binary_search(arr,num,0,len-1);
      printf("Index of %d is %d",num,indx);
      return 0;
}
```

Which of the following lines should be added to complete the above code?
a) hi = mid – 1
b) mid = (lo + hi)/2
c) mid = lo – 1
d) lo = mid + 1

Answer: d
Explanation: The line "lo = mid + 1" should be added to complete the above code.

4. What is the output of the following code?

```
#include<stdio.h>
int recursive_binary_search(int *arr, int num, int lo, int hi)
{
      if(lo > hi)
        return -1;
      int mid = (lo + hi)/2;
      if(arr[mid] == num)
         return mid;
      else if(arr[mid] < num)
         lo = mid + 1;
      else
         hi = mid - 1;
      return recursive_binary_search(arr, num, lo, hi);
}
int main()
{
      int arr[8] = {1,2,3,4,5,6,7,8},num = 7,len = 8;
      int indx = recursive_binary_search(arr,num,0,len-1);
      printf("Index of %d is %d",num,indx);
```

```
      return 0;
}
```

a) Index of 7 is 4
b) Index of 7 is 5
c) Index of 7 is 6
d) Index of 7 is 7

Answer: c
Explanation: The program prints the index of number 7, which is 6.

5. What is the output of the following code?

```
#include<stdio.h>
int recursive_binary_search(int *arr, int num, int lo, int hi)
{
      if(lo > hi)
       return -1;
      int mid = (lo + hi)/2;
      if(arr[mid] == num)
       return mid;
      else if(arr[mid] < num)
          lo = mid + 1;
      else
          hi = mid - 1;
      return recursive_binary_search(arr, num, lo, hi);
}
int main()
{
      int arr[8] = {0,0,0,0,3,5,6,7},num = 0,len = 8;
      int indx = recursive_binary_search(arr,num,0,len-1);
      printf("Index of %d is %d",num,indx);
      return 0;
}
```

a) Index of 0 is 0
b) Index of 0 is 1
c) Index of 0 is 2
d) Index of 0 is 3

Answer: d
Explanation: In this case, when the function recursive_binary_search() is called for the first time we have: lo = 0 and hi = 7. So, the value of mid is:
mid = (lo + hi)/2 = (0 + 7)/2 = 3. Since, arr[mid] = arr[3] = 0, the function returns the value of mid, which is 3.

6. What is the time complexity of the above recursive implementation of binary search?
a) O(n)
b) $O(2^n)$
c) O(logn)
d) O(n!)

Answer: c
Explanation: The time complexity of the above recursive implementation of binary search is O(logn).

7. In which of the below cases will the following code produce a wrong output?

```
int recursive_binary_search(int *arr, int num, int lo, int hi)
{
      if(lo > hi)
       return -1;
      int mid = (lo + hi)/2;
      if(arr[mid] == num)
       return mid;
      else if(arr[mid] < num)
```

```
        lo = mid + 1;
    else
        hi = mid - 1;
    return recursive_binary_search(arr, num, lo, hi);
}
```

a) Array: {0,0,0,0,0,0} Search: -10
b) Array: {1,2,3,4,5} Search: 0
c) Array: {5,4,3,2,1} Search: 1
d) Array: {-5,-4,-3,-2,-1} Search: -1

Answer: c
Explanation: Since the array {5,4,3,2,1} is sorted in descending order, it will produce a wrong output.

8. How many times is the function recursive_binary_search() called when the following code is executed?

```
#include<stdio.h>
int recursive_binary_search(int *arr, int num, int lo, int hi)
{
    if(lo > hi)
     return -1;
    int mid = (lo + hi)/2;
    if(arr[mid] == num)
      return mid;
    else if(arr[mid] < num)
        lo = mid + 1;
    else
        hi = mid - 1;
    return recursive_binary_search(arr, num, lo, hi);
}
int main()
{
    int arr[5] = {1,2,3,4,5},num = 1,len = 5;
    int indx = recursive_binary_search(arr,num,0,len-1);
    printf("Index of %d is %d",num,indx);
    return 0;
}
```

a) 0
b) 1
c) 2
d) 3

Answer: c
Explanation: The function recursive_binary_search() is called 2 times, when the above code is executed.

9. What is the output of the following code?

```
#include<stdio.h>
int recursive_binary_search(int *arr, int num, int lo, int hi)
{
    if(lo > hi)
      return -1;
    int mid = (lo + hi)/2;
    if(arr[mid] == num)
      return mid;
    else if(arr[mid] < num)
        lo = mid + 1;
    else
        hi = mid - 1;
    return recursive_binary_search(arr, num, lo, hi);
}
int main()
{
    int arr[5] = {5,4,3,2,1},num = 1,len = 5;
    int indx = recursive_binary_search(arr,num,0,len-1);
```

```
      printf("Index of %d is %d",num,indx);
      return 0;
}
```

a) Index of 1 is 4
b) Index of 1 is 5
c) Index of 1 is -1
d) Index of 1 is 0

Answer: c
Explanation: Since the array is sorted in descending order, the above implementation of binary search will not work for the given array. Even though 1 is present in the array, binary search won't be able to search it and it will produce -1 as an answer.

1. Which of the following methods can be used to search an element in a linked list?
a) Iterative linear search
b) Iterative binary search
c) Recursive binary search
d) Normal binary search

Answer: a
Explanation: Iterative linear search can be used to search an element in a linked list. Binary search can be used only when the list is sorted.

2. Consider the following code snippet to search an element in a linked list:

```
struct Node
{
     int val;
     struct Node* next;
}*head;
int linear_search(int value)
{
      struct Node *temp = head->next;
      while(temp != 0)
      {
           if(temp->val == value)
              return 1;
           _________;
      }
      return 0;
}
```

Which of the following lines should be inserted to complete the above code?
a) temp = next
b) temp->next = temp
c) temp = temp->next
d) return 0

Answer: c
Explanation: The line "temp = temp->next" should be inserted to complete the above code.

3. What does the following code do?

```
#include<stdio.h>
#include<stdlib.h>
struct Node
{
     int val;
     struct Node* next;
}*head;
int linear_search(int value)
{
```

```
      struct Node *temp = head->next;
      while(temp != 0)
      {
           if(temp->val == value)
              return 1;
           temp = temp->next;
      }
      return 0;
}
int main()
{
      int arr[5] = {1,2,3,4,5};
      int n = 5,i;
      head = (struct Node*)malloc(sizeof(struct Node));
      head->next = 0;
      struct Node *temp;
      temp = head;
      for(i=0; i<n; i++)
      {
           struct Node *newNode = (struct Node*)malloc(sizeof(struct Node));
           newNode->next = 0;
           newNode->val = arr[i];
           temp->next = newNode;
           temp = temp->next;
      }
      int ans = linear_search(60);
      if(ans == 1)
      printf("Found");
      else
      printf("Not found");
      return 0;
}
```

a) Finds the index of the first occurrence of a number in a linked list
b) Finds the index of the last occurrence of a number in a linked list
c) Checks if a number is present in a linked list
d) Checks whether the given list is sorted or not

Answer: c
Explanation: The above code checks if a number is present in a linked list.

4. What is the output of the following code?

```
#include<stdio.h>
#include<stdlib.h>
struct Node
{
     int val;
     struct Node* next;
}*head;
int linear_search(int value)
{
      struct Node *temp = head->next;
      while(temp != 0)
      {
           if(temp->val == value)
             return 1;
           temp = temp->next;
      }
      return 0;
}
int main()
{
     int arr[5] = {1,2,3,4,5};
     int n = 5,i;
     head = (struct Node*)malloc(sizeof(struct Node));
     head->next = 0;
     struct Node *temp;
```

```
     temp = head;
     for(i=0; i<n; i++)
     {
           struct Node *newNode = (struct Node*)malloc(sizeof(struct Node));
           newNode->next = 0;
           newNode->val = arr[i];
           temp->next = newNode;
           temp = temp->next;
     }
     int ans = linear_search(-1);
     if(ans == 1)
     printf("Found");
     else
     printf("Not found");
     return 0;
}
```

a) Found
b) Not found
c) Compile time error
d) Runtime error

Answer: b
Explanation: Since the number -1 is not present in the linked list, the program prints not found.

5. What is the time complexity of the following implementation of linear search on a linked list?

```
#include<stdio.h>
#include<stdlib.h>
struct Node
{
     int val;
     struct Node* next;
}*head;
int linear_search(int value)
{
      struct Node *temp = head->next;
      while(temp != 0)
      {
           if(temp->val == value)
             return 1;
           temp = temp->next;
      }
      return 0;
}
int main()
{
     int arr[5] = {1,2,3,4,5};
     int n = 5,i;
     head = (struct Node*)malloc(sizeof(struct Node));
     head->next = 0;
     struct Node *temp;
     temp = head;
     for(i=0; i<n; i++)
     {
           struct Node *newNode = (struct Node*)malloc(sizeof(struct Node));
           newNode->next = 0;
           newNode->val = arr[i];
           temp->next = newNode;
           temp = temp->next;
     }
     int ans = linear_search(-1);
     if(ans == 1)
     printf("Found");
     else
     printf("Not found");
     return 0;
}
```

a) O(1)
b) O(n)
c) O(n^2)
d) O(n^3)

Answer: b
Explanation: The time complexity of the above implementation of linear search on a linked list is O(n).

6. What is the output of the following code?

```
#include<stdio.h>
#include<stdlib.h>
struct Node
{
     int val;
     struct Node* next;
}*head;
int linear_search(int value)
{
      struct Node *temp = head->next;
      while(temp -> next != 0)
      {
            if(temp->val == value)
            return 1;
            temp = temp->next;
      }
      return 0;
}
int main()
{
     int arr[6] = {1,2,3,4,5,6};
     int n = 6,i;
     head = (struct Node*)malloc(sizeof(struct Node));
     head->next = 0;
     struct Node *temp;
     temp = head;
     for(i=0; i<n; i++)
     {
           struct Node *newNode = (struct Node*)malloc(sizeof(struct Node));
           newNode->next = 0;
           newNode->val = arr[i];
           temp->next = newNode;
           temp = temp->next;
     }
     int ans = linear_search(60);
     if(ans == 1)
       printf("Found");
     else
       printf("Not found");
     return 0;
}
```

a) Found
b) Not found
c) Compile time error
d) Runtime error

Answer: b
Explanation: The condition in the while loop "temp->next == 0", checks if the current element is the last element. If the current element is the last element, the value of the current element is not compared with the value to be searched. So, even though the number 6 is present in the linked list, it will print not found.Answer: a
Explanation: Since linked list doesn't allow random access, binary search cannot be applied on a sorted linked list in O(Logn) time.

8. What will be time complexity when binary search is applied on a linked list?

a) O(1)
b) O(n)
c) O(n^2)
d) O(n^3)

Answer: b
Explanation: The time complexity will be O(n) when binary search is applied on a linked list.

9. Consider the following recursive implementation of linear search on a linked list:

```
struct Node
{
     int val;
     struct Node* next;
}*head;
int linear_search(struct Node *temp,int value)
{
      if(temp == 0)
         return 0;
      if(temp->val == value)
         return 1;
      return _________;
}
```

Which of the following lines should be inserted to complete the above code?
a) 1
b) 0
c) linear_search(temp, value)
d) linear_search(temp->next, value)

Answer: d
Explanation: The line "linear_search(temp->next, value)", should be inserted to complete the above code.

10. What is the output of the following code?

```
#include<stdio.h>
#include<stdlib.h>
struct Node
{
     int val;
     struct Node* next;
}*head;
int linear_search(struct Node *temp,int value)
{
      if(temp == 0)
         return 0;
      if(temp->val == value)
         return 1;
      return linear_search(temp->next, value);
}
int main()
{
     int arr[6] = {1,2,3,4,5,6};
     int n = 6,i;
     head = (struct Node*)malloc(sizeof(struct Node));
     head->next = 0;
     struct Node *temp;
     temp = head;
     for(i=0; i<n; i++)
     {
           struct Node *newNode = (struct Node*)malloc(sizeof(struct Node));
           newNode->next = 0;
           newNode->val = arr[i];
           temp->next = newNode;
           temp = temp->next;
     }
```

```
     int ans = linear_search(head->next,6);
     if(ans == 1)
       printf("Found");
     else
       printf("Not found");
     return 0;
}
```

a) Found
b) Not found
c) Compile time error
d) Runtime error

Answer: a
Explanation: Since the element 6 is present in the linked list, the program prints "Found".

11. How many times is the function linear_search() called when the following code is executed?

```
#include<stdio.h>
#include<stdlib.h>
struct Node
{
     int val;
     struct Node* next;
}*head;
int linear_search(struct Node *temp,int value)
{
      if(temp == 0)
         return 0;
      if(temp->val == value)
         return 1;
      return linear_search(temp->next, value);
}
int main()
{
     int arr[6] = {1,2,3,4,5,6};
     int n = 6,i;
     head = (struct Node*)malloc(sizeof(struct Node));
     head->next = 0;
     struct Node *temp;
     temp = head;
     for(i=0; i<n; i++)
     {
           struct Node *newNode = (struct Node*)malloc(sizeof(struct Node));
           newNode->next = 0;
           newNode->val = arr[i];
           temp->next = newNode;
           temp = temp->next;
     }
     int ans = linear_search(head->next,6);
     if(ans == 1)
       printf("Found");
     else
       printf("Not found");
    return 0;
}
```

a) 5
b) 6
c) 7
d) 8

Answer: b
Explanation: The function linear_search() is called 6 times when the above code is executed.

12. What is the time complexity of the following recursive implementation of linear search?

```
#include<stdio.h>
#include<stdlib.h>
struct Node
{
     int val;
     struct Node* next;
}*head;
int linear_search(struct Node *temp,int value)
{
      if(temp == 0)
         return 0;
      if(temp->val == value)
         return 1;
      return linear_search(temp->next, value);
}
int main()
{
     int arr[6] = {1,2,3,4,5,6};
     int n = 6,i;
     head = (struct Node*)malloc(sizeof(struct Node));
     head->next = 0;
     struct Node *temp;
     temp = head;
     for(i=0; i<n; i++)
     {
           struct Node *newNode = (struct Node*)malloc(sizeof(struct Node));
           newNode->next = 0;
           newNode->val = arr[i];
           temp->next = newNode;
           temp = temp->next;
     }
     int ans = linear_search(head->next,6);
     if(ans == 1)
       printf("Found");
     else
       printf("Not found");
    return 0;
}
```

a) O(1)
b) O(n)
c) O(n^2)
d) O(n^3)

Answer: b
Explanation: The time complexity of the above recursive implementation of linear search is O(n).

1. What will be the output for following code?

```
#include<stdio.h>
int func(int x,  int y)
{
        if (y == 0)
                return 1;
        else if (y%2 == 0)
                return func(x, y/2)*func(x, y/2);
        else
                return x*func(x, y/2)*func(x, y/2);
}
int main()
{
        int x = 2;
    int y = 3;

        printf("%d", func(x, y));
        return 0;
}
```

a) 9
b) 6
c) 8
d) 5

Answer: c
Explanation: The given program calculates the value of x raised to power y. Thus $2^3 = 8$.

2. What will be the time complexity of the following code which raises an integer x to the power y?

```
#include<stdio.h>
int power(int x,  int y)
{
        if (y == 0)
                return 1;
        else if (y%2 == 0)
                return power(x, y/2)*power(x, y/2);
        else
                return x*power(x, y/2)*power(x, y/2);
}
int main()
{
        int x = 2;
    int y = 3;

        printf("%d", power(x, y));
        return 0;
}
```

a) O(n)
b) O(log n)
c) O(n log n)
d) $O(n^2)$

Answer: a
Explanation: The recurrence relation for the above code is given by T(n)=2T(n/2)+c. By using master theorem we can calculate the result for this relation. It is found to be equal to O(n).

3. What is the space complexity of the given code?

```
#include<stdio.h>
int power(int x,  int y)
{
        if (y == 0)
                return 1;
        else if (y%2 == 0)
                return power(x, y/2)*power(x, y/2);
        else
                return x*power(x, y/2)*power(x, y/2);
}
int main()
{
        int x = 2;
    int y = 3;

        printf("%d", power(x, y));
        return 0;
}
```

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

Answer: a
Explanation: The space complexity of the given code will be equal to O(1) as it uses only constant space in the memory.

4. Recursive program to raise an integer x to power y uses which of the following algorithm?
a) Dynamic programming
b) Backtracking
c) Divide and conquer
d) Greedy algorithm

Answer: c
Explanation: The recursive approach uses divide and conquer algorithm as we break the problem into smaller parts and then solve the smaller parts and finally combine their results to get the overall solution.

5. What is the least time in which we can raise a number x to power y?
a) O(x)
b) O(y)
c) O(log x)
d) O(log y)

Answer: d
Explanation: We can optimize the code for finding power of a number by calculating x raised to power y/2 only once and using it depending on whether y is even or odd.

6. What will be the time complexity of the following code?

```
#include<stdio.h>
int power(int x, int y)
{
    int temp;
    if( y == 0)
        return 1;
    temp = power(x, y/2);
    if (y%2 == 0)
        return temp*temp;
    else
        return x*temp*temp;
}
int main()
{
        int x = 2;
    int y = 3;

        printf("%d", power(x, y));
        return 0;
}
```

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

Answer: c
Explanation: The given code is the optimized version for finding the power of a number. It forms a recurrence relation given by T(n)=T(n/2)+c which can be solved using master theorem. It is calculated to be equal to O(log n).

7. What is the advantage of iterative code for finding power of number over recursive code?
a) Iterative code requires less time
b) Iterative code requires less space
c) Iterative code is more compiler friendly
d) It has no advantage

Answer: b

Explanation: Both iterative and recursive approach can be implemented in log n time but the recursive code requires memory in call stack which makes it less preferable.

b)

```
#include <stdio.h>
int power(int x,  int y)
{
    int res = 1;
    while (y > 0)
    {

        if (y & 1)
            res = res * x;
        y = y >> 1;
        x = x * x;
    }
    return res;
}
int main()
{
    int x = 3;
    unsigned int y = 5;
    printf("%d", power(x, y));
    return 0;
}
```

c)

```
#include <stdio.h>
int power(int x,  int y)
{
    int res = 1;
    while (y > 0)
    {

        if (y && 1)
            res = res * x;
        y = y >> 1;
        x = x * x;
    }
    return res;
}
int main()
{
    int x = 3;
    unsigned int y = 5;
    printf("%d", power(x, y));
    return 0;
}
```

d)

```
#include <stdio.h>
int power(int x,  int y)
{
    int res = 1;
    while (y > 0)
    {
        if (y && 1)
        res = x * x;
        y = y >> 1;
        x = x * y;
    }
    return res;
}
int main()
{
```

```
    int x = 3;
    unsigned int y = 5;
    printf("%d", power(x, y));
    return 0;
}
```

Answer: b
Explanation: The recursive code requires memory in call stack which makes it less preferable as compared to iterative approach.

10. What will be the output for following code?

```
#include <stdio.h>
int power(int x,  int y)
{
    int res = 1;
    while (y > 0)
    {
        if (y & 1)
        res = x * x;
        y = y >> 1;
        x = x * y;
    }
    return res;
}
int main()
{
    int x = 3;
    unsigned int y = 5;
    printf("%d", power(x, y));
    return 0;
}
```

a) Error
b) 1/4
c) 4
d) 0.25

Answer: d
Explanation: The given code is capable of handling negative powers too. Thus, the output will be $2^{-2} = 0.25$.

1. What is the objective of tower of hanoi puzzle?
a) To move all disks to some other rod by following rules
b) To divide the disks equally among the three rods by following rules
c) To move all disks to some other rod in random order
d) To divide the disks equally among three rods in random order

Answer: a
Explanation: Objective of tower of hanoi problem is to move all disks to some other rod by following the following rules- 1) Only one disk can be moved at a time. 2) Disk can only be moved if it is the uppermost disk of the stack. 3) No disk should be placed over a smaller disk.

2. Which of the following is NOT a rule of tower of hanoi puzzle?
a) No disk should be placed over a smaller disk
b) Disk can only be moved if it is the uppermost disk of the stack
c) No disk should be placed over a larger disk
d) Only one disk can be moved at a time

Answer: c
Explanation: The rule is to not put a disk over a smaller one. Putting a smaller disk over larger one is allowed.

3. The time complexity of the solution tower of hanoi problem using recursion is _____

a) $O(n^2)$
b) $O(2^n)$
c) O(n log n)
d) O(n)

Answer: b
Explanation: Time complexity of the problem can be found out by solving the recurrence relation: T(n)=2T(n-1)+c. Result of this relation is found to be equal to 2^n. It can be solved using substitution.

4. Recurrence equation formed for the tower of hanoi problem is given by _____
a) T(n) = 2T(n-1)+n
b) T(n) = 2T(n/2)+c
c) T(n) = 2T(n-1)+c
d) T(n) = 2T(n/2)+n

Answer: c
Explanation: As there are 2 recursive calls to n-1 disks and one constant time operation so the recurrence relation will be given by T(n) = 2T(n-1)+c.

5. Minimum number of moves required to solve a tower of hanoi problem with n disks is _____
a) 2^n
b) 2^n-1
c) n^2
d) n^2-1

Answer: b
Explanation: Minimum number of moves can be calculated by solving the recurrence relation – T(n)=2T(n-1)+c. Alternatively we can observe the pattern formed by the series of number of moves 1,3,7,15…..Either way it turn out to be equal to 2^n-1.

6. Space complexity of recursive solution of tower of hanoi puzzle is _____
a) O(1)
b) O(n)
c) O(log n)
d) O(n log n)

Answer: b
Explanation: Space complexity of tower of hanoi problem can be found out by solving the recurrence relation T(n)=T(n-1)+c. Result of this relation is found out to be n. It can be solved using substitution.

b)

```
void ToH(int n,int a,int b,int c)
{
   If(n>0)
   {
       ToH(n-1,a,c,b);
       cout<<"move a disk from" <<a<<" to"<< c;
       ToH(n-1,b,a,c);
   }
}
```

c)

```
void ToH(int n,int a,int b,int c
{
   If(n>0)
   {
       ToH(n-1,a,b,c);
       cout<<"move a disk from" <<a<<" to"<< c;
```

```
        ToH(n-1,b,a,c);
    }
}
```

d)

```
void ToH(int n,int a,int b,int c)
{
     If(n>0)
     {
         ToH(n-1,a,c,b);
         cout<<"move a disk from" <<a<<" to"<< c;
         ToH(n-1,a,b,c);
     }
}
```

8. Recursive solution of tower of hanoi problem is an example of which of the following algorithm?
a) Dynamic programming
b) Backtracking
c) Greedy algorithm
d) Divide and conquer

Answer: d
Explanation: The recursive approach uses divide and conquer algorithm as we break the problem into smaller parts and then solve the smaller parts and finally combine their results to get the overall solution.Answer: a
Explanation: Iterative solution to tower of hanoi puzzle also exists. Its approach depends on whether the total numbers of disks are even or odd.

10. Minimum time required to solve tower of hanoi puzzle with 4 disks assuming one move takes 2 seconds, will be _____
a) 15 seconds
b) 30 seconds
c) 16 seconds
d) 32 seconds

Answer: b
Explanation: Number of moves = 2^4-1=16-1=15
Time for 1 move = 2 seconds
Time for 15 moves = 15×2 = 30 seconds.

1. Master's theorem is used for?
a) solving recurrences
b) solving iterative relations
c) analysing loops
d) calculating the time complexity of any code

Answer: a
Explanation: Master's theorem is a direct method for solving recurrences. We can solve any recurrence that falls under any one of the three cases of master's theorem.

2. How many cases are there under Master's theorem?
a) 2
b) 3
c) 4
d) 5

Answer: b
Explanation: There are primarily 3 cases under master's theorem. We can solve any recurrence that falls under any one of these three cases.

3. What is the result of the recurrences which fall under first case of Master's theorem (let the recurrence be given by T(n)=aT(n/b)+f(n) and f(n)=n^c?

a) T(n) = O(n^$\log_b a$)

b) T(n) = O(n^c log n)

c) T(n) = O(f(n))

d) T(n) = O(n^2)

Answer: a

Explanation: In first case of master's theorem the necessary condition is that $c < \log_b a$. If this condition is true then T(n) = O(n^$\log_b a$).

4. What is the result of the recurrences which fall under second case of Master's theorem (let the recurrence be given by T(n)=aT(n/b)+f(n) and f(n)=n^c?

a) T(n) = O(n$\log_b a$)

b) T(n) = O(n^c log n)

c) T(n) = O(f(n))

d) T(n) = O(n^2)

Answer: b

Explanation: In second case of master's theorem the necessary condition is that $c = \log_b a$. If this condition is true then T(n) = O(n^c log n)

5. What is the result of the recurrences which fall under third case of Master's theorem (let the recurrence be given by T(n)=aT(n/b)+f(n) and f(n)=n^c?

a) T(n) = O(n$\log_b a$)

b) T(n) = O(n^c log n)

c) T(n) = O(f(n))

d) T(n) = O(n^2)

Answer: c

Explanation: In third case of master's theorem the necessary condition is that $c > \log_b a$. If this condition is true then T(n) = O(f(n)).Answer: b

Explanation: No we cannot solve all the recurrences by only using master's theorem. We can solve only those which fall under the three cases prescribed in the theorem.

7. Under what case of Master's theorem will the recurrence relation of merge sort fall?

a) 1

b) 2

c) 3

d) It cannot be solved using master's theorem

Answer: b

Explanation: The recurrence relation of merge sort is given by T(n) = 2T(n/2) + O(n). So we can observe that $c = \text{Log}_b a$ so it will fall under case 2 of master's theorem.

8. Under what case of Master's theorem will the recurrence relation of stooge sort fall?

a) 1

b) 2

c) 3

d) It cannot be solved using master's theorem

Answer: a

Explanation: The recurrence relation of stooge sort is given as T(n) = 3T(2/3n) + O(1). It is found too be equal to O($n^{2.7}$) using master's theorem first case.

9. Which case of master's theorem can be extended further?
a) 1
b) 2
c) 3
d) No case can be extended

Answer: b
Explanation: The second case of master's theorem can be extended for a case where $f(n) = n^c (\log n)^k$ and the resulting recurrence becomes $T(n)= O(n^c (\log n))^{k+1}$.

10. What is the result of the recurrences which fall under the extended second case of Master's theorem (let the recurrence be given by $T(n)=aT(n/b)+f(n)$ and $f(n)=n^c(\log n)k$?
a) $T(n) = O(n\log_b a)$
b) $T(n) = O(n^c \log n)$
c) $T(n)= O(n^c (\log n)^{k+1}$
d) $T(n) = O(n^2)$

Answer: c
Explanation: In the extended second case of master's theorem the necessary condition is that $c = \log_b a$. If this condition is true then $T(n)= O(n^c(\log n))^{k+1}$.

11. Under what case of Master's theorem will the recurrence relation of binary search fall?
a) 1
b) 2
c) 3
d) It cannot be solved using master's theorem

Answer: b
Explanation: The recurrence relation of binary search is given by $T(n) = T(n/2) + O(1)$. So we can observe that $c = \text{Log}_b a$ so it will fall under case 2 of master's theorem.Answer: c
Explanation: The given recurrence can be solved by using the second case of Master's theorem.
$T(n) = O(nc \log n)$ = Here $nc = n^2$
So the solution becomes $T(n) = O(n^2 \log n)$.Answer: c
Explanation: The given recurrence can be solved by using the third case (as $f(n) > \log 21$) of Master's theorem.
$T(n) = O(f(n))$ = Here $f(n) = 2n$.
So the solution becomes $T(n) = O(2n)$. Answer: d
Explanation: The given recurrence can be solved by using the first case of Master's theorem. So the solution becomes $T(n) = O(n^2)$.Answer: d
Explanation: The given recurrence cannot be solved by using the Master's theorem. It is because this recurrence relation does not fit into any case of Master's theorem.Answer: d
Explanation: The given recurrence cannot be solved by using the Master's theorem. It is because in this recurrence relation $a < 1$ so master's theorem cannot be applied.Answer: a
Explanation: The given recurrence can be solved by using the third case of Master's theorem. So the solution becomes $T(n) = O(n!)$.Answer: a
Explanation: The given recurrence can be solved by using the extended second case of Master's theorem. So the solution becomes $T(n) = O(n (\log n)^2)$.

8. What will be the recurrence relation of the following code?

```
Int sum(int n)
{
   If(n==1)
       return 1;
   else
       return n+sum(n-1);
}
```

a) T(n) = T(n/2) + n
b) T(n) = T(n-1) + n
c) T(n) = T(n-1) + O(1)
d) T(n) = T(n/2) + O(1)

Answer: c
Explanation: As after every recursive call the integer up to which the sum is to be calculated decreases by 1. So the recurrence relation for the given code will be T(n) = T(n-1) + O(1).

9. What will be the recurrence relation of the following code?

```
int xpowy(int x, int n)
if (n==0) return 1;
if (n==1) return x;
if ((n % 2) == 0)
return xpowy(x*x, n/2);
else
return xpowy(x*x, n/2) * x;
```

a) T(n) = T(n/2) + n
b) T(n) = T(n-1) + n
c) T(n) = T(n-1) + O(1)
d) T(n) = T(n/2) + O(1)

Answer: d
Explanation: As after every recursive call the integer up to which the power is to be calculated decreases by half. So the recurrence relation for the given code will be T(n) = T(n/2) + O(1). It can be solved by using master's theorem.

10. What will be the time complexity of the following code?

```
int xpowy(int x, int n)
{
    if (n==0)
        return 1;
    if (n==1)
        return x;
    if ((n % 2) == 0)
        return xpowy(x*x, n/2);
    else
        return xpowy(x*x, n/2) * x;
}
```

a) O(log n)
b) O(n)
c) O(n log n)
d) O(n^2)

Answer: a
Explanation: As the recurrence relation of the code is given by T(n) = T(n/2) + O(1) so it can be solved by using master's theorem second case.

1. Fractional knapsack problem is also known as _____
a) 0/1 knapsack problem
b) Continuous knapsack problem
c) Divisible knapsack problem
d) Non continuous knapsack problem

Answer: b
Explanation: Fractional knapsack problem is also called continuous knapsack problem. Fractional knapsack is solved using dynamic programming.

2. Fractional knapsack problem is solved most efficiently by which of the following algorithm?
a) Divide and conquer
b) Dynamic programming
c) Greedy algorithm
d) Backtracking

Answer: c
Explanation: Greedy algorithm is used to solve this problem. We first sort items according to their value/weight ratio and then add item with highest ratio until we cannot add the next item as a whole. At the end, we add the next item as much as we can.

3. What is the objective of the knapsack problem?
a) To get maximum total value in the knapsack
b) To get minimum total value in the knapsack
c) To get maximum weight in the knapsack
d) To get minimum weight in the knapsack

Answer: a
Explanation: The objective is to fill the knapsack of some given volume with different materials such that the value of selected items is maximized.

4. Which of the following statement about 0/1 knapsack and fractional knapsack problem is correct?
a) In 0/1 knapsack problem items are divisible and in fractional knapsack items are indivisible
b) Both are the same
c) 0/1 knapsack is solved using a greedy algorithm and fractional knapsack is solved using dynamic programming
d) In 0/1 knapsack problem items are indivisible and in fractional knapsack items are divisible

Answer: d
Explanation: In fractional knapsack problem we can partially include an item into the knapsack whereas in 0/1 knapsack we have to either include or exclude the item wholly.

5. Time complexity of fractional knapsack problem is _____
a) O(n log n)
b) O(n)
c) $O(n^2)$
d) O(nW)

Answer: a
Explanation: As the main time taking a step is of sorting so it defines the time complexity of our code. So the time complexity will be O(n log n) if we use quick sort for sorting.Answer: a
Explanation: It is possible to solve the problem in O(n) time by adapting the algorithm for finding weighted medians.

7. Given items as {value,weight} pairs {{40,20},{30,10},{20,5}}. The capacity of knapsack=20. Find the maximum value output assuming items to be divisible.
a) 60
b) 80
c) 100
d) 40

Answer: a
Explanation: The value/weight ratio are-{2,3,4}. So we include the second and third items wholly into the knapsack. This leaves only 5 units of volume for the first item. So we include the first item partially.
Final value = 20+30+(40/4)=60.Answer: a
Explanation: As fractional knapsack gives extra liberty to include the object partially which is not possible with 0/1 knapsack, thus we get better results with a fractional knapsack.

9. The main time taking step in fractional knapsack problem is _____

a) Breaking items into fraction
b) Adding items into knapsack
c) Sorting
d) Looping through sorted items

Answer: c
Explanation: The main time taking step is to sort the items according to their value/weight ratio. It defines the time complexity of the code.

10. Given items as {value,weight} pairs {{60,20},{50,25},{20,5}}. The capacity of knapsack=40. Find the maximum value output assuming items to be divisible and nondivisible respectively.
a) 100, 80
b) 110, 70
c) 130, 110
d) 110, 80

Answer: d
Explanation: Assuming items to be divisible-
The value/weight ratio are {3, 2, 4}.So we include third and first items wholly. So, now only 15 units of volume are left for second item. So we include it partially.
Final volume = 20+60+50x(15/25)=80+30=110
Assuming items to be indivisible- In this case we will have to leave one item due to insufficient capacity.
Final volume = 60 + 20 = 80.

1. Which of the following algorithms is the best approach for solving Huffman codes?
a) exhaustive search
b) greedy algorithm
c) brute force algorithm
d) divide and conquer algorithm

Answer: b
Explanation: Greedy algorithm is the best approach for solving the Huffman codes problem since it greedily searches for an optimal solution.

2. How many printable characters does the ASCII character set consists of?
a) 120
b) 128
c) 100
d) 98

Answer: c
Explanation: Out of 128 characters in an ASCII set, roughly, only 100 characters are printable while the rest are non-printable.

3. Which bit is reserved as a parity bit in an ASCII set?
a) first
b) seventh
c) eighth
d) tenth

Answer: c
Explanation: In an ASCII character set, seven bits are reserved for character representation while the eighth bit is a parity bit.

4. How many bits are needed for standard encoding if the size of the character set is X?
a) log X
b) X+1
c) 2X

d) X^2

Answer: a
Explanation: If the size of the character set is X, then [log X] bits are needed for representation in a standard encoding.Answer: b
Explanation: The code length depends on the frequency of occurrence of characters. The more frequent the character occurs, the less is the length of the code.

6. In Huffman coding, data in a tree always occur?
a) roots
b) leaves
c) left sub trees
d) right sub trees

Answer: b
Explanation: In Huffman encoding, data is always stored at the leaves of a tree inorder to compute the codeword effectively.Answer: a
Explanation: By recording the path of the node from root to leaf, the code word for character 'a' is found to be 011.Answer: c
Explanation: By recording the path of the node from root to leaf, assigning left branch as 0 and right branch as 1, the codeword for c is 110.

9. What will be the cost of the code if character c_i is at depth d_i and occurs at frequency f_i?
a) $c_i f_i$
b) $\int c_i f_i$
c) $\sum f_i d_i$
d) $f_i d_i$

Answer: c
Explanation: If character c_i is at depth d_i and occurs at frequency f_i, the cost of the codeword obtained is $\sum f_i d_i$.Answer: a
Explanation: An optimal tree will always have the property that all nodes are either leaves or have two children. Otherwise, nodes with one child could move up a level.

11. The type of encoding where no character code is the prefix of another character code is called?
a) optimal encoding
b) prefix encoding
c) frequency encoding
d) trie encoding

Answer: b
Explanation: Even if the character codes are of different lengths, the encoding where no character code is the prefix of another character code is called prefix encoding.

12. What is the running time of the Huffman encoding algorithm?
a) O(C)
b) O(log C)
c) O(C log C)
d) O(N log C)

Answer: c
Explanation: If we maintain the trees in a priority queue, ordered by weight, then the running time is given by O(C log C).

13. What is the running time of the Huffman algorithm, if its implementation of the priority queue is done using linked lists?
a) O(C)

b) O(log C)
c) O(C log C)
d) O(C^2)

Answer: d
Explanation: If the implementation of the priority queue is done using linked lists, the running time of Huffman algorithm is O(C^2).

1. Which of the problems cannot be solved by backtracking method?
a) n-queen problem
b) subset sum problem
c) hamiltonian circuit problem
d) travelling salesman problem

Answer: d
Explanation: N-queen problem, subset sum problem, Hamiltonian circuit problems can be solved by backtracking method whereas travelling salesman problem is solved by Branch and bound method.

2. Backtracking algorithm is implemented by constructing a tree of choices called as?
a) State-space tree
b) State-chart tree
c) Node tree
d) Backtracking tree

Answer: a
Explanation: Backtracking problem is solved by constructing a tree of choices called as the state-space tree. Its root represents an initial state before the search for a solution begins.

3. What happens when the backtracking algorithm reaches a complete solution?
a) It backtracks to the root
b) It continues searching for other possible solutions
c) It traverses from a different route
d) Recursively traverses through the same route

Answer: b
Explanation: When we reach a final solution using a backtracking algorithm, we either stop or continue searching for other possible solutions.

4. A node is said to be _____ if it has a possibility of reaching a complete solution.
a) Non-promising
b) Promising
c) Succeeding
d) Preceding

Answer: b
Explanation: If a node has a possibility of reaching the final solution, it is called a promising node. Otherwise, it is non-promising.

5. In what manner is a state-space tree for a backtracking algorithm constructed?
a) Depth-first search
b) Breadth-first search
c) Twice around the tree
d) Nearest neighbour first

Answer: a
Explanation: A state-space tree for a backtracking algorithm is constructed in the manner of depth-first search so that it is easy to look into.Answer: b
Explanation: The leaves in a state space tree can either represent non-promising dead ends or complete solutions found

by the algorithm.

7. In general, backtracking can be used to solve?
a) Numerical problems
b) Exhaustive search
c) Combinatorial problems
d) Graph coloring problems

Answer: c
Explanation: Backtracking approach is used to solve complex combinatorial problems which cannot be solved by exhaustive search algorithms.

8. Which one of the following is an application of the backtracking algorithm?
a) Finding the shortest path
b) Finding the efficient quantity to shop
c) Ludo
d) Crossword

Answer: d
Explanation: Crossword puzzles are based on backtracking approach whereas the rest are travelling salesman problem, knapsack problem and dice game.Answer: a
Explanation: Backtracking is faster than brute force approach since it can remove a large set of answers in one test.

10. Which of the following logical programming languages is not based on backtracking?
a) Icon
b) Prolog
c) Planner
d) Fortran

Answer: d
Explanation: Backtracking algorithm form the basis for icon, planner and prolog whereas fortran is an ancient assembly language used in second generation computers.

11. The problem of finding a list of integers in a given specific range that meets certain conditions is called?
a) Subset sum problem
b) Constraint satisfaction problem
c) Hamiltonian circuit problem
d) Travelling salesman problem

Answer: b
Explanation: Constraint satisfaction problem is the problem of finding a list of integers under given constraints. Constraint satisfaction problem is solved using a backtracking approach.

12. Who coined the term 'backtracking'?
a) Lehmer
b) Donald
c) Ross
d) Ford

Answer: a
Explanation: D.H. Lehmer was the first person to coin the term backtracking. Initially, the backtracking facility was provided using SNOBOL.

13. _____ enumerates a list of promising nodes that could be computed to give the possible solutions of a given problem.
a) Exhaustive search
b) Brute force
c) Backtracking

d) Divide and conquer

Answer: c
Explanation: Backtracking is a general algorithm that evaluates partially constructed candidates that can be developed further without violating problem constraints.

14. The problem of finding a subset of positive integers whose sum is equal to a given positive integer is called as?
a) n- queen problem
b) subset sum problem
c) knapsack problem
d) hamiltonian circuit problem

Answer: b
Explanation: Subset sum problem is the problem of finding a subset using the backtracking algorithm when summed, equals a given integer.

15. The problem of placing n queens in a chessboard such that no two queens attack each other is called as?
a) n-queen problem
b) eight queens puzzle
c) four queens puzzle
d) 1-queen problem

Answer: a
Explanation: The problem of placing n-queens in a chessboard such that no two queens are vertical or horizontal or diagonal to each other is an n-queen problem. The problem only exists for n = 1, 4, 8.

1. Who published the eight queens puzzle?
a) Max Bezzel
b) Carl
c) Gauss
d) Friedrich

Answer: a
Explanation: The first Eight Queen Puzzle was published by Max Friedrich William Bezzel, who was a chess composer by profession in 1848. He was a German chess composer and the first person to publish the puzzle.

2. When was the Eight Queen Puzzle published?
a) 1846
b) 1847
c) 1848
d) 1849

Answer: c
Explanation: The first Eight Queen Puzzle was published by Max Friedrich William Bezzel, who was a German chess composer by profession. He published the puzzle in 1848.

3. Who published the first solution of the eight queens puzzle?
a) Franz Nauck
b) Max Bezzel
c) Carl
d) Friedrich

Answer: a
Explanation: The first solution to the Eight Queen Puzzle was given by Franz Nauck in 1850. While the first Eight Queen Puzzle was published by Max Friedrich William Bezzel, who was a German chess composer.

4. When was the first solution to Eight Queen Puzzle published?
a) 1850

b) 1847
c) 1848
d) 1849

Answer: a
Explanation: The first solution to the Eight Queen Puzzle was given by Franz Nauck in 1850. Max Friedrich William Bezzel, who was a German chess composer by profession published the puzzle in 1848.

5. Who published the extended version of eight queens puzzle?
a) Franz Nauck
b) Max Bezzel
c) Carl
d) Friedrich

Answer: a
Explanation: The first extended version to the Eight Queen Puzzle was given by Franz Nauck in 1850. Max Friedrich William Bezzel published the puzzle in 1848.

6. For how many queens was the extended version of Eight Queen Puzzle applicable for n*n squares?
a) 5
b) 6
c) 8
d) n

Answer: d
*Explanation: The extended version given by Franz Nauck of the Eight Queen Puzzle was for n queens on n*n square chessboard. Earlier the puzzle was proposed with 8 queens on 8*8 board.*

7. Who was the first person to find the solution of Eight Queen Puzzle using determinant?
a) Max Bezzel
b) Frank Nauck
c) Gunther
d) Friedrich

Answer: c
Explanation: S. Gunther was the first person to propose a solution to the eight queen puzzle using determinant. Max Friedrich William Bezzel published the puzzle and the first solution to the Eight Queen Puzzle was given by Franz Nauck.

8. Who proposed the depth first backtracking algorithm?
a) Edsger Dijkshtra
b) Max Bezzel
c) Frank Nauck
d) Carl Friedrich

Answer: a
Explanation: In 1972, depth first backtracking algorithm was proposed by Edsger Dijkshtra to illustrate the Eight Queen Puzzle. Max Friedrich William Bezzel published the puzzle and the first solution to the Eight Queen Puzzle was given by Franz Nauck.

9. How many solutions are there for 8 queens on 8*8 board?
a) 12
b) 91
c) 92
d) 93

Answer: c
*Explanation: For 8*8 chess board with 8 queens there are total of 92 solutions for the puzzle. There are total of 12 fundamental solutions to the eight queen puzzle.*

10. Who publish the bitwise operation method to solve the eight queen puzzle?
a) Zongyan Qiu
b) Martin Richard
c) Max Bezzel
d) Frank Nauck

Answer: a
Explanation: The first person to publish the bitwise operation method to solve the eight queen puzzle was Zongyan Qiu. After him, it was published by Martin Richard.

11. How many fundamental solutions are there for the eight queen puzzle?
a) 92
b) 10
c) 11
d) 12

Answer: d
*Explanation: There are total of 12 fundamental solutions to the eight queen puzzle after removing the symmetrical solutions due to rotation. For 8*8 chess board with 8 queens there are total of 92 solutions for the puzzle.Answer: b*
Explanation: No three queens lie in a straight line in one of the fundamental solution of the eight queen puzzle.

13. How many fundamental solutions are the for 3 queens on a 3*3 board?
a) 1
b) 12
c) 3
d) 0

Answer: d
*Explanation: There are in total zero solution to the 3 queen puzzle for 3*3 chess board. Hence there are no fundamental solutions. For 8*8 chess board with 8 queens there are total of 12 fundamental solutions for the puzzle.Answer: a*
Explanation: There are total 4 solutions for the six queen puzzle and one fundamental solution while there are total of 10 solutions for 5 queen puzzle and 2 fundamental solutions.

15. Which ordered board is the highest enumerated board till now?
a) 25*25
b) 26*26
c) 27*27
d) 28*28

Answer: c
*Explanation: The 27*27 board has the highest order board with total 234,907,967,154,122,528 solutions. It also has total of 29,363,495,934,315,694 fundamental solution.*

1. In how many directions do queens attack each other?
a) 1
b) 2
c) 3
d) 4

Answer: c
Explanation: Queens attack each other in three directions- vertical, horizontal and diagonal.

2. Placing n-queens so that no two queens attack each other is called?
a) n-queen's problem
b) 8-queen's problem
c) Hamiltonian circuit problem
d) subset sum problem

Answer: a
Explanation: Placing n queens so that no two queens attack each other is n-queens problem. If n=8, it is called as 8-queens problem.

3. Where is the n-queens problem implemented?
a) carom
b) chess
c) ludo
d) cards

Answer: b
*Explanation: N-queens problem occurs in chess. It is the problem of placing n- queens in a n*n chess board.Answer: b*
Explanation: Unlike a real chess game, n-queens occur in a n-queen problem since it is the problem of dealing with n-queens.

5. In n-queen problem, how many values of n does not provide an optimal solution?
a) 1
b) 2
c) 3
d) 4

Answer: b
Explanation: N-queen problem does not provide an optimal solution of only three values of n (i.e.) n=2,3.

6. Which of the following methods can be used to solve n-queen's problem?
a) greedy algorithm
b) divide and conquer
c) iterative improvement
d) backtracking

Answer: d
Explanation: Of the following given approaches, n-queens problem can be solved using backtracking. It can also be solved using branch and bound.

7. Of the following given options, which one of the following is a correct option that provides an optimal solution for 4-queens problem?
a) (3,1,4,2)
b) (2,3,1,4)
c) (4,3,2,1)
d) (4,2,3,1)

Answer: a
Explanation: Of the following given options for optimal solutions of 4-queens problem, (3, 1, 4, 2) is the right option.

8. How many possible solutions exist for an 8-queen problem?
a) 100
b) 98
c) 92
d) 88

Answer: c
Explanation: For an 8-queen problem, there are 92 possible combinations of optimal solutions.

9. How many possible solutions occur for a 10-queen problem?
a) 850
b) 742
c) 842
d) 724

Answer: d
Explanation: For a 10-queen problem, 724 possible combinations of optimal solutions are available.Answer: b

Explanation: For n=1, the n-queen problem has a trivial and real solution and it is represented by

11. What is the domination number for 8-queen's problem?
a) 8
b) 7
c) 6
d) 5

Answer: d
Explanation: The minimum number of queens needed to occupy every square in n-queens problem is called domination number. While n=8, the domination number is 5.

12. Of the following given options, which one of the following does not provides an optimal solution for 8-queens problem?
a) (5,3,8,4,7,1,6,2)
b) (1,6,3,8,3,2,4,7)
c) (4,1,5,8,6,3,7,2)
d) (6,2,7,1,4,8,5,3)

Answer: b
Explanation: The following given options for optimal solutions of 8-queens problem, (1,6,3,8,3,2,4,7) does not provide an optimal solution.

1. Which of the following is/are property/properties of a dynamic programming problem?
a) Optimal substructure
b) Overlapping subproblems
c) Greedy approach
d) Both optimal substructure and overlapping subproblems

Answer: d
Explanation: A problem that can be solved using dynamic programming possesses overlapping subproblems as well as optimal substructure properties.

2. If an optimal solution can be created for a problem by constructing optimal solutions for its subproblems, the problem possesses _____ property.
a) Overlapping subproblems
b) Optimal substructure
c) Memoization
d) Greedy

Answer: b
Explanation: Optimal substructure is the property in which an optimal solution is found for the problem by constructing optimal solutions for the subproblems.

3. If a problem can be broken into subproblems which are reused several times, the problem possesses _____ property.
a) Overlapping subproblems
b) Optimal substructure
c) Memoization
d) Greedy

Answer: a
Explanation: Overlapping subproblems is the property in which value of a subproblem is used several times.

4. If a problem can be solved by combining optimal solutions to non-overlapping problems, the strategy is called _____
a) Dynamic programming
b) Greedy
c) Divide and conquer
d) Recursion

Answer: c
Explanation: In divide and conquer, the problem is divided into smaller non-overlapping subproblems and an optimal solution for each of the subproblems is found. The optimal solutions are then combined to get a global optimal solution. For example, mergesort uses divide and conquer strategy.Answer: a
Explanation: Dynamic programming calculates the value of a subproblem only once, while other methods that don't take advantage of the overlapping subproblems property may calculate the value of the same subproblem several times. So, dynamic programming saves the time of recalculation and takes far less time as compared to other methods that don't take advantage of the overlapping subproblems property.Answer: b
Explanation: A greedy algorithm gives optimal solution for all subproblems, but when these locally optimal solutions are combined it may NOT result into a globally optimal solution. Hence, a greedy algorithm CANNOT be used to solve all the dynamic programming problems.

7. In dynamic programming, the technique of storing the previously calculated values is called _____
a) Saving value property
b) Storing value property
c) Memoization
d) Mapping

Answer: c
Explanation: Memoization is the technique in which previously calculated values are stored, so that, these values can be used to solve other subproblems.

8. When a top-down approach of dynamic programming is applied to a problem, it usually _____
a) Decreases both, the time complexity and the space complexity
b) Decreases the time complexity and increases the space complexity
c) Increases the time complexity and decreases the space complexity
d) Increases both, the time complexity and the space complexity

Answer: b
Explanation: The top-down approach uses the memoization technique which stores the previously calculated values. Due to this, the time complexity is decreased but the space complexity is increased.

9. Which of the following problems is NOT solved using dynamic programming?
a) 0/1 knapsack problem
b) Matrix chain multiplication problem
c) Edit distance problem
d) Fractional knapsack problem

Answer: d
Explanation: The fractional knapsack problem is solved using a greedy algorithm.

10. Which of the following problems should be solved using dynamic programming?
a) Mergesort
b) Binary search
c) Longest common subsequence
d) Quicksort

Answer: c
Explanation: The longest common subsequence problem has both, optimal substructure and overlapping subproblems. Hence, dynamic programming should be used the solve this problem.Answer: d

Explanation: Each of the above mentioned methods can be used to find the nth fibonacci term.

2. Consider the recursive implementation to find the nth fibonacci number:

```
int fibo(int n)
    if n <= 1
        return n
    return __________
```

Which line would make the implementation complete?
a) fibo(n) + fibo(n)
b) fibo(n) + fibo(n – 1)
c) fibo(n – 1) + fibo(n + 1)
d) fibo(n – 1) + fibo(n – 2)

Answer: d
Explanation: Consider the first five terms of the fibonacci sequence: 0,1,1,2,3. The 6th term can be found by adding the two previous terms, i.e. fibo(6) = fibo(5) + fibo(4) = 3 + 2 = 5. Therefore,the nth term of a fibonacci sequence would be given by:
fibo(n) = fibo(n-1) + fibo(n-2).

3. What is the time complexity of the recursive implementation used to find the nth fibonacci term?
a) O(1)
b) $O(n^2)$
c) O(n!)
d) Exponential

Answer: d
Explanation: The recurrence relation is given by fibo(n) = fibo(n – 1) + fibo(n – 2). So, the time complexity is given by:
$T(n) = T(n – 1) + T(n – 2)$
Approximately,
$T(n) = 2 * T(n – 1)$
$= 4 * T(n – 2)$
$= 8 * T(n – 3)$
:
:
:
$= 2^k * T(n – k)$
This recurrence will stop when $n – k = 0$
i.e. $n = k$
Therefore, $T(n) = 2^n * O(0) = 2^n$
Hence, it takes exponential time.
It can also be proved by drawing the recursion tree and counting the number of leaves.

4. Suppose we find the 8th term using the recursive implementation. The arguments passed to the function calls will be as follows:

```
fibonacci(8)
fibonacci(7) + fibonacci(6)
fibonacci(6) + fibonacci(5) + fibonacci(5) + fibonacci(4)
fibonacci(5) + fibonacci(4) + fibonacci(4) + fibonacci(3) + fibonacci(4)
+ fibonacci(3)  + fibonacci(3) + fibonacci(2)
:
:
:
```

Which property is shown by the above function calls?
a) Memoization
b) Optimal substructure

c) Overlapping subproblems
d) Greedy

Answer: c
Explanation: From the function calls, we can see that fibonacci(4) is calculated twice and fibonacci(3) is calculated thrice. Thus, the same subproblem is solved many times and hence the function calls show the overlapping subproblems property.

5. What is the output of the following program?

```
#include<stdio.h>
int fibo(int n)
{
      if(n<=1)
         return n;
      return fibo(n-1) + fibo(n-2);
}
int main()
{
      int r = fibo(50000);
      printf("%d",r);
      return 0;
}
```

a) 1253556389
b) 5635632456
c) Garbage value
d) Runtime error

Answer: d
Explanation: The value of n is 50000. The function is recursive and it's time complexity is exponential. So, the function will be called almost 2^{50000} times. Now, even though NO variables are stored by the function, the space required to store the addresses of these function calls will be enormous. Stack memory is utilized to store these addresses and only a particular amount of stack memory can be used by any program. So, after a certain function call, no more stack space will be available and it will lead to stack overflow causing runtime error.

6. What is the space complexity of the recursive implementation used to find the nth fibonacci term?
a) O(1)
b) O(n)
c) O(n^2)
d) O(n^3)

Answer: a
Explanation: The recursive implementation doesn't store any values and calculates every value from scratch. So, the space complexity is O(1).

7. Consider the following code to find the nth fibonacci term:

```
int fibo(int n)
    if n == 0
       return 0
    else
       prevFib = 0
       curFib = 1
       for i : 1 to n-1
           nextFib = prevFib + curFib
           __________
           __________
       return curFib
```

8. What is the time complexity of the following for loop method used to compute the nth fibonacci term?

```
int fibo(int n)
    if n == 0
       return 0
    else
       prevFib = 0
       curFib = 1
       for i : 1 to n-1
           nextFib = prevFib + curFib
           prevFib = curFib
           curFib = nextFib
       return curFib
```

a) O(1)
b) O(n)
c) O(n^2)
d) Exponential

Answer: b
Explanation: To calculate the nth term, the for loop runs (n – 1) times and each time a for loop is run, it takes a constant time. Therefore, the time complexity is of the order of n.

9. What is the space complexity of the following for loop method used to compute the nth fibonacci term?

```
int fibo(int n)
    if n == 0
       return 0
    else
       prevFib = 0
       curFib = 1
       for i : 1 to n-1
           nextFib = prevFib + curFib
           prevFib = curFib
           curFib = nextFib
       return curFib
```

a) O(1)
b) O(n)
c) O(n^2)
d) Exponential

Answer: a
Explanation: To calculate the nth term, we just store the previous term and the current term and then calculate the next term using these two terms. It takes a constant space to store these two terms and hence O(1) is the answer.

10. What will be the output when the following code is executed?

```
#include<stdio.h>
int fibo(int n)
{
      if(n==0)
         return 0;
      int i;
      int prevFib=0,curFib=1;
      for(i=1;i<=n-1;i++)
      {
           int nextFib = prevFib + curFib;
           prevFib = curFib;
           curFib = nextFib;
      }
      return curFib;
}
int main()
{
      int r = fibo(10);
      printf("%d",r);
```

```
    return 0;
}
```

a) 34
b) 55
c) Compile error
d) Runtime error

Answer: b
Explanation: The output is the 10th fibonacci number, which is 55.

11. Consider the following code to find the nth fibonacci term using dynamic programming:

```
1. int fibo(int n)
2.   int fibo_terms[100000]  //arr to store the fibonacci numbers
3.   fibo_terms[0] = 0
4.   fibo_terms[1] = 1
5.
6.   for i: 2 to n
7.       fibo_terms[i] = fibo_terms[i - 1] + fibo_terms[i - 2]
8.
9.   return fibo_terms[n]
```

Which property is shown by line 7 of the above code?
a) Optimal substructure
b) Overlapping subproblems
c) Both overlapping subproblems and optimal substructure
d) Greedy substructure

Answer: a
Explanation: We find the nth fibonacci term by finding previous fibonacci terms, i.e. by solving subproblems. Hence, line 7 shows the optimal substructure property.

12. Consider the following code to find the nth fibonacci term using dynamic programming:

```
1. int fibo(int n)
2.      int fibo_terms[100000]  //arr to store the fibonacci numbers
3.      fibo_terms[0] = 0
4.      fibo_terms[1] = 1
5.
6.      for i: 2 to n
7.              fibo_terms[i] = fibo_terms[i - 1] + fibo_terms[i - 2]
8.
9.      return fibo_terms[n]
```

Which technique is used by line 7 of the above code?
a) Greedy
b) Recursion
c) Memoization
d) Overlapping subproblems

Answer: c
Explanation: Line 7 stores the current value that is calculated, so that the value can be used later directly without calculating it from scratch. This is memoization.

13. What is the time complexity of the following dynamic programming implementation used to compute the nth fibonacci term?

```
1. int fibo(int n)
2.      int fibo_terms[100000]  //arr to store the fibonacci numbers
3.      fibo_terms[0] = 0
4.      fibo_terms[1] = 1
5.
```

```
6.      for i: 2 to n
7.              fibo_terms[i] = fibo_terms[i - 1] + fibo_terms[i - 2]
8.
9.      return fibo_terms[n]
```

a) O(1)
b) O(n)
c) O(n^2)
d) Exponential

Answer: b
Explanation: To calculate the nth term, the for loop runs (n – 1) times and each time a for loop is run, it takes a constant time. Therefore, the time complexity is of the order of n.

14. What is the space complexity of the following dynamic programming implementation used to compute the nth fibonacci term?

```
int fibo(int n)
        int fibo_terms[100000]  //arr to store the fibonacci numbers
        fibo_terms[0] = 0
        fibo_terms[1] = 1

        for i: 2 to n
                fibo_terms[i] = fibo_terms[i - 1] + fibo_terms[i - 2]

        return fibo_terms[n]
```

a) O(1)
b) O(n)
c) O(n^2)
d) Exponential

Answer: b
Explanation: To calculate the nth term, we store all the terms from 0 to n – 1. So, it takes O(n) space.

15. What will be the output when the following code is executed?

```
#include<stdio.
int fibo(int n)
{
      int i;
      int fibo_terms[100];
      fibo_terms[0]=0;
      fibo_terms[1]=1;
      for(i=2;i<=n;i++)
          fibo_terms[i] = fibo_terms[i-2] + fibo_terms[i-1];
      return fibo_terms[n];
}
int main()
{
      int r = fibo(8);
      printf("%d",r);
      return 0;
}
```

a) 34
b) 55
c) Compile error
d) 21

Answer: d
Explanation: The program prints the 8th fibonacci term, which is 21.

1. You are given infinite coins of denominations v1, v2, v3,…..,vn and a sum S. The coin change problem is to find the

minimum number of coins required to get the sum S. This problem can be solved using _____
a) Greedy algorithm
b) Dynamic programming
c) Divide and conquer
d) Backtracking

Answer: b
Explanation: The coin change problem has overlapping subproblems(same subproblems are solved multiple times) and optimal substructure(the solution to the problem can be found by finding optimal solutions for subproblems). So, dynamic programming can be used to solve the coin change problem.

2. Suppose you have coins of denominations 1, 3 and 4. You use a greedy algorithm, in which you choose the largest denomination coin which is not greater than the remaining sum. For which of the following sums, will the algorithm NOT produce an optimal answer?
a) 20
b) 12
c) 6
d) 5

Answer: c
Explanation: Using the greedy algorithm, three coins {4,1,1} will be selected to make a sum of 6. But, the optimal answer is two coins {3,3}.

3. Suppose you have coins of denominations 1,3 and 4. You use a greedy algorithm, in which you choose the largest denomination coin which is not greater than the remaining sum. For which of the following sums, will the algorithm produce an optimal answer?
a) 14
b) 10
c) 6
d) 100

Answer: d
Explanation: Using the greedy algorithm, three coins {4,1,1} will be selected to make a sum of 6. But, the optimal answer is two coins {3,3}. Similarly, four coins {4,4,1,1} will be selected to make a sum of 10. But, the optimal answer is three coins {4,3,3}. Also, five coins {4,4,4,1,1} will be selected to make a sum of 14. But, the optimal answer is four coins {4,4,3,3}. For a sum of 100, twenty-five coins {all 4's} will be selected and the optimal answer is also twenty-five coins {all 4's}.

4. Fill in the blank to complete the code.

```
#include<stdio.h>
int main()
{
      int coins[10]={1,3,4},lookup[100000];
      int i,j,tmp,num_coins = 3,sum=100;
      lookup[0]=0;
      for(i = 1; i <= sum; i++)
      {
           int min_coins = i;
           for(j = 0;j < num_coins; j++)
           {
                tmp = i - coins[j];
                if(tmp < 0)
                 continue;
                if(lookup[tmp] < min_coins)
               _____________;
           }
           lookup[i] = min_coins + 1;
      }
      printf("%d",lookup[sum]);
      return 0;
```

```
}
```

a) lookup[tmp] = min_coins
b) min_coins = lookup[tmp]
c) break
d) continue

Answer: b
Explanation: min_coins = lookup[tmp] will complete the code.

5. You are given infinite coins of N denominations v1, v2, v3,…..,vn and a sum S. The coin change problem is to find the minimum number of coins required to get the sum S. What is the time complexity of a dynamic programming implementation used to solve the coin change problem?
a) O(N)
b) O(S)
c) $O(N^2)$
d) O(S*N)

Answer: d
*Explanation: The time complexity is O(S*N).*

6. Suppose you are given infinite coins of N denominations v1, v2, v3,…..,vn and a sum S. The coin change problem is to find the minimum number of coins required to get the sum S. What is the space complexity of a dynamic programming implementation used to solve the coin change problem?
a) O(N)
b) O(S)
c) $O(N^2)$
d) O(S*N)

Answer: b
Explanation: To get the optimal solution for a sum S, the optimal solution is found for each sum less than equal to S and each solution is stored. So, the space complexity is O(S).

7. You are given infinite coins of denominations 1, 3, 4. What is the total number of ways in which a sum of 7 can be achieved using these coins if the order of the coins is not important?
a) 4
b) 3
c) 5
d) 6

Answer: c
Explanation: A sum of 7 can be achieved in the following ways:
{1,1,1,1,1,1,1}, {1,1,1,1,3}, {1,3,3}, {1,1,1,4}, {3,4}.
Therefore, the sum can be achieved in 5 ways.

8. You are given infinite coins of denominations 1, 3, 4. What is the minimum number of coins required to achieve a sum of 7?
a) 1
b) 2
c) 3
d) 4

Answer: b
Explanation: A sum of 7 can be achieved by using a minimum of two coins {3,4}.

9. You are given infinite coins of denominations 5, 7, 9. Which of the following sum CANNOT be achieved using these coins?
a) 50

b) 21
c) 13
d) 23

Answer: c
Explanation: One way to achieve a sum of 50 is to use ten coins of 5. A sum of 21 can be achieved by using three coins of 7. One way to achieve a sum of 23 is to use two coins of 7 and one coin of 9. A sum of 13 cannot be achieved.

10. You are given infinite coins of denominations 3, 5, 7. Which of the following sum CANNOT be achieved using these coins?
a) 15
b) 16
c) 17
d) 4

Answer: d
Explanation: Sums can be achieved as follows:
15 = {5,5,5}
16 = {3,3,5,5}
17 = {3,7,7}
we can't achieve for sum=4 because our available denominations are 3,5,7 and sum of any two denominations is greater than 4.

11. What is the output of the following program?

```
#include<stdio.h>
int main()
{
      int coins[10]={1,3,4},lookup[100];
      int i,j,tmp,num_coins = 3,sum=10;
      lookup[0]=0;
      for(i=1;i<=sum;i++)
      {
            int min_coins = i;
            for(j=0;j<num_coins;j++)
            {
                  tmp=i-coins[j];
                  if(tmp<0)
                   continue;
                  if(lookup[tmp] < min_coins)
                  min_coins=lookup[tmp];
            }
            lookup[i] = min_coins + 1;
      }
      printf("%d",lookup[sum]);
      return 0;
}
```

a) 2
b) 3
c) 4
d) 5

Answer: b
Explanation: The program prints the minimum number of coins required to get a sum of 10, which is 3.

12. What is the output of the following program?

```
#include<stdio.h>
int main()
{
      int coins[10]={1,3,4},lookup[100];
      int i,j,tmp,num_coins = 3,sum=14;
```

```
      lookup[0]=0;
      for(i=1;i<=sum;i++)
      {
           int min_coins = i;
           for(j=0;j<num_coins;j++)
           {
                 tmp=i-coins[j];
                 if(tmp<0)
                  continue;
                 if(lookup[tmp] < min_coins)
                 min_coins=lookup[tmp];
           }
           lookup[i] = min_coins + 1;
      }
      printf("%d",lookup[sum]);
      return 0;
}
```

a) 2
b) 3
c) 4
d) 5

Answer: c
Explanation: The program prints the minimum number of coins required to get a sum of 14, which is 4.

1. Given a one-dimensional array of integers, you have to find a sub-array with maximum sum. This is the maximum sub-array sum problem. Which of these methods can be used to solve the problem?
a) Dynamic programming
b) Two for loops (naive method)
c) Divide and conquer
d) Dynamic programming, naïve method and Divide and conquer methods

Answer: d
Explanation: Dynamic programming, naïve method and Divide and conquer methods can be used to solve the maximum sub array sum problem.Answer: b
Explanation: The maximum sub-array sum is 5.Answer: d
Explanation: All the elements are negative. So, the maximum sub-array sum will be equal to the largest element. The largest element is -1 and therefore, the maximum sub-array sum is -1.

4. Consider the following naive method to find the maximum sub-array sum:

```
#include<stdio.h>
int main()
{
     int arr[1000]={2, -1, 3, -4, 1, -2, -1, 5, -4}, len=9;
     int cur_max, tmp_max, strt_idx, sub_arr_idx;
     cur_max = arr[0];
     for(strt_idx = 0; strt_idx < len; strt_idx++)
     {
          tmp_max=0;
          for(sub_arr_idx = strt_idx; sub_arr_idx < len; sub_arr_idx++)
          {
               tmp_max +=arr[sub_arr_idx];
               if(tmp_max > cur_max)
                 _____________;
          }
     }
     printf("%d",cur_max);
     return 0;
}
```

Which line should be inserted to complete the above code?
a) tmp_max = cur_max

b) break
c) continue
d) cur_max = tmp_max

Answer: d
Explanation: If the tmp_max element is greater than the cur_max element, we make cur_max equal to tmp_max, i.e. cur_max = tmp_max.

5. What is the time complexity of the following naive method used to find the maximum sub-array sum in an array containing n elements?

```
#include<stdio.h>
int main()
{
     int arr[1000]={2, -1, 3, -4, 1, -2, -1, 5, -4}, len=9;
     int cur_max, tmp_max, strt_idx, sub_arr_idx;
     cur_max = arr[0];
     for(strt_idx = 0; strt_idx < len; strt_idx++)
     {
          tmp_max=0;
          for(sub_arr_idx = strt_idx; sub_arr_idx < len; sub_arr_idx++)
          {
               tmp_max +=arr[sub_arr_idx];
               if(tmp_max > cur_max)
                 _____________;
          }
     }
     printf("%d",cur_max);
     return 0;
}
```

a) $O(n^2)$
b) O(n)
c) $O(n^3)$
d) O(1)

Answer: a
Explanation: The naive method uses two for loops. The outer loop runs from 0 to n,
i.e. i = 0 : n.
The inner loop runs from i to n, i.e. j = i : n.
Therefore, the inner loop runs:
n times when the outer loop runs the first time.
(n-1) times when the outer loop runs the second time.
:
:
:
2 times when the outer loop runs (n-1)th time.
1 time when the outer loop runs nth time.
*Therefore, time complexity = n + (n-1) + (n-2) + …… + 2 + 1 = n * (n + 1) /2 = $O(n^2)$.*

6. What is the space complexity of the following naive method used to find the maximum sub-array sum in an array containing n elements?

```
#include<stdio.h>
int main()
{
     int arr[1000]={2, -1, 3, -4, 1, -2, -1, 5, -4}, len=9;
     int cur_max, tmp_max, strt_idx, sub_arr_idx;
     cur_max = arr[0];
     for(strt_idx = 0; strt_idx < len; strt_idx++)
     {
          tmp_max=0;
```

```
        for(sub_arr_idx = strt_idx; sub_arr_idx < len; sub_arr_idx++)
        {
             tmp_max +=arr[sub_arr_idx];
             if(tmp_max > cur_max)
               _____________;
        }
     }
     printf("%d",cur_max);
     return 0;
}
```

a) $O(n^2)$
b) O(1)
c) $O(n^3)$
d) O(n)

Answer: b
Explanation: The naive method uses only a constant space. So, the space complexity is O(1).

7. What is the output of the following naive method used to find the maximum sub-array sum?

```
#include<stdio.h>
int main()
{
     int arr[1000] = {-2, -5, 6, -2, 3, -1, 0,-5, 6}, len = 9;
     int cur_max, tmp_max, strt_idx, sub_arr_idx;
     cur_max = arr[0];
     for(strt_idx = 0; strt_idx < len; strt_idx++)
     {
          tmp_max = 0;
          for(sub_arr_idx = strt_idx; sub_arr_idx < len; sub_arr_idx++)
          {
               tmp_max += arr[sub_arr_idx];
               if(tmp_max > cur_max)
               cur_max = tmp_max;
          }
     }
     printf("%d",cur_max);
     return 0;
}
```

a) 6
b) 9
c) 7
d) 4

Answer: c
Explanation: The naive method prints the maximum sub-array sum, which is 7.

8. What is the time complexity of the divide and conquer algorithm used to find the maximum sub-array sum?
a) O(n)
b) O(logn)
c) O(nlogn)
d) $O(n^2)$

Answer: c
Explanation: The time complexity of the divide and conquer algorithm used to find the maximum sub-array sum is O(nlogn).

9. What is the space complexity of the divide and conquer algorithm used to find the maximum sub-array sum?
a) O(n)
b) O(1)
c) O(n!)

d) $O(n^2)$

Answer: b
Explanation: The divide and conquer algorithm uses a constant space. So, the space complexity is O(1).

1. Which line should be inserted in the blank to complete the following dynamic programming implementation of the maximum sub-array sum problem?

```
#include<stdio.h>
int max_num(int a,int b)
{
      if(a> b)
         return a;
      return b;
}
int maximum_subarray_sum(int *arr, int len)
{
      int sum[len], idx;
      sum[0] = arr[0];
      for(idx = 1; idx < len; idx++)
         sum[idx] = _____________________;
      int mx = sum[0];
      for(idx = 0; idx < len; idx++)
         if(sum[idx] > mx)
             mx =sum[idx];
         return mx;
}
int main()
{
      int arr[] = {-2, -5, 6, -2, 3, -1, 0,-5, 6}, len = 9;
      int ans = maximum_subarray_sum(arr, len);
      printf("%d",ans);
      return 0;
}
```

a) max_num(sum[idx – 1] + arr[idx], arr[idx])
b) sum[idx – 1] + arr[idx].
c) min_num(sum[idx – 1] + arr[idx], arr[idx])
d) arr[idx].

Answer: a
Explanation: The array "sum" is used to store the maximum sub-array sum. The appropriate way to do this is by using: sum[idx] = max_num(sum[idx – 1] + arr[idx], arr[idx]).

2. What is the time complexity of the following dynamic programming algorithm used to find the maximum sub-array sum?

```
#include<stdio.h>
int max_num(int a,int b)
{
      if(a> b)
         return a;
      return b;
}
int maximum_subarray_sum(int *arr, int len)
{
      int sum[len], idx;
      sum[0] = arr[0];
      for(idx = 1; idx < len; idx++)
         sum[idx] = max_num(sum[idx - 1] + arr[idx], arr[idx]);
      int mx = sum[0];
      for(idx = 0; idx < len; idx++)
         if(sum[idx] > mx)
             mx =sum[idx];
         return mx;
}
```

```
int main()
{
      int arr[] = {-2, -5, 6, -2, 3, -1, 0,-5, 6}, len = 9;
      int ans = maximum_subarray_sum(arr, len);
      printf("%d",ans);
      return 0;
}
```

a) O(n)
b) O(logn)
c) O(nlogn)
d) O(n^2)

Answer: a
Explanation: The time complexity of the above dynamic programming algorithm used to solve maximum sub-array sum is O(n).

3. What is the space complexity of the following dynamic programming algorithm used to find the maximum sub-array sum?

```
#include<stdio.h>
int max_num(int a,int b)
{
      if(a> b)
         return a;
      return b;
}
int maximum_subarray_sum(int *arr, int len)
{
      int sum[len], idx;
      sum[0] = arr[0];
      for(idx = 1; idx < len; idx++)
         sum[idx] = max_num(sum[idx - 1] + arr[idx], arr[idx]);
      int mx = sum[0];
      for(idx = 0; idx < len; idx++)
         if(sum[idx] > mx)
             mx =sum[idx];
         return mx;
}
int main()
{
      int arr[] = {-2, -5, 6, -2, 3, -1, 0,-5, 6}, len = 9;
      int ans = maximum_subarray_sum(arr, len);
      printf("%d",ans);
      return 0;
}
```

a) O(n)
b) O(1)
c) O(n!)
d) O(n^2)

Answer: a
Explanation: The above dynamic programming algorithm uses space equal to the length of the array to store the sum values. So, the space complexity is O(n).

4. Consider the following code snippet. Which property is shown by line 4 of the below code snippet?

```
1. int sum[len], idx;
2. sum[0] = arr[0];
3. for(idx = 1; idx < len; idx++)
4.         sum[idx] = max(sum[idx - 1] + arr[idx], arr[idx]);
5. int mx = sum[0];
6. for(idx = 0; idx < len; idx++)
7.       if(sum[idx] > mx)
```

```
8.              mx =sum[idx];
9. return mx;
```

a) Optimal substructure
b) Overlapping subproblems
c) Both overlapping subproblems and optimal substructure
d) Greedy substructure

Answer: a
Explanation: The current sum (i.e. sum[idx]) uses the previous sum (i.e. sum[idx – 1]) to get an optimal value. So, line 4 shows the optimal substructure property.

5. Consider the following code snippet:

```
1. int sum[len], idx;
2. sum[0] = arr[0];
3. for(idx = 1; idx < len; idx++)
4.        sum[idx] = max(sum[idx - 1] + arr[idx], arr[idx]);
5. int mx = sum[0];
6. for(idx = 0; idx < len; idx++)
7.       if(sum[idx] > mx)
8.              mx =sum[idx];
9. return mx;
```

Which method is used by line 4 of the above code snippet?
a) Divide and conquer
b) Recursion
c) Both memoization and divide and conquer
d) Memoization

Answer: d
Explanation: The array "sum" is used to store the previously calculated values, so that they aren't recalculated. So, line 4 uses the memoization technique.Answer: b
Explanation: All the elements of the array are positive. So, the maximum sub-array sum is equal to the sum of all the elements, which is 36.

7. What is the output of the following program?

```
#include<stdio.h>
int max_num(int a,int b)
{
     if(a> b)
        return a;
     return b;
}
int maximum_subarray_sum(int *arr, int len)
{
     int sum[len], idx;
     sum[0] = arr[0];
     for(idx = 1; idx < len; idx++)
        sum[idx] = max_num(sum[idx - 1] + arr[idx], arr[idx]);
     int mx = sum[0];
     for(idx = 0; idx < len; idx++)
        if(sum[idx] > mx)
           mx =sum[idx];
     return mx;
}
int main()
{
     int arr[] = {-20, 23, 10, 3, -10, 11, -5},len = 7;
     int ans = maximum_subarray_sum(arr, len);
     printf("%d",ans);
     return 0;
}
```

a) 27
b) 37
c) 36
d) 26

Answer: b
Explanation: The program prints the value of maximum sub-array sum, which is 37.

8. What is the value stored in sum[4] after the following program is executed?

```
#include<stdio.h>
int max_num(int a,int b)
{
      if(a> b)
          return a;
      return b;
}
int maximum_subarray_sum(int *arr, int len)
{
      int sum[len], idx;
      sum[0] = arr[0];
      for(idx = 1; idx < len; idx++)
           sum[idx] = max_num(sum[idx - 1] + arr[idx], arr[idx]);
      int mx = sum[0];
      for(idx = 0; idx < len; idx++)
          if(sum[idx] > mx)
              mx =sum[idx];
      return mx;
}
int main()
{
      int arr[] = {-2, 14, 11, -13, 10, -5, 11, -6, 3, -5},len = 10;
      int ans = maximum_subarray_sum(arr, len);
      printf("%d",ans);
      return 0;
}
```

a) 28
b) 25
c) 22
d) 12

Answer: c
Explanation: After the program is executed the value stored in sum[4] is 22.
Note: You are asked to find the value stored in sum[4] and NOT the output of the program.

1. Kadane's algorithm is used to find _____
a) Longest increasing subsequence
b) Longest palindrome subsequence
c) Maximum sub-array sum
d) Longest decreasing subsequence

Answer: c
Explanation: Kadane's algorithm is used to find the maximum sub-array sum for a given array.

2. Kadane's algorithm uses which of the following techniques?
a) Divide and conquer
b) Dynamic programming
c) Recursion
d) Greedy algorithm

Answer: b
Explanation: Kadane's algorithm uses dynamic programming.

3. For which of the following inputs would Kadane's algorithm produce the INCORRECT output?
a) {0,1,2,3}
b) {-1,0,1}
c) {-1,-2,-3,0}
d) {-4,-3,-2,-1}

Answer: d
Explanation: Kadane's algorithm works if the input array contains at least one non-negative element. Every element in the array {-4,-3,-2,-1} is negative. Hence Kadane's algorithm won't work.

4. For which of the following inputs would Kadane's algorithm produce a WRONG output?
a) {1,0,-1}
b) {-1,-2,-3}
c) {1,2,3}
d) {0,0,0}

Answer: b
Explanation: Kadane's algorithm doesn't work for all negative numbers. So, the answer is {-1,-2,-3}.

5. Complete the following code for Kadane's algorithm:

```
#include<stdio.h>
int max_num(int a, int b)
{
      if(a > b)
         return a;
      return b;
}
int kadane_algo(int *arr, int len)
{
     int ans, sum, idx;
     ans =0;
     sum =0;
     for(idx =0; idx < len; idx++)
     {
         sum = max_num(0,sum + arr[idx]);
         ans = ___________;
     }
     return ans;
}
int main()
{
     int arr[] = {-2, -3, 4, -1, -2, 1, 5, -3},len=7;
     int ans = kadane_algo(arr,len);
     printf("%d",ans);
     return 0;
}
```

a) max_num(sum, sum + arr[idx])
b) sum
c) sum + arr[idx]
d) max_num(sum,ans)

Answer: d
Explanation: The maximum of sum and ans, is stored in ans. So, the answer is max_num(sum, ans).

6. What is the time complexity of Kadane's algorithm?
a) O(1)
b) O(n)
c) $O(n^2)$
d) O(5)

Answer: b
Explanation: The time complexity of Kadane's algorithm is O(n) because there is only one for loop which scans the entire array exactly once.

7. What is the space complexity of Kadane's algorithm?
a) O(1)
b) O(n)
c) O(n^2)
d) None of the mentioned

Answer: a
Explanation: Kadane's algorithm uses a constant space. So, the space complexity is O(1).

8. What is the output of the following implementation of Kadane's algorithm?

```
#include<stdio.h>
int max_num(int a, int b)
{
       if(a > b)
        return a;
       return b;
}
int kadane_algo(int *arr, int len)
{
       int ans, sum, idx;
       ans =0;
       sum =0;
       for(idx =0; idx < len; idx++)
       {
             sum = max_num(0,sum + arr[idx]);
             ans = max_num(sum,ans);
       }
       return ans;
}
int main()
{
      int arr[] = {2, 3, -3, -1, 2, 1, 5, -3}, len = 8;
      int ans = kadane_algo(arr,len);
      printf("%d",ans);
      return 0;
}
```

a) 6
b) 7
c) 8
d) 9

Answer: d
Explanation: Kadane's algorithm produces the maximum sub-array sum, which is equal to 9.

9. What is the output of the following implementation of Kadane's algorithm?

```
#include<stdio.h>
int max_num(int a, int b)
{
      if(a > b)
         return a;
      return b;
}
int kadane_algo(int *arr, int len)
{
      int ans, sum, idx;
      ans =0;
      sum =0;
      for(idx =0; idx < len; idx++)
```

```
      {
          sum = max_num(0,sum + arr[idx]);
          ans = max_num(sum,ans);
      }
      return ans;
}
int main()
{
      int arr[] = {-2, -3, -3, -1, -2, -1, -5, -3},len = 8;
      int ans = kadane_algo(arr,len);
      printf("%d",ans);
      return 0;
}
```

a) 1
b) -1
c) -2
d) 0

Answer: d
Explanation: Kadane's algorithm produces a wrong output when all the elements are negative. The output produced is 0.

10. Consider the following implementation of Kadane's algorithm:

```
1. #include<stdio.h>
2. int max_num(int a, int b)
3. {
4.     if(a > b)
5.       return a;
6.     return b;
7. }
8. int kadane_algo(int *arr, int len)
9. {
10.       int ans = 0, sum = 0, idx;
11.       for(idx =0; idx < len; idx++)
12.       {
13.             sum = max_num(0,sum + arr[idx]);
14.             ans = max_num(sum,ans);
15.       }
16.       return ans;
17. }
18. int main()
19. {
20.        int arr[] = {-2, -3, -3, -1, -2, -1, -5, -3},len = 8;
21.        int ans = kadane_algo(arr,len);
22.        printf("%d",ans);
23.        return 0;
24. }
```

What changes should be made to the Kadane's algorithm so that it produces the right output even when all array elements are negative?

```
        Change 1 = Line 10: int sum = arr[0], ans = arr[0]
        Change 2 = Line 13: sum = max_num(arr[idx],sum+arr[idx])
```

a) Only Change 1 is sufficient
b) Only Change 2 is sufficient
c) Both Change 1 and Change 2 are necessary
d) No change is required

Answer: c
Explanation: Both change 1 and change 2 should be made to Kadane's algorithm so that it produces the right output even when all the array elements are negative.

1. The longest increasing subsequence problem is a problem to find the length of a subsequence from a sequence of

array elements such that the subsequence is sorted in increasing order and it's length is maximum. This problem can be solved using _____

a) Recursion

b) Dynamic programming

c) Brute force

d) Recursion, Dynamic programming, Brute force

Answer: d

Explanation: The longest increasing subsequence problem can be solved using all of the mentioned methods.Answer: d

Explanation: The longest increasing subsequence is {-10, 9, 10, 13, 14}.Answer: d

Explanation: The longest increasing subsequence is {-10, 24, 35, 55, 76, 84} and it's length is 6.Answer: b

Explanation: For a given sequence, it is possible that there is more than one subsequence with the longest length. Consider, the following sequence: {10,11,12,1,2,3}:

There are two longest increasing subsequences: {1,2,3} and {10,11,12}.Answer: d

Explanation: Each array element individually forms a longest increasing subsequence and so, the length of the longest increasing subsequence is 1. So, the number of increasing subsequences with the longest length is 6.

6. In the brute force implementation to find the longest increasing subsequence, all the subsequences of a given sequence are found. All the increasing subsequences are then selected and the length of the longest subsequence is found. What is the time complexity of this brute force implementation?

a) O(n)

b) O(n^2)

c) O(n!)

d) O(2^n)

Answer: d

Explanation: The time required to find all the subsequences of a given sequence is 2^n, where 'n' is the number of elements in the sequence. So, the time complexity is O(2^n).

7. Complete the following dynamic programming implementation of the longest increasing subsequence problem:

```
#include<stdio.h>
int longest_inc_sub(int *arr, int len)
{
      int i, j, tmp_max;
      int LIS[len];  // array to store the lengths of the longest increasing subsequence
      LIS[0]=1;
      for(i = 1; i < len; i++)
      {
           tmp_max = 0;
           for(j = 0; j < i; j++)
           {
                if(arr[j] < arr[i])
                {
                    if(LIS[j] > tmp_max)
                     ___________;
                }
           }
           LIS[i] = tmp_max + 1;
      }
      int max = LIS[0];
      for(i = 0; i < len; i++)
        if(LIS[i] > max)
           max = LIS[i];
      return max;
}
int main()
{
      int arr[] = {10,22,9,33,21,50,41,60,80}, len = 9;
      int ans = longest_inc_sub(arr, len);
      printf("%d",ans);
      return 0;
```

```
}
```

a) tmp_max = LIS[j]
b) LIS[i] = LIS[j]
c) LIS[j] = tmp_max
d) tmp_max = LIS[i]

Answer: a
Explanation: tmp_max is used to store the maximum length of an increasing subsequence for any 'j' such that: arr[j] < arr[i] and 0 < j < i.
So, tmp_max = LIS[j] completes the code.

8. What is the time complexity of the following dynamic programming implementation used to find the length of the longest increasing subsequence?

```
#include<stdio.h>
int longest_inc_sub(int *arr, int len)
{
      int i, j, tmp_max;
      int LIS[len];  // array to store the lengths of the longest increasing subsequence
      LIS[0]=1;
      for(i = 1; i < len; i++)
      {
           tmp_max = 0;
           for(j = 0; j < i; j++)
           {
                if(arr[j] < arr[i])
                {
                    if(LIS[j] > tmp_max)
                     tmp_max = LIS[j];
                }
           }
           LIS[i] = tmp_max + 1;
      }
      int max = LIS[0];
      for(i = 0; i < len; i++)
        if(LIS[i] > max)
           max = LIS[i];
      return max;
}
int main()
{
      int arr[] = {10,22,9,33,21,50,41,60,80}, len = 9;
      int ans = longest_inc_sub(arr, len);
      printf("%d",ans);
      return 0;
}
```

a) O(1)
b) O(n)
c) $O(n^2)$
d) O(nlogn)

Answer: c
Explanation: The time complexity of the above dynamic programming implementation used to find the length of the longest increasing subsequence is $O(n^2)$.

9. What is the space complexity of the following dynamic programming implementation used to find the length of the longest increasing subsequence?

```
#include<stdio.h>
int longest_inc_sub(int *arr, int len)
{
      int i, j, tmp_max;
      int LIS[len];  // array to store the lengths of the longest increasing subsequence
```

```
      LIS[0]=1;
      for(i = 1; i < len; i++)
      {
           tmp_max = 0;
           for(j = 0; j < i; j++)
           {
                if(arr[j] < arr[i])
                {
                    if(LIS[j] > tmp_max)
                     tmp_max = LIS[j];
                }
           }
           LIS[i] = tmp_max + 1;
      }
      int max = LIS[0];
      for(i = 0; i < len; i++)
        if(LIS[i] > max)
           max = LIS[i];
      return max;
}
int main()
{
      int arr[] = {10,22,9,33,21,50,41,60,80}, len = 9;
      int ans = longest_inc_sub(arr, len);
      printf("%d",ans);
      return 0;
}
```

a) O(1)
b) O(n)
c) O(n^2)
d) O(nlogn)

Answer: b
Explanation: The above dynamic programming implementation uses space equal to the length of the sequence. So, the space complexity of the above dynamic programming implementation used to find the length of the longest increasing subsequence is O(n).

10. What is the output of the following program?

```
#include<stdio.h>
int longest_inc_sub(int *arr, int len)
{
      int i, j, tmp_max;
      int LIS[len];  // array to store the lengths of the longest increasing subsequence
      LIS[0]=1;
      for(i = 1; i < len; i++)
      {
            tmp_max = 0;
            for(j = 0; j < i; j++)
            {
                if(arr[j] < arr[i])
                {
                     if(LIS[j] > tmp_max)
                       tmp_max = LIS[j];
                }
            }
            LIS[i] = tmp_max + 1;
      }
      int max = LIS[0];
      for(i = 0; i < len; i++)
          if(LIS[i] > max)
              max = LIS[i];
      return max;
}
int main()
{
```

```
      int arr[] = {10,22,9,33,21,50,41,60,80}, len = 9;
      int ans = longest_inc_sub(arr, len);
      printf("%d",ans);
      return 0;
}
```

a) 3
b) 4
c) 5
d) 6

Answer: d
Explanation: The program prints the length of the longest increasing subsequence, which is 6.

11. What is the value stored in LIS[5] after the following program is executed?

```
#include<stdio.h>
int longest_inc_sub(int *arr, int len)
{
      int i, j, tmp_max;
      int LIS[len];  // array to store the lengths of the longest increasing subsequence
      LIS[0]=1;
      for(i = 1; i < len; i++)
      {
           tmp_max = 0;
           for(j = 0; j < i; j++)
           {
                if(arr[j] < arr[i])
                {
                     if(LIS[j] > tmp_max)
                     tmp_max = LIS[j];
                }
           }
           LIS[i] = tmp_max + 1;
       }
      int max = LIS[0];
      for(i = 0; i < len; i++)
         if(LIS[i] > max)
             max = LIS[i];
      return max;
}
int main()
{
      int arr[] = {10,22,9,33,21,50,41,60,80}, len = 9;
      int ans = longest_inc_sub(arr, len);
      printf("%d",ans);
      return 0;
}
```

a) 2
b) 3
c) 4
d) 5

Answer: c
Explanation: The value stored in LIS[5] after the program is executed is 4.

1. Given a rod of length n and the selling prices of all pieces smaller than equal to n, find the most beneficial way of cutting the rod into smaller pieces. This problem is called the rod cutting problem. Which of these methods can be used to solve the rod cutting problem?
a) Brute force
b) Dynamic programming
c) Recursion
d) Brute force, Dynamic programming and Recursion

Answer: d
Explanation: Brute force, Dynamic programming and Recursion can be used to solve the rod cutting problem.

What is the maximum value that you can get after cutting the rod and selling the pieces?
a) 10
b) 11
c) 12
d) 13

Answer: c
Explanation: The pieces {1,2 2} give the maximum value of 12.

3. Consider the brute force implementation of the rod cutting problem in which all the possible cuts are found and the maximum value is calculated. What is the time complexity of this brute force implementation?
a) $O(n^2)$
b) $O(n^3)$
c) O(nlogn)
d) $O(2^n)$

Answer: d
Explanation: The brute force implementation finds all the possible cuts. This takes $O(2^n)$ time.

Which of these pieces give the maximum price?
a) {1,2,7}
b) {10}
c) {2,2,6}
d) {1,4,5}

Answer: c
Explanation: The pieces {2,2,6} give the maximum value of 27.

5. Consider the following recursive implementation of the rod cutting problem:

```
#include<stdio.h>
#include<limits.h>
int max_of_two(int a, int b)
{
      if(a > b)
            return a;
      return b;
}
int rod_cut(int *prices, int len)
{
      int max_price = INT_MIN; // INT_MIN is the min value an integer can take
      int i;
      if(len <= 0 )
        return 0;
      for(i = 0; i < len; i++)
         max_price = max_of_two(_____________); // subtract 1 because index starts from 0
      return max_price;
}
int main()
{
      int prices[]={2, 5, 6, 9, 9, 17, 17, 18, 20, 22},len_of_rod = 10;
      int ans = rod_cut(prices, len_of_rod);
      printf("%d",ans);
      return 0;
}
```

Complete the above code.
a) max_price, prices[i] + rod_cut(prices,len – i – 1)
b) max_price, prices[i – 1].

c) max_price, rod_cut(prices, len – i – 1)
d) max_price, prices[i – 1] + rod_cut(prices,len – i – 1)

Answer: a
Explanation: max_price, prices[i] + rod_cut(prices,len – i – 1) completes the above code.

6. What is the time complexity of the following recursive implementation?

```
#include<stdio.h>
#include<limits.h>
int max_of_two(int a, int b)
{
      if(a > b)
            return a;
      return b;
}
int rod_cut(int *prices, int len)
{
      int max_price = INT_MIN; // INT_MIN is the min value an integer can take
      int i;
      if(len <= 0 )
        return 0;
      for(i = 0; i < len; i++)
         // subtract 1 because index starts from 0
         max_price = max_of_two(max_price, prices[i] + rod_cut(prices,len - i - 1));
      return max_price;
}
int main()
{
      int prices[]={2, 5, 6, 9, 9, 17, 17, 18, 20, 22},len_of_rod = 10;
      int ans = rod_cut(prices, len_of_rod);
      printf("%d",ans);
      return 0;
}
```

a) O(n)
b) $O(n^2)$
c) $O(n^3)$
d) $O(2^n)$

Answer: d
Explanation: The time complexity of the above recursive implementation is $O(2^n)$.

7. What is the space complexity of the following recursive implementation?

```
#include<stdio.h>
#include<limits.h>
int max_of_two(int a, int b)
{
      if(a > b)
            return a;
      return b;
}
int rod_cut(int *prices, int len)
{
      int max_price = INT_MIN; // INT_MIN is the min value an integer can take
      int i;
      if(len <= 0 )
        return 0;
      for(i = 0; i < len; i++)
         // subtract 1 because index starts from 0
         max_price = max_of_two(max_price, prices[i] + rod_cut(prices,len - i - 1));
      return max_price;
}
int main()
{
```

```
      int prices[]={2, 5, 6, 9, 9, 17, 17, 18, 20, 22},len_of_rod = 10;
      int ans = rod_cut(prices, len_of_rod);
      printf("%d",ans);
      return 0;
}
```

a) O(1)
b) O(logn)
c) O(nlogn)
d) O(n!)

Answer: a
Explanation: The space complexity of the above recursive implementation is O(1) because it uses a constant space.

8. What will be the value stored in max_value when the following code is executed?

```
#include<stdio.h>
#include<limits.h>
int max_of_two(int a, int b)
{
       if(a > b)
         return a;
       return b;
}
int rod_cut(int *prices, int len)
{
       int max_price = INT_MIN; // INT_MIN is the min value an integer can take
       int i;
       if(len <= 0 )
          return 0;
       for(i = 0; i < len; i++)
          // subtract 1 because index starts from 0
          max_price = max_of_two(prices[i] + rod_cut(prices,len - i - 1), max_price);
       return max_price;
}
int main()
{
       int prices[]={2, 5, 6, 9, 9, 17, 17, 18, 20, 22},len_of_rod = 3;
       int ans = rod_cut(prices, len_of_rod);
       printf("%d",ans);
       return 0;
}
```

a) 5
b) 6
c) 7
d) 8

Answer: c
Explanation: The value stored in max_value after the code is executed is equal to 7.Answer: b
Explanation: Consider a rod of length 3. The prices are {2,3,6} for lengths {1,2,3} respectively. The pieces {1,1,1} and {3} both give the maximum value of 6.

10.Consider the following dynamic programming implementation of the rod cutting problem:

```
#include<stdio.h>
#include<limits.h>
int rod_cut(int *prices, int len)
{
      int max_val[len + 1];
      int i,j,tmp_price,tmp_idx;
      max_val[0] = 0;
      for(i = 1; i <= len; i++)
      {
           int tmp_max = INT_MIN; // minimum value an integer can hold
           for(j = 1; j <= i; j++)
```

```
            {
                  tmp_idx = i - j;
                  //subtract 1 because index of prices starts from 0
                  tmp_price = _____________;
                  if(tmp_price > tmp_max)
                    tmp_max = tmp_price;
             }
             max_val[i] = tmp_max;
       }
       return max_val[len];
}
int main()
{
       int prices[]={2, 5, 6, 9, 9, 17, 17, 18, 20, 22},len_of_rod = 5;
       int ans = rod_cut(prices, len_of_rod);
       printf("%d",ans);
       return 0;
}
```

Which line will complete the ABOVE code?
a) prices[j-1] + max_val[tmp_idx]
b) prices[j] + max_val[tmp_idx]
c) prices[j-1] + max_val[tmp_idx – 1]
d) prices[j] + max_val[tmp_idx – 1]

Answer: a
Explanation: prices[j-1] + max_val[tmp_idx] completes the code.

11. What is the time complexity of the following dynamic programming implementation of the rod cutting problem?

```
#include<stdio.h>
#include<limits.h>
int rod_cut(int *prices, int len)
{
      int max_val[len + 1];
      int i,j,tmp_price,tmp_idx;
      max_val[0] = 0;
      for(i = 1; i <= len; i++)
      {
            int tmp_max = INT_MIN; // minimum value an integer can hold
            for(j = 1; j <= i; j++)
            {
                  tmp_idx = i - j;
                  //subtract 1 because index of prices starts from 0
                  tmp_price = prices[j-1] + max_val[tmp_idx];
                  if(tmp_price > tmp_max)
                    tmp_max = tmp_price;
             }
             max_val[i] = tmp_max;
       }
       return max_val[len];
}
int main()
{
       int prices[]={2, 5, 6, 9, 9, 17, 17, 18, 20, 22},len_of_rod = 5;
       int ans = rod_cut(prices, len_of_rod);
       printf("%d",ans);
       return 0;
}
```

a) O(1)
b) O(n)
c) $O(n^2)$
d) $O(2^n)$

Answer: c

Explanation: The time complexity of the above dynamic programming implementation of the rod cutting problem is $O(n^2)$.

12. What is the space complexity of the following dynamic programming implementation of the rod cutting problem?

```
#include<stdio.h>
#include<limits.h>
int rod_cut(int *prices, int len)
{
      int max_val[len + 1];
      int i,j,tmp_price,tmp_idx;
      max_val[0] = 0;
      for(i = 1; i <= len; i++)
      {
           int tmp_max = INT_MIN; // minimum value an integer can hold
           for(j = 1; j <= i; j++)
           {
                 tmp_idx = i - j;
                 //subtract 1 because index of prices starts from 0
                 tmp_price = prices[j-1] + max_val[tmp_idx];
                 if(tmp_price > tmp_max)
                   tmp_max = tmp_price;
            }
            max_val[i] = tmp_max;
       }
       return max_val[len];
}
int main()
{
       int prices[]={2, 5, 6, 9, 9, 17, 17, 18, 20, 22},len_of_rod = 5;
       int ans = rod_cut(prices, len_of_rod);
       printf("%d",ans);
       return 0;
}
```

a) O(1)
b) O(n)
c) $O(n^2)$
d) $O(2^n)$

Answer: b
Explanation: The space complexity of the above dynamic programming implementation of the rod cutting problem is O(n) because it uses a space equal to the length of the rod.

13. What is the output of the following program?

```
#include<stdio.h>
#include<limits.h>
int rod_cut(int *prices, int len)
{
      int max_val[len + 1];
      int i,j,tmp_price,tmp_idx;
      max_val[0] = 0;
      for(i = 1; i <= len; i++)
      {
           int tmp_max = INT_MIN; // minimum value an integer can hold
           for(j = 1; j <= i; j++)
           {
                tmp_idx = i - j;
                //subtract 1 because index of prices starts from 0
                tmp_price = prices[j-1] + max_val[tmp_idx];
                if(tmp_price > tmp_max)
                  tmp_max = tmp_price;
           }
           max_val[i] = tmp_max;
      }
```

```
      return max_val[len];
}
int main()
{
      int prices[]={2, 5, 6, 9, 9, 17, 17, 18, 20, 22},len_of_rod = 5;
      int ans = rod_cut(prices, len_of_rod);
      printf("%d",ans);
      return 0;
}
```

a) 9
b) 10
c) 11
d) 12

Answer: d
Explanation: The program prints the maximum price that can be achieved by cutting the rod into pieces, which is equal to 27.

14. What is the value stored in max_val[5] after the following program is executed?

```
#include<stdio.h>
#include<limits.h>
int rod_cut(int *prices, int len)
{
      int max_val[len + 1];
      int i,j,tmp_price,tmp_idx;
      max_val[0] = 0;
      for(i = 1; i <= len; i++)
      {
            int tmp_max = INT_MIN; // minimum value an integer can hold
            for(j = 1; j <= i; j++)
            {
                  tmp_idx = i - j;
                 //subtract 1 because index of prices starts from 0
                  tmp_price = prices[j-1] + max_val[tmp_idx];
                  if(tmp_price > tmp_max)
                     tmp_max = tmp_price;
            }
            max_val[i] = tmp_max;
     }
     return max_val[len];
}
int main()
{
      int prices[]={2, 5, 6, 9, 9, 17, 17, 18, 20, 22},len_of_rod = 5;
      int ans = rod_cut(prices, len_of_rod);
      printf("%d",ans);
      return 0;
}
```

a) 12
b) 27
c) 10
d) 17

Answer: a
Explanation: The value stored in max_val[5] after the program is executed is 12.

1. You are given an array of elements where each array element represents the MAXIMUM number of jumps that can be made in the forward direction from that element. You have to find the minimum number of jumps that are required to reach the end of the array. Which of these methods can be used to solve the problem?
a) Dynamic Programming
b) Greedy Algorithm
c) Recursion

d) Recursion and Dynamic Programming

Answer: d
Explanation: Both recursion and dynamic programming can be used to solve minimum number of jumps problem.Answer: c
Explanation: The jumps made will be:{1 -> 2 -> 4 -> 9}. So, the number of jumps is three.

3. Consider the following recursive implementation:

```
#include<stdio.h>
#include<limits.h>
int min_jumps(int *arr, int strt, int end)
{
     int idx;
     if(strt == end)
        return 0;
     if(arr[strt] == 0) // jump cannot be made
        return INT_MAX;
     int min = INT_MAX;
     for(idx = 1; idx <= arr[strt] && strt + idx <= end; idx++)
     {
          int jumps = min_jumps(____,____,____) + 1;
          if(jumps < min)
              min  = jumps;
     }
     return min;
}
int main()
{
     int arr[] ={1, 3, 5, 8, 9, 2, 6, 7, 6},len = 9;
     int ans = min_jumps(arr, 0, len-1);
     printf("%d\n",ans);
     return 0;
}
```

Which of these arguments should be passed by the min_jumps function represented by the blanks?
a) arr, strt + idx, end
b) arr + idx, strt, end
c) arr, strt, end
d) arr, strt, end + idx

Answer: a
Explanation: arr, strt + idx, end should be passed as arguments.Answer: a
Explanation: Consider the array {1,2,3,4,5}. It is possible to reach the end in the following ways: {1 -> 2 -> 3 -> 5} or {1 -> 2 -> 4 -> 5}.
In both the cases the number of jumps is 3, which is minimum. Hence, it is possible to reach the end of the array in multiple ways using minimum number of jumps.

5. What is the output of the following program?

```
#include<stdio.h>
#include<limits.h>
int min_jumps(int *arr, int strt, int end)
{
      int idx;
      if(strt == end)
         return 0;
      if(arr[strt] == 0) // jump cannot be made
         return INT_MAX;
      int min = INT_MAX;
      for(idx = 1; idx <= arr[strt] && strt + idx <= end; idx++)
      {
           int jumps = min_jumps(arr, strt + idx, end) + 1;
           if(jumps < min)
             min  = jumps;
```

```
      }
      return min;
}
int main()
{
      int arr[] ={1, 2, 3, 4, 5, 4, 3, 2, 1},len = 9;
      int ans = min_jumps(arr, 0, len-1);
      printf("%d\n",ans);
      return 0;
}
```

a) 4
b) 5
c) 6
d) 7

Answer: a
Explanation: The program prints the minimum number of jumps required to reach the end of the array. One way reach the end using minimum number of jumps is
{1 -> 2 -> 4 -> 8 -> 9}. So, the number of jumps is 4.Answer: b
Explanation: Consider the array {1,0,2,3,4}.
In this case, only one element is 0 but it is not possible to reach the end of the array.

7. Consider the following dynamic programming implementation of the minimum jumps problem:

```
#include<stdio.h>
#include<limits.h>
int min_jump(int *arr, int len)
{
     int j, idx, jumps[len];
     jumps[len - 1] = 0;
     for(idx = len - 2; idx >= 0; idx--)
     {
          int tmp_min = INT_MAX;
          for(j = 1; j <= arr[idx] && idx + j < len; j++)
          {
                 if(jumps[idx + j] + 1 < tmp_min)
                     tmp_min = jumps[idx + j] + 1;
          }
          jumps[idx] = tmp_min;
     }
     return jumps[0];
}
int main()
{
      int arr[] ={1, 1, 1, 1, 1, 1, 1, 1, 1},len = 9;
      int ans = min_jump(arr,len);
      printf("%d\n",ans);
      return 0;
}
```

Which of the following "for" loops can be used instead of the inner for loop so that the output doesn't change?
a) for(j = 1; j < arr[idx] + len; j++)
b) for(j = 0; j < arr[idx] – len; j++)
c) for(j = idx + 1; j < len && j <= arr[idx] + idx; j++)
d) No change is required

Answer: d
Explanation: None of the above mentioned "for" loops can be used instead of the inner for loop. Note, for(j = idx + 1; j < len && j <= arr[idx] + idx; j++) covers the same range as the inner for loop but it produces the wrong output because the indexing inside the loops changes as "j" takes different values in the two "for" loops.

8. What is the time complexity of the following dynamic programming implementation used to find the minimum number of jumps?

```
#include<stdio.h>
#include<limits.h>
int min_jump(int *arr, int len)
{
     int j, idx, jumps[len];
     jumps[len - 1] = 0;
     for(idx = len - 2; idx >= 0; idx--)
     {
          int tmp_min = INT_MAX;
          for(j = 1; j <= arr[idx] && idx + j < len; j++)
          {
                 if(jumps[idx + j] + 1 < tmp_min)
                     tmp_min = jumps[idx + j] + 1;
          }
          jumps[idx] = tmp_min;
     }
     return jumps[0];
}
int main()
{
      int arr[] ={1, 1, 1, 1, 1, 1, 1, 1, 1},len = 9;
      int ans = min_jump(arr,len);
      printf("%d\n",ans);
      return 0;
}
```

a) O(1)
b) O(n)
c) O(n^2)
d) None of the mentioned

Answer: c
Explanation: The time complexity of the above dynamic programming implementation is O(n^2).

9. What is the space complexity of the following dynamic programming implementation used to find the minimum number of jumps?

```
#include<stdio.h>
#include<limits.h>
int min_jump(int *arr, int len)
{
     int j, idx, jumps[len];
     jumps[len - 1] = 0;
     for(idx = len - 2; idx >= 0; idx--)
     {
          int tmp_min = INT_MAX;
          for(j = 1; j <= arr[idx] && idx + j < len; j++)
          {
                 if(jumps[idx + j] + 1 < tmp_min)
                     tmp_min = jumps[idx + j] + 1;
          }
          jumps[idx] = tmp_min;
     }
     return jumps[0];
}
int main()
{
      int arr[] ={1, 1, 1, 1, 1, 1, 1, 1, 1},len = 9;
      int ans = min_jump(arr,len);
      printf("%d\n",ans);
      return 0;
}
```

a) O(1)
b) O(n)
c) O(n^2)

d) O(5)

Answer: b
Explanation: The space complexity of the above dynamic programming implementation is O(n).

10. What is the output of the following program?

```
#include<stdio.h>
#include<limits.h>
int min_jump(int *arr, int len)
{
      int j, idx, jumps[len];
      jumps[len - 1] = 0;
      for(idx = len - 2; idx >= 0; idx--)
      {
            int tmp_min = INT_MAX;
            for(j = 1; j <= arr[idx] && idx + j < len; j++)
            {
                  if(jumps[idx + j] + 1 < tmp_min)
                     tmp_min = jumps[idx + j] + 1;
            }
            jumps[idx] = tmp_min;
      }
      return jumps[0];
}
int main()
{
      int arr[] ={1, 1, 1, 1, 1, 1, 1, 1, 1},len = 9;
      int ans = min_jump(arr,len);
      printf("%d\n",ans);
      return 0;
}
```

a) 7
b) 8
c) 9
d) 10

Answer: b
Explanation: The program prints the minimum jumps required to reach the end of the array, which is 8 and so the output is 8.

11. What is the output of the following program?

```
#include<stdio.h>
#include<limits.h>
int min_jump(int *arr, int len)
{
      int j, idx, jumps[len];
      jumps[len - 1] = 0;
      for(idx = len - 2; idx >= 0; idx--)
      {
          int tmp_min = INT_MAX;
          for(j = 1; j <= arr[idx] && idx + j < len; j++)
          {
                if(jumps[idx + j] + 1 < tmp_min)
                  tmp_min = jumps[idx + j] + 1;
          }
          jumps[idx] = tmp_min;
      }
      return jumps[0];
}
int main()
{
      int arr[] ={9, 9, 9, 9, 9, 9, 9, 9, 9},len = 9;
      int ans = min_jump(arr,len);
      printf("%d\n",ans);
```

```
    return 0;
}
```

a) 1
b) 6
c) 2
d) 7

Answer: a
Explanation: The program prints the minimum jumps required to reach the end of the array, which is 1 and so the output is 1.

1. The Knapsack problem is an example of _____
a) Greedy algorithm
b) 2D dynamic programming
c) 1D dynamic programming
d) Divide and conquer

Answer: b
Explanation: Knapsack problem is an example of 2D dynamic programming.

2. Which of the following methods can be used to solve the Knapsack problem?
a) Brute force algorithm
b) Recursion
c) Dynamic programming
d) Brute force, Recursion and Dynamic Programming

Answer: d
Explanation: Brute force, Recursion and Dynamic Programming can be used to solve the knapsack problem.

3. You are given a knapsack that can carry a maximum weight of 60. There are 4 items with weights {20, 30, 40, 70} and values {70, 80, 90, 200}. What is the maximum value of the items you can carry using the knapsack?
a) 160
b) 200
c) 170
d) 90

Answer: a
Explanation: The maximum value you can get is 160. This can be achieved by choosing the items 1 and 3 that have a total weight of 60.

4. Which of the following problems is equivalent to the 0-1 Knapsack problem?
a) You are given a bag that can carry a maximum weight of W. You are given N items which have a weight of {w1, w2, w3,…., wn} and a value of {v1, v2, v3,…., vn}. You can break the items into smaller pieces. Choose the items in such a way that you get the maximum value
b) You are studying for an exam and you have to study N questions. The questions take {t1, t2, t3,…., tn} time(in hours) and carry {m1, m2, m3,…., mn} marks. You can study for a maximum of T hours. You can either study a question or leave it. Choose the questions in such a way that your score is maximized
c) You are given infinite coins of denominations {v1, v2, v3,….., vn} and a sum S. You have to find the minimum number of coins required to get the sum S
d) You are given a suitcase that can carry a maximum weight of 15kg. You are given 4 items which have a weight of {10, 20, 15,40} and a value of {1, 2, 3,4}. You can break the items into smaller pieces. Choose the items in such a way that you get the maximum value

Answer: b
Explanation: In this case, questions are used instead of items. Each question has a score which is same as each item having a value. Also, each question takes a time t which is same as each item having a weight w. You have to maximize the score in time T which is same as maximizing the value using a bag of weight W.

5. What is the time complexity of the brute force algorithm used to solve the Knapsack problem?
a) O(n)
b) O(n!)
c) O(2^n)
d) O(n^3)

Answer: c
Explanation: In the brute force algorithm all the subsets of the items are found and the value of each subset is calculated. The subset of items with the maximum value and a weight less than equal to the maximum allowed weight gives the answer. The time taken to calculate all the subsets is O(2^n).Answer: b
Explanation: The Knapsack problem cannot be solved using the greedy algorithm.

7. Consider the following dynamic programming implementation of the Knapsack problem:

```
#include<stdio.h>
int find_max(int a, int b)
{
      if(a > b)
         return a;
      return b;
}
int knapsack(int W, int *wt, int *val,int n)
{
     int ans[n + 1][W + 1];
     int itm,w;
     for(itm = 0; itm <= n; itm++)
         ans[itm][0] = 0;
     for(w = 0;w <= W; w++)
        ans[0][w] = 0;
     for(itm = 1; itm <= n; itm++)
     {
          for(w = 1; w <= W; w++)
          {
               if(wt[itm - 1] <= w)
                  ans[itm][w] = _____________;
               else
                  ans[itm][w] = ans[itm - 1][w];
          }
     }
     return ans[n][W];
}
int main()
{
     int w[] = {10,20,30}, v[] = {60, 100, 120}, W = 50;
     int ans = knapsack(W, w, v, 3);
     printf("%d",ans);
     return 0;
}
```

Which of the following lines completes the above code?
a) find_max(ans[itm – 1][w – wt[itm – 1]] + val[itm – 1], ans[itm – 1][w])
b) find_max(ans[itm – 1][w – wt[itm – 1]], ans[itm – 1][w])
c) ans[itm][w] = ans[itm – 1][w];
d) ans[itm+1][w] = ans[itm – 1][w];

Answer: a
Explanation: find_max(ans[itm – 1][w – wt[itm – 1]] + val[itm – 1], ans[itm – 1][w]) completes the above code.

8. What is the time complexity of the following dynamic programming implementation of the Knapsack problem with n items and a maximum weight of W?

```
#include<stdio.h>
int find_max(int a, int b)
{
```

```
      if(a > b)
         return a;
      return b;
}
int knapsack(int W, int *wt, int *val,int n)
{
     int ans[n + 1][W + 1];
     int itm,w;
     for(itm = 0; itm <= n; itm++)
         ans[itm][0] = 0;
     for(w = 0;w <= W; w++)
        ans[0][w] = 0;
     for(itm = 1; itm <= n; itm++)
     {
          for(w = 1; w <= W; w++)
          {
               if(wt[itm - 1] <= w)
                 ans[itm][w] = find_max(ans[itm - 1][w-wt[itm - 1]]+val[itm - 1], ans[itm - 1][w])
               else
                  ans[itm][w] = ans[itm - 1][w];
          }
     }
     return ans[n][W];
}
int main()
{
     int w[] = {10,20,30}, v[] = {60, 100, 120}, W = 50;
     int ans = knapsack(W, w, v, 3);
     printf("%d",ans);
     return 0;
}
```

a) O(n)
b) O(n + w)
c) O(nW)
d) O(n^2)

Answer: c
Explanation: The time complexity of the above dynamic programming implementation of the Knapsack problem is O(nW).

9. What is the space complexity of the following dynamic programming implementation of the Knapsack problem?

```
#include<stdio.h>
int find_max(int a, int b)
{
      if(a > b)
         return a;
      return b;
}
int knapsack(int W, int *wt, int *val,int n)
{
     int ans[n + 1][W + 1];
     int itm,w;
     for(itm = 0; itm <= n; itm++)
         ans[itm][0] = 0;
     for(w = 0;w <= W; w++)
        ans[0][w] = 0;
     for(itm = 1; itm <= n; itm++)
     {
          for(w = 1; w <= W; w++)
          {
               if(wt[itm - 1] <= w)
                ans[itm][w] = find_max(ans[itm - 1][w - wt[itm - 1]]+val[itm - 1], ans[itm - 1][w]
               else
                ans[itm][w] = ans[itm - 1][w];
          }
```

```
    }
    return ans[n][W];
}
int main()
{
    int w[] = {10,20,30}, v[] = {60, 100, 120}, W = 50;
    int ans = knapsack(W, w, v, 3);
    printf("%d",ans);
    return 0;
}
```

a) O(n)
b) O(n + w)
c) O(nW)
d) O(n^2)

Answer: c
Explanation: The space complexity of the above dynamic programming implementation of the Knapsack problem is O(nW).

10. What is the output of the following code?

```
#include<stdio.h>
int find_max(int a, int b)
{
     if(a > b)
       return a;
     return b;
}
int knapsack(int W, int *wt, int *val,int n)
{
    int ans[n + 1][W + 1];
    int itm,w;
    for(itm = 0; itm <= n; itm++)
        ans[itm][0] = 0;
    for(w = 0;w <= W; w++)
        ans[0][w] = 0;
    for(itm = 1; itm <= n; itm++)
    {
         for(w = 1; w <= W; w++)
         {
              if(wt[itm - 1] <= w)
               ans[itm][w] = find_max(ans[itm - 1][w-wt[itm - 1]]+val[itm - 1], ans[itm - 1][w]);
              else
               ans[itm][w] = ans[itm - 1][w];
         }
    }
    return ans[n][W];
}
int main()
{
    int w[] = {10,20,30}, v[] = {60, 100, 120}, W = 50;
    int ans = knapsack(W, w, v, 3);
    printf("%d",ans);
    return 0;
}
```

a) 120
b) 100
c) 180
d) 220

Answer: d
Explanation: The output of the above code is 220.

1. Which of the following methods can be used to solve the matrix chain multiplication problem?

a) Dynamic programming
b) Brute force
c) Recursion
d) Dynamic Programming, Brute force, Recursion

Answer: d
Explanation: Dynamic Programming, Brute force, Recursion methods can be used to solve the matrix chain multiplication problem.Answer: d
Explanation: The recurrence relation is given by:
dp[i,j] = 0 if i=j
*dp[i,j] = min{dp[i,k] + dp[k+1,j]} + mat[i-1]*mat[k]*mat[j].*

3. Consider the two matrices P and Q which are 10 x 20 and 20 x 30 matrices respectively. What is the number of multiplications required to multiply the two matrices?
a) 10*20
b) 20*30
c) 10*30
d) 10*20*30

Answer: d
*Explanation: The number of multiplications required is 10*20*30.*

4. Consider the matrices P, Q and R which are 10 x 20, 20 x 30 and 30 x 40 matrices respectively. What is the minimum number of multiplications required to multiply the three matrices?
a) 18000
b) 12000
c) 24000
d) 32000

Answer: a
*Explanation: The minimum number of multiplications are 18000. This is the case when the matrices are parenthesized as (P*Q)*R.*

5. Consider the matrices P, Q, R and S which are 20 x 15, 15 x 30, 30 x 5 and 5 x 40 matrices respectively. What is the minimum number of multiplications required to multiply the four matrices?
a) 6050
b) 7500
c) 7750
d) 12000

Answer: c
Explanation: The minimum number of multiplications required is 7750.

6. Consider the brute force implementation in which we find all the possible ways of multiplying the given set of n matrices. What is the time complexity of this implementation?
a) O(n!)
b) $O(n^3)$
c) $O(n^2)$
d) Exponential

Answer: d
Explanation: The time complexity of finding all the possible ways of multiplying a set of n matrices is given by $(n-1)^{th}$ Catalan number which is exponential.

7. Consider the following dynamic programming implementation of the matrix chain problem:

```
#include<stdio.h>
#include<limits.h>
```

```
int mat_chain_multiplication(int *mat, int n)
{
      int arr[n][n];
      int i,k,row,col,len;
      for(i=1;i<n;i++)
          arr[i][i] = 0;
      for(len = 2; len < n; len++)
      {
           for(row = 1; row <= n - len + 1; row++)
           {
               col = row + len - 1;
               arr[row][col] = INT_MAX;
               for(k = row; k <= col - 1; k++)
               {
                    int tmp = _____________________;
                    if(tmp < arr[row][col])
                    arr[row][col] = tmp;
               }
           }
      }
      return arr[1][n - 1];
}
int main()
{
     int mat[6] = {20,5,30,10,40};
     int ans = mat_chain_multiplication(mat,5);
     printf("%d",ans);
     return 0;
}
```

Which of the following lines should be inserted to complete the above code?
a) arr[row][k] – arr[k + 1][col] + mat[row – 1] * mat[k] * mat[col];
b) arr[row][k] + arr[k + 1][col] – mat[row – 1] * mat[k] * mat[col];
c) arr[row][k] + arr[k + 1][col] + mat[row – 1] * mat[k] * mat[col];
d) arr[row][k] – arr[k + 1][col] – mat[row – 1] * mat[k] * mat[col];

Answer: c
*Explanation: The line arr[row][k] + arr[k + 1][col] + mat[row – 1] * mat[k] * mat[col] should be inserted to complete the above code.*

8. What is the time complexity of the following dynamic programming implementation of the matrix chain problem?

```
#include<stdio.h>
#include<limits.h>
int mat_chain_multiplication(int *mat, int n)
{
      int arr[n][n];
      int i,k,row,col,len;
      for(i=1;i<n;i++)
          arr[i][i] = 0;
      for(len = 2; len < n; len++)
      {
           for(row = 1; row <= n - len + 1; row++)
           {
               col = row + len - 1;
               arr[row][col] = INT_MAX;
               for(k = row; k <= col - 1; k++)
               {
                    int tmp = arr[row][k] + arr[k + 1][col] + mat[row - 1] * mat[k] * mat[col];
                    if(tmp < arr[row][col])
                    arr[row][col] = tmp;
               }
           }
      }
      return arr[1][n - 1];
}
int main()
```

```
{
     int mat[6] = {20,5,30,10,40};
     int ans = mat_chain_multiplication(mat,5);
     printf("%d",ans);
     return 0;
}
```

a) O(1)
b) O(n)
c) O(n^2)
d) O(n^3)

Answer: d
Explanation: The time complexity of the above dynamic programming implementation of the matrix chain multiplication is O(n^3).

9. What is the space complexity of the following dynamic programming implementation of the matrix chain problem?

```
#include<stdio.h>
#include<limits.h>
int mat_chain_multiplication(int *mat, int n)
{
      int arr[n][n];
      int i,k,row,col,len;
      for(i=1;i<n;i++)
          arr[i][i] = 0;
      for(len = 2; len < n; len++)
      {
           for(row = 1; row <= n - len + 1; row++)
           {
               col = row + len - 1;
               arr[row][col] = INT_MAX;
               for(k = row; k <= col - 1; k++)
               {
                    int tmp = arr[row][k] + arr[k + 1][col] + mat[row - 1] * mat[k] * mat[col];
                    if(tmp < arr[row][col])
                    arr[row][col] = tmp;
               }
           }
      }
      return arr[1][n - 1];
}
int main()
{
     int mat[6] = {20,5,30,10,40};
     int ans = mat_chain_multiplication(mat,5);
     printf("%d",ans);
     return 0;
}
```

a) O(1)
b) O(n)
c) O(n^2)
d) O(n^3)

Answer: c
Explanation: The space complexity of the above dynamic programming implementation of the matrix chain multiplication is O(n^2).

10. What is the output of the following code?

```
#include<stdio.h>
#include<limits.h>
int mat_chain_multiplication(int *mat, int n)
{
```

```
    int arr[n][n];
    int i,k,row,col,len;
    for(i=1;i<n;i++)
        arr[i][i] = 0;
    for(len = 2; len < n; len++)
    {
         for(row = 1; row <= n - len + 1; row++)
         {
              col = row + len - 1;
              arr[row][col] = INT_MAX;
              for(k = row; k <= col - 1; k++)
              {
                   int tmp = arr[row][k] + arr[k + 1][col] + mat[row - 1] * mat[k] * mat[col];
                   if(tmp < arr[row][col])
                   arr[row][col] = tmp;
              }
          }
    }
    return arr[1][n - 1];
}
int main()
{
    int mat[6] = {20,30,40,50};
    int ans = mat_chain_multiplication(mat,4);
    printf("%d",ans);
    return 0;
}
```

a) 64000
b) 70000
c) 120000
d) 150000

Answer: a
Explanation: The program prints the minimum number of multiplications required, which is 64000.

11. What is the value stored in arr[2][3] when the following code is executed?

```
#include<stdio.h>
#include<limits.h>
int mat_chain_multiplication(int *mat, int n)
{
    int arr[n][n];
    int i,k,row,col,len;
    for(i=1;i<n;i++)
        arr[i][i] = 0;
    for(len = 2; len < n; len++)
    {
         for(row = 1; row <= n - len + 1; row++)
         {
              col = row + len - 1;
              arr[row][col] = INT_MAX;
              for(k = row; k <= col - 1; k++)
              {
                    int tmp = arr[row][k] + arr[k + 1][col] + mat[row - 1] * mat[k] * mat[col];
                    if(tmp < arr[row][col])
                    arr[row][col] = tmp;
              }
         }
    }
    return arr[1][n - 1];
}
int main()
{
    int mat[6] = {20,30,40,50};
    int ans = mat_chain_multiplication(mat,4);
    printf("%d",ans);
    return 0;
```

```
}
```

a) 64000
b) 60000
c) 24000
d) 12000

Answer: b
Explanation: The value stored in arr[2][3] when the above code is executed is 60000.

12. What is the output of the following code?

```
#include<stdio.h>
#include<limits.h>
int mat_chain_multiplication(int *mat, int n)
{
     int arr[n][n];
     int i,k,row,col,len;
     for(i=1;i<n;i++)
         arr[i][i] = 0;
     for(len = 2; len < n; len++)
     {
          for(row = 1; row <= n - len + 1; row++)
          {
               col = row + len - 1;
               arr[row][col] = INT_MAX;
               for(k = row; k <= col - 1; k++)
               {
                    int tmp = arr[row][k] + arr[k + 1][col] + mat[row - 1] * mat[k] * mat[col];
                    if(tmp < arr[row][col])
                    arr[row][col] = tmp;
               }
          }
     }
     return arr[1][n-1];
}
int main()
{
     int mat[6] = {10,10,10,10,10,10};
     int ans = mat_chain_multiplication(mat,6);
     printf("%d",ans);
     return 0;
}
```

a) 2000
b) 3000
c) 4000
d) 5000

Answer: c
Explanation: The program prints the minimum number of multiplications required to multiply the given matrices, which is 4000.

13. What is the output of the following code?

```
#include<stdio.h>
#include<limits.h>
int mat_chain_multiplication(int *mat, int n)
{
     int arr[n][n];
     int i,k,row,col,len;
     for(i=1;i<n;i++)
         arr[i][i] = 0;
     for(len = 2; len < n; len++)
     {
          for(row = 1; row <= n - len + 1; row++)
```

```
          {
               col = row + len - 1;
               arr[row][col] = INT_MAX;
               for(k = row; k <= col - 1; k++)
               {
                    int tmp = arr[row][k] + arr[k + 1][col] + mat[row - 1] * mat[k] * mat[col];
                    if(tmp < arr[row][col])
                    arr[row][col] = tmp;
               }
          }
     }
     return arr[1][n-1];
}
int main()
{
     int mat[6] = {20,25,30,35,40};
     int ans = mat_chain_multiplication(mat,5);
     printf("%d",ans);
     return 0;
}
```

a) 32000
b) 28000
c) 64000
d) 70000

Answer: c
Explanation: The output of the program is 64000.

1. Which of the following methods can be used to solve the longest common subsequence problem?
a) Recursion
b) Dynamic programming
c) Both recursion and dynamic programming
d) Greedy algorithm

Answer: c
Explanation: Both recursion and dynamic programming can be used to solve the longest subsequence problem.

2. Consider the strings "PQRSTPQRS" and "PRATPBRQRPS". What is the length of the longest common subsequence?
a) 9
b) 8
c) 7
d) 6

Answer: c
Explanation: The longest common subsequence is "PRTPQRS" and its length is 7.

3. Which of the following problems can be solved using the longest subsequence problem?
a) Longest increasing subsequence
b) Longest palindromic subsequence
c) Longest bitonic subsequence
d) Longest decreasing subsequence

Answer: b
Explanation: To find the longest palindromic subsequence in a given string, reverse the given string and then find the longest common subsequence in the given string and the reversed string.

4. Longest common subsequence is an example of _____
a) Greedy algorithm
b) 2D dynamic programming

c) 1D dynamic programming
d) Divide and conquer

Answer: b
Explanation: Longest common subsequence is an example of 2D dynamic programming.

5. What is the time complexity of the brute force algorithm used to find the longest common subsequence?
a) O(n)
b) $O(n^2)$
c) $O(n^3)$
d) $O(2^n)$

Answer: d
Explanation: The time complexity of the brute force algorithm used to find the longest common subsequence is $O(2^n)$.

6. Consider the following dynamic programming implementation of the longest common subsequence problem:

```
#include<stdio.h>
#include<string.h>
int max_num(int a, int b)
{
      if(a > b)
        return a;
      return b;
}
int lcs(char *str1, char *str2)
{
      int i,j,len1,len2;
      len1 = strlen(str1);
      len2 = strlen(str2);
      int arr[len1 + 1][len2 + 1];
      for(i = 0; i <= len1; i++)
          arr[i][0] = 0;
      for(i = 0; i <= len2; i++)
          arr[0][i] = 0;
      for(i = 1; i <= len1; i++)
      {
            for(j = 1; j <= len2; j++)
            {
                 if(str1[i-1] == str2[j - 1])
                  _______________;
                else
                   arr[i][j] = max_num(arr[i - 1][j], arr[i][j - 1]);
            }
      }
      return arr[len1][len2];
}
int main()
{
      char str1[] = " abcedfg", str2[] = "bcdfh";
      int ans = lcs(str1,str2);
      printf("%d",ans);
      return 0;
}
```

Which of the following lines completes the above code?
a) arr[i][j] = 1 + arr[i][j].
b) arr[i][j] = 1 + arr[i – 1][j – 1].
c) arr[i][j] = arr[i – 1][j – 1].
d) arr[i][j] = arr[i][j].

Answer: b
Explanation: The line, arr[i][j] = 1 + arr[i – 1][j – 1] completes the above code.

7. What is the time complexity of the following dynamic programming implementation of the longest common subsequence problem where length of one string is “m” and the length of the other string is “n”?

```
#include<stdio.h>
#include<string.h>
int max_num(int a, int b)
{
      if(a > b)
        return a;
      return b;
}
int lcs(char *str1, char *str2)
{
      int i,j,len1,len2;
      len1 = strlen(str1);
      len2 = strlen(str2);
      int arr[len1 + 1][len2 + 1];
      for(i = 0; i <= len1; i++)
          arr[i][0] = 0;
      for(i = 0; i <= len2; i++)
          arr[0][i] = 0;
      for(i = 1; i <= len1; i++)
      {
            for(j = 1; j <= len2; j++)
            {
                 if(str1[i-1] == str2[j - 1])
                  arr[i][j] = 1 + arr[i - 1][j - 1];
                else
                   arr[i][j] = max_num(arr[i - 1][j], arr[i][j - 1]);
            }
      }
      return arr[len1][len2];
}
int main()
{
      char str1[] = " abcedfg", str2[] = "bcdfh";
      int ans = lcs(str1,str2);
      printf("%d",ans);
      return 0;
}
```

a) O(n)
b) O(m)
c) O(m + n)
d) O(mn)

Answer: d
Explanation: The time complexity of the above dynamic programming implementation of the longest common subsequence is O(mn).

8. What is the space complexity of the following dynamic programming implementation of the longest common subsequence problem where length of one string is “m” and the length of the other string is “n”?

```
#include<stdio.h>
#include<string.h>
int max_num(int a, int b)
{
      if(a > b)
        return a;
      return b;
}
int lcs(char *str1, char *str2)
{
      int i,j,len1,len2;
      len1 = strlen(str1);
      len2 = strlen(str2);
      int arr[len1 + 1][len2 + 1];
```

```
      for(i = 0; i <= len1; i++)
          arr[i][0] = 0;
      for(i = 0; i <= len2; i++)
          arr[0][i] = 0;
      for(i = 1; i <= len1; i++)
      {
            for(j = 1; j <= len2; j++)
            {
                 if(str1[i-1] == str2[j - 1])
                  arr[i][j] = 1 + arr[i - 1][j - 1];
                else
                   arr[i][j] = max_num(arr[i - 1][j], arr[i][j - 1]);
            }
      }
      return arr[len1][len2];
}
int main()
{
      char str1[] = " abcedfg", str2[] = "bcdfh";
      int ans = lcs(str1,str2);
      printf("%d",ans);
      return 0;
}
```

a) O(n)
b) O(m)
c) O(m + n)
d) O(mn)

Answer: d
Explanation: The space complexity of the above dynamic programming implementation of the longest common subsequence is O(mn).

9. What is the output of the following code?

```
#include<stdio.h>
#include<string.h>
int max_num(int a, int b)
{
      if(a > b)
         return a;
      return b;
}
int lcs(char *str1, char *str2)
{
      int i,j,len1,len2;
      len1 = strlen(str1);
      len2 = strlen(str2);
      int arr[len1 + 1][len2 + 1];
      for(i = 0; i <= len1; i++)
          arr[i][0] = 0;
      for(i = 0; i <= len2; i++)
          arr[0][i] = 0;
      for(i = 1; i <= len1; i++)
      {
          for(j = 1; j <= len2; j++)
          {
              if(str1[i-1] == str2[j - 1])
                  arr[i][j] = 1 + arr[i - 1][j - 1];
              else
                  arr[i][j] = max_num(arr[i - 1][j], arr[i][j - 1]);
          }
      }
      return arr[len1][len2];
}
int main()
{
      char str1[] = "hbcfgmnapq", str2[] = "cbhgrsfnmq";
```

```
      int ans = lcs(str1,str2);
      printf("%d",ans);
      return 0;
}
```

a) 3
b) 4
c) 5
d) 6

Answer: b
Explanation: The program prints the length of the longest common subsequence, which is 4.

10. Which of the following is the longest common subsequence between the strings "hbcfgmnapq" and "cbhgrsfnmq" ?
a) hgmq
b) cfnq
c) bfmq
d) fgmna

Answer: d
Explanation: The length of the longest common subsequence is 4. But 'fgmna' is not the longest common subsequence as its length is 5.

11. What is the value stored in arr[2][3] when the following code is executed?

```
#include<stdio.h>
#include<string.h>
int max_num(int a, int b)
{
      if(a > b)
         return a;
      return b;
}
int lcs(char *str1, char *str2)
{
      int i,j,len1,len2;
      len1 = strlen(str1);
      len2 = strlen(str2);
      int arr[len1 + 1][len2 + 1];
      for(i = 0; i <= len1; i++)
          arr[i][0] = 0;
      for(i = 0; i <= len2; i++)
          arr[0][i] = 0;
      for(i = 1; i <= len1; i++)
      {
           for(j = 1; j <= len2; j++)
           {
                if(str1[i-1] == str2[j - 1])
                    arr[i][j] = 1 + arr[i - 1][j - 1];
                else
                    arr[i][j] = max_num(arr[i - 1][j], arr[i][j - 1]);
           }
      }
      return arr[len1][len2];
}
int main()
{
      char str1[] = "hbcfgmnapq", str2[] = "cbhgrsfnmq";
      int ans = lcs(str1,str2);
      printf("%d",ans);
      return 0;
}
```

a) 1

b) 2
c) 3
d) 4

Answer: a
Explanation: The value stored in arr[2][3] is 1.

12. What is the output of the following code?

```
#include<stdio.h>
#include<string.h>
int max_num(int a, int b)
{
      if(a > b)
        return a;
      return b;
}
int lcs(char *str1, char *str2)
{
      int i,j,len1,len2;
      len1 = strlen(str1);
      len2 = strlen(str2);
      int arr[len1 + 1][len2 + 1];
      for(i = 0; i <= len1; i++)
          arr[i][0] = 0;
      for(i = 0; i <= len2; i++)
          arr[0][i] = 0;
      for(i = 1; i <= len1; i++)
      {
           for(j = 1; j <= len2; j++)
           {
                if(str1[i-1] == str2[j - 1])
                   arr[i][j] = 1 + arr[i - 1][j - 1];
                else
                   arr[i][j] = max_num(arr[i - 1][j], arr[i][j - 1]);
           }
      }
      return arr[len1][len2];
}
int main()
{
     char str1[] = "abcd", str2[] = "efgh";
     int ans = lcs(str1,str2);
     printf("%d",ans);
     return 0;
}
```

a) 3
b) 2
c) 1
d) 0

Answer: d
Explanation: The program prints the length of the longest common subsequence, which is 0.

1. Which of the following methods can be used to solve the longest palindromic subsequence problem?
a) Dynamic programming
b) Recursion
c) Brute force
d) Dynamic programming, Recursion, Brute force

Answer: d
Explanation: Dynamic programming, Recursion, Brute force can be used to solve the longest palindromic subsequence problem.

2. Which of the following is not a palindromic subsequence of the string "ababcdabba"?
a) abcba
b) abba
c) abbbba
d) adba

Answer: d
Explanation: 'adba' is not a palindromic sequence.

3. For which of the following, the length of the string is not equal to the length of the longest palindromic subsequence?
a) A string that is a palindrome
b) A string of length one
c) A string that has all the same letters(e.g. aaaaaa)
d) Some strings of length two

Answer: d
Explanation: A string of length 2 for eg: ab is not a palindrome.

4. What is the length of the longest palindromic subsequence for the string "ababcdabba"?
a) 6
b) 7
c) 8
d) 9

Answer: b
Explanation: The longest palindromic subsequence is "abbabba" and its length is 7.

5. What is the time complexity of the brute force algorithm used to find the length of the longest palindromic subsequence?
a) O(1)
b) $O(2^n)$
c) O(n)
d) $O(n^2)$

Answer: b
Explanation: In the brute force algorithm, all the subsequences are found and the length of the longest palindromic subsequence is calculated. This takes exponential time.Answer: a
Explanation: A single character of any string can always be considered as a palindrome and its length is one.

7. Longest palindromic subsequence is an example of _____
a) Greedy algorithm
b) 2D dynamic programming
c) 1D dynamic programming
d) Divide and conquer

Answer: b
Explanation: Longest palindromic subsequence is an example of 2D dynamic programming.

8. Consider the following code:

```
#include<stdio.h>
#include<string.h>
int max_num(int a, int b)
{
      if(a > b)
        return a;
      return b;
}
```

```
int lps(char *str1)
{
      int i,j,len;
      len = strlen(str1);
      char str2[len + 1];
      strcpy(str2, str1);
      _____________;
      int arr[len + 1][len + 1];
      for(i = 0; i <= len; i++)
          arr[i][0] = 0;
      for(i = 0; i <= len; i++)
          arr[0][i] = 0;
      for(i = 1; i <= len; i++)
      {
          for(j = 1; j <= len; j++)
          {
               if(str1[i-1] == str2[j - 1])
                   arr[i][j] = 1 + arr[i - 1][j - 1];
               else
                   arr[i][j] = max_num(arr[i - 1][j], arr[i][j - 1]);
          }
      }
      return arr[len][len];
}
int main()
{
     char str1[] = "ababcdabba";
     int ans = lps(str1);
     printf("%d",ans);
     return 0;
}
```

Which of the following lines completes the above code?
a) strrev(str2)
b) str2 = str1
c) len2 = strlen(str2)
d) strlen(str2)

Answer: a
Explanation: To find the longest palindromic subsequence, we need to reverse the copy of the string, which is done by strrev.

9. What is the time complexity of the following dynamic programming implementation to find the longest palindromic subsequence where the length of the string is n?

```
#include<stdio.h>
#include<string.h>
int max_num(int a, int b)
{
      if(a > b)
        return a;
      return b;
}
int lps(char *str1)
{
      int i,j,len;
      len = strlen(str1);
      char str2[len + 1];
      strcpy(str2, str1);
      strrev(str2);
      int arr[len + 1][len + 1];
      for(i = 0; i <= len; i++)
          arr[i][0] = 0;
      for(i = 0; i <= len; i++)
          arr[0][i] = 0;
      for(i = 1; i <= len; i++)
      {
```

```
        for(j = 1; j <= len; j++)
        {
             if(str1[i-1] == str2[j - 1])
                 arr[i][j] = 1 + arr[i - 1][j - 1];
             else
                 arr[i][j] = max_num(arr[i - 1][j], arr[i][j - 1]);
        }
      }
      return arr[len][len];
}
int main()
{
     char str1[] = "ababcdabba";
     int ans = lps(str1);
     printf("%d",ans);
     return 0;
}
```

a) O(n)
b) O(1)
c) O(n^2)
d) O(2)

Answer: c
Explanation: The time complexity of the above dynamic programming implementation to find the longest palindromic subsequence is O(n^2).

10. What is the space complexity of the following dynamic programming implementation to find the longest palindromic subsequence where the length of the string is n?

```
#include<stdio.h>
#include<string.h>
int max_num(int a, int b)
{
      if(a > b)
        return a;
      return b;
}
int lps(char *str1)
{
      int i,j,len;
      len = strlen(str1);
      char str2[len + 1];
      strcpy(str2, str1);
      strrev(str2);
      int arr[len + 1][len + 1];
      for(i = 0; i <= len; i++)
          arr[i][0] = 0;
      for(i = 0; i <= len; i++)
          arr[0][i] = 0;
      for(i = 1; i <= len; i++)
      {
          for(j = 1; j <= len; j++)
          {
               if(str1[i-1] == str2[j - 1])
                   arr[i][j] = 1 + arr[i - 1][j - 1];
               else
                   arr[i][j] = max_num(arr[i - 1][j], arr[i][j - 1]);
          }
      }
      return arr[len][len];
}
int main()
{
     char str1[] = "ababcdabba";
     int ans = lps(str1);
     printf("%d",ans);
```

```
    return 0;
}
```

a) O(n)
b) O(1)
c) O(n^2)
d) O(2)

Answer: c
Explanation: The space complexity of the above dynamic programming implementation to find the longest palindromic subsequence is O(n^2).

11. What is the value stored in arr[3][3] when the following code is executed?

```
#include<stdio.h>
#include<string.h>
int max_num(int a, int b)
{
      if(a > b)
        return a;
      return b;
}
int lps(char *str1)
{
      int i,j,len;
      len = strlen(str1);
      char str2[len + 1];
      strcpy(str2, str1);
      strrev(str2);
      int arr[len + 1][len + 1];
      for(i = 0; i <= len; i++)
          arr[i][0] = 0;
      for(i = 0; i <= len; i++)
          arr[0][i] = 0;
      for(i = 1; i <= len; i++)
      {
          for(j = 1; j <= len; j++)
          {
                if(str1[i-1] == str2[j - 1])
                    arr[i][j] = 1 + arr[i - 1][j - 1];
                else
                    arr[i][j] = max_num(arr[i - 1][j], arr[i][j - 1]);
          }
      }
      return arr[len][len];
}
int main()
{
      char str1[] = "ababcdabba";
      int ans = lps(str1);
      printf("%d",ans);
      return 0;
}
```

a) 2
b) 3
c) 4
d) 5

Answer: a
Explanation: The value stored in arr[3][3] when the above code is executed is 2.

12. What is the output of the following code?

```
#include<stdio.h>
#include<string.h>
```

```
int max_num(int a, int b)
{
      if(a > b)
        return a;
      return b;
}
int lps(char *str1)
{
      int i,j,len;
      len = strlen(str1);
      char str2[len + 1];
      strcpy(str2, str1);
      strrev(str2);
      int arr[len + 1][len + 1];
      for(i = 0; i <= len; i++)
          arr[i][0] = 0;
      for(i = 0; i <= len; i++)
          arr[0][i] = 0;
      for(i = 1; i <= len; i++)
      {
          for(j = 1; j <= len; j++)
          {
               if(str1[i-1] == str2[j - 1])
                  arr[i][j] = 1 + arr[i - 1][j - 1];
               else
                  arr[i][j] = max_num(arr[i - 1][j], arr[i][j - 1]);
          }
      }
      return arr[len][len];
}
int main()
{
      char str1[] = "abcd";
      int ans = lps(str1);
      printf("%d",ans);
      return 0;
}
```

a) 0
b) 1
c) 2
d) 3

Answer: b
Explanation: The program prints the length of the longest palindromic subsequence, which is 1.

13. What is the output of the following code?

```
#include<stdio.h>
#include<string.h>
int max_num(int a, int b)
{
      if(a > b)
        return a;
      return b;
}
int lps(char *str1)
{
      int i,j,len;
      len = strlen(str1);
      char str2[len + 1];
      strcpy(str2, str1);
      strrev(str2);
      int arr[len + 1][len + 1];
      for(i = 0; i <= len; i++)
          arr[i][0] = 0;
      for(i = 0; i <= len; i++)
          arr[0][i] = 0;
```

```
      for(i = 1; i <= len; i++)
      {
          for(j = 1; j <= len; j++)
          {
              if(str1[i-1] == str2[j - 1])
                  arr[i][j] = 1 + arr[i - 1][j - 1];
              else
                  arr[i][j] = max_num(arr[i - 1][j], arr[i][j - 1]);
          }
      }
      return arr[len][len];
}
int main()
{
      char str1[] = "abdgkagdjbccbba";
      int ans = lps(str1);
      printf("%d",ans);
      return 0;
}
```

a) 5
b) 7
c) 9
d) 11

Answer: c
Explanation: The program prints the length of the longest palindromic subsequence, which is 9.

1. Which of the following methods can be used to solve the edit distance problem?
a) Recursion
b) Dynamic programming
c) Both dynamic programming and recursion
d) Greedy Algorithm

Answer: c
Explanation: Both dynamic programming and recursion can be used to solve the edit distance problem.Answer: a
Explanation: d(s,s) = 0, since each string can be transformed into itself without any change.
d(s1, s2) > 0 when s1 != s2, since the transformation would require at least one operation.
d(s1, s2) = d(s2, s1)
d(s1, s3) <= d(s1, s2) + d(s2, s3)
Thus, the edit distance satisfies the axioms of a metric.

3. Which of the following is an application of the edit distance problem?
a) Approximate string matching
b) Spelling correction
c) Similarity of DNA
d) Approximate string matching, Spelling Correction and Similarity of DNA

Answer: d
Explanation: All of the mentioned are the applications of the edit distance problem.

4. In which of the following cases will the edit distance between two strings be zero?
a) When one string is a substring of another
b) When the lengths of the two strings are equal
c) When the two strings are equal
d) The edit distance can never be zero

Answer: c
Explanation: The edit distance will be zero only when the two strings are equal.Answer: a
Explanation: Consider the strings "abcd" and "efghi". The string "efghi" can be converted to "abcd" by deleting "i" and converting "efgh" to "abcd". The cost of transformation is 5, which is equal to the length of the larger string.

6. Consider the strings “monday” and “tuesday”. What is the edit distance between the two strings?
a) 3
b) 4
c) 5
d) 6

Answer: b
Explanation: “monday” can be converted to “tuesday” by replacing “m” with “t”, “o” with “u”, “n” with “e” and inserting “s” at the appropriate position. So, the edit distance is 4.

7. Consider the two strings “”(empty string) and “abcd”. What is the edit distance between the two strings?
a) 0
b) 4
c) 2
d) 3

Answer: b
Explanation: The empty string can be transformed into “abcd” by inserting “a”, “b”, “c” and “d” at appropriate positions. Thus, the edit distance is 4.

8. Consider the following dynamic programming implementation of the edit distance problem:

```
#include<stdio.h>
#include<string.h>
int get_min(int a, int b)
{
      if(a < b)
        return a;
      return b;
}
int edit_distance(char *s1, char *s2)
{
     int len1,len2,i,j,min;
     len1 = strlen(s1);
     len2 = strlen(s2);
     int arr[len1 + 1][len2 + 1];
     for(i = 0;i <= len1; i++)
       arr[i][0] = i;
     for(i = 0; i <= len2; i++)
       arr[0][i] = i;
     for(i = 1; i <= len1; i++)
     {
         for(j = 1; j <= len2; j++)
         {
               min = get_min(arr[i-1][j],arr[i][j-1]) + 1;
               if(s1[i - 1] == s2[j - 1])
               {
                    if(arr[i-1][j-1] < min)
                       min = arr[i-1][j-1];
               }
               else
               {
                     if(arr[i-1][j-1] + 1 < min)
                         min = arr[i-1][j-1] + 1;
               }
                _____________;
         }
     }
     return arr[len1][len2];
}
int main()
{
     char s1[] = "abcd", s2[] = "defg";
     int ans = edit_distance(s1, s2);
     printf("%d",ans);
```

```
    return 0;
}
```

Which of the following lines should be added to complete the above code?
a) arr[i-1][j] = min
b) arr[i][j-1] = min
c) arr[i-1][j-1] = min
d) arr[i][j] = min

Answer: d
Explanation: The line arr[i][j] = min completes the above code.

9. What is the time complexity of the following dynamic programming implementation of the edit distance problem where "m" and "n" are the lengths of two strings?

```
#include<stdio.h>
#include<string.h>
int get_min(int a, int b)
{
      if(a < b)
        return a;
      return b;
}
int edit_distance(char *s1, char *s2)
{
     int len1,len2,i,j,min;
     len1 = strlen(s1);
     len2 = strlen(s2);
     int arr[len1 + 1][len2 + 1];
     for(i = 0;i <= len1; i++)
       arr[i][0] = i;
     for(i = 0; i <= len2; i++)
       arr[0][i] = i;
     for(i = 1; i <= len1; i++)
     {
         for(j = 1; j <= len2; j++)
         {
               min = get_min(arr[i-1][j],arr[i][j-1]) + 1;
               if(s1[i - 1] == s2[j - 1])
               {
                    if(arr[i-1][j-1] < min)
                       min = arr[i-1][j-1];
               }
               else
               {
                     if(arr[i-1][j-1] + 1 < min)
                         min = arr[i-1][j-1] + 1;
               }
                arr[i][j] = min;
         }
     }
     return arr[len1][len2];
}
int main()
{
     char s1[] = "abcd", s2[] = "defg";
     int ans = edit_distance(s1, s2);
     printf("%d",ans);
     return 0;
}
```

a) O(1)
b) O(m + n)
c) O(mn)
d) O(n)

Answer: c
Explanation: The time complexity of the above dynamic programming implementation of the edit distance problem is O(mn).

10. What is the space complexity of the following dynamic programming implementation of the edit distance problem where "m" and "n" are the lengths of the two strings?

```
#include<stdio.h>
#include<string.h>
int get_min(int a, int b)
{
      if(a < b)
        return a;
      return b;
}
int edit_distance(char *s1, char *s2)
{
     int len1,len2,i,j,min;
     len1 = strlen(s1);
     len2 = strlen(s2);
     int arr[len1 + 1][len2 + 1];
     for(i = 0;i <= len1; i++)
       arr[i][0] = i;
     for(i = 0; i <= len2; i++)
       arr[0][i] = i;
     for(i = 1; i <= len1; i++)
     {
         for(j = 1; j <= len2; j++)
         {
               min = get_min(arr[i-1][j],arr[i][j-1]) + 1;
               if(s1[i - 1] == s2[j - 1])
               {
                    if(arr[i-1][j-1] < min)
                       min = arr[i-1][j-1];
               }
               else
               {
                     if(arr[i-1][j-1] + 1 < min)
                         min = arr[i-1][j-1] + 1;
               }
                arr[i][j] = min;
         }
     }
     return arr[len1][len2];
}
int main()
{
     char s1[] = "abcd", s2[] = "defg";
     int ans = edit_distance(s1, s2);
     printf("%d",ans);
     return 0;
}
```

a) O(1)
b) O(m + n)
c) O(mn)
d) O(n)

Answer: c
Explanation: The space complexity of the above dynamic programming implementation of the edit distance problem is O(mn).

11. What is the output of the following code?

```
#include<stdio.h>
#include<string.h>
```

```
int get_min(int a, int b)
{
      if(a < b)
        return a;
      return b;
}
int edit_distance(char *s1, char *s2)
{
     int len1,len2,i,j,min;
     len1 = strlen(s1);
     len2 = strlen(s2);
     int arr[len1 + 1][len2 + 1];
     for(i = 0;i <= len1; i++)
      arr[i][0] = i;
     for(i = 0; i <= len2; i++)
      arr[0][i] = i;
     for(i = 1; i <= len1; i++)
     {
         for(j = 1; j <= len2; j++)
         {
              min = get_min(arr[i-1][j],arr[i][j-1]) + 1;
              if(s1[i - 1] == s2[j - 1])
              {
                  if(arr[i-1][j-1] < min)
                      min = arr[i-1][j-1];
              }
              else
              {
                  if(arr[i-1][j-1] + 1 < min)
                     min = arr[i-1][j-1] + 1;
              }
              arr[i][j] = min;
         }
     }
     return arr[len1][len2];
}
int main()
{
     char s1[] = "abcd", s2[] = "defg";
     int ans = edit_distance(s1, s2);
     printf("%d",ans);
     return 0;
}
```

a) 1
b) 2
c) 3
d) 4

Answer: d
Explanation: The program prints the edit distance between the strings “abcd” and “defg”, which is 4.

12. What is the value stored in arr[2][2] when the following code is executed?

```
#include<stdio.h>
#include<string.h>
int get_min(int a, int b)
{
      if(a < b)
        return a;
      return b;
}
int edit_distance(char *s1, char *s2)
{
      int len1,len2,i,j,min;
      len1 = strlen(s1);
      len2 = strlen(s2);
      int arr[len1 + 1][len2 + 1];
```

```
      for(i = 0;i <= len1; i++)
        arr[i][0] = i;
      for(i = 0; i <= len2; i++)
         arr[0][i] = i;
      for(i = 1; i <= len1; i++)
      {
           for(j = 1; j <= len2; j++)
           {
                 min = get_min(arr[i-1][j],arr[i][j-1]) + 1;
                 if(s1[i - 1] == s2[j - 1])
                 {
                      if(arr[i-1][j-1] < min)
                        min = arr[i-1][j-1];
                 }
                 else
                 {
                      if(arr[i-1][j-1] + 1 < min)
                        min = arr[i-1][j-1] + 1;
                 }
                 arr[i][j] = min;
           }
      }
      return arr[len1][len2];
}
int main()
{
      char s1[] = "abcd", s2[] = "defg";
      int ans = edit_distance(s1, s2);
      printf("%d",ans);
      return 0;
}
```

a) 1
b) 2
c) 3
d) 4

Answer: b
Explanation: The value stored in arr[2][2] when the above code is executed is 2.

13. What is the output of the following code?

```
#include<stdio.h>
#include<string.h>
int get_min(int a, int b)
{
      if(a < b)
        return a;
      return b;
}
int edit_distance(char *s1, char *s2)
{
     int len1,len2,i,j,min;
     len1 = strlen(s1);
     len2 = strlen(s2);
     int arr[len1 + 1][len2 + 1];
     for(i = 0;i <= len1; i++)
       arr[i][0] = i;
     for(i = 0; i <= len2; i++)
       arr[0][i] = i;
     for(i = 1; i <= len1; i++)
     {
         for(j = 1; j <= len2; j++)
         {
              min = get_min(arr[i-1][j],arr[i][j-1]) + 1;
              if(s1[i - 1] == s2[j - 1])
              {
                  if(arr[i-1][j-1] < min)
```

```
                    min = arr[i-1][j-1];
                }
                else
                {
                    if(arr[i-1][j-1] + 1 < min)
                        min = arr[i-1][j-1] + 1;
                }
                arr[i][j] = min;
            }
        }
        return arr[len1][len2];
}
int main()
{
        char s1[] = "pqrstuv", s2[] = "prstuv";
        int ans = edit_distance(s1, s2);
        printf("%d",ans);
        return 0;
}
```

a) 1
b) 2
c) 3
d) 4

Answer: a
Explanation: The code prints the edit distance between the two strings, which is 1.

1. Wagner–Fischer is a _____ algorithm.
a) Brute force
b) Greedy
c) Dynamic programming
d) Recursive

Answer: c
Explanation: Wagner–Fischer belongs to the dynamic programming type of algorithms.

2. Wagner–Fischer algorithm is used to find _____
a) Longest common subsequence
b) Longest increasing subsequence
c) Edit distance between two strings
d) Longest decreasing subsequence

Answer: c
Explanation: Wagner–Fischer algorithm is used to find the edit distance between two strings.

3. What is the edit distance between the strings “abcd” and “acbd” when the allowed operations are insertion, deletion and substitution?
a) 1
b) 2
c) 3
d) 4

Answer: b
Explanation: The string “abcd” can be changed to “acbd” by substituting “b” with “c” and “c” with “b”. Thus, the edit distance is 2.

4. For which of the following pairs of strings is the edit distance maximum?
a) sunday & monday
b) monday & tuesday
c) tuesday & wednesday

d) wednesday & thursday

Answer: d
Explanation: The edit distances are 2, 4, 4 and 5 respectively. Hence, the maximum edit distance is between the strings wednesday and thursday.

5. Consider the following implementation of the Wagner–Fischer algorithm:

```
int get_min(int a, int b)
{
     if(a < b)
        return a;
     return b;
}
int edit_distance(char *s1, char *s2)
{
     int len1,len2,i,j,min;
     len1 = strlen(s1);
     len2 = strlen(s2);
     int arr[len1 + 1][len2 + 1];
     for(i = 0;i <= len1; i++)
       arr[i][0] = i;
     for(i = 0; i <= len2; i++)
       arr[0][i] = i;
     for(i = 1; i <= len1; i++)
     {
         for(j = 1; j <= len2; j++)
         {
              min = get_min(arr[i-1][j],arr[i][j-1]) + 1;
              if(s1[i - 1] == s2[j - 1])
              {
                  if(arr[i-1][j-1] < min)
                      ____________;
              }
              else
              {
                  if(arr[i-1][j-1] + 1 < min)
                      min = arr[i-1][j-1] + 1;
              }
              arr[i][j] = min;
         }
     }
     return arr[len1][len2];
}
```

Which of the following lines should be inserted to complete the above code?
a) arr[i][j] = min
b) min = arr[i-1][j-1] – 1;
c) min = arr[i-1][j-1].
d) min = arr[i-1][j-1] + 1;

Answer: c
Explanation: The line min = arr[i-1][j-1] completes the above code.

6. What is the time complexity of the Wagner–Fischer algorithm where “m” and “n” are the lengths of the two strings?
a) O(1)
b) O(n+m)
c) O(mn)
d) O(nlogm)

Answer: c
Explanation: The time complexity of the Wagner–Fischer algorithm is O(mn).

7. What is the space complexity of the above implementation of Wagner–Fischer algorithm where “m” and “n” are the lengths of the two strings?
a) O(1)
b) O(n+m)
c) O(mn)
d) O(nlogm)

Answer: c
Explanation: The space complexity of the above Wagner–Fischer algorithm is O(mn).

8. What is the output of the following code?

```
#include<stdio.h>
#include<string.h>
int get_min(int a, int b)
{
      if(a < b)
        return a;
      return b;
}
int edit_distance(char *s1, char *s2)
{
      int len1,len2,i,j,min;
      len1 = strlen(s1);
      len2 = strlen(s2);
      int arr[len1 + 1][len2 + 1];
      for(i = 0;i <= len1; i++)
         arr[i][0] = i;
      for(i = 0; i <= len2; i++)
         arr[0][i] = i;
      for(i = 1; i <= len1; i++)
      {
          for(j = 1; j <= len2; j++)
          {
               min = get_min(arr[i-1][j],arr[i][j-1]) + 1;
               if(s1[i - 1] == s2[j - 1])
               {
                   if(arr[i-1][j-1] < min)
                      min = arr[i-1][j-1];
               }
               else
               {
                   if(arr[i-1][j-1] + 1 < min)
                     min = arr[i-1][j-1] + 1;
               }
               arr[i][j] = min;
          }
      }
      return arr[len1][len2];
}
int main()
{
      char s1[] = "somestring", s2[] = "anotherthing";
      int ans = edit_distance(s1, s2);
      printf("%d",ans);
      return 0;
}
```

a) 6
b) 7
c) 8
d) 9

Answer: a
Explanation: The program prints the edit distance between the strings “somestring” and “anotherthing”, which is 6.

9. What is the value stored in arr[3][3] when the below code is executed?

```
#include<stdio.h>
#include<string.h>
int get_min(int a, int b)
{
      if(a < b)
        return a;
      return b;
}
int edit_distance(char *s1, char *s2)
{
      int len1,len2,i,j,min;
      len1 = strlen(s1);
      len2 = strlen(s2);
      int arr[len1 + 1][len2 + 1];
      for(i = 0;i <= len1; i++)
        arr[i][0] = i;
      for(i = 0; i <= len2; i++)
         arr[0][i] = i;
      for(i = 1; i <= len1; i++)
      {
           for(j = 1; j <= len2; j++)
           {
                min = get_min(arr[i-1][j],arr[i][j-1]) + 1;
                if(s1[i - 1] == s2[j - 1])
                {
                     if(arr[i-1][j-1] < min)
                       min = arr[i-1][j-1];
                }
                else
                {
                    if(arr[i-1][j-1] + 1 < min)
                     min = arr[i-1][j-1] + 1;
                }
                arr[i][j] = min;
          }
     }
     return arr[len1][len2];
}
int main()
{
      char s1[] = "somestring", s2[] = "anotherthing";
      int ans = edit_distance(s1, s2);
      printf("%d",ans);
      return 0;
}
```

a) 1
b) 2
c) 3
d) 4

Answer: c
Explanation: The value stored in arr[3][3] is 3.

10. What is the output of the following code?

```
#include<stdio.h>
#include<string.h>
int get_min(int a, int b)
{
     if(a < b)
         return a;
     return b;
}
int edit_distance(char *s1, char *s2)
{
```

```
    int len1,len2,i,j,min;
    len1 = strlen(s1);
    len2 = strlen(s2);
    int arr[len1 + 1][len2 + 1];
    for(i = 0;i <= len1; i++)
      arr[i][0] = i;
    for(i = 0; i <= len2; i++)
      arr[0][i] = i;
    for(i = 1; i <= len1; i++)
    {
        for(j = 1; j <= len2; j++)
        {
             min = get_min(arr[i-1][j],arr[i][j-1]) + 1;
             if(s1[i - 1] == s2[j - 1])
             {
                 if(arr[i-1][j-1] < min)
                    min = arr[i-1][j-1];
             }
             else
             {
                 if(arr[i-1][j-1] + 1 < min)
                     min = arr[i-1][j-1] + 1;
             }
             arr[i][j] = min;
        }
    }
    return arr[len1][len2];
}
int main()
{
    char s1[] = "abcd", s2[] = "dcba";
    int ans = edit_distance(s1, s2);
    printf("%d",ans);
    return 0;
}
```

a) 1
b) 2
c) 3
d) 4

Answer: d
Explanation: The program prints the edit distance between the strings "abcd" and "dcba", which is 4.

1. Which of the following is NOT a Catalan number?
a) 1
b) 5
c) 14
d) 43

Answer: d
Explanation: Catalan numbers are given by: (2n!)/((n+1)!n!).
For n = 0, we get C0 = 1.
For n = 3, we get C3 = 5.
For n = 4, we get C4 = 14.
For n = 5, we get C3 = 42.

2. Which of the following numbers is the 6th Catalan number?
a) 14
b) 429
c) 132
d) 42

Answer: d

Explanation: Catalan numbers are given by: (2n!)/((n+1)!n!).
First Catalan number is given by n = 0.
So the 6th Catalan number will be given by n = 5, which is 42.

3. Which of the following is not an application of Catalan Numbers?
a) Counting the number of Dyck words
b) Counting the number of expressions containing n pairs of parenthesis
c) Counting the number of ways in which a convex polygon can be cut into triangles by connecting vertices with straight lines
d) Creation of head and tail for a given number of tosses

Answer: d
Explanation: Counting the number of Dyck words, Counting the number of expressions containing n pairs of parenthesis, Counting the number of ways in which a convex polygon can be cut into triangles by connecting vertices with straight lines are the applications of Catalan numbers where as creation of head and tails for a given number of tosses is an application of Pascal's triangle.

4. Which of the following methods can be used to find the nth Catalan number?
a) Recursion
b) Binomial coefficients
c) Dynamic programming
d) Recursion, Binomial Coefficients, Dynamic programming

Answer: d
Explanation: All of the mentioned methods can be used to find the nth Catalan number.

Which of the following lines completes the above code?
a) arr[i] = arr[j] * arr[k];
b) arr[j] += arr[i] * arr[k];
c) arr[i] += arr[j] * arr[k].
d) arr[j] = arr[i] * arr[k];

Answer: c
*Explanation: The line arr[i] += arr[j] * arr[k] reflects the recursive formula Cn = ∑Ci*C(n-i).*

6. Which of the following implementations of Catalan numbers has the smallest time complexity?
a) Dynamic programming
b) Binomial coefficients
c) Recursion
d) All have equal time complexity

Answer: b
Explanation: The time complexities are as follows:
Dynamic programming: $O(n^2)$
Recursion: Exponential
Binomial coefficients: O(n).

7. What is the output of the following code?

```
#include<stdio.h>
int cat_number(int n)
{
     int i,j,arr[n],k;
     arr[0] = 1;
     for(i = 1; i < n; i++)
     {
         arr[i] = 0;
         for(j = 0,k = i - 1; j < i; j++,k--)
           ________________________;
     }
```

```
    return arr[n-1];

}
int main()
{
    int ans, n = 8;
    ans = cat_number(n);
    printf("%d\n",ans);
    return 0;
}
```

a) 42
b) 132
c) 429
d) 1430

Answer: c
Explanation: The program prints the 8th Catalan number, which is 429.

8. Which of the following implementations of Catalan numbers has the largest space complexity(Don't consider the stack space)?
a) Dynamic programming
b) Binomial coefficients
c) Recursion
d) All have equal space complexities

Answer: a
Explanation: The space complexities are as follows:
Dynamic programming: O(n)
Recursion: O(1)
Binomial coefficients: O(1).

9. What will be the value stored in arr[5] when the following code is executed?

```
#include<stdio.h>
int cat_number(int n)
{
    int i,j,arr[n],k;
    arr[0] = 1;
    for(i = 1; i < n; i++)
    {
        arr[i] = 0;
        for(j = 0,k = i - 1; j < i; j++,k--)
        arr[i] += arr[j] * arr[k];
    }
    return arr[n-1];
}
int main()
{
    int ans, n = 8;
    ans = cat_number(n);
    printf("%d\n",ans);
    return 0;
}
```

a) 14
b) 42
c) 132
d) 429

Answer: b
Explanation: The 6th Catalan number will be stored in arr[5], which is 42.

10. Which of the following errors will occur when the below code is executed?

```
#include<stdio.h>
int cat_number(int n)
{
     int i,j,arr[n],k;
     arr[0] = 1;
     for(i = 1; i < n; i++)
     {
         arr[i] = 0;
         for(j = 0,k = i - 1; j < i; j++,k--)
          arr[i] += arr[j] * arr[k];
     }
     return arr[n-1];
}
int main()
{
     int ans, n = 10;
     ans = cat_number(n);
     printf("%d\n",ans);
     return 0;
}
```

a) Segmentation fault
b) Array size too large
c) Integer value out of range
d) Array index out of range

Answer: c
Explanation: The 100th Catalan number is too large to be stored in an integer. So, the error produced will be integer value out of range.(It will be a logical error and the compiler wont show it).

1. Which of the following methods can be used to solve the assembly line scheduling problem?
a) Recursion
b) Brute force
c) Dynamic programming
d) All of the mentioned

Answer: d
Explanation: All of the above mentioned methods can be used to solve the assembly line scheduling problem.

2. What is the time complexity of the brute force algorithm used to solve the assembly line scheduling problem?
a) O(1)
b) O(n)
c) $O(n^2)$
d) $O(2^n)$

Answer: d
Explanation: In the brute force algorithm, all the possible ways are calculated which are of the order of 2^n.

3. In the dynamic programming implementation of the assembly line scheduling problem, how many lookup tables are required?
a) 0
b) 1
c) 2
d) 3

Answer: c
Explanation: In the dynamic programming implementation of the assembly line scheduling problem, 2 lookup tables are required one for storing the minimum time and the other for storing the assembly line number.

4. Consider the following assembly line problem:

```
time_to_reach[2][3] = {{17, 2, 7}, {19, 4, 9}}
time_spent[2][4] = {{6, 5, 15, 7}, {5, 10, 11, 4}}
entry_time[2] = {8, 10}
exit_time[2] = {10, 7}
num_of_stations = 4
```

For the optimal solution which should be the starting assembly line?
a) Line 1
b) Line 2
c) All of the mentioned
d) None of the mentioned

Answer: b
Explanation: For the optimal solution, the starting assembly line is line 2.

5. Consider the following assembly line problem:

```
time_to_reach[2][3] = {{17, 2, 7}, {19, 4, 9}}
time_spent[2][4] = {{6, 5, 15, 7}, {5, 10, 11, 4}}
entry_time[2] = {8, 10}
exit_time[2] = {10, 7}
num_of_stations = 4
```

For the optimal solution, which should be the exit assembly line?
a) Line 1
b) Line 2
c) All of the mentioned
d) None of the mentioned

Answer: b
Explanation: For the optimal solution, the exit assembly line is line 2.

6. Consider the following assembly line problem:

```
time_to_reach[2][3] = {{17, 2, 7}, {19, 4, 9}}
time_spent[2][4] = {{6, 5, 15, 7}, {5, 10, 11, 4}}
entry_time[2] = {8, 10}
exit_time[2] = {10, 7}
num_of_stations = 4
```

What is the minimum time required to build the car chassis?
a) 40
b) 41
c) 42
d) 43

Answer: d
Explanation: The minimum time required is 43.
The path is S2,1 -> S1,2 -> S2,3 -> S2,4, where Si,j : i = line number, j = station number

7. Consider the following code:

```
#include<stdio.h>
int get_min(int a, int b)
{
     if(a<b)
        return a;
     return b;
}
int minimum_time_required(int reach[][3],int spent[][4], int *entry, int *exit, int n)
{
```

```
    int t1[n], t2[n],i;
    t1[0] = entry[0] + spent[0][0];
    t2[0] = entry[1] + spent[1][0];
    for(i = 1; i < n; i++)
    {
        t1[i] = get_min(t1[i-1]+spent[0][i], t2[i-1]+reach[1][i-1]+spent[0][i]);
        __________;
    }
    return get_min(t1[n-1]+exit[0], t2[n-1]+exit[1]);
}
```

Which of the following lines should be inserted to complete the above code?
a) t2[i] = get_min(t2[i-1]+spent[1][i], t1[i-1]+reach[0][i-1]+spent[1][i])
b) t2[i] = get_min(t2[i-1]+spent[1][i], t1[i-1]+spent[1][i])
c) t2[i] = get_min(t2[i-1]+spent[1][i], t1[i-1]+reach[0][i-1])
d) none of the mentioned

Answer: a
Explanation: The line t2[i] = get_min(t2[i-1]+spent[1][i], t1[i-1]+reach[0][i-1]+spent[1][i]) should be added to complete the above code.

8. What is the time complexity of the above dynamic programming implementation of the assembly line scheduling problem?
a) O(1)
b) O(n)
c) $O(n^2)$
d) $O(n^3)$

Answer: b
Explanation: The time complexity of the above dynamic programming implementation of the assembly line scheduling problem is O(n).

9. What is the space complexity of the above dynamic programming implementation of the assembly line scheduling problem?
a) O(1)
b) O(n)
c) $O(n^2)$
d) $O(n^3)$

Answer: b
Explanation: The space complexity of the above dynamic programming implementation of the assembly line scheduling problem is O(n).

10. What is the output of the following code?

```
#include<stdio.h>
int get_min(int a, int b)
{
    if(a<b)
       return a;
    return b;
}
int minimum_time_required(int reach[][3],int spent[][4], int *entry, int *exit, int n)
{
    int t1[n], t2[n], i;
    t1[0] = entry[0] + spent[0][0];
    t2[0] = entry[1] + spent[1][0];
    for(i = 1; i < n; i++)
    {
        t1[i] = get_min(t1[i-1]+spent[0][i], t2[i-1]+reach[1][i-1]+spent[0][i]);
        t2[i] = get_min(t2[i-1]+spent[1][i], t1[i-1]+reach[0][i-1]+spent[1][i]);
    }
```

```
    return get_min(t1[n-1]+exit[0], t2[n-1]+exit[1]);
}
int main()
{
    int time_to_reach[][3] = {{6, 1, 5},
                              {2, 4, 7}};
    int time_spent[][4] = {{6, 5, 4, 7},
                           {5, 10, 2, 6}};
    int entry_time[2] = {5, 6};
    int exit_time[2] = {8, 9};
    int num_of_stations = 4;
    int ans = minimum_time_required(time_to_reach, time_spent,
              entry_time, exit_time, num_of_stations);
    printf("%d",ans);
    return 0;
}
```

a) 32
b) 33
c) 34
d) 35

Answer: c
Explanation: The program prints the optimal time required to build the car chassis, which is 34.

11. What is the value stored in t1[2] when the following code is executed?

```
#include<stdio.h>
int get_min(int a, int b)
{
     if(a<b)
        return a;
     return b;
}
int minimum_time_required(int reach[][3],int spent[][4], int *entry, int *exit, int n)
{
      int t1[n], t2[n],i;
      t1[0] = entry[0] + spent[0][0];
      t2[0] = entry[1] + spent[1][0];
      for(i = 1; i < n; i++)
      {
          t1[i] = get_min(t1[i-1]+spent[0][i], t2[i-1]+reach[1][i-1]+spent[0][i]);
          t2[i] = get_min(t2[i-1]+spent[1][i], t1[i-1]+reach[0][i-1]+spent[1][i]);
      }
    return get_min(t1[n-1]+exit[0], t2[n-1]+exit[1]);
}
int main()
{
     int time_to_reach[][3] = {{6, 1, 5},
                               {2, 4, 7}};
     int time_spent[][4] = {{6, 5, 4, 7},
                            {5, 10, 2, 6}};
     int entry_time[2] = {5, 6};
     int exit_time[2] = {8, 9};
     int num_of_stations = 4;
     int ans = minimum_time_required(time_to_reach, time_spent,
               entry_time, exit_time, num_of_stations);
     printf("%d",ans);
     return 0;
}
```

a) 16
b) 18
c) 20
d) 22

Answer: c

Explanation: The value stored in t1[2] when the above code is executed is 20.

12. What is the value stored in t2[3] when the following code is executed?

```
#include<stdio.h>
int get_min(int a, int b)
{
     if(a<b)
        return a;
     return b;
}
int minimum_time_required(int reach[][3],int spent[][4], int *entry, int *exit, int n)
{
     int t1[n], t2[n],i;
     t1[0] = entry[0] + spent[0][0];
     t2[0] = entry[1] + spent[1][0];
     for(i = 1; i < n; i++)
     {
         t1[i] = get_min(t1[i-1]+spent[0][i], t2[i-1]+reach[1][i-1]+spent[0][i]);
         t2[i] = get_min(t2[i-1]+spent[1][i], t1[i-1]+reach[0][i-1]+spent[1][i]);
     }
     return get_min(t1[n-1]+exit[0], t2[n-1]+exit[1]);
}
int main()
{
    int time_to_reach[][3] = {{6, 1, 5},
                              {2, 4, 7}};
    int time_spent[][4] = {{6, 5, 4, 7},
                           {5, 10, 2, 6}};
    int entry_time[2] = {5, 6};
    int exit_time[2] = {8, 9};
    int num_of_stations = 4;
    int ans = minimum_time_required(time_to_reach, time_spent,
              entry_time, exit_time, num_of_stations);
    printf("%d",ans);
    return 0;
}
```

a) 19
b) 23
c) 25
d) 27

Answer: c
Explanation: The value stored in t2[3] when the above code is executed is 25.

13. What is the output of the following code?

```
#include<stdio.h>
int get_min(int a, int b)
{
      if(a<b)
        return a;
      return b;
}
int minimum_time_required(int reach[][4],int spent[][5], int *entry, int *exit, int n)
{
      int t1[n], t2[n], i;
      t1[0] = entry[0] + spent[0][0];
      t2[0] = entry[1] + spent[1][0];
      for(i = 1; i < n; i++)
      {
          t1[i] = get_min(t1[i-1]+spent[0][i], t2[i-1]+reach[1][i-1]+spent[0][i]);
          t2[i] = get_min(t2[i-1]+spent[1][i], t1[i-1]+reach[0][i-1]+spent[1][i]);
      }
      return get_min(t1[n-1]+exit[0], t2[n-1]+exit[1]);
}
```

```
int main()
{
     int time_to_reach[][4] = {{16, 10, 5, 12},
                               {12, 4, 17, 8}};
     int time_spent[][5] = {{13, 5, 20, 19, 9},
                            {15, 10, 12, 16, 13}};
     int entry_time[2] = {12, 9};
     int exit_time[2] = {10, 13};
     int num_of_stations = 5;
     int ans = minimum_time_required(time_to_reach, time_spent,
               entry_time, exit_time, num_of_stations);
     printf("%d",ans);
     return 0;
}
```

a) 62
b) 69
c) 75
d) 88

Answer: d
Explanation: The program prints the optimal time required to build the car chassis, which is 88.

1. Given a string, you have to find the minimum number of characters to be inserted in the string so that the string becomes a palindrome. Which of the following methods can be used to solve the problem?
a) Greedy algorithm
b) Recursion
c) Dynamic programming
d) Both recursion and dynamic programming

Answer: d
Explanation: Dynamic programming and recursion can be used to solve the problem.

2. In which of the following cases the minimum no of insertions to form palindrome is maximum?
a) String of length one
b) String with all same characters
c) Palindromic string
d) Non palindromic string

Answer: d
Explanation: In string of length one, string with all same characters and a palindromic string the no of insertions is zero since the strings are already palindromes. To convert a non-palindromic string to a palindromic string, the minimum length of string to be added is 1 which is greater than all the other above cases. Hence the minimum no of insertions to form palindrome is maximum in non-palindromic strings.Answer: b
Explanation: In the worst case, the minimum number of insertions to be made to convert the string into a palindrome is equal to length of the string minus one. For example, consider the string "abc". The string can be converted to "abcba" by inserting "a" and "b". The number of insertions is two, which is equal to length minus one.

4. Consider the string "efge". What is the minimum number of insertions required to make the string a palindrome?
a) 0
b) 1
c) 2
d) 3

Answer: b
Explanation: The string can be converted to "efgfe" by inserting "f" or to "egfge" by inserting "g". Thus, only one insertion is required.

5. Consider the string "abbccbba". What is the minimum number of insertions required to make the string a palindrome?

a) 0
b) 1
c) 2
d) 3

Answer: a
Explanation: The given string is already a palindrome. So, no insertions are required.

6. Which of the following problems can be used to solve the minimum number of insertions to form a palindrome problem?
a) Minimum number of jumps problem
b) Longest common subsequence problem
c) Coin change problem
d) Knapsack problems

Answer: b
Explanation: A variation of longest common subsequence can be used to solve the minimum number of insertions to form a palindrome problem.

7. Consider the following dynamic programming implementation:

```
#include<stdio.h>
#include<string.h>
int max(int a, int b)
{
      if(a > b)
        return a;
      return b;
}
int min_ins(char *s)
{
      int len = strlen(s), i, j;
      int arr[len + 1][len + 1];
      char rev[len + 1];
      strcpy(rev, s);
      strrev(rev);
      for(i = 0;i <= len; i++)
         arr[i][0] = 0;
      for(i = 0; i <= len; i++)
         arr[0][i] = 0;
      for(i = 1; i <= len; i++)
      {
          for(j = 1; j <= len; j++)
          {
              if(s[i - 1] == rev[j - 1])
                 arr[i][j] = arr[i - 1][j - 1] + 1;
              else
                 arr[i][j] = max(arr[i - 1][j], arr[i][j - 1]);
          }
      }
      return _____________;
}
int main()
{
      char s[] = "abcda";
      int ans = min_ins(s);
      printf("%d",ans);
      return 0;
}
```

Which of the following lines should be added to complete the code?
a) arr[len][len]
b) len + arr[len][len]
c) len

d) len – arr[len][len]

Answer: d
Explanation: arr[len][len] contains the length of the longest palindromic subsequence. So, len – arr[len][len] gives the minimum number of insertions required to form a palindrome.

8. What is the time complexity of the following dynamic programming implementation of the minimum number of insertions to form a palindrome problem?

```
#include<stdio.h>
#include<string.h>
int max(int a, int b)
{
      if(a > b)
        return a;
      return b;
}
int min_ins(char *s)
{
      int len = strlen(s), i, j;
      int arr[len + 1][len + 1];
      char rev[len + 1];
      strcpy(rev, s);
      strrev(rev);
      for(i = 0;i <= len; i++)
         arr[i][0] = 0;
      for(i = 0; i <= len; i++)
         arr[0][i] = 0;
      for(i = 1; i <= len; i++)
      {
          for(j = 1; j <= len; j++)
          {
              if(s[i - 1] == rev[j - 1])
                 arr[i][j] = arr[i - 1][j - 1] + 1;
              else
                 arr[i][j] = max(arr[i - 1][j], arr[i][j - 1]);
          }
      }
      return len - arr[len][len];
}
int main()
{
      char s[] = "abcda";
      int ans = min_ins(s);
      printf("%d",ans);
      return 0;
}
```

a) O(1)
b) O(n)
c) $O(n^2)$
d) O(mn)

Answer: c
Explanation: The time complexity of the above dynamic programming implementation is $O(n^2)$.

9. What is the space complexity of the following dynamic programming implementation of the minimum number of insertions to form a palindrome problem?

```
#include<stdio.h>
#include<string.h>
int max(int a, int b)
{
      if(a > b)
        return a;
      return b;
```

```
}
int min_ins(char *s)
{
      int len = strlen(s), i, j;
      int arr[len + 1][len + 1];
      char rev[len + 1];
      strcpy(rev, s);
      strrev(rev);
      for(i = 0;i <= len; i++)
         arr[i][0] = 0;
      for(i = 0; i <= len; i++)
         arr[0][i] = 0;
      for(i = 1; i <= len; i++)
      {
          for(j = 1; j <= len; j++)
          {
              if(s[i - 1] == rev[j - 1])
                 arr[i][j] = arr[i - 1][j - 1] + 1;
              else
                 arr[i][j] = max(arr[i - 1][j], arr[i][j - 1]);
          }
      }
      return len - arr[len][len];
}
int main()
{
      char s[] = "abcda";
      int ans = min_ins(s);
      printf("%d",ans);
      return 0;
}
```

a) O(1)
b) O(n)
c) O(n^2)
d) O(mn)

Answer: c
Explanation: The space complexity of the above dynamic programming implementation is O(n^2).

10. What is the output of the following code?

```
#include<stdio.h>
#include<string.h>
int max(int a, int b)
{
      if(a > b)
        return a;
      return b;
}
int min_ins(char *s)
{
      int len = strlen(s), i, j;
      int arr[len + 1][len + 1];
      char rev[len + 1];
      strcpy(rev, s);
      strrev(rev);
      for(i = 0;i <= len; i++)
         arr[i][0] = 0;
      for(i = 0; i <= len; i++)
         arr[0][i] = 0;
      for(i = 1; i <= len; i++)
      {
          for(j = 1; j <= len; j++)
          {
              if(s[i - 1] == rev[j - 1])
                 arr[i][j] = arr[i - 1][j - 1] + 1;
              else
```

```
                arr[i][j] = max(arr[i - 1][j], arr[i][j - 1]);
        }
    }
    return len - arr[len][len];
}
int main()
{
    char s[] = "abcda";
    int ans = min_ins(s);
    printf("%d",ans);
    return 0;
}
```

a) 1
b) 2
c) 3
d) 4

Answer: b
Explanation: The length of the longest palindromic subsequence is 3. So, the output will be 5 – 3 = 2.

11. What is the value stored in arr[2][4] when the following code is executed?

```
#include<stdio.h>
#include<string.h>
int max(int a, int b)
{
    if(a > b)
      return a;
    return b;
}
int min_ins(char *s)
{
    int len = strlen(s), i, j;
    int arr[len + 1][len + 1];
    char rev[len + 1];
    strcpy(rev, s);
    strrev(rev);
    for(i = 0;i <= len; i++)
       arr[i][0] = 0;
    for(i = 0; i <= len; i++)
       arr[0][i] = 0;
    for(i = 1; i <= len; i++)
    {
        for(j = 1; j <= len; j++)
        {
            if(s[i - 1] == rev[j - 1])
               arr[i][j] = arr[i - 1][j - 1] + 1;
            else
                arr[i][j] = max(arr[i - 1][j], arr[i][j - 1]);
        }
    }
    return len - arr[len][len];
}
int main()
{
    char s[] = "abcda";
    int ans = min_ins(s);
    printf("%d",ans);
    return 0;
}
```

a) 2
b) 3
c) 4
d) 5

Answer: a
Explanation: The value stored in arr[2][4] when the above code is executed is 2.

12. What is the output of the following code?

```
#include<stdio.h>
#include<string.h>
int max(int a, int b)
{
      if(a > b)
        return a;
      return b;
}
int min_ins(char *s)
{
      int len = strlen(s), i, j;
      int arr[len + 1][len + 1];
      char rev[len + 1];
      strcpy(rev, s);
      strrev(rev);
      for(i = 0;i <= len; i++)
          arr[i][0] = 0;
      for(i = 0; i <= len; i++)
          arr[0][i] = 0;
      for(i = 1; i <= len; i++)
      {
          for(j = 1; j <= len; j++)
          {
              if(s[i - 1] == rev[j - 1])
                arr[i][j] = arr[i - 1][j - 1] + 1;
              else
                  arr[i][j] = max(arr[i - 1][j], arr[i][j - 1]);
          }
      }
      return len - arr[len][len];
}
int main()
{
      char s[] = "efgfe";
      int ans = min_ins(s);
      printf("%d",ans);
      return 0;
}
```

a) 0
b) 2
c) 4
d) 6

Answer: a
Explanation: Since the string "efgfe" is already a palindrome, the number of insertions required is 0.

1. Given a 2D matrix, find a submatrix that has the maximum sum. Which of the following methods can be used to solve this problem?
a) Brute force
b) Recursion
c) Dynamic programming
d) Brute force, Recursion, Dynamic programming

Answer: d
Explanation: Brute force, Recursion and Dynamic programming can be used to find the submatrix that has the maximum sum.

2. In which of the following cases, the maximum sum rectangle is the 2D matrix itself?
a) When all the elements are negative

b) When all the elements are positive
c) When some elements are positive and some negative
d) When diagonal elements are positive and rest are negative

Answer: a
Explanation: When all the elements of a matrix are positive, the maximum sum rectangle is the 2D matrix itself.Answer: d
Explanation: If the matrix size is 1×1 then the element itself is the maximum sum of that 1×1 matrix. If all elements are zero, then the sum of any submatrix of the given matrix is zero. If all elements are negative, then the maximum element in that matrix is the highest sum in that matrix which is again 1X1 submatrix of the given matrix. Hence all three statements are correct.Answer: a
Explanation: If a matrix contains all non-zero elements with at least one positive and at least on negative element, then the sum of elements of the maximum sum rectangle cannot be zero.

5. Consider the 2×3 matrix {{1,2,3},{1,2,3}}. What is the sum of elements of the maximum sum rectangle?
a) 3
b) 6
c) 12
d) 18

Answer: c
Explanation: Since all the elements of the 2×3 matrix are positive, the maximum sum rectangle is the matrix itself and the sum of elements is 12.

6. Consider the 2×2 matrix {{-1,-2},{-3,-4}}. What is the sum of elements of the maximum sum rectangle?
a) 0
b) -1
c) -7
d) -12

Answer: b
Explanation: Since all the elements of the 2×2 matrix are negative, the maximum sum rectangle is {-1}, a 1×1 matrix containing the largest element. The sum of elements of the maximum sum rectangle is -1.

7. Consider the 3×3 matrix {{2,1,-3},{6,3,4},{-2,3,0}}. What is the sum of the elements of the maximum sum rectangle?
a) 13
b) 16
c) 14
d) 19

Answer: c
Explanation: The complete matrix represents the maximum sum rectangle and it's sum is 14.

8. What is the time complexity of the brute force implementation of the maximum sum rectangle problem?
a) O(n)
b) $O(n^2)$
c) $O(n^3)$
d) $O(n^4)$

Answer: d
Explanation: The time complexity of the brute force implementation of the maximum sum rectangle problem is $O(n^4)$.

9. The dynamic programming implementation of the maximum sum rectangle problem uses which of the following algorithm?
a) Hirschberg's algorithm
b) Needleman-Wunsch algorithm

c) Kadane's algorithm
d) Wagner Fischer algorithm

Answer: c
Explanation: The dynamic programming implementation of the maximum sum rectangle problem uses Kadane's algorithm.

10. Consider the following code snippet:

```
int max_sum_rectangle(int arr[][3],int row,int col)
{
      int left, right, tmp[row], mx_sm = INT_MIN, idx, val;
      for(left = 0; left < col; left++)
      {
           for(right = left; right < col; right++)
           {
               if(right == left)
               {
                   for(idx = 0; idx < row; idx++)
                     tmp[idx] = arr[idx][right];
               }
               else
               {
                   for(idx = 0; idx < row; idx++)
                      tmp[idx] += arr[idx][right];
               }
               val = kadane_algo(tmp,row);
               if(val > mx_sm)
                  ______;
           }
      }
      return mx_sm;
}
```

Which of the following lines should be inserted to complete the above code?
a) val = mx_sm
b) return val
c) mx_sm = val
d) return mx_sm

Answer: c
Explanation: The line "mx_sm = val" should be inserted to complete the above code.

11. What is the time complexity of the following dynamic programming implementation of the maximum sum rectangle problem?

```
int max_sum_rectangle(int arr[][3],int row,int col)
{
      int left, right, tmp[row], mx_sm = INT_MIN, idx, val;
      for(left = 0; left < col; left++)
      {
           for(right = left; right < col; right++)
           {
               if(right == left)
               {
                   for(idx = 0; idx < row; idx++)
                     tmp[idx] = arr[idx][right];
               }
               else
               {
                   for(idx = 0; idx < row; idx++)
                      tmp[idx] += arr[idx][right];
               }
               val = kadane_algo(tmp,row);
               if(val > mx_sm)
                 mx_sm = val;
```

```
            }
      }
      return mx_sm;
}
```

a) O(row*col)
b) O(row)
c) O(col)
d) O(row*col*col)

Answer: d
*Explanation: The time complexity of the above dynamic programming implementation of the maximum sum rectangle is O(row*col*col).*

12. What is the space complexity of the following dynamic programming implementation of the maximum sum rectangle problem?

```
int max_sum_rectangle(int arr[][3],int row,int col)
{
      int left, right, tmp[row], mx_sm = INT_MIN, idx, val;
      for(left = 0; left < col; left++)
      {
           for(right = left; right < col; right++)
           {
               if(right == left)
               {
                   for(idx = 0; idx < row; idx++)
                     tmp[idx] = arr[idx][right];
               }
               else
               {
                   for(idx = 0; idx < row; idx++)
                      tmp[idx] += arr[idx][right];
               }
               val = kadane_algo(tmp,row);
               if(val > mx_sm)
                 mx_sm = val;
           }
      }
      return mx_sm;
}
```

a) O(row*col)
b) O(row)
c) O(col)
d) O(row*col*col)

Answer: b
Explanation: The space complexity of the above dynamic programming implementation of the maximum sum rectangle problem is O(row).

13. What is the output of the following code?

```
#include<stdio.h>
#include<limits.h>
int max_num(int a, int b)
{
      if(a > b)
        return a;
      return b;
}
int kadane_algo(int *arr, int len)
{
      int ans, sum, idx;
      ans =0;
```

```
      sum =0;
      for(idx =0; idx < len; idx++)
      {
          sum = max_num(0,sum + arr[idx]);
          ans = max_num(sum,ans);
      }
      return ans;
}
int max_sum_rectangle(int arr[][5],int row,int col)
{
      int left, right, tmp[row], mx_sm = INT_MIN, idx, val;
      for(left = 0; left < col; left++)
      {
           for(right = left; right < col; right++)
           {
                if(right == left)
                {
                    for(idx = 0; idx < row; idx++)
                     tmp[idx] = arr[idx][right];
                }
                else
                {
                     for(idx = 0; idx < row; idx++)
                       tmp[idx] += arr[idx][right];
                }
                val = kadane_algo(tmp,row);
                if(val > mx_sm)
                mx_sm = val;
           }
      }
      return mx_sm;
}
int main()
{
       int arr[2][5] ={{1, 2, -1, -4, -20}, {-4, -1, 1, 7, -6}};
       int row = 2, col = 5;
       int ans = max_sum_rectangle(arr,row,col);
       printf("%d",ans);
       return 0;
}
```

a) 7
b) 8
c) 9
d) 10

Answer: b
Explanation: The program prints the sum of elements of the maximum sum rectangle, which is 8.

14. What is the output of the following code?

```
#include<stdio.h>
#include<limits.h>
int max_num(int a, int b)
{
      if(a > b)
        return a;
      return b;
}
int kadane_algo(int *arr, int len)
{
      int ans, sum, idx;
      ans =0;
      sum =0;
      for(idx =0; idx < len; idx++)
      {
          sum = max_num(0,sum + arr[idx]);
          ans = max_num(sum,ans);
```

```
      }
      return ans;
}
int max_sum_rectangle(int arr[][5],int row,int col)
{
      int left, right, tmp[row], mx_sm = INT_MIN, idx, val=0;
      for(left = 0; left < col; left++)
      {
           for(right = left; right < col; right++)
           {
               if(right == left)
               {
                   for(idx = 0; idx < row; idx++)
                      tmp[idx] = arr[idx][right];
               }
               else
               {
                   for(idx = 0; idx < row; idx++)
                      tmp[idx] += arr[idx][right];
               }
               val = kadane_algo(tmp,row);
               if(val > mx_sm)
               mx_sm = val;
           }
      }
      return mx_sm;
}
int main()
{
      int arr[4][5] ={{ 7,  1, -3, -6, -15},
                      { 10,  -6,  3,  -4,  11},
                      { -2, -3, -1,  2, -5},
                      { 3,  0,  1,  0,  3}};
      int row = 4, col = 5;
      int ans = max_sum_rectangle(arr,row,col);
      printf("%d",ans);
      return 0;
}
```

a) 17
b) 18
c) 19
d) 20

Answer: b
Explanation: The program prints the sum of elements of the maximum sum rectangle, which is 18.

1. Given an array, check if the array can be divided into two subsets such that the sum of elements of the two subsets is equal. This is the balanced partition problem. Which of the following methods can be used to solve the balanced partition problem?
a) Dynamic programming
b) Recursion
c) Brute force
d) Dynamic programming, Recursion, Brute force

Answer: d
Explanation: All of the mentioned methods can be used to solve the balanced partition problem.

2. In which of the following cases, it is not possible to have two subsets with equal sum?
a) When the number of elements is odd
b) When the number of elements is even
c) When the sum of elements is odd
d) When the sum of elements is even

Answer: c
Explanation: When the sum of all the elements is odd, it is not possible to have two subsets with equal sum.

3. What is the time complexity of the brute force algorithm used to solve the balanced partition problem?
a) O(1)
b) O(n)
c) O(n^2)
d) O(2^n)

Answer: d
Explanation: In the brute force implementation, all the possible subsets will be formed. This takes exponential time.

4. Consider a variation of the balanced partition problem in which we find two subsets such that |S1 – S2| is minimum. Consider the array {1, 2, 3, 4, 5}. Which of the following pairs of subsets is an optimal solution for the above problem?
a) {5, 4} & {3, 2, 1}
b) {5} & {4, 3, 2, 1}
c) {4, 2} & {5, 3, 1}
d) {5, 3} & {4, 2, 1}

Answer: d
Explanation: For S1 = {5, 3} and S2 = {4, 2, 1}, sum(S1) – sum(S2) = 1, which is the optimal solution.

5. Consider the following code:

```
#include<stdio.h>
int balanced_partition(int *arr, int len)
{
     int sm = 0, i, j;
     for(i = 0;i < len; i++)
      sm += arr[i];
     if(sm % 2 != 0)
        return 0;
     int ans[sm/2 + 1][len + 1];
     for(i = 0; i <= len; i++)
      ans[0][i] = 1;
     for(i = 1; i <= sm/2; i++)
      ans[i][0] = 0;
     for(i = 1; i <= sm/2; i++)
     {
         for(j = 1;j <= len; j++)
         {
             ans[i][j] = ans[i][j-1];
             if(i >= arr[j - 1])
                ans[i][j] = _________________;
         }
     }
     return ans[sm/2][len];
}
int main()
{
     int arr[] = {3, 4, 5, 6, 7, 1}, len = 6;
     int ans = balanced_partition(arr,len);
     if(ans == 0)
       printf("false");
     else
       printf("true");
     return 0;
}
```

Which of the following lines should be inserted to complete the above code?
a) ans[i – arr[j – 1]][j – 1]
b) ans[i][j]
c) ans[i][j] || ans[i – arr[j – 1]][j – 1]

d) ans[i][j] && ans[i – arr[j – 1]][j – 1]

Answer: c
Explanation: The line "ans[i][j] || ans[i – arr[j – 1]][j – 1]" completes the above code.

6. What is the time complexity of the following dynamic programming implementation of the balanced partition problem where "n" is the number of elements and "sum" is their sum?

```
#include<stdio.h>
int balanced_partition(int *arr, int len)
{
     int sm = 0, i, j;
     for(i = 0;i < len; i++)
      sm += arr[i];
     if(sm % 2 != 0)
        return 0;
     int ans[sm/2 + 1][len + 1];
     for(i = 0; i <= len; i++)
      ans[0][i] = 1;
     for(i = 1; i <= sm/2; i++)
      ans[i][0] = 0;
     for(i = 1; i <= sm/2; i++)
     {
         for(j = 1;j <= len; j++)
         {
             ans[i][j] = ans[i][j-1];
             if(i >= arr[j - 1])
                ans[i][j] = ans[i][j] || ans[i - arr[j - 1]][j - 1];
         }
     }
     return ans[sm/2][len];
}
int main()
{
     int arr[] = {3, 4, 5, 6, 7, 1}, len = 6;
     int ans = balanced_partition(arr,len);
     if(ans == 0)
       printf("false");
     else
       printf("true");
     return 0;
}
```

a) O(sum)
b) O(n)
c) O(sum * n)
d) O(sum + n)

Answer: c
*Explanation: The time complexity of the above dynamic programming implementation of the balanced partition problem is O(sum * n).*

7. What is the space complexity of the following dynamic programming implementation of the balanced partition problem?

```
#include<stdio.h>
int balanced_partition(int *arr, int len)
{
     int sm = 0, i, j;
     for(i = 0;i < len; i++)
      sm += arr[i];
     if(sm % 2 != 0)
        return 0;
     int ans[sm/2 + 1][len + 1];
     for(i = 0; i <= len; i++)
      ans[0][i] = 1;
```

```
    for(i = 1; i <= sm/2; i++)
     ans[i][0] = 0;
    for(i = 1; i <= sm/2; i++)
    {
        for(j = 1;j <= len; j++)
        {
            ans[i][j] = ans[i][j-1];
            if(i >= arr[j - 1])
               ans[i][j] = ans[i][j] || ans[i - arr[j - 1]][j - 1];
        }
    }
    return ans[sm/2][len];
}
int main()
{
    int arr[] = {3, 4, 5, 6, 7, 1}, len = 6;
    int ans = balanced_partition(arr,len);
    if(ans == 0)
      printf("false");
    else
      printf("true");
    return 0;
}
```

a) O(sum)
b) O(n)
c) O(sum * n)
d) O(sum + n)

Answer: c
*Explanation: The space complexity of the above dynamic programming implementation of the balanced partition problem is O(sum * n).*

8. What is the output of the following code?

```
#include<stdio.h>
int balanced_partition(int *arr, int len)
{
     int sm = 0, i, j;
     for(i = 0;i < len; i++)
     sm += arr[i];
     if(sm % 2 != 0)
       return 0;
     int ans[sm/2 + 1][len + 1];
     for(i = 0; i <= len; i++)
     ans[0][i] = 1;
     for(i = 1; i <= sm/2; i++)
     ans[i][0] = 0;
     for(i = 1; i <= sm/2; i++)
     {
         for(j = 1;j <= len; j++)
         {
             ans[i][j] = ans[i][j-1];
             if(i >= arr[j - 1])
               ans[i][j] = ans[i][j] || ans[i - arr[j - 1]][j - 1];
         }
     }
     return ans[sm/2][len];
}
int main()
{
     int arr[] = {3, 4, 5, 6, 7, 1}, len = 6;
     int ans = balanced_partition(arr,len);
     if(ans == 0)
        printf("false");
     else
        printf("true");
     return 0;
```

```
}
```

Answer: a
Explanation: The partitions are S1 = {6, 7} and S2 = {1, 3, 4, 5} and the sum of each partition is 13. So, the array can be divided into balanced partitions.

9. What is the value stored in ans[3][3] when the following code is executed?

```
#include<stdio.h>
int balanced_partition(int *arr, int len)
{
     int sm = 0, i, j;
     for(i = 0;i < len; i++)
      sm += arr[i];
     if(sm % 2 != 0)
        return 0;
     int ans[sm/2 + 1][len + 1];
     for(i = 0; i <= len; i++)
      ans[0][i] = 1;
     for(i = 1; i <= sm/2; i++)
      ans[i][0] = 0;
     for(i = 1; i <= sm/2; i++)
     {
         for(j = 1;j <= len; j++)
         {
             ans[i][j] = ans[i][j-1];
             if(i >= arr[j - 1])
                ans[i][j] = ans[i][j] || ans[i - arr[j - 1]][j - 1];
         }
     }
     return ans[sm/2][len];
}
int main()
{
     int arr[] = {3, 4, 5, 6, 7, 1}, len = 6;
     int ans = balanced_partition(arr,len);
     if(ans == 0)
        printf("false");
     else
        printf("true");
     return 0;
}
```

a) 0
b) 1
c) -1
d) -2

Answer: b
Explanation: The value stored in ans[3][3] indicates if a sum of 3 can be obtained using a subset of the first 3 elements. Since the sum can be obtained the value stored is 1.

10. What is the sum of each of the balanced partitions for the array {5, 6, 7, 10, 3, 1}?
a) 16
b) 32
c) 0
d) 64

Answer: a
Explanation: The sum of all the elements of the array is 32. So, the sum of all the elements of each partition should be 16.

11. What is the output of the following code?

```
#include<stdio.h>
```

```
int balanced_partition(int *arr, int len)
{
     int sm = 0, i, j;
     for(i = 0;i < len; i++)
      sm += arr[i];
     if(sm % 2 != 0)
        return 0;
     int ans[sm/2 + 1][len + 1];
     for(i = 0; i <= len; i++)
      ans[0][i] = 1;
     for(i = 1; i <= sm/2; i++)
      ans[i][0] = 0;
     for(i = 1; i <= sm/2; i++)
     {
         for(j = 1;j <= len; j++)
         {
             ans[i][j] = ans[i][j-1];
             if(i >= arr[j - 1])
                 ans[i][j] = ans[i][j] || ans[i - arr[j - 1]][j - 1];
         }
     }
     return ans[sm/2][len];
}
int main()
{
     int arr[] = {5, 6, 7, 10, 3, 1}, len = 6;
     int ans = balanced_partition(arr,len);
     if(ans == 0)
       printf("false");
     else
       printf("true");
     return 0;
}
```

Answer: a
Explanation: The array can be divided into two partitions S1 = {10, 6} and S2 = {5, 7, 3, 1} and the sum of all the elements of each partition is 16. So, the answer is true.

12. What is the output of the following code?

```
#include<stdio.h>
int balanced_partition(int *arr, int len)
{
     int sm = 0, i, j;
     for(i = 0;i < len; i++)
      sm += arr[i];
     if(sm % 2 != 0)
        return 0;
     int ans[sm/2 + 1][len + 1];
     for(i = 0; i <= len; i++)
      ans[0][i] = 1;
     for(i = 1; i <= sm/2; i++)
      ans[i][0] = 0;
     for(i = 1; i <= sm/2; i++)
     {
         for(j = 1;j <= len; j++)
         {
             ans[i][j] = ans[i][j-1];
             if(i >= arr[j - 1])
                 ans[i][j] = ans[i][j] || ans[i - arr[j - 1]][j - 1];
         }
     }
     return ans[sm/2][len];
}
int main()
{
     int arr[] = {1, 2, 3, 4, 5, 6, 7, 8, 9}, len = 9;
     int ans = balanced_partition(arr,len);
     if(ans == 0)
```

```
      printf("false");
    else
      printf("true");
    return 0;
}
```

Answer: b
Explanation: Since the sum of all the elements of the array is 45, the array cannot be divided into two partitions of equal sum and the answer is false.

1. You are given n dice each having f faces. You have to find the number of ways in which a sum of S can be achieved. This is the dice throw problem. Which of the following methods can be used to solve the dice throw problem?
a) Brute force
b) Recursion
c) Dynamic programming
d) Brute force, Recursion and Dynamic Programming

Answer: d
Explanation: Brute force, Recursion and Dynamic Programming can be used to solve the dice throw problem.

2. You have n dice each having f faces. What is the number of permutations that can be obtained when you roll the n dice together?
a) n*n*n…f times
b) f*f*f…n times
c) n*n*n…n times
d) f*f*f…f times

Answer: b
*Explanation: Each die can take f values and there are n dice. So, the total number of permutations is f*f*f...n times.*

3. You have 3 dice each having 6 faces. What is the number of permutations that can be obtained when you roll the 3 dice together?
a) 27
b) 36
c) 216
d) 81

Answer: c
*Explanation: The total number of permutations that can be obtained is 6 * 6 * 6 = 216.*

4. You have 2 dice each of them having 6 faces numbered from 1 to 6. What is the number of ways in which a sum of 11 can be achieved?
a) 0
b) 1
c) 2
d) 3

Answer: c
Explanation: The sum of 11 can be achieved when the dice show {6, 5} or {5, 6}.

5. There are n dice with f faces. The faces are numbered from 1 to f. What is the minimum possible sum that can be obtained when the n dice are rolled together?
a) 1
b) f
c) n
d) n*f

Answer: c
Explanation: The sum will be minimum when all the faces show a value 1. The sum in this case will be n.

6. There are n dice with f faces. The faces are numbered from 1 to f. What is the maximum possible sum that can be obtained when the n dice are rolled together?
a) 1
b) f*f
c) n*n
d) n*f

Answer: d
*Explanation: The sum will be maximum when all the faces show a value f. The sum in this case will be n*f.*

7. There are 10 dice having 5 faces. The faces are numbered from 1 to 5. What is the number of ways in which a sum of 4 can be achieved?
a) 0
b) 2
c) 4
d) 8

Answer: a
Explanation: Since there are 10 dice and the minimum value each die can take is 1, the minimum possible sum is 10. Hence, a sum of 4 cannot be achieved.

8. Consider the following dynamic programming implementation of the dice throw problem:

```
#include<stdio.h>
int get_ways(int num_of_dice, int num_of_faces, int S)
{
     int arr[num_of_dice + 1][S + 1];
     int dice, face, sm;
     for(dice = 0; dice <= num_of_dice; dice++)
         for(sm = 0; sm <= S; sm++)
           arr[dice][sm] = 0;
     for(sm = 1; sm <= S; sm++)
         arr[1][sm] = 1;
     for(dice = 2; dice <= num_of_dice; dice++)
     {
         for(sm = 1; sm <= S; sm++)
         {
             for(face = 1; face <= num_of_faces && face < sm; face++)
                 arr[dice][sm] += arr[dice - 1][sm - face];
         }
     }
     return _____________;
}
int main()
{
     int num_of_dice = 3, num_of_faces = 4, sum = 6;
     int ans = get_ways(num_of_dice, num_of_faces, sum);
     printf("%d",ans);
     return 0;
}
```

Which of the following lines should be added to complete the above code?
a) arr[num_of_dice][S]
b) arr[dice][sm]
c) arr[dice][S]
d) arr[S][dice]

Answer: a
Explanation: The line arr[num_of_dice][S] completes the above code.

9. What is time complexity of the following dynamic programming implementation of the dice throw problem where f is the number of faces, n is the number of dice and s is the sum to be found?

```
#include<stdio.h>
int get_ways(int num_of_dice, int num_of_faces, int S)
{
     int arr[num_of_dice + 1][S + 1];
     int dice, face, sm;
     for(dice = 0; dice <= num_of_dice; dice++)
         for(sm = 0; sm <= S; sm++)
           arr[dice][sm] = 0;
     for(sm = 1; sm <= S; sm++)
         arr[1][sm] = 1;
     for(dice = 2; dice <= num_of_dice; dice++)
     {
         for(sm = 1; sm <= S; sm++)
         {
             for(face = 1; face <= num_of_faces && face < sm; face++)
                 arr[dice][sm] += arr[dice - 1][sm - face];
         }
     }
     return arr[num_of_dice][S];
}
int main()
{
     int num_of_dice = 3, num_of_faces = 4, sum = 6;
     int ans = get_ways(num_of_dice, num_of_faces, sum);
     printf("%d",ans);
     return 0;
}
```

a) O(n*f)
b) O(f*s)
c) O(n*s)
d) O(n*f*s)

Answer: d
*Explanation: The time complexity of the above dynamic programming implementation is O(n*f*s).*

10. What is space complexity of the following dynamic programming implementation of the dice throw problem where f is the number of faces, n is the number of dice and s is the sum to be found?

```
#include<stdio.h>
int get_ways(int num_of_dice, int num_of_faces, int S)
{
     int arr[num_of_dice + 1][S + 1];
     int dice, face, sm;
     for(dice = 0; dice <= num_of_dice; dice++)
         for(sm = 0; sm <= S; sm++)
           arr[dice][sm] = 0;
     for(sm = 1; sm <= S; sm++)
         arr[1][sm] = 1;
     for(dice = 2; dice <= num_of_dice; dice++)
     {
         for(sm = 1; sm <= S; sm++)
         {
             for(face = 1; face <= num_of_faces && face < sm; face++)
                 arr[dice][sm] += arr[dice - 1][sm - face];
         }
     }
     return arr[num_of_dice][S];
}
int main()
{
     int num_of_dice = 3, num_of_faces = 4, sum = 6;
     int ans = get_ways(num_of_dice, num_of_faces, sum);
     printf("%d",ans);
     return 0;
}
```

a) O(n*f)
b) O(f*s)
c) O(n*s)
d) O(n*f*s)

Answer: c
*Explanation: The space complexity of the above dynamic programming implementation is O(n*s).*

11. What is the output of the following code?

```
#include<stdio.h>
int get_ways(int num_of_dice, int num_of_faces, int S)
{
     int arr[num_of_dice + 1][S + 1];
     int dice, face, sm;
     for(dice = 0; dice <= num_of_dice; dice++)
         for(sm = 0; sm <= S; sm++)
           arr[dice][sm] = 0;
     for(sm = 1; sm <= S; sm++)
         arr[1][sm] = 1;
     for(dice = 2; dice <= num_of_dice; dice++)
     {
         for(sm = 1; sm <= S; sm++)
         {
             for(face = 1; face <= num_of_faces && face < sm; face++)
                 arr[dice][sm] += arr[dice - 1][sm - face];
         }
     }
     return arr[num_of_dice][S];
}
int main()
{
     int num_of_dice = 3, num_of_faces = 4, sum = 6;
     int ans = get_ways(num_of_dice, num_of_faces, sum);
     printf("%d",ans);
     return 0;
}
```

a) 10
b) 12
c) 14
d) 16

Answer: a
Explanation: The output of the above code is 10.

12. What will be the value stored in arr[2][2] when the following code is executed?

```
#include<stdio.h>
int get_ways(int num_of_dice, int num_of_faces, int S)
{
      int arr[num_of_dice + 1][S + 1];
      int dice, face, sm;
      for(dice = 0; dice <= num_of_dice; dice++)
         for(sm = 0; sm <= S; sm++)
           arr[dice][sm] = 0;
      for(sm = 1; sm <= S; sm++)
           arr[1][sm] = 1;
      for(dice = 2; dice <= num_of_dice; dice++)
      {
          for(sm = 1; sm <= S; sm++)
          {
              for(face = 1; face <= num_of_faces && face < sm; face++)
                  arr[dice][sm] += arr[dice - 1][sm - face];
          }
      }
```

```
      return arr[num_of_dice][S];
}
int main()
{
      int num_of_dice = 3, num_of_faces = 4, sum = 6;
      int ans = get_ways(num_of_dice, num_of_faces, sum);
      printf("%d",ans);
      return 0;
}
```

a) 0
b) 1
c) 2
d) 3

Answer: b
Explanation: The value stored in arr[2][2] is 1.

13. What is the output of the following code?

```
#include<stdio.h>
int get_ways(int num_of_dice, int num_of_faces, int S)
{
     int arr[num_of_dice + 1][S + 1];
     int dice, face, sm;
     for(dice = 0; dice <= num_of_dice; dice++)
         for(sm = 0; sm <= S; sm++)
           arr[dice][sm] = 0;
     for(sm = 1; sm <= S; sm++)
         arr[1][sm] = 1;
     for(dice = 2; dice <= num_of_dice; dice++)
     {
         for(sm = 1; sm <= S; sm++)
         {
             for(face = 1; face <= num_of_faces && face < sm; face++)
                 arr[dice][sm] += arr[dice - 1][sm - face];
         }
     }
     return arr[num_of_dice][S];
}
int main()
{
     int num_of_dice = 4, num_of_faces = 6, sum = 3;
     int ans = get_ways(num_of_dice, num_of_faces, sum);
     printf("%d",ans);
     return 0;
}
```

a) 0
b) 1
c) 2
d) 3

Answer: a
Explanation: The minimum possible sum is 4. So, the output for sum = 3 is 0.

14. What is the output of the following code?

```
#include<stdio.h>
int get_ways(int num_of_dice, int num_of_faces, int S)
{
      int arr[num_of_dice + 1][S + 1];
      int dice, face, sm;
      for(dice = 0; dice <= num_of_dice; dice++)
          for(sm = 0; sm <= S; sm++)
            arr[dice][sm] = 0;
```

```
      for(sm = 1; sm <= S; sm++)
         arr[1][sm] = 1;
      for(dice = 2; dice <= num_of_dice; dice++)
      {
          for(sm = 1; sm <= S; sm++)
          {
              for(face = 1; face <= num_of_faces && face < sm; face++)
                 arr[dice][sm] += arr[dice - 1][sm - face];
          }
      }
      return arr[num_of_dice][S];
}
int main()
{
      int num_of_dice = 2, num_of_faces = 6, sum = 5;
      int ans = get_ways(num_of_dice, num_of_faces, sum);
      printf("%d",ans);
      return 0;
}
```

a) 2
b) 3
c) 4
d) 5

Answer: c
Explanation: The output of the above code is 4.

1. You are given a boolean expression which consists of operators &, | and ∧ (AND, OR and XOR) and symbols T or F (true or false). You have to find the number of ways in which the symbols can be parenthesized so that the expression evaluates to true. This is the boolean parenthesization problem. Which of the following methods can be used to solve the problem?
a) Dynamic programming
b) Recursion
c) Brute force
d) Dynamic programming, Recursion and Brute force

Answer: d
Explanation: Dynamic programming, Recursion and Brute force can be used to solve the Boolean parenthesization problem.

2. Consider the expression T & F | T. What is the number of ways in which the expression can be parenthesized so that the output is T (true)?
a) 0
b) 1
c) 2
d) 3

Answer: c
Explanation: The expression can be parenthesized as T & (F | T) and (T & F) | T so that the output is T.

3. Consider the expression T & F ∧ T. What is the number of ways in which the expression can be parenthesized so that the output is T (true)?
a) 0
b) 1
c) 2
d) 3

Answer: c
Explanation: The expression can be parenthesized as (T & F) ∧ T or T & (F ∧ T), so that the output is T.

4. Consider the expression T | F ∧ T. In how many ways can the expression be parenthesized so that the output is F (false)?
a) 0
b) 1
c) 2
d) 3

Answer: b
Explanation: The expression can be parenthesized as (T | F) ∧ T, so that the output is F (false).

5. Which of the following gives the total number of ways of parenthesizing an expression with n + 1 terms?
a) n factorial
b) n square
c) n cube
d) nth catalan number

Answer: d
Explanation: The nth Catalan number gives the total number of ways of parenthesizing an expression with n + 1 terms.

6. What is the maximum number of ways in which a boolean expression with n + 1 terms can be parenthesized, such that the output is true?
a) nth catalan number
b) n factorial
c) n cube
d) n square

Answer: a
Explanation: The number of ways will be maximum when all the possible parenthesizations result in a true value. The number of possible parenthesizations is given by the nth catalan number.

7. Consider the following dynamic programming implementation of the boolean parenthesization problem:

```
int count_bool_parenthesization(char *sym, char *op)
{
      int str_len = strlen(sym);
      int True[str_len][str_len],False[str_len][str_len];
      int row,col,length,l;
      for(row = 0, col = 0; row < str_len; row++,col++)
      {
          if(sym[row] == 'T')
          {
              True[row][col] = 1;
              False[row][col] = 0;
          }
          else
          {
              True[row][col] = 0;
              False[row][col] = 1;
          }
      }
      for(length = 1; length < str_len; length++)
      {
          for(row = 0, col = length; col < str_len; col++, row++)
          {
              True[row][col] = 0;
              False[row][col] = 0;
              for(l = 0; l < length; l++)
              {
                  int pos = row + l;
                  int t_row_pos = True[row][pos] + False[row][pos];
                  int t_pos_col = True[pos+1][col] + False[pos+1][col];
                  if(op[pos] == '|')
                  {
```

```
                    _________________;
                }
                if(op[pos] == '&')
                {
                    _________________;
                }
                if(op[pos] == '^')
                {
                    _________________;
                }
            }
        }
    }
    return True[0][str_len-1];
}
```

Answer: d
Explanation: The following lines should be added:
*False[row][col] += False[row][pos] * False[pos+1][col];*
*True[row][col] += t_row_pos * t_pos_col + False[row][pos] * False[pos+1][col];*

8. Which of the following lines should be added to complete the "if(op[k] == '&')" part of the following code?

```
int count_bool_parenthesization(char *sym, char *op)
{
    int str_len = strlen(sym);
    int True[str_len][str_len],False[str_len][str_len];
    int row,col,length,l;
    for(row = 0, col = 0; row < str_len; row++,col++)
    {
        if(sym[row] == 'T')
        {
            True[row][col] = 1;
            False[row][col] = 0;
        }
        else
        {
            True[row][col] = 0;
            False[row][col] = 1;
        }
    }
    for(length = 1; length < str_len; length++)
    {
        for(row = 0, col = length; col < str_len; col++, row++)
        {
            True[row][col] = 0;
            False[row][col] = 0;
            for(l = 0; l < length; l++)
            {
                int pos = row + l;
                int t_row_pos = True[row][pos] + False[row][pos];
                int t_pos_col = True[pos+1][col] + False[pos+1][col];
                if(op[pos] == '|')
                {
                    _________________;
                }
                if(op[pos] == '&')
                {
                    _________________;
                }
                if(op[pos] == '^')
                {
                    _________________;
                }
            }
        }
    }
    return True[0][str_len-1];
}
```

Answer: b
Explanation: The following lines should be added:
*True[row][col] += True[row][pos] * True[pos+1][col];*
*False[row][col] += t_row_pos * t_pos_col – True[row][pos] * True[pos+1][col];*

9. What is the time complexity of the following dynamic programming implementation of the boolean parenthesization problem?

```
int count_bool_parenthesization(char *sym, char *op)
{
      int str_len = strlen(sym);
      int True[str_len][str_len],False[str_len][str_len];
      int row,col,length,l;
      for(row = 0, col = 0; row < str_len; row++,col++)
      {
          if(sym[row] == 'T')
          {
              True[row][col] = 1;
              False[row][col] = 0;
          }
          else
          {
              True[row][col] = 0;
              False[row][col] = 1;
          }
      }
      for(length = 1; length < str_len; length++)
      {
          for(row = 0, col = length; col < str_len; col++, row++)
          {
              True[row][col] = 0;
              False[row][col] = 0;
              for(l = 0; l < length; l++)
              {
                  int pos = row + l;
                  int t_row_pos = True[row][pos] + False[row][pos];
                  int t_pos_col = True[pos+1][col] + False[pos+1][col];
                  if(op[pos] == '|')
                  {
                      False[row][col] += False[row][pos] * False[pos+1][col];
                      True[row][col] += t_row_pos * t_pos_col - False[row][pos] * False[pos+1][col
                  }
                  if(op[pos] == '&')
                  {
                     True[row][col] += True[row][pos] * True[pos+1][col];
                     False[row][col] += t_row_pos * t_pos_col - True[row][pos] * True[pos+1][col];
                  }
                  if(op[pos] == '^')
                  {
                     True[row][col] += True[row][pos] * False[pos+1][col]
                                       + False[row][pos] * True[pos + 1][col];
                     False[row][col] += True[row][pos] * True[pos+1][col]
                                        + False[row][pos] * False[pos+1][col];
                  }
              }
          }
      }
      return True[0][str_len-1];
}
```

a) O(1)
b) O(n)
c) O(n^2)
d) O(n^3)

Answer: d

Explanation: The time complexity of the above dynamic programming implementation is $O(n^3)$.

10. What is the space complexity of the following dynamic programming implementation of the boolean parenthesization problem?

```
int count_bool_parenthesization(char *sym, char *op)
{
      int str_len = strlen(sym);
      int True[str_len][str_len],False[str_len][str_len];
      int row,col,length,l;
      for(row = 0, col = 0; row < str_len; row++,col++)
      {
          if(sym[row] == 'T')
          {
              True[row][col] = 1;
              False[row][col] = 0;
          }
          else
          {
              True[row][col] = 0;
              False[row][col] = 1;
          }
      }
      for(length = 1; length < str_len; length++)
      {
          for(row = 0, col = length; col < str_len; col++, row++)
          {
              True[row][col] = 0;
              False[row][col] = 0;
              for(l = 0; l < length; l++)
              {
                  int pos = row + l;
                  int t_row_pos = True[row][pos] + False[row][pos];
                  int t_pos_col = True[pos+1][col] + False[pos+1][col];
                  if(op[pos] == '|')
                  {
                      False[row][col] += False[row][pos] * False[pos+1][col];
                      True[row][col] += t_row_pos * t_pos_col - False[row][pos] * False[pos+1][col
                  }
                  if(op[pos] == '&')
                  {
                     True[row][col] += True[row][pos] * True[pos+1][col];
                     False[row][col] += t_row_pos * t_pos_col - True[row][pos] * True[pos+1][col];
                  }
                  if(op[pos] == '^')
                  {
                     True[row][col] += True[row][pos] * False[pos+1][col]
                                       + False[row][pos] * True[pos + 1][col];
                     False[row][col] += True[row][pos] * True[pos+1][col]
                                       + False[row][pos] * False[pos+1][col];
                  }
              }
          }
      }
      return True[0][str_len-1];
}
```

a) O(1)
b) O(n)
c) $O(n^2)$
d) $O(n^3)$

Answer: c
Explanation: The space complexity of the above dynamic programming implementation is $O(n^2)$.

11. What is the output of the following code?

```
#include<stdio.h>
#include<string.h>
int count_bool_parenthesization(char *sym, char *op)
{
     int str_len = strlen(sym);
     int True[str_len][str_len],False[str_len][str_len];
     int row,col,length,l;
     for(row = 0, col = 0; row < str_len; row++,col++)
     {
         if(sym[row] == 'T')
         {
             True[row][col] = 1;
             False[row][col] = 0;
         }
         else
         {
             True[row][col] = 0;
             False[row][col] = 1;
         }
     }
     for(length = 1; length < str_len; length++)
     {
         for(row = 0, col = length; col < str_len; col++, row++)
         {
             True[row][col] = 0;
             False[row][col] = 0;
             for(l = 0; l < length; l++)
             {
                 int pos = row + l;
                 int t_row_pos = True[row][pos] + False[row][pos];
                 int t_pos_col = True[pos+1][col] + False[pos+1][col];
                 if(op[pos] == '|')
                 {
                     False[row][col] += False[row][pos] * False[pos+1][col];
                     True[row][col] += t_row_pos * t_pos_col - False[row][pos] * False[pos+1][col]
                 }
                 if(op[pos] == '&')
                 {
                     True[row][col] += True[row][pos] * True[pos+1][col];
                     False[row][col] += t_row_pos * t_pos_col - True[row][pos] * True[pos+1][col];
                 }
                 if(op[pos] == '^')
                 {
                     True[row][col] += True[row][pos] * False[pos+1][col]
                                       + False[row][pos] * True[pos + 1][col];
                     False[row][col] += True[row][pos] * True[pos+1][col]
                                       + False[row][pos] * False[pos+1][col];
                 }
             }
         }
     }
     return True[0][str_len-1];
}
int main()
{
     char sym[] = "TTTT";
     char op[] = "|^^";
     int ans = count_bool_parenthesization(sym,op);
     printf("%d",ans);
     return 0;
}
```

a) 1
b) 2
c) 3
d) 4

Answer: d

Explanation: The output of the above program is 4.

1. What is the meaning of cipher in cryptography?
a) an algorithm that performs encryption
b) an algorithm that generates a secret code
c) an algorithm that performs encryption or decryption
d) a secret code

Answer: c
Explanation: Cipher is an algorithm for performing encryption or decryption. In cryptography, a set of defined steps are followed to generate ciphers.

2. Which of the following is a type of traditional cipher?
a) transportation cipher
b) transposition cipher
c) transforming cipher
d) vigenere cipher

Answer: b
Explanation: There are two types of a traditional cipher. First is transposition cipher and the second is substitution cipher.

3. Which of the following ciphers are created by shuffling the letters of a word?
a) substitution cipher
b) transposition cipher
c) vigenere cipher
d) hill cipher

Answer: b
Explanation: There are two types of traditional ciphers – Transposition and substitution cipher. In transposition cipher the letters of the given data are shuffled in a particular order, fixed by a given rule.

4. Which of the following is a type of substitution cipher?
a) Mono alphabetic cipher
b) transposition cipher
c) transportation cipher
d) transforming cipher

Answer: a
Explanation: In substitution cipher the plain text is replaced by cipher text according to a fixed rule. There are two types of substitution cipher – Mono alphabetic and Poly alphabetic cipher.

5. Which of the following is not a type of mono alphabetic cipher?
a) additive cipher
b) multiplicative cipher
c) afffine cipher
d) hill cipher

Answer: d
Explanation: In mono alphabetic cipher each symbol of plain text is replaced by a particular respective symbol in the cipher text. There are three types of mono alphabetic ciphers- additive, multiplicative and affine.

6. Which of the following is not a type of poly alphabetic cipher?
a) Auto key cipher
b) Hill cipher
c) Playfair cipher
d) Additive cipher

Answer:d
Explanation: In poly alphabetic cipher each symbol of plain text is replaced by a different cipher text regardless of its occurrence. Out of the given options, only additive cipher is not a poly alphabetic cipher.

7. What will be the ciphered text for the input string "sanfoundry" with key string as "code" to the program of keyword cipher?
a) SCMBNUMERY
b) SSCMBNUMERY
c) NIFO
d) NILO

Answer: a
Explanation: Keyword cipher is type of mono alphabetic cipher. In this algorithm the letters {A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,W,X,Y,Z} by {C,O,D,E,A,B,F,G,H,I,J,K,L,M,N,P,Q,R,S,T,U,V,W,X,Y,Z} respectively.

8. Which of the following is a type of transposition cipher?
a) Rail Fence cipher
b) Hill cipher
c) Rotor cipher
d) One time pad

Answer: a
Explanation: In transposition cipher the letters of the given data are shuffled in a particular order, fixed by a given rule. There are two types of transposition cipher – Rail fence cipher and Columnar transposition cipher.

9. What will be output for the given code?

```
#include<bits/stdc++.h>
using namespace std;
string encrypter(string keyword)
{
        string encoded = "";
        bool arr[26] = {0};
        for (int i=0; i<keyword.size(); i++)
        {
                if(keyword[i] >= 'A' && keyword[i] <= 'Z')
                {
                        if (arr[keyword[i]-65] == 0)
                        {
                                encoded += keyword[i];
                                arr[keyword[i]-65] = 1;
                        }
                }
                else if (keyword[i] >= 'a' && keyword[i] <= 'z')
                {
                        if (arr[keyword[i]-97] == 0)
                        {
                                encoded += keyword[i] - 32;
                                alpha[keyword[i]-97] = 1;
                        }
                }
        }
        for (int i=0; i<26; i++)
        {
                if(arr[i] == 0)
                {
                        arr[i]=1;
                        encoded += char(i + 65);
                }
        }
        return encoded;
}
string ciphertxt(string msg, string encoded)
```

```
{
        string cipher="";
        for (int i=0; i<msg.size(); i++)
        {
                if (msg[i] >='a' && msg[i] <='z')
                {
                        int pos = msg[i] - 97;
                        cipher += encoded[pos];
                }
                else if (msg[i] >='A' && msg[i] <='Z')
                {
                        int pos = msg[i] - 65;
                        cipher += encoded[pos];
                }
                else
                {
                        cipher += msg[i];
                }
        }
        return cipher;
}
int main()
{
        string keyword;
        keyword = "cipher";
        string encoded = encrypter(keyword);
        string message = "hello";
        cout  << ciphertxt(message,encoded) << endl;
        return 0;
}
```

a) bejjm
b) LFPDAR
c) BEJJM
d) lfpdar

Answer: c
Explanation: The given code is the implementation of keyword cipher. It is an example of mono alphabetic cipher. The given string is always converted into an uppercase ciphered text.

10. What will be output for the given code taking input string as “sanfoundry”?

```
package com.sanfoundry.setandstring;
import java.util.Scanner;
public class MonoalphabeticCipher
{
      public static char p[]  = { 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j',
                                  'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't',
                                  'u', 'v', 'w', 'x', 'y', 'z' };
      public static char ch[] = { 'Q', 'W', 'E', 'R', 'T', 'Y', 'U', 'I', 'O', 'P',
                                  'A', 'S', 'D', 'F', 'G', 'H', 'J', 'K', 'L', 'Z',
                                  'X', 'C', 'V', 'B', 'N', 'M' };
      public static String doEncryption(String s)
      {
           char c[] = new char[(s.length())];
           for (int i = 0; i < s.length(); i++)
           {
                for (int j = 0; j < 26; j++)
                {
                     if (p[j] == s.charAt(i))
                     {
                         c[i] = ch[j];
                          break;
                     }
                }
            }  return (new String(c));
        }
        public static void main(String args[])
```

```
        {
            Scanner sc = new Scanner(System.in);
            System.out.println("Enter the message: ");
            String en = doEncryption(sc.next().toLowerCase());
            System.out.println("Encrypted message: " + en);
            sc.close();
        }
}
```

a) Encrypted message: LQFYGXFRKN
b) Encrypted message: NKRFXGYFQL
c) Encrypted message: lqfygxfrkn
d) Encrypted message: nkrfxgyfql

Answer: a
Explanation: The given code is an example of mono-alphabetic cipher. The code replaces the letters of the input string by corresponding keyboard letters.

11. In which of the following cipher the plain text and the ciphered text does not have same number of letters?
a) keyword cipher
b) vigenere cipher
c) transposition cipher
d) additive cipher

Answer: b
Explanation: In transposition cipher and mono alphabetic cipher the number of letters in the plain text and ciphered text remain same. But in poly alphabetic cipher the number of letters change. So here as vigenere cipher is the only poly alphabetic cipher so it will be the answer.

1. What is the meaning of cipher in cryptography?
a) an algorithm that performs encryption
b) an algorithm that generates a secret code
c) an algorithm that performs encryption or decryption
d) a secret code

Answer: c
Explanation: Cipher is an algorithm for performing encryption or decryption. In cryptography, a set of defined steps are followed to generate ciphers. These are necessary to prevent data breach.

2. Which of the following ciphers are created by shuffling the letters of a word?
a) rotor cipher
b) rail fence cipher
c) vigenere cipher
d) hill cipher

Answer: b
Explanation: There are two types of traditional ciphers- Transposition and substitution cipher. In transposition cipher the letters of the given data are shuffled in a particular order, fixed by a given rule. Rail fence cipher is an example of transposition cipher.

3. Which of the following is a type of substitution cipher?
a) poly alphabetic cipher
b) transposition cipher
c) columnar cipher
d) rail fence cipher

Answer: a
Explanation: In substitution cipher the plain text is replaced by cipher text according to a fixed rule. There are two types of substitution cipher- Mono alphabetic and Poly alphabetic cipher.

4. Which of the following correctly defines poly alphabetic cipher?
a) a substitution based cipher which uses multiple substitution at different positions
b) a substitution based cipher which uses fixed substitution over entire message
c) a transposition based cipher which uses multiple substitution at different positions
d) a transposition based cipher which uses fixed substitution over entire message

Answer: a
Explanation: Poly alphabetic cipher is a type of substitution cipher. It uses multiple substitution at different positions in order to cipher the plain text.

5. Which of the following is not a type of poly alphabetic cipher?
a) Rotor cipher
b) Hill cipher
c) One time pad cipher
d) Affine cipher

Answer: d
Explanation: In poly alphabetic cipher each symbol of plain text is replaced by a different cipher text regardless of its occurrence. Out of the given options, only affine cipher is not a poly alphabetic cipher.

6. Which of the following is a type of traditional cipher?
a) transportation cipher
b) transposition cipher
c) transforming cipher
d) translating cipher

Answer: b
Explanation: There are two types of a traditional cipher. First is transposition cipher and the second is substitution cipher. Answer: a
Explanation: Mono alphabetic ciphers can be decoded by using the method frequency analysis. But in poly alphabetic cipher each symbol of plain text is replaced by a different cipher text regardless of its occurrence. This makes it very difficult to be decoded by using frequency analysis.

8. Which cipher is represented by the following function?

```
void Cipher(string msg, string key)
{
        // Get key matrix from the key string
        int keyMat[3][3];
        getKeyMatrix(key, keyMat);
        int msgVector[3][1];
        for (int i = 0; i <=2; i++)
                msgVector[i][0] = (msg[i]) % 65;
        int cipherMat[3][1];
        // Following function generates
        // the encrypted vector
        encrypt(cipherMat, keyMat, msgVector);
        string CipherText;
        for (int i = 0; i <=2; i++)
                CipherText += cipherMat[i][0] + 65;
        cout  << CipherText;
}
```

a) vigenere cipher
b) hill cipher
c) keyword cipher
d) rotor cipher

Answer: b
Explanation: The given function represents hill cipher. It is a type of poly alphabetic substitution which is based on linear algebra.

9. Which cipher is represented by the following function?

```
public class Cipher
{
    public static String encrypt(String text, final String key)
    {
        String res = "";
        text = text.toUpperCase();
        for (int i = 0, j = 0; i < text.length(); i++)
        {
            char c = text.charAt(i);
            if (c < 'A' || c > 'Z')
                continue;
            res += (char) ((c + key.charAt(j) - 2 * 'A') % 26 + 'A');
            j = ++j % key.length();
        }
        return res;
    }
```

a) vigenere cipher
b) hill cipher
c) keyword cipher
d) rotor cipher

Answer: a
Explanation: The given function represents the function of hill cipher. It is a type of poly alphabetic substitution which makes use of the vigenere table for making substitutions in the plain text.

10. In which of the following cipher the plain text and the ciphered text does not have same number of letters?
a) affine cipher
b) hill cipher
c) columnar cipher
d) additive cipher

Answer: b
Explanation: In transposition cipher and mono alphabetic cipher the number of letters remains the same in ciphered and deciphered text. But in poly alphabetic cipher the number of letters are different. So here as hill cipher is the only poly alphabetic cipher so it will be the answer.

11. What will be the ciphered text if the string "SANFOUNDRY" is given as input to the code of vigenere cipher with keyword as "HELLO"?
a) UEWIIDKLL
b) ZEYQCOCM
c) ZEYQCBROCM
d) ZEYQCBROCMJDH

Answer: c
Explanation: Vigenere cipher is a type of poly alphabetic substitution which uses vigenere table for making substitutions in the plain text. Using the table we find the solution to be ZEYQCBROCM.

12. What will be the ciphered text if the string "HELLO" is given as input to the code of hill cipher with keyword as "SANFOUNDRY"?
a) EZJ
b) JEZ
c) ZEJ
d) JZE

Answer: d
Explanation: Hill cipher is a type of poly alphabetic substitution. It uses a n x n matrix in order to cipher the given plain text. Using this method we find the ciphered text to be JZE.

1. What is the meaning of cipher in computer terminology?
a) an algorithm that performs encryption
b) an algorithm that generates a secret code
c) an algorithm that performs encryption or decryption
d) a secret code

Answer: c
Explanation: Cipher can be defined to be an algorithm that performs encryption or decryption. In cryptography, a set of defined steps are followed to generate ciphers.

2. Which of the following cipher is created by shuffling the letters of a word?
a) substitution cipher
b) transposition cipher
c) mono alphabetic cipher
d) poly alphabetic cipher

Answer: b
Explanation: Types of traditional ciphers- Transposition and substitution cipher. In transposition cipher the letters of the given message are shuffled in a particular order, fixed by a given rule.

3. Which of the following is not a type of transposition cipher?
a) Rail fence cipher
b) Columnar transposition cipher
c) One time pad cipher
d) Route cipher

Answer: c
Explanation: Out of the given options only One time pad cipher is not a type of transposition cipher. It is a type of substitution cipher.

4. Which of the following is not a type of mono alphabetic cipher?
a) additive cipher
b) multiplicative cipher
c) afffine cipher
d) hill cipher

Answer: d
Explanation: In mono alphabetic cipher each symbol of plain text is replaced by a particular respective symbol in the cipher text. There are three types of mono alphabetic ciphers- additive, multiplicative and affine.

5. Route cipher falls under the category of?
a) mono-alphabetic cipher
b) poly-alphabetic cipher
c) transposition cipher
d) additive cipher

Answer: c
Explanation: Route cipher is a transposition cipher. It falls under the category of transposition cipher as it encrypts the plain text by rearranging its letters.

6. Which of the following ciphered text would have used transposition cipher for encryption of the plain text "SANFOUNDRY"?
a) SSCMBNUMERY
b) TBMGPVOESZ
c) UCNHQWPFTA
d) SNONRAFUDY

Answer: d

Explanation: We know that transposition cipher encrypts the plain text by shuffling the letters of the plain text. So out of the given options, only "SNONRAFUDY" has the same set of letters as "SANFOUNDRY".

7. Which of the following is a type of transposition cipher?
a) Rail Fence cipher
b) Hill cipher
c) Rotor cipher
d) One time pad

Answer: a
Explanation: In transposition cipher the letters of the given message are shuffled in a particular order, fixed by a given rule. Two types of transposition cipher are – Rail fence cipher and Columnar transposition cipher.

8. In which of the following cipher the plain text and the ciphered text have same set of letters?
a) one time pad cipher
b) columnar transposition cipher
c) playfair cipher
d) additive cipher

Answer: b
Explanation: In transposition cipher, the letters remain same in ciphered and plain text. Their position is only changed whereas in substitution cipher the letters become different in encrypted text. So columnar transposition cipher will be the correct option.Answer: a
Explanation: Combining transposition cipher with substitution cipher helps in overcoming its weaknesses. But it causes the cipher to become more laborious to decode and it becomes more prone to errors.

10. What will be the encrypted text corresponding to plain text "SANFOUNDRY" using rail fence cipher with key value given to be 2?
a) SNONRAFUDY
b) SORAFUDYNN
c) SNAUDNORFY
d) SANFOUNDRY

Answer: a
Explanation: For encrypting a plain text using rail fence cipher we use a table having a number of rows equal to key value as shown below. Then we read along the rows to get the ciphered text. So the ciphered text will be "SNONRAFUDY".

11. What will be the encrypted text corresponding to plain text "SANFOUNDRY" using columnar transposition cipher with the keyword as "GAMES"?
a) SNONRAFUDY
b) SORAFUDYNN
c) SNAUDNORFY
d) ANFRSUNDOY

Answer: d
Explanation: For encrypting using columnar cipher we have to arrange the letters of the plain text in a table which has the same number of columns as the letters of the keyword. Then the letters of the keyword are arranged in alphabetical order and we read along each column.
3 1 4 2 5
G A M E S
S A N F O
U N D R Y
So the ciphered text will be "ANFRSUNDOY".

1. Which of the following cipher replaces letters with symbols?
a) polybius square ciper

b) affine cipher
c) caesar cipher
d) pigpen cipher

Answer: d
Explanation: Pigpen cipher encrypts the plain text by replacing each letter with a corresponding symbol. Pigpen cipher is an example of geometric substitution cipher.

2. Which of the following is not an alternative name of pigpen cipher?
a) pen cipher
b) freemason's cipher
c) napoleon cipher
d) tic-tac-toe cipher

Answer: a
Explanation: Pigpen cipher is also known by the names like:- freemason's cipher, napoleon cipher, tic-tac-toe cipher, masonic cipher. It is a cipher which is believed to be used by freemasons in the 18th century.

3. Which of the following cipher was used by freemasons?
a) Vigenere cipher
b) Autokey cipher
c) Pigpen cipher
d) Hill cipher

Answer: c
Explanation: Pigpen cipher is believed to be used by freemasons in the 18th century. Pigpen cipher is an example of geometric substitution cipher.

4. Choose the weakest cipher from the following?
a) vigenere cipher
b) rail fence cipher
c) hill cipher
d) pigpen cipher

Answer: d
Explanation: Pigpen cipher is the weakest cipher out of the given options. This is because it is a simple geometric substitution cipher so can be easily cracked using frequency analysis.

5. Which of the following is not a poly alphabetic substitution cipher?
a) vigenere cipher
b) one time pad cipher
c) play fair cipher
d) pigpen cipher

Answer: d
Explanation: Pigpen cipher is the only non poly alphabetic substitution cipher out of the given options. It is an example of geometric substitution cipher.

6. Pigpen cipher is an example of?
a) Mono alphabetic substitution cipher
b) Transposition cipher
c) Poly alphabetic substitution cipher
d) Geometric substitution cipher

Answer: a
Explanation: Pigpen cipher encrypts the plain text by replacing each letter with a corresponding symbol. Pigpen cipher is an example of geometric substitution cipher.Answer: a
Explanation: Vigenere cipher is more secure as compared to pigpen cipher. It is because pigpen cipher is geometric

substitution cipher and vigenere cipher is poly alphabetic substitution cipher.Answer: b
Explanation: Pigpen cipher is a very weak cipher as it just replaces letters with corresponding symbols. It can be easily broken using frequency analysis.

9. What will be the ciphered text corresponding to plain text "ACT" if pigpen cipher is used for encryption?
a) _| |_ >
b) |_ > |_
c) _| |_<
d) |_< _|

Answer: a
Explanation: Pigpen cipher replaces letters of the plain text with corresponding symbols. These symbols are fragment of a grid.

10. What will be the plain text corresponding to ciphered text "|_ _| >" if pigpen cipher is used for encryption?
a) cat
b) hat
c) dog
d) rat

Answer: a
Explanation: Pigpen cipher replaces letters of the plain text with corresponding symbols. So by using the grid we can replace these symbols by their corresponding letters which are "cat" here.

11. What is common between affine cipher and pigpen cipher.
a) both are mono alphabetic substitution cipher
b) both are poly alphabetic substitution cipher
c) both can be cracked using frequency analysis
d) both are transposition cipher

Answer: c
Explanation: Both affine cipher and pigpen cipher can be broken using frequency analysis. Pigpen cipher is an example of geometric substitution cipher.

1. What is the meaning of cipher in cryptography?
a) an algorithm that performs encryption
b) an algorithm that generates a secret code
c) an algorithm that performs encryption or decryption
d) a secret code

Answer: c
Explanation: Cipher is an algorithm for performing encryption or decryption. In cryptography, a set of defined steps are followed to generate ciphers. These are necessary to prevent data breach.

2. Vigenere cipher is an example of _____
a) mono-alphabetic cipher
b) poly-alphabetic cipher
c) transposition cipher
d) additive cipher

Answer: b
Explanation: Vigenere cipher is a substitution cipher. It falls under the category of poly alphabetic cipher as it uses multiple substitution at different positions in order to cipher the plain text.

3. Encryption in Vigenere cipher is done using _____
a) vigenere formula
b) vigenere cycle
c) vigenere square

d) vigenere addition

Answer: c
Explanation: Encryption of plain text using vigenre cipher is done by making use of vigenere table. It is also known as vigenere square.

4. Which of the following correctly defines poly alphabetic cipher?
a) a substitution based cipher which uses multiple substitution at different positions
b) a substitution based cipher which uses fixed substitution over entire message
c) a transposition based cipher which uses multiple substitution at different positions
d) a transposition based cipher which uses fixed substitution over entire message

Answer: a
Explanation: Poly alphabetic cipher is a type of substitution cipher. It uses multiple substitution at different positions in order to cipher the plain text.

5. Which of the following is not a type of poly alphabetic cipher?
a) Rotor cipher
b) Hill cipher
c) One time pad cipher
d) Multiplicative cipher

Answer:d
Explanation: In poly alphabetic cipher each symbol of plain text is replaced by a different cipher text regardless of its occurrence. Out of the given options only multiplicative cipher is not a poly alphabetic cipher.

6. Vigenere table consists of _____
a) 26 rows and 26 columns
b) 26 rows and 1 column
c) 1 row and 26 columns
d) 27 rows and 27 columns

Answer: a
Explanation: Encryption of plain text using vigenre cipher is done by making use of vigenere table. It consists of 26 rows and 26 columns which have alphabets written 26 times. These are shifted towards left after each row.Answer: a
Explanation: Keyword cipher is less secure than vigenere cipher. It is due to the fact that keyword cipher is mono alphabetic and thus can be cracked using frequency analysis. But vigenere cipher being a poly alphabetic cipher cannot be deciphered using this method.

8. What will be the plain text corresponding to cipher text "PROTO" if vigenere cipher is used with keyword as "HELLO"?
a) SANFOUNDRY
b) WORLD
c) INDIA
d) AMERICA

Answer: c
Explanation: Vigenere cipher is a type of poly alphabetic substitution which uses vigenere table for making substitutions in the plain text. Using the table we find the plain text to be "INDIA".

9. Which cipher is represented by the following function?

```
public class Cipher
{
    public static String encrypt(String text, final String key)
    {
        String res = "";
        text = text.toUpperCase();
        for (int i = 0, j = 0; i < text.length(); i++)
```

```
        {
            char c = text.charAt(i);
            if (c < 'A' || c > 'Z')
                continue;
            res += (char) ((c + key.charAt(j) - 2 * 'A') % 26 + 'A');
            j = ++j % key.length();
        }
        return res;
    }
```

a) vigenere cipher
b) hill cipher
c) keyword cipher
d) rotor cipher

Answer: a
Explanation: The given function represents the function of hill cipher. It is a type of poly alphabetic substitution which makes use of the vigenere table for making substitutions in the plain text.

10. In which of the following cipher the plain text and the ciphered text does not have a same number of letters?
a) affine cipher
b) vigenere cipher
c) columnar cipher
d) additive cipher

Answer: b
Explanation: In transposition cipher and mono alphabetic cipher the number of letters remain same in ciphered and deciphered text. But in poly alphabetic cipher the number of letters are different. So here as vigenere cipher is the only poly alphabetic cipher so it will be the answer.

11. What will be the ciphered text if the string "SANFOUNDRY" is given as input to the code of vigenere cipher with keyword as "HELLO"?
a) UEWIIDKLL
b) ZEYQCOCM
c) ZEYQCBROCM
d) ZEYQCBROCMJDH

Answer: c
Explanation: Vigenere cipher is a type of poly alphabetic substitution which uses vigenere table for making substitutions in the plain text. Using the table we find the solution to be ZEYQCBROCM.

1. Which of the following cipher does not require a key for encrypting plain text?
a) atbash cipher
b) affine cipher
c) playfair cipher
d) vigenere cipher

Answer: a
Explanation: Out of the given options only atbash cipher does not require a key for encryption. Atbash cipher is an example of a substitution cipher.

2. Atbash cipher was originally used for encrypting _____
a) english alphabet
b) greek alphabet
c) hebrew alphabet
d) hindi alphabet

Answer: c
Explanation: Atbash cipher was originally built to encrypt hebrew alphabet. But it can be used to encrypt any type of

alphabets.

3. Which of the following cipher is a special case of affine cipher?
a) Vigenere cipher
b) Autokey cipher
c) Atbash cipher
d) Hill cipher

Answer: c
Explanation: Atbash cipher is a special case of affine cipher. We have to assume a=b=25 as the key in affine cipher to get the same encryption.

4. Choose the weakest cipher from the following?
a) vigenere cipher
b) rail fence cipher
c) hill cipher
d) atbash cipher

Answer: d
Explanation: Atbash cipher is the weakest cipher out of the given options. This is because it is a simple mono alphabetic substitution cipher with almost no security. It can be cracked without even knowing which technique has been used for encryption.

5. Which of the following is a mono alphabetic substitution cipher?
a) vigenere cipher
b) one time pad cipher
c) play fair cipher
d) atbash cipher

Answer: d
Explanation: Atbash cipher is the only mono alphabetic substitution cipher out of the given options. All the remaining options fall under the category of poly alphabetic cipher.Answer: a
Explanation: Affine cipher is more secure as compared to pigpen cipher. Atbash cipher is a special case of affine cipher and can be cracked without even knowing which technique has been used for encryption.

7. Atbash cipher is an example of?
a) Mono alphabetic substitution cipher
b) Transposition cipher
c) Poly alphabetic substitution cipher
d) Geometric substitution cipher

Answer: a
Explanation: Atbash cipher is an example of mono alphabetic substitution cipher. It replaces each alphabet of the plain text with a corresponding letter which we can see in the table below.

```
A - Z      J - Q      S - H
B - Y      K - P      T - G
C - X      L - O      U - F
D - W      M - N      V - E
E - V      N - M      W - D
F - U      O - L      X - C
G - T      P - K      Y - B
H - S      Q - J      Z - A
I - R      R - I
```

Answer: b
Explanation: Atbash cipher is a very weak cipher. It can be easily broken without even knowing the exact technique which has been used for encryption. We can just assume the ciphered text to be a substitution cipher in order to crack it.

9. What will be the ciphered text corresponding to plain text "SECRET" if atbash cipher is used for encryption?
a) VHIXVG
b) HVXIVG
c) HXVIVG
d) VIHXIV

Answer: b
Explanation: Atbash cipher replaces each alphabet of the plain text with a corresponding letter which we can see in the table below.

```
A - Z      J - Q      S - H
B - Y      K - P      T - G
C - X      L - O      U - F
D - W      M - N      V - E
E - V      N - M      W - D
F - U      O - L      X - C
G - T      P - K      Y - B
H - S      Q - J      Z - A
I - R      R - I
```

So the ciphered text would be "HVXIVG".

10. What will be the plain text corresponding to ciphered text "MAO" if atbash cipher is used for encryption?
a) NZL
b) IND
c) AUS
d) ENG

Answer: a
Explanation: Atbash cipher replaces each alphabet of the plain text with a corresponding letter which we can see in the table below.

```
A - Z      J - Q      S - H
B - Y      K - P      T - G
C - X      L - O      U - F
D - W      M - N      V - E
E - V      N - M      W - D
F - U      O - L      X - C
G - T      P - K      Y - B
H - S      Q - J      Z - A
I - R      R - I
```

So the plain text would be "NZL".

11. Which of the following is not true about atbash cipher?
a) it is a mono alphabetic substitution cipher
b) it can only be used to encrypt hebrew alphabet
c) it is a special case of affine cipher
d) it is weaker than playfair cipher

Answer: b
Explanation: Atbash cipher was originally built to encrypt hebrew alphabet. But it can be used to encrypt any type of alphabets.

1. Gronsfeld cipher is an example of _____
a) mono-alphabetic cipher
b) poly-alphabetic cipher
c) transposition cipher
d) additive cipher

Answer: b

Explanation: Gronsfeld cipher is a substitution cipher. It falls under the category of poly alphabetic cipher as it uses multiple substitutions at different positions in order to cipher the plain text.

2. Which of the following is true about encryption in gronsfeld cipher?
a) It replaces letters with numbers
b) It shifts the letters of plain text on the basis of key value
c) It makes substitutions by using tabula recta
d) It makes substitutions on the basis of a mathematical formula

Answer: b
Explanation: Encryption of plain text in gronsfeld cipher is done by shifting the letters of plain text on the basis of key value. So it is similar to caesar cipher.

3. Gronsfeld cipher is similar to?
a) additive cipher
b) multiplicative cipher
c) caesar cipher
d) affine cipher

Answer: c
Explanation: In gronsfeld cipher we shift the letters of the plain text on the basis of the key value. So it becomes a modified version of the caesar cipher.

4. Which of the following cipher uses a numeric key?
a) gronsfeld cipher
b) vigenere cipher
c) trithemius cipher
d) running key cipher

Answer: a
Explanation: Gronsfeld cipher is a variation of vigenere cipher. The difference is that vigenere cipher uses an alphabetical key whereas gronsfeld cipher uses numeric key.

5. Which of the following cipher does not require the use of tabula recta?
a) running key cipher
b) vigenere cipher
c) gronsfeld cipher
d) trithemius cipher

Answer: c
Explanation: Ciphers like running key cipher, vigenere cipher, trithemius cipher etc. makes use of tabula recta. Whereas gronsfeld cipher does not require tabula recta for encryption of plain text.

6. Gronsfeld cipher is a variation of _____
a) autokey cipher
b) vigenere cipher
c) hill cipher
d) route cipher

Answer: b
Explanation: Gronsfeld cipher is a variation of vigenere cipher. The difference is that vigenere cipher uses an alphabetical key whereas the key of gronsfeld cipher consists of numbers.Answer: a
Explanation: In gronsfeld cipher we shift the letters of the plain text on the basis of the key value. So it becomes a more complex version of the caesar cipher and thus harder to crack.

8. What will be the plain text corresponding to cipher text "EETPEG" if gronsfeld cipher is used with key "4321"?
a) ABACUS
b) ABROAD

c) ABRUPT
d) ABUSED

Answer: b
Explanation: Encryption in gronsfeld cipher takes place by shifting the letters of the plain text by a number given by the numeric key. So by applying the reverse of this method, we get the deciphered text to be "ABROAD".Answer: a
Explanation: Gronsfeld cipher is a type of substitution cipher. It is a variation of vigenere cipher.

10. Which of the following is not a transposition cipher?
a) rail fence cipher
b) gronsfeld cipher
c) hill cipher
d) route cipher

Answer: b
Explanation: Rail fence, hill and route cipher are examples of transposition cipher. Gronsfeld cipher is a substitution cipher and so is the correct option.

11. What will be the ciphered text corresponding to "SANFOUNDRY" if gronsfeld cipher is used for encryption with key as "1234"?
a) TCQJPWQHSA
b) TCQJTAULAI
c) TCQJOUNDRY
d) UETNQYTLTC

Answer: a
Explanation: Encryption in gronsfeld cipher takes place by shifting the letters of the plain text by a number given by the numeric key. So by applying this method, we get the ciphered text to be "TCQJPWQHSA".

12. What will be the ciphered text corresponding to "ALGORITHM" if gronsfeld cipher is used for encryption with key "4321"?
a) BNJSSKWLN
b) EOIPVLVIQ
c) BNJSRITHM
d) EOIPRITHM

Answer: b
Explanation: Encryption in gronsfeld cipher takes place by shifting the letters of the plain text by a number given by the numeric key. So by applying this method, we get the ciphered text to be "EOIPVLVIQ".

13. What will be the plain text corresponding to cipher text "SCEFJV" if gronsfeld cipher is used with key "1234"?
a) RABBIT
b) RUSSIA
c) RANGER
d) FRIEND

Answer: a
Explanation: Encryption in gronsfeld cipher takes place by shifting the letters of the plain text by a number given by the numeric key. So by applying the reverse of this method, we get the deciphered text to be "RABBIT".

1. Beaufort cipher is an example of _____
a) mono-alphabetic cipher
b) poly-alphabetic cipher
c) transposition cipher
d) additive cipher

Answer: b
Explanation: Beaufort cipher is a substitution cipher. It falls under the category of poly alphabetic cipher as it uses

multiple substitutions at different positions in order to cipher the plain text.

2. Encryption in beaufort cipher is done using _____
a) trithemius table
b) vigenere cycle
c) tabula recta
d) four square table

Answer: c
Explanation: Encryption of plain text in beaufort cipher is done by making use of tabula recta. The same table is also used for encryption in vigenere cipher and running key cipher.

3. Which of the following is reciprocal cipher?
a) vigenere cipher
b) autokey cipher
c) running key cipher
d) beaufort cipher

Answer: d
Explanation: A reciprocal cipher is a cipher in which if we try to encrypt the cipher text (using the same cipher) then we gets the plain text as a result. In other words, the process of decryption and encryption is exactly the same.

4. Which of the following is a difference between beaufort cipher and vigenere cipher?
a) they use different tables for encryption
b) vigenere cipher is poly alphabetic whereas beaufort cipher is mono alphabetic
c) vigenere cipher uses a key whereas no key is required for using beaufort cipher
d) they use the same table for encryption, but in a different manner

Answer: d
Explanation: Beaufort cipher is a variation of vigenere cipher. They use the same table i.e. tabula recta for encryption but the table is used in a different manner.

5. Beaufort cipher is a variant of _____
a) autokey cipher
b) vigenere cipher
c) hill cipher
d) route cipher

Answer: b
Explanation: Beaufort cipher is a variant of vigenere cipher. The difference is that beaufort cipher uses tabula recta in a different manner for encryption.Answer: a
Explanation: Beaufort cipher is an example of reciprocal cipher. So that is why the process of decryption is exactly same as that of encryption in beaufort cipher.

7. What will be the plain text corresponding to cipher text "SEDGKG" if beaufort cipher is used with key as "KEY"?
a) PRISON
b) DEATHS
c) SAVEUS
d) ABUSED

Answer: c
Explanation: The process of decryption and encryption are exactly the same in beaufort cipher. So the corresponding plain text would be "SAVEUS".Answer: b
Explanation: Beaufort cipher is different from variant beaufort cipher. Variant beaufort cipher is a variant of beaufort cipher.

9. What will be the ciphered text corresponding to "SANFOUNDRY" if beaufort cipher is used for encryption with key as "KEY"?

a) SBPISZTKZH
b) TCQJTAULAI
c) SELFQEXBHM
d) SPBISZKTZH

Answer: c
Explanation: For encrypting plain text choose the first letter of plain text (i.e. S) in the first horizontal row. Then find the first letter of key (i.e. K) in the column obtained in the first step. Finally, choose the leftmost letter of the row (i.e. S) obtained in the previous step. So the corresponding cipher text is "SELFQEXBHM".

10. What will be the ciphered text corresponding to "ALGORITHM" if beaufort cipher is used for encryption with key as "KEY"?
a) BNJSWOAPV
b) KTSWNQRXM
c) AMIRVNZOU
d) MBPHJSNIU

Answer: b
Explanation: For encrypting plain text choose the first letter of plain text (i.e. A) in the first horizontal row. Then find the first letter of key (i.e. K) in the column obtained in the first step. Finally, choose the leftmost letter of the row (i.e. K) obtained in the previous step. So the corresponding cipher text is "KTSWNQRXM".

1. What is the meaning of cipher in cryptography?
a) an algorithm that performs encryption
b) an algorithm that generates a secret code
c) an algorithm that performs encryption or decryption
d) a secret code

Answer: c
Explanation: Cipher is an algorithm for performing encryption or decryption. In cryptography, a set of defined steps are followed to generate ciphers. These are necessary to prevent a data breach.

2. Autokey cipher is an example of _____
a) mono-alphabetic cipher
b) poly-alphabetic cipher
c) transposition cipher
d) additive cipher

Answer: b
Explanation: Autokey cipher is a substitution cipher. It falls under the category of poly alphabetic cipher as it uses multiple substitutions at different positions in order to cipher the plain text.

3. Encryption in Autokey cipher is done using _____
a) vigenere formula
b) vigenere cycle
c) vigenere table
d) vigenere addition

Answer: c
Explanation: Encryption of plain text in Autokey cipher is done using the same table which is used for encryption in vigenere cipher. It is known as vigenere table.

4. Which of the following correctly defines poly alphabetic cipher?
a) a substitution based cipher which uses multiple substitution at different positions
b) a substitution based cipher which uses fixed substitution over entire message
c) a transposition based cipher which uses multiple substitution at different positions
d) a transposition based cipher which uses fixed substitution over entire message

Answer: a
Explanation: Poly alphabetic cipher is a type of substitution cipher. It uses multiple substitution at different positions in order to cipher the plain text.

5. Which of the following is not a type of poly alphabetic cipher?
a) Autokey cipher
b) Hill cipher
c) One time pad cipher
d) Additive cipher

Answer: d
Explanation: In poly alphabetic cipher each symbol of plain text is replaced by a different cipher text regardless of its occurrence. Out of the given options only additive cipher is not a poly alphabetic cipher.

6. Autokey cipher is closely related to _____
a) Vigenere cipher
b) Hill cipher
c) Play fair cipher
d) Rotor cipher

Answer: a
Explanation: Autokey cipher like vigenere cipher uses vigenere table in order to encrypt the plain text. The difference in these cipher is the usage of the keyword.Answer: a
Explanation: Keyword cipher is less secure than Autokey cipher. It is due to the fact that keyword cipher is mono alphabetic and thus can be cracked using frequency analysis. But Autokey cipher being a poly alphabetic cipher cannot be cracked using this method.

8. What will be the plain text corresponding to cipher text "ZEYQC" if autokey cipher is used with keyword as "SANFOUNDRY"?
a) INDIA
b) WORLD
c) HELLO
d) SECRET

Answer: c
Explanation: Autokey cipher is a type of poly alphabetic substitution which uses vigenere table for making substitutions in the plain text after generating key using plain text. Using the table we find the plain text to be "HELLO".

9. Autokey cipher is also known as?
a) vigenere cipher
b) autoclave cipher
c) auto key cipher
d) auto cipher

Answer: b
Explanation: Autoclave cipher is an alternative name given to autokey cipher. It is because the encrypted text is formed using a key that is generated from a message(plain text) itself.

10. In which of the following cipher the plain text and the ciphered text does not have same number of letters?
a) affine cipher
b) autokey cipher
c) columnar cipher
d) additive cipher

Answer: b
Explanation: In transposition cipher and mono alphabetic cipher the number of letters remain same in ciphered and deciphered text. But in poly alphabetic cipher the number of letters are different. So here as autokey cipher is the only poly alphabetic cipher so it will be the answer.

11. What will be the ciphered text if the string "SECRET" is given as input to the code of autokey cipher with keyword as "SANFOUNDRY"?
a) KEPWSN
b) EKWPSN
c) EKWPSNFDED
d) KEPWSNFDED

Answer: a
Explanation: For encrypting a plain text in autokey cipher we first add the keyword at the beginning of the plain text in order to obtain the key for encryption. Then we use the vigenere table for making respective substitutions. Answer: b
Explanation: Autokey cipher is harder to crack than vigenere cipher. It is because in autokey cipher the key is formed using the plain text itself whereas in vigenere cipher the key is independent of the plain text. This makes autokey cipher more secure.

1. What is the alternative name of playfair cipher?
a) Wadsworth's cipher
b) Wheatstone playfair cipher
c) Playfair rectangle
d) Wheatstone cipher

Answer: b
Explanation: Playfair cipher is also known by the name of Wheatstone playfair cipher. It is because it was discovered by Charles Wheatstone but was promoted by Lord Playfair.

2. Playfair cipher is an example of _____
a) mono-alphabetic cipher
b) poly-alphabetic cipher
c) transposition cipher
d) additive cipher

Answer: b
Explanation: Playfair cipher is a substitution cipher. It falls under the category of poly alphabetic cipher as it uses multiple substitution at different positions in order to cipher the plain text.

3. Encryption in Autokey cipher is done using _____
a) a 5×5 table
b) a 13×2 table
c) vigenere table
d) a 6×6 table

Answer: c
Explanation: Encryption of plain text in playfair cipher is done using a 5×5 table. In this table, all the alphabets of the keyword are arranged row wise by eliminating any duplicates then the remaining letters of the alphabet are placed. One of the alphabet has to be omitted in order to fit all the alphabets in the table.

4. Which of the following correctly defines poly graphic substitution cipher?
a) a substitution based cipher which uses multiple substitutions at different positions
b) a substitution based cipher which uses fixed substitution over entire plain text
c) a substitution based cipher in which substitution is performed over a block of letters
d) a transposition based cipher which uses fixed substitution over entire plain text

Answer: c
Explanation: Poly graphic cipher is a type of substitution cipher in which substitution is performed over a block of letters. Playfair cipher is an example of poly graphic cipher.

5. Which of the following was the first diagram substitution cipher?
a) autokey cipher
b) hill cipher

c) one time pad cipher
d) playfair cipher

Answer: d
Explanation: Poly graphic cipher is a type of substitution cipher in which substitution is performed over a block of letters. In diagram substitution, two adjacent letters are substituted simultaneously. Playfair cipher was the first diagram substitution cipher.

6. Which of the following is hardest to break using frequency analysis?
a) Vigenere cipher
b) Autokey cipher
c) Playfair cipher
d) Rotor cipher

Answer: c
Explanation: Out of the given options playfair cipher is the hardest cipher to break using frequency analysis. It is because it does not substitute letters of the word individually but it encrypts them in pairs of two.Answer: a
Explanation: Keyword cipher is less secure than playfair cipher. It is due to the fact that keyword cipher is mono alphabetic and thus can be cracked using frequency analysis. But playfair cipher being a poly graphic substitution cipher is harder to break using this method.

8. What will be the plain text corresponding to cipher text "BPKYFS" if playfair cipher is used with keyword as "SECRET" (assuming j is combined with i)?
a) INDIAN
b) WORLD
c) DOLLAR
d) HELLO

Answer: c
Explanation: To decrypt the message we follow the reverse procedure. The table is formed in the same manner. Applying this we get the plain text to be "DOLLAR".

9. What is the rule for encryption in playfair cipher if the letters in a pair are identical?
a) then that pair is neglected
b) a null is added in between the letters
c) one of the identical letter is replaced by some other letter
d) then both of the letters are replaced by the letter appearing just next in the row

Answer: b
Explanation: In playfair cipher if the letters in a pair are identical then a null is added in between the letters. Any letter can be used as a null as long as that letter is not the one being repeated.

10. What is the rule for encryption in playfair cipher if the letters in a pair appear in same row?
a) they are replaced by the letter appearing immediately below them respectively
b) they are replaced by the letter appearing immediately right to them respectively
c) they are replaced by the letter at the corner of the row
d) that pair is neglected

Answer: b
Explanation: If the letters in a pair appear in same row then they are replaced by the letters appearing immediately right to them respectively. If the element to be replaced appears at the corner of the row then we wrap around to the left side of that row.

11. What will be the ciphered text if the string "SANFOUNDRY" is given as input to the code of playfair cipher with keyword as "SECRET" (assuming j is combined with i)?
a) ZHQAPNPAFR
b) AHQAPNPAFR
c) HAQAPNPAFR

d) QHAAPNPAFR

Answer: b
Explanation: For encrypting the plain text using playfair cipher we use a 5×5 table that is constructed by using keyword. Then we apply rules for encryption in order to get the ciphered text. Table is given as under-

```
S E C R T
A B D F G
H I K L M
N O P Q U
V W X Y Z
```

12. What is the rule for encryption in playfair cipher if the letters in a pair appear in same column?
a) they are replaced by the letter appearing immediately below them respectively
b) they are replaced by the letter appearing immediately right to them respectively
c) they are replaced by the letters at the corner of the row
d) that pair is neglected

Answer: a
Explanation: If the letters in a pair appear in the same column then they are replaced by the letters appearing immediately below them respectively. If the element to be replaced appears at the corner of the column then we wrap around to the top side of that column.

13. What is the rule for encryption in playfair cipher if the letters in a pair does not appear in same row or column?
a) they are replaced by the letter appearing immediately below them respectively
b) they are replaced by the letter appearing immediately right to them respectively
c) they are replaced by the letter of the same row at the corner of the rectangle defined by the original pair respectively
d) that pair is neglected

Answer: c
Explanation: If the letters in a pair does not appear in same row or column then they are replaced by the letters of the same row at the corner of the rectangle defined by the original pair respectively. The order of letters should be maintained.

1. Hill cipher requires prerequisite knowledge of?
a) integration
b) differentiation
c) matrix algebra
d) differential equation

Answer: c
Explanation: Hill cipher uses matrix multiplication in order to encrypt the given plain text. So it requires prerequisite knowledge of matrix algebra.

2. Hill cipher is an example of _____
a) mono-alphabetic cipher
b) substitution cipher
c) transposition cipher
d) additive cipher

Answer: b
Explanation: Hill cipher is a substitution cipher. It falls under the category of poly alphabetic cipher as it uses multiple substitutions at different positions in order to cipher the plain text.

3. Encryption in hill cipher is done using _____
a) matrix multiplication
b) a 5×5 table
c) vigenere table

d) matrix inversion

Answer: a
Explanation: Encryption of plain text in playfair cipher is done using matrix multiplication. Then modulo 26 of the resulting matrix is taken so as to get the ciphered text.

4. What is poly graphic substitution cipher?
a) a substitution based cipher which uses multiple substitutions at different positions
b) a substitution based cipher which uses fixed substitution over entire plain text
c) a substitution based cipher in which substitution is performed over a block of letters
d) a transposition based cipher which uses fixed substitution over entire plain text

Answer: c
Explanation: Poly graphic cipher is a type of substitution cipher in which substitution is performed over a block of letters. Hill cipher is an example of poly graphic cipher.

5. Which of the following was the first poly graphic cipher to be able to operate on more than 3 letters at once?
a) autokey cipher
b) hill cipher
c) one time pad cipher
d) playfair cipher

Answer:b
Explanation: Poly graphic cipher is a type of substitution cipher in which substitution is performed over a block of letters. Hill cipher was the first poly graphic cipher to be able to operate on more than 3 letters at once.

6. Which of the following is hardest to break using frequency analysis?
a) Vigenere cipher
b) Hill cipher
c) Caesar cipher
d) Affine cipher

Answer: b
Explanation: Out of the given options hill cipher is the hardest cipher to break using frequency analysis. Although it is quite vulnerable to other forms of attack. Answer: b
Explanation: Both hill cipher and playfair cipher are less vulnerable to frequency analysis. But hill cipher is quite vulnerable to other forms of attack and thus less secure than playfair cipher.

8. What will be the plain text corresponding to cipher text "YGQ" if hill cipher is used with keyword as "GYBNQKURP"?
a) SECRET
b) WORLD
c) DOLLAR
d) HELLO

Answer: a
Explanation: To decrypt the message we follow the reverse procedure. We first find the inverse of the keyword matrix then multiply it with cipher matrix.

9. What will be the size of a key matrix if the plain text is "SECRET"?
a) 1×6
b) 5×1
c) 6×1
d) 6×6

Answer: d
Explanation: Hill cipher uses a n x n matrix in order to cipher the plain text. In this case n=6 so a 6×6 matrix will be used.

10. A key matrix used for encryption in hill cipher must be?
a) invertible matrix
b) non invertible matrix
c) square matrix
d) rectangular matrix

Answer: a
Explanation: A key matrix used for encryption in hill cipher must be invertible matrix. Otherwise it will be impossible to decipher the message.

11. What will be the ciphered text if the plain text "SAN" is encrypted using hill cipher with keyword as "GYBNQKURP"?
a) RAJ
b) JAR
c) ARJ
d) AJR

Answer: a
Explanation: We first convert the keyword and plain text into their corresponding matrix respectively. Then we multiply those matrix and take modulo 26 of the resulting matrix to get the ciphered text.

1. What is the alternative name given to Rail fence cipher?
a) random cipher
b) matrix cipher
c) zig zag cipher
d) columnar cipher

Answer: c
Explanation: Rail fence cipher is also known as zig zag cipher. It is so because the plain text gets ciphered by arranging the letters in a zig zag fashion.

2. Rail fence cipher is an example of _____
a) mono-alphabetic cipher
b) poly-alphabetic cipher
c) transposition cipher
d) additive cipher

Answer: c
Explanation: Rail fence cipher is a transposition cipher. It falls under the category of transposition cipher as it encrypts the plain text by rearranging its letters.

3. Encryption in Rail fence cipher is done using _____
a) by arranging the letters in a zig zag fashion in a table
b) by randomly arranging letters
c) by arranging letters in vigenere table
d) by swapping adjacent letters

Answer: a
Explanation: Rail fence cipher is a transposition cipher. It encrypts a given plain text by arranging the letters in a zig zag fashion in a table.

4. Which of the following ciphers are created by shuffling the letters of a word?
a) substitution cipher
b) transposition cipher
c) vigenere cipher
d) hill cipher

Answer: b

Explanation: There are two types of traditional ciphers- Transposition and substitution cipher. In transposition cipher the letters of the given data are shuffled in a particular order, fixed by a given rule.

5. Which of the following is not a type of poly alphabetic cipher?
a) Autokey cipher
b) Hill cipher
c) One time pad cipher
d) Additive cipher

Answer:d
Explanation: In poly alphabetic cipher each symbol of plain text is replaced by a different cipher text regardless of its occurrence. Out of the given options, only additive cipher is not a poly alphabetic cipher.

6. Which of the following is are two types of traditional cipher?
a) transposition cipher and replacement cipher
b) transposition cipher and substitution cipher
c) transforming cipher and substitution cipher
d) transforming cipher and replacement cipher

Answer: b
Explanation: There are two types of a traditional cipher. First is transposition cipher and the second is substitution cipher.Answer: b
Explanation: The number of columns in the table of rail fence cipher is always equal to the number of letters in the plain text. It is independent of the given key value.

8. What will be the plain text corresponding to cipher text "SCSEMG" if rail fence cipher is used with key value 2?
a) MSGSEC
b) SECMSG
c) GSMSEC
d) SECGSM

Answer: b
Explanation: For decryption we reverse the process used in encryption.

S C S
E M G

So the resulting plain text is "SECMSG".

Answer: b
Explanation: One time pad cipher is one of the most secure traditional cipher as it is almost impossible to crack. Rail fence cipher is not very hard to crack. Thus rail fence cipher is less secure than one time pad cipher.

10. In which of the following cipher the plain text and the ciphered text have same letters?
a) autokey cipher
b) rail fence cipher
c) vigenere cipher
d) additive cipher

Answer: b
Explanation: In transposition cipher the letters remain the same in ciphered and plain text. Their position is only changed whereas in substitution cipher the letters become different in encrypted text. So rail fence cipher will be the correct option.

11. What will be the ciphered text if rail fence cipher is used for encrypting the plain text "SANFOUNDRY" with the key value given to be 2?
a) SNONRAFUDY
b) SORAFUDYNN

c) SNAUDNORFY
d) SANFOUNDRY

Answer: a
Explanation: For encrypting a plain text using rail fence cipher we use a table having the number of rows equal to key value as shown below. Then we read along the rows to get the ciphered text.

S			N			O			N			R		
	A			F			U			D			Y	

So the ciphered text becomes "SNONRAFUDY".

1. What is route cipher?
a) a transposition cipher that performs encryption by following an imaginary pattern on a grid
b) a substitution cipher that performs encryption by following an imaginary pattern on a grid
c) a transposition cipher that performs encryption by following a zig zag pattern on a grid
d) a substitution cipher that performs encryption by following a zig zag pattern on a grid

Answer: a
Explanation: Route cipher is a transposition cipher. It encrypts the plain text by following an imaginary pattern on a grid. This pattern can vary from one version of route cipher to others.

2. Route cipher is an example of _____
a) mono-alphabetic cipher
b) poly-alphabetic cipher
c) transposition cipher
d) additive cipher

Answer: c
Explanation: Route cipher is a transposition cipher. It falls under the category of transposition cipher as it encrypts the plain text by rearranging its letters.

3. Encryption in Route cipher is done _____
a) by arranging the letters in a zig zag fashion in a table
b) by randomly arranging letters
c) by following an imaginary pattern drawn on a grid
d) by swapping adjacent letters

Answer: c
Explanation: Route cipher is a transposition cipher. It encrypts the plain text by following an imaginary pattern on a grid.

4. Which of the following ciphers are created by shuffling the letters of a word?
a) playfair cipher
b) route cipher
c) vigenere cipher
d) hill cipher

Answer: b
Explanation: In transposition cipher the letters of the given data are shuffled in a particular order, fixed by a given rule. Route cipher is the only transposition cipher out of the given options.

5. Route cipher is closely related to?
a) Autokey cipher
b) Hill cipher
c) Rail fence cipher
d) Columnar transposition cipher

Answer: c
Explanation: Route cipher is closely related to rail fence cipher. In rail fence cipher the plain text is written in a zig zag fashion in a grid whereas in route cipher the plain text is written horizontally.Answer: b
Explanation: Rail fence is cipher is less secure than route cipher. It is because the key of route cipher not only defines the size of the grid but also the path that is to be followed.Answer: b
Explanation: The key of route cipher not only defines the size of grid but also the path that is to be followed. So longer the message is, the harder it becomes to guess the path.

8. What will be the plain text corresponding to cipher text "RSEADC" if with the number of columns are given to be 3 and route of reading is down the columns?
a) SACRED
b) DERSAC
c) REDSAC
d) SEDRAC

Answer: c
Explanation: For decryption we reverse the process used in encryption. So we arrange the letters of cipher text down the columns and then read the grid horizontally.

R	E	D
S	A	C

So the resulting plain text is "REDSAC".

9. What will be the plain text corresponding to cipher text "SFNUFACT" if with number of columns are given to be 3 and route of reading is from the bottom right anti clockwise?
a) FACTSFUN
b) FACTFUNS
c) FUNSFACT
d) FUNFACTS

Answer: d
Explanation: For decryption we reverse the process used in encryption. So we arrange the letters of cipher text from bottom right anti clockwise and then read the grid horizontally.

F	U	N	F
A	C	T	S

So the resulting plain text is "FUNFACTS".

10. What will be the ciphered text if route cipher is used for encrypting the plain text "SANFOUNDRY" with number of columns given to be 5 and route from the bottom right anti clockwise?
a) SUANNDFROY
b) SORAFUDYNN
c) YOFNASUNDRY
d) SANFOUNDRY

Answer: c
Explanation: For encrypting a plain text using route cipher we use a table having 5 columns as shown below. Then we read from the bottom right anti clockwise to cipher the plain text.

S	A	N	F	O
U	N	D	R	Y

So the ciphered text becomes "YOFNASUNDRY".

11. What will be the ciphered text if route cipher is used for encrypting the plain text "SANFOUNDRY" with a

number of columns given to be 5 and route of reading is down the columns?
a) SUANNDFROY
b) SORAFUDYNN
c) SNAUDNORFY
d) SANFOUNDRY

Answer: a
Explanation: For encrypting a plain text using route cipher we use a table having 5 columns as shown below. Then we read down the columns to cipher the plain text.

S	A	N	F	O
U	N	D	R	Y

So the ciphered text becomes "SUANNDFROY".

1. Trithemius cipher is an example of _____
a) mono-alphabetic cipher
b) poly-alphabetic cipher
c) transposition cipher
d) additive cipher

Answer: b
Explanation: Trithemius cipher is a substitution cipher. It falls under the category of poly alphabetic cipher as it uses multiple substitutions at different positions in order to cipher the plain text.

2. Encryption in trithemius cipher is done using _____
a) trithemius table
b) vigenere cycle
c) tabula recta
d) any table provided by the person performing the encryption

Answer: c
Explanation: Encryption of plain text in trithemius cipher is done by making use of tabula recta. The same table is also used for encryption in vigenere cipher and running key cipher.

3. Which of the following is a modified version of Caesar cipher?
a) vigenere cipher
b) autokey cipher
c) running key cipher
d) trithemius cipher

Answer: d
Explanation: If in caesar cipher we consider a shift that increases by 1 by each letter starting at 0 then it is equivalent to trithemius cipher. So trithemius cipher is a special case of caesar cipher.

4. Which of the following is a difference between trithemius cipher and vigenere cipher?
a) they use different tables for encryption
b) vigenere cipher is poly alphabetic whereas running key cipher is mono alphabetic
c) vigenere cipher uses a key whereas no key is required for using trithemius cipher
d) vigenere cipher is substitution cipher whereas trithemius cipher is transposition cipher

Answer: c
Explanation: Trithemius cipher is a special case of vigenere cipher. The difference is that vigenere cipher uses a different key every time but a fixed key is used by trithemius cipher.

5. Which of the following cipher require the use of tabula recta?
a) hill cipher
b) route cipher

c) rail fence cipher
d) trithemius cipher

Answer: d
Explanation: Ciphers like running key cipher, vigenere cipher, trithemius cipher, etc. makes use of tabula recta. Whereas hill cipher, rail fence cipher and route cipher does not require tabula recta for encryption of plain text.

6. Trithemius cipher is a special case of _____
a) autokey cipher
b) vigenere cipher
c) hill cipher
d) route cipher

Answer: b
Explanation: Trithemius cipher is a special case of vigenere cipher. The difference is that vigenere cipher uses a different key every time but a fixed key is used by trithemius cipher.Answer: b
Explanation: Trithemius cipher is a special case of vigenere cipher with ABCDEFGHIJKLMNOPQRSTUVWXYZ as key. So trithemius cipher is easier to crack as the key being used remains same every time.

8. What will be the plain text corresponding to cipher text "ACCFYX" if trithemius cipher is used?
a) ABACUS
b) ABROAD
c) ABRUPT
d) ABUSED

Answer: a
Explanation: Running key cipher is a type of poly alphabetic substitution which uses tabula recta for making substitutions in the plain text. Using the table and key as ABCDEFGHIJKLMNOPQRSTUVWXYZ we find the plain text to be "ABACUS".Answer: a
Explanation: Trithemius cipher uses a more complex version of caesar cipher. So trithemius cipher is harder to crack as compared to caesar cipher.

10. Which of the following cipher is easiest to crack?
a) vigenere cipher
b) running key cipher
c) trithemius cipher
d) all are equally secure

Answer: c
Explanation: Trithemius cipher is a special case of vigenere cipher and running key cipher is a variation of vigenere cipher. If one figures out that the cipher being used is trithemius then it is very easy to crack, unlike running key and vigenere ciphers as these use a secret key for encryption.

11. What will be the ciphered text corresponding to "SANFOUNDRY" if trithemius cipher is used for encryption?
a) SBPISZTKZH
b) TCQJTAULAI
c) TBOGPVOESZ
d) SPBISZKTZH

Answer: a
Explanation: Encryption in trithemius cipher takes place exactly as in vigenere cipher if we consider the key to be ABCDEFGHIJKLMNOPQRSTUVWXYZ. So by using the tabula recta we can find the encrypted text which is "SBPISZTKZH".

12. What will be the ciphered text corresponding to "ALGORITHM" if trithemius cipher is used for encryption?
a) BNJSWOAPV
b) BMHPSJUIN
c) AMIRVNZOU

d) MBPHJSNIU

Answer: c
Explanation: Encryption in trithemius cipher takes place exactly as in vigenere cipher if we consider the key to be ABCDEFGHIJKLMNOPQRSTUVWXYZ. So by using the tabula recta we can find the encrypted text which is "AMIRVNZOU".

13. What will be the plain text corresponding to cipher text "RVUVMF" if trithemius cipher is used?
a) RABBIT
b) RUSSIA
c) RANGER
d) FRIEND

Answer: b
Explanation: Trithemius cipher is a type of poly alphabetic substitution which uses tabula recta for making substitutions in the plain text. Using the table and using the key as ABCDEFGHIJKLMNOPQRSTUVWXYZ we find the plain text to be "RUSSIA".

1. Polybius square is also known by the name of?
a) Polybius checkboard
b) Polybius table
c) Ploybius board
d) Ploybius keypad

Answer: a
Explanation: Polybius square is similar to substitution cipher. It is also known by the name of Polybius checkboard.

2. How many keys are required for encryption and decryption of data when we use asymmetric cipher?
a) 0
b) 1
c) 2
d) 3

Answer: c
Explanation: Asymmetric cipher makes use of 2 keys for the purpose of encryption. One is known as public key whereas other is called private key.

3. Which of the following is made possible by the use of Polybius square?
a) To represent the plain text by smaller set of symbols
b) To represent the plain text by larger set of symbols
c) To represent the plain text by the letters of some other language
d) To represent the plain text by the same set of symbols

Answer: a
Explanation: Polybius square is similar to substitution cipher. Polybius square allows us to cipher the plain text in such a way that a minimum number of symbols are used in the encrypted text.

4. What is the usual size of polybius square used for encrypting English alphabets?
a) 5 X 5
b) 6 X 6
c) 26 X 26
d) 25 X 25

Answer: a
Explanation: The usual size of poybius square for encrypting English alphabets is 5 X 5. Usually, I and J are combined so as to fit all English letters in this table.

5. Polybius square cipher is most closely related to?

a) mono-alphabetic cipher
b) poly-alphabetic cipher
c) transposition cipher
d) additive cipher

Answer: a
Explanation: Polybius square cipher is much like any mono alphabetic cipher. It is because it applies a fixed substitution to the letters present in plain text.

6. Which two English letters are usually combined in polybius table?
a) A and B
b) Y and Z
c) I and J
d) J and K

Answer: c
Explanation: The usual size of poybius square for encrypting English alphabets is 5 X 5. Usually, I and J are combined so as to fit all English letters in this table. Answer: a
Explanation: Polybius square is closely related to mono alphabetic substitution cipher and thus is more vulnerable to frequency analysis. Whereas polybius square being a poly alphabetic cipher is less vulnerable to frequency analysis and so is more secure.Answer: b
Explanation: Polybius square cipher is closely related to mono alphabetic cipher. Thus it is quite vulnerable to frequency analysis much like any other mono alphabetic cipher.

9. Which of the following cipher uses polybius square cipher in its first step of encrypting data?
a) Autokey cipher
b) One time pad cipher
c) ADFGVX cipher
d) Rail fence cipher

Answer: c
Explanation: ADFGVX cipher uses polybius square cipher in its first step of encrypting data. It uses a 6 X 6 version of polybius square.

10. What will be the plain text corresponding to ciphered text "134325" if standard polybius square cipher is used for encryption?
a) SRH
b) CSK
c) RCB
d) KKR

Answer: b
Explanation: For decoding ciphered text we have to use the polybius square in and find out the letters corresponding to each pair of coordinate. So in this case the plain text is found to be "CSK".

11. What will be the encrypted text corresponding to plain text "SAN" using standard polybius square cipher?
a) 431133
b) 341133
c) 441133
d) 114433

Answer: a
Explanation: For encrypting using polybius square cipher we have to use the table which is shown below and write (row, col) coordinates corresponding to each letter.
1 2 3 4 5
1 A B C D E
2 F G H I/J K

3 L M N O P
4 Q R S T U
5 V W X Y Z
So the ciphered text will be “431133”.

12.What will be the output of the following C++ code?

```
#include <cmath>
#include <iostream>
using namespace std;
void Cipher(string str)
{
    int r, c;
    for (int i = 0; str[i]; i++)
    {

        r = ceil((str[i] - 'a') / 5) + 1;


        c = ((str[i] - 'a') % 5) + 1;


        if (str[i] == 'k')
        {
            r = r - 1;
            c = 5 - c + 1;
        }


        else if (str[i] >= 'j')
        {
            if (c == 1)
            {
                c = 6;
                r = r - 1;
            }
            c = c - 1;
        }
        cout << r << c;
    }
    cout << endl;
}

int main()
{
    string str = "nsit";
    Cipher(str);
}
```

a) 33344244
b) 44332434
c) 33432444
d) 11444323

Answer: c
Explanation: The given code implements polybius square cipher on a given plain text. So here as the plain text is “nsit” so the output of the code should be “33432444”.

1. What is the significance of indicator block in running key cipher?
a) it helps in encryption
b) it strengthens the cipher
c) it gives information regarding the book/text from where the key is taken
d) it makes encryption easy

Answer: c

Explanation: The purpose of having an indicator block in running key cipher is to give information to the receiver about the source of a key being used. It is usually included in the second last block of the message.

2. Running key cipher is an example of _____
a) mono-alphabetic cipher
b) poly-alphabetic cipher
c) transposition cipher
d) additive cipher

Answer: b
Explanation: Running key cipher is a substitution cipher. It falls under the category of poly alphabetic cipher as it uses multiple substitutions at different positions in order to cipher the plain text.

3. Encryption in running key cipher is done using _____
a) running key table
b) vigenere cycle
c) tabula recta
d) any table provided by the person performing the encryption

Answer: c
Explanation: Encryption of plain text in running key cipher is done by making use of tabula recta. The same table is also used for encryption in vigenere cipher.

4. Which of the following cipher uses a key book or a key text instead of a keyword?
a) vigenere cipher
b) autokey cipher
c) running key cipher
d) affine cipher

Answer: c
Explanation: Running key cipher is a poly alphabetic cipher. It uses a key book or key text instead of a keyword which is agreed by both parties before encryption takes place.

5. Which of the following is a difference between running key cipher and vigenere cipher?
a) they use different tables for encryption
b) vigenere cipher is poly alphabetic whereas running key cipher is mono alphabetic
c) in vigenere cipher the key is repeated whereas in running key cipher key is not repeated
d) vigenere cipher was used in ancient time whereas running key cipher is used in modern world

Answer: c
Explanation: In running key cipher key is not repeated unlike vigenere cipher. They both use the same table for encryption.

6. Tabula recta consists of _____
a) 26 rows and 26 columns
b) 26 rows and 1 column
c) 1 row and 26 columns
d) 27 rows and 27 columns

Answer: a
Explanation: Encryption of plain text using running key cipher is done by making use of tabula recta. It consists of 26 rows and 26 columns which have alphabets written 26 times. These are shifted towards left after each row.Answer: a
Explanation: Keyword cipher is less secure than running key cipher. It is due to the fact that keyword cipher is mono alphabetic and thus can be cracked using frequency analysis. But running key cipher being a poly alphabetic cipher is harder to crack.

8. What will be the plain text corresponding to cipher text "KEPWSN" if running key cipher is used with keyword as "SANFOUNDRY"?

a) SECRET
b) QWERTY
c) INDIAN
d) FRIEND

Answer: a
Explanation: Running key cipher is a type of poly alphabetic substitution which uses tabula recta for making substitutions in the plain text. Using the table we find the plain text to be "SECRET".Answer: b
Explanation: Running key cipher is a poly alphabetic substitution cipher. Encryption is done by using tabula recta.

10. Running key cipher is a variation of?
a) vigenere cipher
b) autokey cipher
c) hill cipher
d) route cipher

Answer: a
Explanation: Vigenere cipher is a variation of vigenere cipher. The only difference between them is that in vigenere cipher a random key is chosen whereas in running key cipher key is chosen from a book or a text.

11. What will be the ciphered text corresponding to "SANFOUNDRY" if running key cipher is used for encryption with keyword as "ONCEUPONATIME"?
a) GNPJIJBQRR
b) GNPIJJBRRQ
c) GPNJJOBQRR
d) GJJNPOBQRI

Answer: a
Explanation: Encryption in running key cipher takes place exactly as in vigenere cipher if we don't include the indicator block. So by using the tabula recta we can find the encrypted text which is "GNPJIJBQRR".

12. What will be the ciphered text corresponding to "ALGORITHM" if running key cipher is used for encryption with keyword as "DATASTRUCTURE"?
a) LDJOZOBBK
b) DLZOJBKBO
c) ZOLDJBKBO
d) OLZBJDKBO

Answer: b
Explanation: Encryption in running key cipher takes place exactly as in vigenere cipher if we don't include the indicator block. So by using the tabula recta we can find the encrypted text which is "DLZOJBKBO".

13. What will be the plain text corresponding to cipher text "IWRWHS" if running key cipher is used with keyword as "SANFOUNDRY"?
a) SECRET
b) QWERTY
c) INDIAN
d) FRIEND

Answer: b
Explanation: Running key cipher is a type of poly alphabetic substitution which uses tabula recta for making substitutions in the plain text. Using the table we find the plain text to be "QWERTY".

1. What is the period in bifid cipher?
a) length of blocks in which the plain text is divided before encryption
b) number of letters after which the key is repeated
c) number of blocks into which the plain text is divided before encryption
d) number of keys used for encryption

Answer: a
Explanation: Polybius square is similar to substitution cipher. It is also known by the name of Polybius checkboard.

2. Which of the following cipher uses polybius square?
a) bifid cipher
b) beaufort cipher
c) trithemius cipher
d) gronsfeld cipher

Answer: a
Explanation: Bifid cipher uses polybius square for encrypting the plain text. Bifid cipher combines polybius square cipher with transposition.

3. Bifid cipher combines transposition with which of the following cipher?
a) Polybius square cipher
b) Playfair cipher
c) gronsfeld cipher
d) trifid cipher

Answer: a
Explanation: Bifid cipher combines polybius square cipher with transposition. It uses fractionation to obtain diffusion.Answer: b
Explanation: Trifid cipher is a variation of bifid cipher. The only difference between them is that trifid cipher uses a 3×3 key square instead of 5×5 square.Answer: b
Explanation: Bifid square combines polybius square with transposition. So the given statement is false.

7. What will be the ciphered text corresponding to "SANFOUNDRY" if bifid cipher is used for encryption with key as "KEY" with period as 5?
a) SBPISZTKZH
b) PLNSOWGKQM
c) SELFQEXBHM
d) YFSGWNFBW

Answer: b
Explanation: For encrypting the plain text using bifid cipher we first form the polybius square with the help of given key. After this, the plain text is divided into blocks of length 5 and corresponding coordinates are noted. So the corresponding cipher text would be "PLNSOWGKQM".

8. What will be the ciphered text corresponding to "ALGORITHM" if bifid cipher is used for encryption with key as "KEY" with a period as 5?
a) SBPISZTKZH
b) PLNSOWGKQM
c) SELFQEXBHM
d) YFSGWNFBW

Answer: d
Explanation: For encrypting the plain text using bifid cipher we first form the polybius square with the help of given key. After this, the plain text is divided into blocks of length 5 and corresponding coordinates are noted. So the corresponding cipher text would be "YFSGWNFBW".

9. What will be the plain text corresponding to cipher text "XKS" if the bifid cipher is used with key as "KEY" and period as 5?
a) IND
b) USA
c) RSA
d) AUS

Answer: b

Explanation: The decryption in bifid cipher begins identically to encryption. Then we find the corresponding coordinates and then we get the plain text. So the plain text is "USA".

10. What will be the plain text corresponding to ciphered text "BKC" if the bifid cipher is used for encryption with key as "KEY" and period 5?
a) MSD
b) SRT
c) KVK
d) VVS

Answer: c
Explanation: The decryption in bifid cipher begins identically to encryption. Then we find the corresponding coordinates and then we get the plain text. So the plain text is "KVK".

1. What is the length of the dot in Morse code?
a) 1 Unit
b) 7 Units
c) 5 Units
d) 3 Units

Answer: d
Explanation: In Morse Code, the length of the single dot is defined to be 1 Unit. While dash is defined to be 3 Unit. In Morse Code, the space between words is 7 unit.

2. Which of the following is used as signal duration in Morse Code?
a) Dash
b) Forward Slash
c) Apostrophe
d) Ampersand

Answer: a
Explanation: The two signal duration used in Morse code are Dot and Dash. They are also called Dits and Dahs. While other symbols are expressed in terms of dots and dash.

3. Morse Code is named after which scientist?
a) Samuel F. B. Morse
b) Karl Morse
c) Judea Pearl
d) Daniel C. Morse

Answer: a
Explanation: Morse Code is named after Samuel F. B. Morse who is the inventor of the telegraph. The best first search algorithm using heuristic evaluation rule or function was proposed by an Israeli – American computer scientist and philosopher Judea Pearl.

4. What is the basic unit of time measurement in Morse code transmission?
a) Dot duration
b) Dash Duration
c) Numeric Duration
d) Space Duration

Answer: a
Explanation: Dot duration is the basic unit of time measurement in Morse code transmission. Both dot and dash signal are used as time measurement in Morse code.

5. The dash duration is how many times the dot duration?
a) 1
b) 2

c) 3
d) 4

Answer: c
Explanation: The duration of a dash is 3 times the duration of a dot as duration of dot is defined to be 1 unit in Morse Code and duration of dash is 3 units in Morse Code.

6. In Morse Code, each dot or dash within a character is followed by a period of signal absence. What is the name of that signal?
a) Slash
b) Space
c) Ampersand
d) Asterisk

Answer: b
Explanation: Each dot or dash within a character is followed by a period of signal absence called Space Signal which is of 1 Unit. The space signal is used after every word in Morse Code.

7. What is the duration of Space Signal?
a) Equal to Dot Duration
b) More than Dot Duration
c) Equal to Dash Duration
d) More than Dash Duration

Answer: a
Explanation: The duration of Space Signal is 1 Unit and Dot Duration is also 1 Unit. So the duration of Space Signal is equal to Dot Duration.

8. The letters of a word are separated by a space of how many durations?
a) 1 Dot Duration
b) 2 Dot Duration
c) 3 Dot Duration
d) 4 Dot Duration

Answer: c
Explanation: The letters of a word are separated by a space of 3 dot durations. Since the dot duration is of 1 Unit. So, the letters of a word are separated by a space of 3 Unit.

9. The words are separated by a space of how many durations?
a) 5
b) 6
c) 7
d) 4

Answer: c
Explanation: The words are separated by a space of 7 dot durations. Since the dot duration is of 1 Unit. So, the words are separated by a space of 7 Unit.Answer: a
Explanation: Morse code was designed so that the length of each symbol is inverse to the frequency of occurrence in text. Therefore, the letter the letter "E", has a single dot.Answer: a
Explanation: The letter in English, "E", has a single dot. It is considered to be the smallest Morse Code ever defined till now while other letters are larger in time duration.

12. What is meant by the single dash in Morse code?
a) A
b) T
c) Z
d) E

Answer: b
Explanation: The common letter in English, the letter "T", has the shortest dash code: a single dash. While the letter in English, "E", has a single dot. It is considered to be the smallest Morse Code ever defined till now while other letters are larger in time duration.

13. What is the Morse code for the distress signal is SOS?
a) 3 Dots, 3Dashes, and 3 Dots
b) 3 Dots, 3 Dashes, and 2 Dots
c) 3 Dots, 2 Dashes, and 3 Dots
d) 2 Dots, 3 Dashes, and 3 Dots

Answer: a
Explanation: In an emergency the distress signal is SOS– 3 dots, 3 dashes, and 3 dots – internationally recognized by treaty. It is used to call emergency help when someone is in need.

14. For which device was Morse code developed for?
a) Telegraphy
b) Stethoscope
c) Telephone
d) SONAR

Answer: a
Explanation: A code was needed to transmit natural language using only the pulses in a telegraph system. Morse developed the modern International Morse code.

15. When was Morse system for telegraphy first used?
a) 1843
b) 1844
c) 1845
d) 1846

Answer: b
Explanation: Morse system for telegraphy was first used in 1844 to make indentations on a paper tape when electric currents were received.

1. Which letter of the English alphabet has the shortest code in Morse Code?
a) A
b) C
c) B
d) E

Answer: d
Explanation: Morse code was designed so that the length of each symbol is inverse to the frequency of occurrence in text. Thus the letter in English, "E", has a single dot.

2. Which word tells a word rate of Morse code's shorter code durations for common characters such as "e" and "t".?
a) PARIS
b) CODEX
c) PARCOD
d) SPACE

Answer: a
Explanation: PARIS tells about word rate that is typical of natural language words and reflects the benefits of Morse code's shorter code durations for common characters such as "e" and "t".Answer: a
Explanation: Morse code speed is measured in words per minute (wpm). The transmission rate of the Morse code is also measured is groups per unit (gpm).Answer: a
Explanation: Morse code speed is measured in characters per minute (cpm) as well as in words per minute (wpm). The transmission rate of the Morse code is also measured is groups per unit (gpm).

5. Who is the creator of Modern International Morse Code?
a) Samuel F. B. Morse
b) Karl Morse
c) Alexander Morse
d) Friedrich Clemens Gerke

Answer: d
Explanation: It was created by Friedrich Clemens Gerke in 1848 and initially used for telegraphy in Germany. Samuel Morse is the inventor of the telegraph after whom Morse code was named.

6. What is meant by the single dot in Morse code?
a) A
b) C
c) B
d) E

Answer: d
Explanation: The most common letter in English, the letter "E", has a single dot. Morse code was designed so that the length of each symbol is inverse to frequency of occurrence in text.

7. Which word offers a word rate that is typical of 5-letter code groups?
a) PARIS
b) CODEX
c) PARCOD
d) SPACE

Answer: b
Explanation: CODEX word offers a word rate that is typical of 5-letter code groups. PARIS is a standard word to measure the operator transmission speed of Morse Code.

8. What is the name of special procedural signals that are used to indicate changes in communications protocol status?
a) Prosigns
b) Aldis
c) BIT
d) Asterisk

Answer: b
Explanation: Prosigns are special unwritten procedural signals which are used to show changes in communications protocol status or white space text.

9. Space signal is of how many unit?
a) 1
b) 2
c) 3
d) 4

Answer: a
Explanation: Since the duration of Space Signal is equal to Dot Duration. So, the space signal is of 1 Unit. While a dash is 3 units and space between letters is also three units.

10. The letters of a word are separated by a space of how many units?
a) 1 Unit
b) 2 Units
c) 3 Units
d) 4 Units

Answer: c

Explanation: The letters of a word are separated by a space of 3 Unit. The representation of dot symbol has 1 unit, while the dash symbol has 2 units.

11. Which symbol is not defined inside the ITU recommendation on Morse code?
a) $
b) .
c) +
d) ~

Answer: a
Explanation: The symbol $ and & are not defined inside the ITU recommendation on Morse code. International Telecommunication Union is a union that mandated Morse code worldwide.

12. For which symbol there is no standard representation in Morse Code?
a) /
b) .
c) *
d) !

Answer: d
Explanation: There is no standard representation for the exclamation mark (!) in Morse code. The other symbols have been defined by the International Telecommunication Union.

13. In which year the symbol @ was added to the official Morse character set by ITU-R?
a) 2003
b) 2004
c) 2005
d) 2006

Answer: b
Explanation: On May 24, 2004, the symbol @ was added to the official Morse character set by International Telecommunication Union.

14. Which device was used to generate high speed Morse Code?
a) Iambic Paddle
b) Note Paddle
c) Vibrolex
d) KSM Radio

Answer: a
Explanation: Iambic Paddle was used to generate high speed Morse Code in conjunction with an electronic keyer. It was commercially manufactured.

15. Who along with Samuel Morse developed Morse code?
a) Alfred Vail
b) Alan Turing
c) Jedidiah Morse
d) Lucretia Pickering Walker

Answer: a
Explanation: Alfred Vail along with Samuel Morse developed Morse code. While Jedidiah Morse, Lucretia Pickering Walker were his father and spouse respectively.

1. Which of the following is not a type of traditional cipher?
a) Substitution cipher
b) Transposition cipher
c) Mono alphabetic cipher
d) PKCS cipher

Answer: d
Explanation: There are two types of the traditional cipher. One is transposition cipher and the other is substitution cipher. Whereas PKCS is a modern asymmetric cipher.

2. Which of the following cipher uses two keys to encrypt data?
a) substitution cipher
b) transposition cipher
c) symmetric cipher
d) asymmetric cipher

Answer: d
Explanation: Asymmetric cipher makes use of 2 keys for the purpose of encryption. One is known as public key whereas other is called private key.

3. Asymmetric encryption is also known as?
a) Private key cryptography
b) Public key cryptography
c) Public private key cryptography
d) Traditional cryptography

Answer: b
Explanation: Asymmetric encryption is also known as public key cryptography. Asymmetric cipher makes use of 2 keys for the purpose of encryption.

4. Which of the following is an example of asymmetric encryption technique?
a) one-time pad
b) one-time password
c) DSA
d) blowfish

Answer: c
Explanation: Asymmetric cipher makes use of 2 keys for the purpose of encryption. DSA is an example of asymmetric encryption technique.

5. Columnar cipher falls under the category of?
a) mono-alphabetic cipher
b) poly-alphabetic cipher
c) transposition cipher
d) additive cipher

Answer: c
Explanation: Columnar cipher is a transposition cipher. It falls under the category of transposition cipher as it encrypts the plain text by rearranging its letters.

6. Which of the following ciphered text would have NOT used transposition cipher for encryption of the plain text "CIPHER"?
a) EPIHRC
b) EHIPCR
c) DTIPRC
d) HRIPEC

Answer: c
Explanation: We know that transposition cipher encrypts the plain text by shuffling the letters of the plain text. So out of the given options, only "DTIPRC" does not have the same set of letters as "CIPHER".

7. Which of the following cipher is formed by applying columnar transposition cipher twice?
a) Rail Fence cipher
b) Route cipher

c) Double transposition cipher
d) One time pad

Answer: c
Explanation: Double transposition cipher is formed by applying columnar transposition cipher twice. For the purpose of encryption, we may use the same key twice or we can use two different keys.

8. In which of the following cipher the plain text and the ciphered text does not have the same set of letters?
a) route cipher
b) columnar transposition cipher
c) myszkowski cipher
d) additive cipher

Answer: d
Explanation: In transposition cipher, the letters remain the same in ciphered and plain text. Their position is only changed whereas in substitution cipher the letters become different in encrypted text. As additive cipher is the only non transposition cipher out of the given options so it will be the correct option.Answer: b
Explanation: Double transposition cipher is formed by applying columnar transposition cipher twice. So it is harder to crack than a simple columnar transposition cipher.

10. How many columns do we need to have in the table, that is used for encryption in columnar transposition cipher when a given keyword is "SECRET" and plain text is "SANFOUNDRY"?
a) 4
b) 5
c) 6
d) 7

Answer: c
Explanation: The number of columns in the table used for the purpose encryption in columnar transposition cipher will always be equal to the number of letters in the keyword. So in this case it will be equal to 6.

11. What will be the encrypted text corresponding to plain text "CLASSIFIED" using columnar transposition cipher with a keyword as "GAMES"?
a) LFDSIASECI
b) SECIAISDFL
c) CILFAISESD
d) LFSECIAISD

Answer: d
Explanation: For encrypting using columnar cipher we have to arrange the letters of the plain text in a table which has the same number of columns as the letters of the keyword. Then the letters of the keyword are arranged in alphabetical order and we read along each column.
3 1 4 2 5
G A M E S
C L A S S
I F I E D
So the ciphered text will be "IFSECIAISD".

12. How many rows will the letters of the plain text occupy in the table, that is used for encryption in columnar transposition cipher when a given keyword is "SECRET" and plain text is "SANFOUNDRY"?
a) 1
b) 2
c) 3
d) 4

Answer: b
Explanation: The number of columns in the table used for the purpose encryption in columnar transposition cipher will

always be equal to the number of letters in the keyword.So when we will write the letters of the plain text row wise then there will be 2 rows of plain text in this case. The table is shown below :-
S E C R E T
1 S A N F O U
2 N D R Y

13. Which of the following statement is not true regarding columnar transposition cipher?
a) it is a weak cipher
b) probability of error is high while deciphering
c) it cannot be combined with other ciphers
d) it is a traditional symmetric cipher

Answer: c
Explanation: Although columnar transposition cipher is a weak cipher in itself. But it can be combined with other substitution ciphers so as to improve its security. The probability of error remains high while decoding columnar cipher as it is a lengthy process.

1. Which of the following cipher makes use of linear algebra for encrypting data?
a) polybius square cipher
b) affine cipher
c) caesar cipher
d) rail fence cipher

Answer: b
Explanation: Affine cipher is the only cipher out of the given options that make use of linear algebra for the purpose of encryption. It is a type of mono alphabetic cipher.

2. Which of the following cipher requires only one key for decoding the ciphered text?
a) Affine cipher
b) RSA
c) DSA
d) PKCS

Answer: a
Explanation: Asymmetric cipher makes use of 2 keys for the purpose of encryption. As affine cipher is the only symmetric cipher out of the given options so it requires only one key.

3. Choose the weakest cipher from the following?
a) Vigenere cipher
b) Autokey cipher
c) Affine cipher
d) Hill cipher

Answer: c
Explanation: Affine cipher is the weakest cipher out of the given options as it is a mono alphabetic cipher and other options are poly alphabetic ciphers. So it is quite vulnerable to frequency analysis.

4. What is the formula used for encryption of data using affine cipher(a,b are constants and x is the numerical equivalent of a letter to be encrypted)?
a) ax+b
b) (ax+b)%26
c) ax^2+bx
d) $(ax^2+bx)\%26$

Answer: b
Explanation: Affine cipher uses linear algebra for the purpose of encryption. It uses the formula (ax+b)%26 for this purpose.

5. What is the formula used for decoding the ciphered text using affine cipher(a,b are constants and x is the numerical equivalent of a letter to be encrypted)?

a) $a^{-1}(x-b)\%26$

b) $(ax+b)\%26$

c) $b^{-1}(x-a)\%26$

d) $b^{-1}(x-a)$

Answer: a

Explanation: Affine cipher uses linear algebra for the purpose of encryption. It uses the formula $a^{-1}(x-b)\%26$ for decryption of ciphered text.

6. Affine cipher is an example of?

a) Mono alphabetic cipher

b) Poly alphabetic cipher

c) Transposition cipher

d) Asymmetric cipher

Answer: a

Explanation: Affine cipher falls in the category of mono alphabetic substitution cipher. It uses linear algebra for encrypting the plain text.Answer: b

Explanation: Affine cipher is more secure as compared to caesar cipher. But affine cipher is a very weak cipher as it can be easily broken using frequency analysis.Answer: b

Explanation: Affine cipher is a very weak cipher as it can be easily broken using frequency analysis. It can be easily cracked even if 2 characters are known.

9. What will be the ciphered text corresponding to plain text "sanfoundry" if an affine cipher is used with key values as a=5, b=10?

a) wkxjcgxzra

b) gkxteuxfzw

c) ukxhmyxdfg

d) rfsexbsumv

Answer: a

Explanation: Affine cipher uses linear algebra for the purpose of encryption. It uses the formula $(ax+b)\%26$ for this purpose. So the ciphered text will be "wkxjcgxzra".

10. What will be the plain text corresponding to ciphered text "rmw " if an affine cipher is used with key values as a=5, b=10?

a) csk

b) kkr

c) srt

d) msd

Answer: c

Explanation: Affine cipher uses linear algebra for the purpose of encryption It uses the formula $a^{-1}(x-b)\%26$ for decryption of ciphered text. So the plain text will be "srt".

11. While choosing the value of a and m (m is the no. of alphabets) in affine cipher it must be ensured that?

a) a and m are prime numbers

b) a and m are odd numbers

c) a and m are coprime

d) a and m are even numbers

Answer: c

Explanation: While choosing the values of a and m it should be ensured that a and m are coprime. Otherwise decoding the ciphered text would be difficult.

1. The most common hamming codes are a generalized version of?
a) Hamming(7, 4) code
b) Hamming(8, 4) code
c) Hamming(6, 3) code
d) Hamming(5, 7) code

Answer: a
Explanation: The most common hamming codes generalize to form hamming(7, 4) code. It encodes four bits of data into seven bits by adding three parity bits.

2. What is the minimal Hamming distance between any two correct codewords?
a) 1
b) 2
c) 3
d) 4

Answer: c
Explanation: Since we use a generalized version of Hamming(7, 4) code, the minimal hamming distance is 3. It cannot correct burst errors.

3. Why do we require hamming codes?
a) Error correction
b) Encryption only
c) Decryption
d) Bit stuffing

Answer: a
Explanation: Hamming codes are used for the purpose of error detection and correction. It is also used for channel encoding and decoding. They are linear-error correcting codes.Answer: b
Explanation: Hamming bits are suitable only for single-bit error detection and correction and two bit error detection. It is very unlikely to detect burst errors.

5. Who invented Hamming codes?
a) Richard Hamming
b) Ross Hamming
c) Shannon
d) Huffman

Answer: a
Explanation: Richard W. Hamming invented hamming codes in Bell Telephone Laboratory to minimize the errors in punched card readers. Huffman invented huffman codes. Shannon invented Shannon-Fanno codes.

6. What is the total block length 'n' of a Hamming code?
a) 2^r
b) 2^r-1
c) $2^{r-1}-1$
d) 2^r+1

Answer: b
Explanation: Hamming codes are a class of binary linear codes, hence r>=2. For a hamming(7, 4) code, the block length 'n' is 2^r-1 where r is the parity bit. Here, r=3.

7. What is the message length 'k' of a Hamming(7,4) code?
a) 2^r-1
b) 2^r-r+1
c) 2^r-r-1

d) 2^{r+1}-r

Answer: c
Explanation: Hamming codes are a class of binary linear codes, hence r>=2. For a hamming(7,4) code, the message length 'k' is 2^r-r-1 where r is the parity bit. Here, r=3.

8. What is the rate of hamming codes?
a) 1-[r/(2^r-1)]
b) 1-(r/2^r)
c) 1+(r/2^r)
d) r/2^r+1

Answer: a
Explanation: Rate of a hamming code is message length divided by block length (i.e.) 2^r-r-1/2^r-1 = 1-[r/(2^r-1)]. It is the highest rate for a minimum distance of three.

9. A two-out-of-five code consists of _____
a) Two 0s and three 1s
b) Three 0s and two 1s
c) Four 0s and one 1s
d) One 0s and four 1s

Answer: b
Explanation: A two-out-of-five code consists of three 0s and two 1s. Hence, it contains ten possible combinations to represent digits from 0-9.Answer: b
Explanation: If error has occurred in a data string, parity will change inorder to indicate errors. However, if the error occurs in parity bit, the error goes undetected.

11. _____ is the mechanism of sending data bits multiple times to ensure consistency.
a) Repetition
b) Duplication
c) Mirroring
d) Redundancy

Answer: a
Explanation: Repeating data bits multiple times is done to ensure consistency. If the data bit to be sent is a 1, a n=3 repetition code will send 111. If the bits are not the same, an error has occurred.

12. An Extended hamming code is also called as _____
a) SEDDEC
b) SEDDED
c) SECDED
d) SECDEC

Answer: c
Explanation: An Extended Hamming code is also called as SECDED (Single Error Correction Double Error Detection). The most popular codes are (72, 64) code and (127,120) code.

13. What is the code rate of a repetition Hamming code (3, 1)?
a) 1
b) 3
c) 1/3
d) 1.3

Answer: c
Explanation: The code rate of a repetition hamming code is the second number divided by the first number. Here, it is 1/3.

14. For a hamming code of parity bit m=8, what is the total bits and data bits?
a) (255, 247)
b) (127, 119)
c) (31, 26)
d) (0, 8)

Answer: a
Explanation: Total bits are computed as, 2^m-1 = 2^8-1 =255.
Data bits are computed as 2^m-m-1= 2^8-8-1= 247.

15. What is the rate of the hamming code of parity bit m=8?
a) 0.94
b) 0.92
c) 0.90
d) 0.97

Answer: d
Explanation: For a hamming code of parity bit 8, total bits = 255 and data bits = 247. Rate= data bits/ total bits = 247/255 = 0.969 = 0.97.

1. The worst-case efficiency of solving a problem in polynomial time is?
a) O(p(n))
b) O(p(n log n))
c) O(p(n^2))
d) O(p(m log n))

Answer: a
Explanation: The worst-case efficiency of solving an problem in polynomial time is O(p(n)) where p(n) is the polynomial time of input size.

2. Problems that can be solved in polynomial time are known as?
a) intractable
b) tractable
c) decision
d) complete

Answer: b
Explanation: Problems that can be solved in polynomial time are known as tractable. Problems that cannot be solved in polynomial time are intractable.Answer: a
Explanation: One of the properties of polynomial functions states that the sum and composition of two polynomials are always polynomials.

4. _____ is the class of decision problems that can be solved by non-deterministic polynomial algorithms.
a) NP
b) P
c) Hard
d) Complete

Answer: a
Explanation: NP problems are called as non-deterministic polynomial problems. They are a class of decision problems that can be solved using NP algorithms.

5. Problems that cannot be solved by any algorithm are called?
a) tractable problems
b) intractable problems
c) undecidable problems
d) decidable problems

Answer: c
Explanation: Problems cannot be solved by any algorithm are called undecidable problems. Problems that can be solved in polynomial time are called Tractable problems.

6. The Euler's circuit problem can be solved in?
a) O(N)
b) O(N log N)
c) O(log N)
d) O(N^2)

Answer: d
Explanation: Mathematically, the run time of Euler's circuit problem is determined to be O(N^2).

7. To which class does the Euler's circuit problem belong?
a) P class
b) NP class
c) Partition class
d) Complete class

Answer: a
Explanation: Euler's circuit problem can be solved in polynomial time. It can be solved in O(N^2).

8. Halting problem is an example for?
a) decidable problem
b) undecidable problem
c) complete problem
d) trackable problem

Answer: b
Explanation: Halting problem by Alan Turing cannot be solved by any algorithm. Hence, it is undecidable.

9. How many stages of procedure does a non-deterministic algorithm consist of?
a) 1
b) 2
c) 3
d) 4

Answer: b
Explanation: A non-deterministic algorithm is a two-stage procedure- guessing stage and verification stage.Answer: a
Explanation: One of the properties of NP class problems states that A non-deterministic algorithm is said to be non-deterministic polynomial if the time-efficiency of its verification stage is polynomial.

11. How many conditions have to be met if an NP- complete problem is polynomially reducible?
a) 1
b) 2
c) 3
d) 4

Answer: b
Explanation: A function t that maps all yes instances of decision problems D1 and D2 and t should be computed in polynomial time are the two conditions.

12. To which of the following class does a CNF-satisfiability problem belong?
a) NP class
b) P class
c) NP complete
d) NP hard

Answer: c
Explanation: The CNF satisfiability problem belongs to NP complete class. It deals with Boolean expressions.

13. How many steps are required to prove that a decision problem is NP complete?
a) 1
b) 2
c) 3
d) 4

Answer: b
Explanation: First, the problem should be NP. Next, it should be proved that every problem in NP is reducible to the problem in question in polynomial time.

14. Which of the following problems is not NP complete?
a) Hamiltonian circuit
b) Bin packing
c) Partition problem
d) Halting problem

Answer: d
Explanation: Hamiltonian circuit, bin packing, partition problems are NP complete problems. Halting problem is an undecidable problem.

15. The choice of polynomial class has led to the development of an extensive theory called _____
a) computational complexity
b) time complexity
c) problem complexity
d) decision complexity

Answer: a
Explanation: An extensive theory called computational complexity seeks to classify problems according to their inherent difficulty.

1. Which of the following algorithm can be used to solve the Hamiltonian path problem efficiently?
a) branch and bound
b) iterative improvement
c) divide and conquer
d) greedy algorithm

Answer: a
Explanation: The Hamiltonian path problem can be solved efficiently using branch and bound approach. It can also be solved using a backtracking approach.

2. The problem of finding a path in a graph that visits every vertex exactly once is called?
a) Hamiltonian path problem
b) Hamiltonian cycle problem
c) Subset sum problem
d) Turnpike reconstruction problem

Answer: a
Explanation: Hamiltonian path problem is a problem of finding a path in a graph that visits every node exactly once whereas Hamiltonian cycle problem is finding a cycle in a graph.

3. Hamiltonian path problem is _____
a) NP problem
b) N class problem
c) P class problem
d) NP complete problem

Answer: d
Explanation: Hamiltonian path problem is found to be NP complete. Hamiltonian cycle problem is also an NP- complete problem.Answer: b
Explanation: There is a relationship between Hamiltonian path problem and Hamiltonian circuit problem. The Hamiltonian path in graph G is equal to Hamiltonian cycle in graph H under certain conditions.

5. Which of the following problems is similar to that of a Hamiltonian path problem?
a) knapsack problem
b) closest pair problem
c) travelling salesman problem
d) assignment problem

Answer: c
Explanation: Hamiltonian path problem is similar to that of a travelling salesman problem since both the problem traverses all the nodes in a graph exactly once.

6. Who formulated the first ever algorithm for solving the Hamiltonian path problem?
a) Martello
b) Monte Carlo
c) Leonard
d) Bellman

Answer: a
Explanation: The first ever problem to solve the Hamiltonian path was the enumerative algorithm formulated by Martello.

7. In what time can the Hamiltonian path problem can be solved using dynamic programming?
a) O(N)
b) O(N log N)
c) $O(N^2)$
d) $O(N^2\ 2^N)$

Answer: d
Explanation: Using dynamic programming, the time taken to solve the Hamiltonian path problem is mathematically found to be $O(N^2\ 2^N)$.Answer: a
Explanation: According to a handshaking lemma, in graphs, in which all vertices have an odd degree, the number of Hamiltonian cycles through any fixed edge is always even.

9. Who invented the inclusion-exclusion principle to solve the Hamiltonian path problem?
a) Karp
b) Leonard Adleman
c) Andreas Bjorklund
d) Martello

Answer: c
Explanation: Andreas Bjorklund came up with the inclusion-exclusion principle to reduce the counting of number of Hamiltonian cycles.

10. For a graph of degree three, in what time can a Hamiltonian path be found?
a) $O(0.251^n)$
b) $O(0.401^n)$
c) $O(0.167^n)$
d) $O(0.151^n)$

Answer: a
Explanation: For a graph of maximum degree three, a Hamiltonian path can be found in time $O(0.251^n)$.

11. What is the time complexity for finding a Hamiltonian path for a graph having N vertices (using permutation)?
a) O(N!)
b) O(N! * N)
c) O(log N)
d) O(N)

Answer: b
*Explanation: For a graph having N vertices traverse the permutations in N! iterations and it traverses the permutations to see if adjacent vertices are connected or not takes N iterations (i.e.) O(N! * N).Answer: a*
Explanation: The above graph has only one Hamiltonian path that is from a-b-c-d-e.Answer: c
Explanation: The above graph has no Hamiltonian paths. That is, we cannot traverse the graph with meeting vertices exactly once.

1. Under what condition any set A will be a subset of B?
a) if all elements of set B are also present in set A
b) if all elements of set A are also present in set B
c) if A contains more elements than B
d) if B contains more elements than A

Answer: b
Explanation: Any set A will be called a subset of set B if all elements of set A are also present in set B. So in such a case set A will be a part of set B.

2. What is a subset sum problem?
a) finding a subset of a set that has sum of elements equal to a given number
b) checking for the presence of a subset that has sum of elements equal to a given number and printing true or false based on the result
c) finding the sum of elements present in a set
d) finding the sum of all the subsets of a set

Answer: b
Explanation: In subset sum problem check for the presence of a subset that has sum of elements equal to a given number. If such a subset is present then we print true otherwise false.

3. Which of the following is true about the time complexity of the recursive solution of the subset sum problem?
a) It has an exponential time complexity
b) It has a linear time complexity
c) It has a logarithmic time complexity
d) it has a time complexity of O(n2)

Answer: a
Explanation: Subset sum problem has both recursive as well as dynamic programming solution. The recursive solution has an exponential time complexity as it will require to check for all subsets in worst case.

4. What is the worst case time complexity of dynamic programming solution of the subset sum problem(sum=given subset sum)?
a) O(n)
b) O(sum)
c) O(n^2)
d) O(sum*n)

Answer: d
*Explanation: Subset sum problem has both recursive as well as dynamic programming solution. The dynamic programming solution has a time complexity of O(n*sum) as it as a nested loop with limits from 1 to n and 1 to sum respectively.Answer: a*
Explanation: Subset sum problem takes exponential time when we implement a recursive solution. Subset sum problem is known to be a part of NP complete problems.Answer: b

Explanation: The recursive solution to subset sum problem takes exponential time complexity whereas the dynamic programming solution takes polynomial time complexity. So dynamic programming solution is faster in terms of time complexity.

7. Which of the following is not true about subset sum problem?
a) the recursive solution has a time complexity of O(2n)
b) there is no known solution that takes polynomial time
c) the recursive solution is slower than dynamic programming solution
d) the dynamic programming solution has a time complexity of O(n log n)

Answer: d
*Explanation: Recursive solution of subset sum problem is slower than dynamic problem solution in terms of time complexity. Dynamic programming solution has a time complexity of O(n*sum).*

9. What will be the output for the following code?

```
#include <stdio.h>
bool func(int arr[], int n, int sum)
{
    if (sum == 0)
        return true;
    if (n == 0 && sum != 0)
        return false;
    if (arr[n-1] > sum)
        return func(arr, n-1, sum);

    return func(arr, n-1, sum) || func(arr, n-1, sum-arr[n-1]);
}
int main()
{
    int arr[] = {4,6, 12, 2};
    int sum = 12;
    int n = sizeof(arr)/sizeof(arr[0]);
    if (func(arr, n, sum) == true)
        printf("true");
    else
        printf("false");
    return 0;
}
```

a) 12
b) 4 6 2
c) True
d) False

Answer: c
Explanation: The given code represents the recursive approach of solving the subset sum problem. The output for the code will be true if any subset is found to have sum equal to the desired sum, otherwise false will be printed.

10. What will be the output for the following code?

```
#include <stdio.h>
bool func(int arr[], int n, int sum)
{
        bool subarr[n+1][sum+1];
        for (int i = 0; i <= n; i++)
        subarr[i][0] = true;
        for (int i = 1; i <= sum; i++)
        subarr[0][i] = false;
        for (int i = 1; i <= n; i++)
        {
            for (int j = 1; j <= sum; j++)
            {
                if(j<arr[i-1])
                subarr[i][j] = subarr[i-1][j];
```

```
                if (j >= arr[i-1])
                subarr[i][j] = subarr[i-1][j] ||
                subarr[i - 1][j-arr[i-1]];
            }
        }
        return subarr[n][sum];
}

int main()
{
    int arr[] = {3, 3, 4, 4, 7};
    int sum = 5;
    int n = sizeof(arr)/sizeof(arr[0]);
    if (func(arr, n, sum) == true)
        printf("true");
    else
        printf("false");
    return 0;
}
```

a) true
b) false
c) 0
d) error in code

Answer: b
Explanation: The given code represents the dynamic programming approach of solving the subset sum problem. The output for the code will be true if any subset is found to have sum equal to the desired sum, otherwise false will be printed.

11. What will be the worst case time complexity for the following code?

```
#include <stdio.h>
bool func(int arr[], int n, int sum)
{
    if (sum == 0)
        return true;
    if (n == 0 && sum != 0)
        return false;
    if (arr[n-1] > sum)
        return func(arr, n-1, sum);
    return func(arr, n-1, sum) || func(arr, n-1, sum-arr[n-1]);
}
int main()
{
    int arr[] = {4,6, 12, 2};
    int sum = 12;
    int n = sizeof(arr)/sizeof(arr[0]);
    if (func(arr, n, sum) == true)
        printf("true");
    else
        printf("false");
    return 0;
}
```

a) O(n log n)
b) $O(n^2)$
c) $O(2^n)$
d) $O(n^2 \log n)$

Answer: c
Explanation: The given code represents the recursive approach solution of the subset sum problem. It has an exponential time complexity as it will require to check for all subsets in the worst case. It is equal to $O(2^n)$.

1. What is meant by the power set of a set?

a) subset of all sets
b) set of all subsets
c) set of particular subsets
d) an empty set

Answer: b
Explanation: Power set of a set is defined as the set of all subsets. Ex- if there is a set S={1,3} then power set of set S will be P={{},{1},{3}{1,3}}.

2. What is the set partition problem?
a) finding a subset of a set that has sum of elements equal to a given number
b) checking for the presence of a subset that has sum of elements equal to a given number
c) checking whether the set can be divided into two subsets of with equal sum of elements and printing true or false based on the result
d) finding subsets with equal sum of elements

Answer: c
Explanation: In set partition problem we check whether a set can be divided into 2 subsets such that the sum of elements in each subset is equal. If such subsets are present then we print true otherwise false.

3. Which of the following is true about the time complexity of the recursive solution of set partition problem?
a) It has an exponential time complexity
b) It has a linear time complexity
c) It has a logarithmic time complexity
d) it has a time complexity of O(n2)

Answer: a
Explanation: Set partition problem has both recursive as well as dynamic programming solution. The recursive solution has an exponential time complexity as it will require to check for all subsets in the worst case.

4. What is the worst case time complexity of dynamic programming solution of set partition problem(sum=sum of set elements)?
a) O(n)
b) O(sum)
c) O(n^2)
d) O(sum*n)

Answer: d
*Explanation: Set partition problem has both recursive as well as dynamic programming solution. The dynamic programming solution has a time complexity of O(n*sum) as it as a nested loop with limits from 1 to n and 1 to sum respectively.Answer: a*
Explanation: Set partition problem takes exponential time when we implement a recursive solution. Set partition problem is known to be a part of NP complete problems.Answer: b
Explanation: The recursive solution to set partition problem takes exponential time complexity whereas the dynamic programming solution takes polynomial time complexity. So dynamic programming solution is faster in terms of time complexity.

7. Which of the following is not true about set partition problem?
a) the recursive solution has a time complexity of O(2n)
b) there is no known solution that takes polynomial time
c) the recursive solution is slower than dynamic programming solution
d) the dynamic programming solution has a time complexity of O(n log n)

Answer: d
*Explanation: Recursive solution of set partition problem is slower than dynamic problem solution in terms of time complexity. Dynamic programming solution has a time complexity of O(n*sum).*

9. What will be the output for the given code?

```
#include <stdio.h>
#include <stdbool.h>
bool func1(int arr[], int n, int sum)
{
    if (sum == 0)
        return true;
    if (n == 0 && sum != 0)
        return false;
    if (arr[n-1] > sum)
        return func1(arr, n-1, sum);
    return func1(arr, n-1, sum) || func1(arr, n-1, sum-arr[n-1]);
}
bool func (int arr[], int n)
{
        int sum = 0;
        for (int i = 0; i < n; i++)
        sum += arr[i];
        if (sum%2 != 0)
        return false;
        return func1 (arr, n, sum/2);
}
int main()
{
    int arr[] = {4,6, 12, 2};
    int n = sizeof(arr)/sizeof(arr[0]);
    if (func(arr, n) == true)
        printf("true");
    else
        printf("false");
    return 0;
}
```

a) true
b) false
c) 4 6 2
d) 12

Answer: a
Explanation: The given code represents the recursive approach of solving the set partition problem. The code checks whether a set can be divided into 2 subsets such that the sum of elements in each subset is equal. If such a partition is possible then we print true otherwise false. In this case true should be printed.

10. What will be the output for the given code?

```
#include <stdio.h>
bool func (int arr[], int n)
{
        int sum = 0;
        int i, j;
        for (i = 0; i < n; i++)
        sum += arr[i];
        if (sum%2 != 0)
        return false;
        bool partition[sum/2+1][n+1];
        for (i = 0; i <= n; i++)
        partition[0][i] = true;
        for (i = 1; i <= sum/2; i++)
        partition[i][0] = false;
        for (i = 1; i <= sum/2; i++)
        {
            for (j = 1; j <= n; j++)
            {
                partition[i][j] = partition[i][j-1];
                if (i >= arr[j-1])
                partition[i][j] = partition[i][j] || partition[i - arr[j-1]][j-1];
```

```
            }
        }
        return partition[sum/2][n];
}
int main()
{
    int arr[] = {3, 3, 4, 4, 7};
    int n = sizeof(arr)/sizeof(arr[0]);
    if (func(arr, n) == true)
        printf("true");
    else
        printf("false");
    return 0;
}
```

a) true
b) false
c) 0
d) error

Answer: b
Explanation: The given code represents the dynamic programming approach of solving set partition problem. The code checks whether a set can be divided into 2 subsets such that the sum of elements in each subset is equal. If such a partition is possible then we print true otherwise false. In this case, false should be printed.

11. What will be the auxiliary space complexity of dynamic programming solution of set partition problem(sum=sum of set elements)?
a) O(n log n)
b) O(n^2)
c) O(2^n)
d) O(sum*n)

Answer: d
*Explanation: The auxiliary space complexity of set partition problem is required in order to store the partition table. It takes up a space of n*sum, so its auxiliary space requirement becomes O(n*sum).*

1. _____ has the lowest fault rate of all the page replacement algorithms.
a) Optimal page replacement algorithm
b) LRU replacement algorithm
c) FIFO
d) Counting based

Answer: a
Explanation: Optimal page replacement algorithm has the lowest fault rate as it has the knowledge of all the pages beforehand.

2. Optimal page replacement algorithm is also called as _____
a) LIFO
b) NRU
c) Clairvoyant replacement algorithm
d) Page buffering

Answer: c
Explanation: Optimal page replacement algorithm is also called a Clairvoyant replacement algorithm or Belady's optimal replacement algorithm.

3. In a optimal page replacement algorithm, when a page is to be replaced, which of the following pages is chosen?
a) Oldest page
b) Newest page
c) Frequently occurred page in the future

d) Not frequently occurred page in the future

Answer: d
Explanation: The page which doesn't occur more frequently in the future is chosen to be replaced with the page in the frame.Answer: a
Explanation: In an optimal page replacement algorithm, the page that is to be used later in the future is swapped out over a page that is to be used immediately.

5. Analysis of the optimal paging problem has been done through_____
a) Deterministic algorithm
b) Online algorithm
c) Euclid algorithm
d) Optimal algorithm

Answer: b
Explanation: Analysis of the optimal paging algorithm is done through an online algorithm. Efficiency is calculated through amortized analysis.

6. Optimal page replacement algorithm is implemented in _____
a) General-purpose operating system
b) Special-purpose operating system
c) In any kind of operating system
d) In Windows only

Answer: b
Explanation: Optimal page replacement algorithm is used in special-purpose operating system because it is impossible to compute time before which a page is used.

7. Optimal page replacement algorithm is said to satisfy _____
a) Online algorithm
b) Stack algorithm
c) Queue algorithm
d) Array algorithm

Answer: b
Explanation: Optimal page replacement algorithm is said to satisfy the stack algorithm. It is also called as inclusion property.

8. In a stack algorithm, the set of pages in a k-frame memory is always a subset of pages in a _____ frame memory.
a) k-1
b) k
c) k+1
d) k(k+1)

Answer: c
Explanation: Stack algorithm is satisfied if the set of pages in a k-frame memory is always a subset of pages in a k+1 frame memory.Answer: a
Explanation: The algorithm can be implemented in a general-purpose algorithm if the software is known beforehand and if it is amenable to the static analysis of the memory.

10. Consider a reference string 7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1 of frame size 3. Calculate the number of page faults using optimal page replacement algorithm.
a) 10
b) 9
c) 8
d) 7

Answer: b
Explanation: For the given string, the number of page faults using optimal page replacement algorithm is 9. It is solved in the given diagram.Answer: c
Explanation: For the given reference string, the number of page faults using optimal page replacement algorithm is said to be 8. It is solved in the given diagram.Answer: c
Explanation: For the given reference string, the number of page faults using optimal page replacement algorithm is said to be 14.

1. _____ is a typical online problem from the competitive analysis to determine the optimal solution.
a) Page replacement algorithm
b) Segmentation
c) Paging
d) Segmentation with paging

Answer: a
Explanation: Page replacement is a typical online problem from the competitive analysis. They determine which pages to page out or write to disk.

2. Which of the following is the simplest page replacement algorithm?
a) FIFO
b) Optimal page replacement
c) LRU replacement
d) Counting based replacement

Answer: a
Explanation: FIFO is the simplest page replacement algorithm since LRU and optimal replacement algorithms require past and future data patterns respectively.

3. _____ algorithm associates with each page the time when the page was brought into memory.
a) Optimal page replacement
b) FIFO
c) LRU replacement algorithm
d) Counting based replacement

Answer: b
Explanation: FIFO algorithm associates with each page the time when the page was brought into memory. The new page is inserted at the tail of the queue.Answer: a
Explanation: One of the rules of the page replacement algorithm is that, as the number of frames available increases, the number of page faults decreases.

5. Which of the following page replacement algorithms return the minimum number of page faults?
a) LRU replacement algorithm
b) Optimal page replacement algorithm
c) FIFO
d) Counting based replacement

Answer: b
Explanation: Though FIFO is the simplest of all algorithms, optimal page replacement algorithm returns the minimum number of page faults.

6. Which of the following is the main drawback of FIFO page replacement algorithm?
a) Requirement of large memory
b) Frame allocation
c) Reduction in multiprogramming
d) Reduced optimality

Answer: c
Explanation: The main drawback of FIFO page replacement algorithm is that it reduces the level of multiprogramming

and also causes Belady's anomaly.

7. Which of the following is required to determine the number of page faults in FIFO?
a) Page number
b) Page frame number
c) Memory capacity
d) Segment number

Answer: b
Explanation: To determine the number of page faults in a page replacement algorithm using FIFO, we require page frame number.

8. In a FIFO algorithm, when a page is to be replaced, which of the following page is chosen?
a) Oldest page
b) Newest page
c) Frequently occurred page in past
d) Frequently occurred page in future

Answer: a
Explanation: In FIFO page replacement algorithm, when a page is to be replaced, the oldest page is chosen and replaced at the tail of the queue.Answer: b
Explanation: The cost of a FIFO algorithm is cheap and intuitive and it is used in poor practical applications.

10. FIFO algorithm is used by _____ operating system.
a) Linux
b) Mac
c) Windows
d) VAX/VMS

Answer: d
Explanation: Of the following given operating systems, VAX/VMS uses a FIFO algorithm.

11. What is the competitive analysis of the FIFO algorithm?
a) k/k+1
b) k+1
c) k(k+1)
d) k/(k-h+1)

Answer: d
Explanation: The competitive analysis of a FIFO algorithm is mathematically found to be k/(k-h+1) where k and h are some constants used in page replacement and always, h<=k.

12. Under which of the following scenarios is page replacement algorithm required?
a) When total memory exceeds physical memory
b) To determine the number of frames for each process
c) When paging and segmentation are to be used
d) Execution of a process, not in memory

Answer: a
Explanation: An appropriate page replacement algorithm is required when the total memory requirements exceed the physical memory.Answer: d
Explanation: For the given reference string of frame size 3, the number of page faults is calculated to be 15. It is explained in the diagram.

Answer: c
Explanation: For the given reference string of frame size 4, the number of page faults is calculated to be 10. It is explained in the diagram.

15. _____ states that, on a page fault, the frame that has been in memory the longest is replaced.
a) Belady's anomaly
b) Second chance algorithm
c) Partial second chance algorithm
d) LRU replacement algorithm

Answer: a
Explanation: Belady's anomaly states that, on a page fault, the frame that has been in memory the longest is replaced. It occurs in FIFO algorithm.

1. Topological sort can be applied to which of the following graphs?
a) Undirected Cyclic Graphs
b) Directed Cyclic Graphs
c) Undirected Acyclic Graphs
d) Directed Acyclic Graphs

Answer: d
Explanation: Every Directed Acyclic Graph has one or more topological ordering whereas Cyclic and Undirected graphs can't be ordered topologically.

2. Most Efficient Time Complexity of Topological Sorting is? (V – number of vertices, E – number of edges)
a) O(V + E)
b) O(V)
c) O(E)
d) O(V*E)

Answer: a
Explanation: The topological sort algorithm has complexity same as Depth First Search. So, DFS has a complexity O(V+E).

3. In most of the cases, topological sort starts from a node which has _____
a) Maximum Degree
b) Minimum Degree
c) Any degree
d) Zero Degree

Answer: d
Explanation: Topological sort starts with a node which has zero degree. If multiple such nodes exists then it can start with any node.

4. Which of the following is not an application of topological sorting?
a) Finding prerequisite of a task
b) Finding Deadlock in an Operating System
c) Finding Cycle in a graph

d) Ordered Statistics

Answer: d
Explanation: Topological sort tells what task should be done before a task can be started. It also detects cycle in the graph which is why it is used in the Operating System to find the deadlock. Ordered statistics is an application of Heap sort.

5. Topological sort of a Directed Acyclic graph is?
a) Always unique
b) Always Not unique
c) Sometimes unique and sometimes not unique
d) Always unique if graph has even number of vertices

Answer: c
Explanation: The topological sort of a graph can be unique if we assume the graph as a single linked list and we can have multiple topological sort order if we consider a graph as a complete binary tree.

6. Topological sort can be implemented by?
a) Using Depth First Search
b) Using Breadth First Search
c) Using Depth and Breadth First Search
d) Using level ordered search

Answer: c
Explanation: We can implement topological sort by both BFS and DFS. In BFS, we use queue as data structure and in DFS, we use Linked list (if recursive) or Stack (if not recursive) as data structure.

7. Topological sort is equivalent to which of the traversals in trees?
a) Pre-order traversal
b) Post-order traversal
c) In-order traversal
d) Level-order traversal

Answer: a
Explanation: In pre-order traversal of trees, we process the root first and then child from left to right.

8. A man wants to go different places in the world. He has listed them down all. But there are some places where he wants to visit before some other places. What application of graph can he use to determine that?
a) Depth First Search
b) Breadth First Search
c) Topological Sorting
d) Dijkstra's Shortest path algorithm

Answer: c
Explanation: As the definition of topological sorting suggests, it is the way to do tasks in prescribed order. So, if he does topological sorting, it will be easy for him to recognize what should be the order to visit different places.

9. When the topological sort of a graph is unique?
a) When there exists a hamiltonian path in the graph
b) In the presence of multiple nodes with indegree 0
c) In the presence of single node with indegree 0
d) In the presence of single node with outdegree 0

Answer: a
Explanation: A hamiltonian path exists in a Directed Acyclic Graph when all pairs of consecutive vertices are in sorted order and are connected by edges. In such a case, there exists a unique topological sorting order.

1. Which of the following is an alternative name of the quickselect algorithm?

a) quick sort
b) hoare's selection algorithm
c) tony's selection algorithm
d) kruskal's algorithm

Answer: b
Explanation: Quick select is a selection algorithm. It was developed by Tony Hoare, thus it is also known as Hoare's selection algorithm.

2. Quickselect is an example of _____
a) sorting algorithm
b) selection algorithm
c) greedy algorithm
d) searching algorithm

Answer: b
Explanation: Quickselect is an example of a selection algorithm. It finds the kth smallest element from the given list.

3. What will be the output if quickselect algorithm is applied to the array arr={1,5,4,3,7} with k given as 4?
a) 1
b) 3
c) 4
d) 5

Answer: d
Explanation: Quickselect algorithm finds the kth smallest element from the given list. So as here the given value of k is 4 so we need to find the fourth smallest element which is 5 in the given array.

4. What is the auxiliary space requirement of the quickselect algorithm?
a) $O(n^2)$
b) O(n)
c) O(n log n)
d) O(1)

Answer: d
Explanation: Quickselect algorithm requires no extra space in order to calculate the desired result. It performs manipulations in the given array itself so its auxiliary space requirement will be O(1). Answer: a
Explanation: Quickselect's auxiliary space requirement is O(1). So quickselect qualifies as an in-place algorithm.

6. What is the best case time complexity of quickselect?
a) O(n log n)
b) $O(n^2)$
c) O(n)
d) O(log n)

Answer: c
Explanation: Best case time complexity of quickselect is O(n). It is observed in the case where good pivots are chosen consistently then the array size decreases exponentially and thus the required element is found in linear time.

7. Quickselect's algorithm is similar to which of the following algorithm?
a) Merge sort
b) Quicksort
c) Selection sort
d) Counting sort

Answer: b
Explanation: Both quicksort and quickselect algorithms are closely related. They were developed by the same person. Like quicksort, quickselect also works by choosing a pivot and partitioning array.

8. What is the worst case time complexity of quickselect?
a) O(n log n)
b) $O(n^2)$
c) O(n)
d) O(log n)

Answer: b
Explanation: Worst case complexity occurs in the case where bad pivots are chosen consistently due to which the size of the array decreases in the steps of 1 only. This leads to a time complexity of $O(n^2)$.

9. What is the average case time complexity of quickselect?
a) O(n log n)
b) $O(n^2)$
c) O(n)
d) O(log n)

Answer: c
Explanation: In quickselect, we don't recur for both portions of the array. Only that portion is considered where the smallest element lies. So this causes the average time complexity to be O(n).

10. Which of the following is a disadvantage of quickselect?
a) Poor space complexity
b) Poor best case time complexity
c) Poor average case time complexity
d) Poor worst case time complexity

Answer: d
Explanation: Quickselect has a poor worst case time complexity of $O(n^2)$. There are algorithms which have O(n) time complexity in the worst case.

b)

```
function quickSelect(list, left, right, k)
   if left = right
      return list[left]
   Select a pivotIndex between left and right
   pivotIndex := partition(list, left, right,  pivotIndex)
   if k = pivotIndex
      return list[k]
   else if k < pivotIndex
      right := pivotIndex - 1
   else
      left := pivotIndex + 1
```

c)

```
function quickSelect(list, left, right, k)
   if left = right
      return list[right]
   Select a pivotIndex between left and right
   pivotIndex := partition(list, left, right,  pivotIndex)
   if k = pivotIndex
      return list[k]
   else if k > pivotIndex
      right := pivotIndex - 1
   else
      left := pivotIndex + 1
```

d)

```
function quickSelect(list, left, right, k)
```

```
   if left = right
      return list[left]
   Select a pivotIndex between left and right
   pivotIndex := partition(list, left, right,  pivotIndex)
   if k = pivotIndex
      return list[k]
   else if k < pivotIndex
      right := pivotIndex +1
   else
      left := pivotIndex -1
```

1. What is co-ordinate compression?
a) process of reassigning co-ordinates to remove gaps
b) inserting gaps in a co-ordinate system
c) removing redundant co-ordinates
d) adding extra gaps

Answer: a
Explanation: Co-ordinate compression is the process of reassigning co-ordinates in order to remove gaps. This helps in improving efficiency of a solution.

2. What is the need for co-ordinate compression?
a) for improving time complexity
b) for improving space complexity
c) for improving both time and space complexity
d) for making code simpler

Answer: c
Explanation:Co-ordinate compression is the process of reassigning co-ordinates in order to remove gaps. This helps in improving both time and space complexity of a solution.

3. What is the output for the following code?

```
#include <bits/stdc++.h>
using namespace std;
void convert(int a[], int n)
{
        vector <pair<int, int> > vec;
        for (int i = 0; i < n; i++)
                vec.push_back(make_pair(a[i], i));
        sort(vec.begin(), vec.end());
        for (int i=0; i<n; i++)
                a[vec[i].second] = i;
}
void printArr(int a[], int n)
{
        for (int i=0; i<n; i++)
                cout << a[i] << " ";
}
int main()
{
        int arr[] = {10,8,2,5,7};
        int n = sizeof(arr)/sizeof(arr[0]);
        convert(arr , n);
        printArr(arr, n);
        return 0;
}
```

a) 4 3 0 1 2
b) 1 2 3 4 5
c) 5 4 1 2 3
d) 0 1 2 3 4

Answer: a

Explanation: The given code converts the elements of the input array. They are replaced with their respective position number in the sorted array.

4. What will be the time complexity of given code?

```
#include <bits/stdc++.h>
using namespace std;
void convert(int a[], int n)
{
        vector <pair<int, int> > vec;
        for (int i = 0; i < n; i++)
                vec.push_back(make_pair(a[i], i));
        sort(vec.begin(), vec.end());
        for (int i=0; i<n; i++)
                a[vec[i].second] = i;
}
void printArr(int a[], int n)
{
        for (int i=0; i<n; i++)
                cout << a[i] << " ";
}
int main()
{
        int arr[] = {10,8,2,5,7};
        int n = sizeof(arr)/sizeof(arr[0]);
        convert(arr , n);
        printArr(arr, n);
        return 0;
}
```

a) O(n)
b) O(n log n)
c) O(n^2)
d) O(log n)

Answer: b
Explanation: The time complexity of the given code will be governed by the time complexity of the sorting algorithm used. As this code uses in built sorting of C++ so it will take O(n log n) time.

5. What is the auxiliary space complexity of the given code?

```
#include <bits/stdc++.h>
using namespace std;
void convert(int a[], int n)
{
        vector <pair<int, int> > vec;
        for (int i = 0; i < n; i++)
                vec.push_back(make_pair(a[i], i));
        sort(vec.begin(), vec.end());
        for (int i=0; i<n; i++)
                a[vec[i].second] = i;
}
void printArr(int a[], int n)
{
        for (int i=0; i<n; i++)
                cout << a[i] << " ";
}
int main()
{
        int arr[] = {10,8,2,5,7};
        int n = sizeof(arr)/sizeof(arr[0]);
        convert(arr , n);
        printArr(arr, n);
        return 0;
}
```

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

Answer: b
Explanation: The given code uses an auxiliary space of O(n). It is used by a vector which pairs each element of the array with their respective index number of the original array.

6. What will be the output of the following code?

```
#include <bits/stdc++.h>
using namespace std;
void convert(int arr[], int n)
{
        int temp[n];
        memcpy(temp, arr, n*sizeof(int));
        sort(temp, temp + n);
        unordered_map<int, int> map;
        int sort_index = 0;
        for (int i = 0; i < n; i++)
                map[temp[i]] = sort_index++;
        for (int i = 0; i < n; i++)
                arr[i] = map[arr[i]];
}
void printArr(int arr[], int n)
{
        for (int i=0; i<n; i++)
                cout << arr[i] << " ";
}
int main()
{
        int arr[] = {3,5,2,4};
        int n = sizeof(arr)/sizeof(arr[0]);
        convert(arr , n);
        printArr(arr, n);
        return 0;
}
```

a) 0 2 3 4
b) 1 3 0 2
c) 2 4 1 3
d) 1 2 3 4

Answer: b
Explanation: The given code converts the elements of input array. They are replaced with their respective position number in the sorted array.

7. What is the time complexity of the following code?

```
#include <bits/stdc++.h>
using namespace std;
void convert(int arr[], int n)
{
        int temp[n];
        memcpy(temp, arr, n*sizeof(int));
        sort(temp, temp + n);
        unordered_map<int, int> map;
        int sort_index = 0;
        for (int i = 0; i < n; i++)
                map[temp[i]] = sort_index++;
        for (int i = 0; i < n; i++)
                arr[i] = map[arr[i]];
}
void printArr(int arr[], int n)
{
```

```
        for (int i=0; i<n; i++)
                cout << arr[i] << " ";
}
int main()
{
        int arr[] = {10, 20, 15, 12, 11, 50};
        int n = sizeof(arr)/sizeof(arr[0]);
        convert(arr , n);
        printArr(arr, n);
        return 0;
}
```

a) O(n)
b) O(1)
c) O(n log n)
d) O(n^2)

Answer: c
Explanation: The time complexity of the given code will be governed by the time complexity of the sorting algorithm used. As this code uses inbuilt sorting of C++ so it will take O(n log n) time.

8. What will be the auxiliary space complexity of the following code?

```
#include <bits/stdc++.h>
using namespace std;
void convert(int arr[], int n)
{
        int temp[n];
        memcpy(temp, arr, n*sizeof(int));
        sort(temp, temp + n);
        unordered_map<int, int> map;
        int sort_index = 0;
        for (int i = 0; i < n; i++)
                map[temp[i]] = sort_index++;
        for (int i = 0; i < n; i++)
                arr[i] = map[arr[i]];
}
void printArr(int arr[], int n)
{
        for (int i=0; i<n; i++)
                cout << arr[i] << " ";
}
int main()
{
        int arr[] = {10, 20, 15, 12, 11, 50};
        int n = sizeof(arr)/sizeof(arr[0]);
        convert(arr , n);
        printArr(arr, n);
        return 0;
}
```

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

Answer: a
Explanation: The given code uses an auxiliary space of O(n). It is used by a vector which pairs each element of the array with their respective index number of the original array.Answer: a
Explanation: The idea behind co-ordinate compression is to reduce the number of squares in a graph by converting them into rectangles of viable size. This reduces the time complexity of traversal.Answer: b
Explanation: Co-ordinate compression technique can be applied where such optimization is suitable. It does not require to co-ordinate system problem only.

1. What is the purpose of using square root decomposition?
a) to reduce the time complexity of a code
b) to increase the space complexity of a code
c) to reduce the space complexity of a code
d) to reduce the space and time complexity of a code

Answer: a
Explanation: Square decomposition is mainly used in competitive programming to optimize code. It reduces the time complexity by a factor of √n.

2. By what factor time complexity is reduced when we apply square root decomposition to a code?
a) n
b) √n
c) n^2
d) $n^{-1/2}$

Answer: b
Explanation: In square root decomposition a given array is decomposed into small parts each of size √n. This reduces the time complexity of the code by a factor of √n.

3. What will be the worst case time complexity of finding the sum of elements in a given range of (l,r) in an array of size n?
a) O(n)
b) O(l+r)
c) O(l-r)
d) O(r-l)

Answer: a
Explanation: For a given array of size n we have to traverse all n elements in the worst case. In such a case l=0, r=n-1 so the time complexity will be O(n).

4. What will be the worst case time complexity of finding the sum of elements in a given range of (l,r) in an array of size n when we use square root optimization?
a) O(n)
b) O(l+r)
c) O(√n)
d) O(r-l)

Answer: c
Explanation: When we use square root optimization we decompose the given array into √n chunks each of size √n. So after calculating the sum of each chunk individually, we require to iterate only 3√n times to calculate the sum in the worst case.*

5. Total how many iterations are required to find the sum of elements in a given range of (l,r) in an array of size n when we use square root optimization?
a) √n
b) 2*√n
c) 3*√n
d) n*√n

Answer: c
Explanation: After calculating the sum of each chunk individually we require to iterate only 3√n times to calculate the sum in the worst case. It is because two of the √n factors consider the worst case time complexity of summing elements in the first and last block. Whereas the third √n considers the factor of summing the √n chunks.*

6. What will be the time complexity of update query operation in an array of size n when we use square root optimization?

a) O(√n)
b) O(n)
c) O(1)
d) O(n^2)

Answer:c
Explanation: The time complexity of query operation remains the same in both square root optimized code and non optimized code. We simply find the chunk in which the update requires to be performed and then add the new updated value at the desired index.Answer: b
Explanation: Square root decomposition technique can be applied to an array with any number of indices. It does not require this number to be a perfect square.

8. What will be the worst case time complexity of code to find sum in given query range (l,r) in an array of size n with q number of such queries?
a) O(n)
b) O(q)
c) O(n*q)
d) O(n+q)

Answer: c
*Explanation: For finding the result of one query the worst case time complexity will be n. So for q queries the time complexity will be O(q*n). This can be reduced by using square root optimization.*

9. What will be the worst case time complexity of code to find sum in given query range (l,r) in an array of size n with q number of such queries when we apply MO's algorithm?
a) O(n*q)
b) O(n)
c) O((q+n)√n)
d) O(q*√n)

Answer: c
Explanation: Mo's algorithm requires O(q√n) + O(n*√n) time for processing all the queries. It is better than the naive solution where O(n*q) time is required.Answer: a*
Explanation: Mo's algorithm uses the result of the previous query in order to compute the result of the given query. It cannot be implemented where such a scenario is not possible.

11. What will be the time complexity of the code to find a minimum element from an array of size n and uses square root decomposition(exclude pre processing time)?
a) O(√n)
b) O(n)
c) O(1)
d) O(n^2)

Answer: a
Explanation: For finding the minimum element in a given array of size n using square root decomposition we first divide the array into √n chunks and calculate the result for them individually. So for a given query, the result of middle blocks has to be calculated along with the extreme elements. This takes O(√n) time in the worst case.