|  |  |  |  |
| --- | --- | --- | --- |
| Answer All Questions PART- B (5 x 16 = 80 Marks) | | | |
| B1a)i) | What are the asymptotic notations used for worst case, best case and average case analysis of an algorithm?  ANSWER:  Worst case🡪Big Oh notation(O)  f(n) <= C g(n) for all n >= n0, C > 0 and n0 >= 1. Then we can represent f(n) as O(g(n)).    Best case🡪 Big Omega Notation  f(n)>=C\*g(n) for all n>=n0,C >0 and n0>= 1. Then we can represent f(n) as Ω(g(n)).    Average case🡪Big Theta Notation  C1g(n)<=f(n)<=C2g(n) for all n>=n0,C1,C2>0 and n0>=1,then we can represent f(n) as Θ(g(n)). | | 06 |
| B1a)ii) | Given an array of characters of size ‘N’. Propose a sorting algorithm to arrange the characters in alphabetical order using divide and conquer technique.  Examples:  Input: arr[ ] = {“advanceddatastructures”}  Output: arr[ ]={“aaaaccdddeenrrssttuuv”}  Input: arr[ ] = {“applepp“}  Output: arr[ ] = {“aelpppp”}  ANSWER  main()  {  char temp;  int i = 0;  char Strings[NUM][LEN];  printf("Please enter %d strings, one per line:\n", NUM);  for(i; i<25; i++){  fgets(&Strings[i][0], LEN,stdin);  }  i=0;  puts("\nHere are the strings in the order you entered:");  for (i; i<25; i++){  printf("%s\n",Strings[i]);  }  mergesort(0,NUM,Strings);  i=0;  puts("\nHere are the strings in alphabetical order");  for (i; i<25; i++){  printf("%s\n",Strings[i]);  }  }  int mergesort(int left, int right, char list[NUM][LEN]) {  if (right - left <= 1)  {  return 0;  }  int left\_start = left;  int left\_end = (left+right)/2;  int right\_start = left\_end;  int right\_end = right;  mergesort( left\_start, left\_end, list);  mergesort( right\_start, right\_end, list);  merge(list, left\_start, left\_end, right\_start, right\_end);  }  int merge(char list[NUM][LEN], int left\_start, int left\_end, int right\_start, int right\_end)  {  int left\_length = left\_end - left\_start;  int right\_length = right\_end - right\_start;  char \*left\_half[left\_length];  char \*right\_half[right\_length];  int r = 0;  int l = 0;  int i = 0;  for (i = left\_start; i < left\_end; i++, l++)  {  strcpy(left\_half[l], list[i]);  }  for (i = right\_start; i < right\_end; i++, r++)  {  strcpy(right\_half[r], list[i]);  }  for ( i = left\_start, r = 0, l = 0; l < left\_length && r < right\_length; i++)  {  if ( strcmp(left\_half[l], right\_half[r])<0 )  { strcpy(list[i], left\_half[l++]); }  else { strcpy(list[i], right\_half[r++]); }  }  for ( ; l < left\_length; i++, l++) { strcpy(list[i], left\_half[l]); }  for ( ; r < right\_length; i++, r++) { strcpy(list[i], right\_half[r]); }  return 0;  } | | 10 |
| (OR) | | | |
| B1b)i) | Write the recurrence equation for the best and worst case analysis of Quick Sort algorithm. Solve it using substitution method.  ANSWER | | 06 |
| B1b)ii) | Jim wishes to find Kim, the dragon who is like a father to him. But in order to find him he has to complete certain number of tasks. Tasks are defined by string of lowercase English alphabets and he may be asked to complete the same task more than once. He has a list of N tasks. But he wants to complete the tasks which occur multiple times together. Also he want to complete the tasks which occurs least frequently first. If two tasks have same frequency the lexicographically smaller will be completed first. Help him prepare such a list representing frequency of tasks and task name separated by a space. Write an algorithm to help Jim to find the solution and state its time complexity.  **Input:**  First line contains the number of test cases T. For each of the T test cases first line is number N the number of Tasks followed by N strings of lowercase English alphabet.  **Output:**  For each test case output the list as explained above.  SAMPLE INPUT  1 //test case  10 //no of tasks  abcd  bcd  abc  abc  abc  bcd  bge  dbaa  bcd  bge  SAMPLE OUTPUT  1 abcd  1 dbaa  2 bge  3 abc  3 bcd  ANSWER  *#include*<stdio.h>  struct task{  char name[10];  int freq;  };  int cmp(const void\*a,const void\*b){  struct task x= \*(struct task\*)a;  struct task y= \*(struct task\*)b;  if(x.freq > y.freq)  return 1;  else if(strcmp(x.name,y.name)>0 && x.freq == y.freq)  return 1;  return 0;  }  int main()  {  int n,test,i;  struct task t[10240];  scanf("%d",&test);  while(test--){  scanf("%d",&n);  for(i=0; i<n; i++){  scanf("%s",t[i].name);  t[i].freq=1;  }  qsort(t,n,sizeof(struct task),cmp);  int j=0;  for(i=1; i<n; i++){  if(strcmp(t[i].name,t[i-1].name)==0){  t[j].freq++;  t[i].freq=0;  }  else  j=i;  }  qsort(t,n,sizeof(struct task),cmp);  for(i=0; i<n; i++){  if(t[i].freq!=0)  printf("%d %s\n",t[i].freq,t[i].name);  }  }  return 0;  }  Time Complexity:O(n log n) | | 10 |
| B2a)i) | Write the steps involved in analyzing recursive algorithm with an example of your own.  ANSWER   1. Decide on input size parameter 2. Identify the basic operation 3. Does C(n) depends also on input type? 4. Set up a recurrence relation 5. Solve the recurrence or, at least establish the order of growth of its solution | | 04 |
| B2a)ii) | Write a C function to find if a given integer X appears more than N/2 times in a sorted array of N integers. Write a function say isMajority( ) that takes an array (arr[ ] ), array’s size (N) and a number to be searched (X) as parameters and returns true if X is a majority element (present more than N/2 times).  Examples:  Input: arr[] = {1, 2, 3, 3, 3, 3, 10}, X = 3  Output: True (x appears more than N/2 times in the given array)  Input: arr[] = {1, 1, 2, 4, 4, 4, 6, 6}, X = 4  Output: False (x doesn't appear more than N/2 times in the given array)  Input: arr[] = {1, 1, 1, 2, 2}, X = 1  Output: True (X appears more than N/2 times in the given array)   1. Solve the problem using brute force approach and state its running time. 2. If the designed algorithm is not having O(N) linear running time then redesign it to optimize the running time.   ANSWER  BRUTE FORCE APPROACH  void findMajority(int arr[], int n)  {      int maxCount = 0;      int index = -1; // sentinels      for(int i = 0; i < n; i++)      {          int count = 0;          for(int j = 0; j < n; j++)          {              if(arr[i] == arr[j])              count++;          }          if(count > maxCount)          {              maxCount = count;              index = i;          }      }      if (maxCount > n/2)  printf(“%d\n”,arr[index]);      else  printf(“No Majority Element\n”);  }  int main()  {      int arr[] = {1, 1, 2, 1, 3, 5, 1};      int n = sizeof(arr) / sizeof(arr[0]);      findMajority(arr, n);      return 0;  }  Time Complexity:O(n2)  O(N)-REDESIGNED ALGORITHM  # include <stdio.h>  # include <stdbool.h>  bool isMajority(int arr[], int n, int x)  {      int i;      int last\_index = n%2? (n/2+1): (n/2);      for (i = 0; i < last\_index; i++)      {          if (arr[i] == x && arr[i+n/2] == x)              return 1;      }      return 0;  }  int main()  {       int arr[] ={1, 2, 3, 4, 4, 4, 4};       int n = sizeof(arr)/sizeof(arr[0]);       int x = 4;       if (isMajority(arr, n, x))          printf("%d appears more than %d times in arr[]", x, n/2);       else          printf("%d does not appear more than %d times in arr[]",x, n/2);     return 0;  } | | 12 |
|  | OR | |  |
| B2b)i) | Given a sorted array of N distinct integers A[1]…..A[N], describe an O(logN) algorithm to find out whether there is an index ‘i’ such that A[i]=i. Why does your algorithm run in the claimed time bound? Justify.  ANSWER:  #include<stdio.h>  int binarySearch(int arr[], int low, int high)  {      if(high >= low)      {          int mid = (low + high)/2;          if(mid == arr[mid])              return mid;          if(mid > arr[mid])              return binarySearch(arr, (mid + 1), high);          else              return binarySearch(arr, low, (mid -1));      }      return -1;  }  int main()  {      int arr[10] = {-10, -1, 0, 3, 10, 11, 30, 50, 100};      int n = sizeof(arr)/sizeof(arr[0]);      printf("Fixed Point is %d", binarySearch(arr, 0, n-1));      getchar();      return 0;  }  Output:  Fixed Point is 3  Algorithmic Paradigm: Divide & Conquer Time Complexity: O(Logn)  Justification:  The above algorithm runs in Divide and conquer algorithm as it uses Binary search technique in which the input array should be sorted. | | 06 |
| B2b)ii) | Karl has an array of integers. He wants to reduce the array until all remaining elements are equal. Determine the fewest number of elements to delete to reach his goal. Write an algorithm for the given problem.  For example, if his array is arr=[1,2,2,3], we see that he can delete the 2 elements 1 and 3 leaving arr=[2,2]. He could also delete both twos and either the 1 or the 3, but that would take 3 deletions. The minimum number of deletions is 2.  **Input Format**  The first line contains an integer n, the number of elements in arr. The next line contains m space-separated integers arr[i].  **Output Format**  Print a single integer denoting the minimum number of elements Karl must delete for all elements in the array to be equal.  **Sample Input**  5  3 3 2 1 3  **Sample Output**  2  **Explanation**  Array arr=[3,3,2,1,3]. If we delete arr[2]=2and arr[3]=1, all of the elements in the resulting array, A=[3,3,3], will be equal. Deleting these 2 elements is minimal. Our only other options would be to delete 4 elements to get an array of either [1] or [2].  ANSWER  int main() {  int n,i,j,max=0,cnt;  scanf("%d",&n);  int a[n];  for(i=0;i<n;i++)  scanf("%d",&a[i]);  for(i=0;i<n;i++)  {  cnt=0;  for(j=0;j<n;j++)  {  if(a[i]==a[j])  cnt++;  }  if(cnt>max)  max=cnt;  }  printf("%d",n-max);  return 0;  } | | 10 |
| B3a)i) | Perform the following operations using substitution method:  1) Solve the given recurrence equation.  T(n)= 2T(n-1) -1  T(0)=1  2) Write the recurrence for the following recursive function and solve.  void function(int n)  {  if( n<=1)  return;  else  {  print(\*);  function(n/2);  function(n/2);  }  }  ANSWER:  1) T(n)= 2T(n-1) -1 🡪1  T(0)=1  Sub n=n-1  T(n-1)=2T(n-2)-1 🡪2  Sub 2 in 1  T(n)=2(2T(n-2)-1)-1  T(n)=22T(n-2)-2-1  ….  T(n)=2iT(n-i)-2i-1-2i-2-…..-1  Sub n-i=0  n=i  T(n)=2nT(0)-2n-1-2n-2-…..-1  =2n-2n-1-……..-1  =2n-(2n-1)  =1  Time complexity=O(1)  2) | | 08 |
| B3a)ii) | Given an integer X, find square root of it. If X is not a perfect square, then return floor(√X). Write an algorithm to solve the given problem using appropriate design strategy. State its time complexity.  Examples :  Input: X = 4 Output: 2  Input: X= 11 Output: 3  ANSWER   |  | | --- | | int floorSqrt(int x)  {      if (x == 0 || x == 1)      return x;      int i = 1, result = 1;      while (result <= x)      {        i++;        result = i \* i;      }      return i - 1;  }  int main()  {      int x = 11;  printf(“square root=%d”,floorSqrt(x));      return 0;}  Time complexity: O(n) | | | 08 |
| (OR) | | | |
| B3b)i) | What is the output of following program and write the time complexity for the same?   |  | | --- | | #include <stdio.h>  void funprint(int n, int j)  {     if (j >= n)        return;     if (n-j > 0 && n-j >= j)          printf("%d %dn", j, n-j);     funprint(n, j+1);  }  int main()  {      int n = 8;      funprint(n, 1);  }  ANSWER  Output  1 7  2 6  3 5  4 4  Time Complexity:O(n) | | | 04 |
| B3b)ii) | HackerLand University has the following grading policy:   * Every student receives a grade in the inclusive range from 0 to 100. * Any grade less than 40 is a failing grade.   Sam is a professor at the university and likes to round each student's grade according to these rules:   * If the difference between the grade and the next multiple of 5 is less than 3, round grade up to the next multiple of 5. * If the value of grade is less than 38, no rounding occurs as the result will still be a failing grade.   For example, grade=84 will be rounded to 85 but grade=29 will not be rounded because the rounding would result in a number that is less than 40.  Given the initial value of grade for each of Sam's n students, write code to automate the rounding process. Complete the function solve( ) that takes an integer array of all grades, and return an integer array consisting of the rounded grades. For each gradei, round it according to the rules above and print the result on a new line.  Input Format  The first line contains a single integer denoting n (the number of students).  Each line ‘i‘of the ‘n’ subsequent lines contains a single integer, gradei, denoting student i's grade.  Constraints   * 1<=n<=60 * 0<= gradei<=100   Output Format  For each gradei of the ‘n’ grades, print the rounded grade on a new line.  Sample Input 0  4  73  67  38  33  Sample Output 0  75  67  40  33  Explanation 0   1. Student 1 received a 73, and the next multiple of 5 from 73 is 75. Since 75-73<3, the student's grade is rounded to 75. 2. Student 2 received a 67, and the next multiple of 5 from 67 is 70. Since 70-67=3, the grade will not be modified and the student's final grade is 67. 3. Student 3 received a 38, and the next multiple of 5 from 38 is 40. Since 40-38<3, the student's grade will be rounded to 40. 4. Student 4 received a grade below 38, so the grade will not be modified and the student's final grade is 33.   ANSWER  int main(){  int n;  scanf(“%d”,&n);  for(int a0 = 0; a0 < n; a0++){  int grade;  scanf(“%d”,&grade);  if (grade >= 38) {  int rem = grade % 5;  if (rem >= 3) grade += 5 - rem;  }  Printf(“Grade=%d\n”,grade);  }  return 0;  } | | 12 |
| B4a)i) | Write a function rotate(ar[], d, n) that rotates arr[] of size n by d elements.    Rotation of the above array by 2 to the left will make array    1) Write an algorithm to solve the rotate problem using your own brute force approach.  2) Devise a procedure for the same rotate problem using the function reverse(arr, n), which reverses the array of given size n.  ANSWER  **Brute Force Approach**  #include <stdio.h>  void leftRotate(int arr[], int d, int n)  {    int i;    for (i = 0; i < d; i++)      leftRotatebyOne(arr, n);  }  void leftRotatebyOne(int arr[], int n)  {    int temp = arr[0], i;    for (i = 0; i < n-1; i++)       arr[i] = arr[i+1];    arr[i] = temp;  }  void printArray(int arr[], int n)  {    int i;    for (i = 0; i < n; i++)      printf("%d ", arr[i]);  }  int main()  {     int arr[] = {1, 2, 3, 4, 5, 6, 7};     leftRotate(arr, 2, 7);     printArray(arr, 7);     return 0;  }  **Using Reverse Algorithm**  include<stdio.h>  void leftRotate(int arr[], int d, int n)  {      rvereseArray(arr, 0, d-1);      rvereseArray(arr, d, n-1);      rvereseArray(arr, 0, n-1);  }  void printArray(int arr[], int size)  {      int i;      for (i = 0; i < size; i++)          printf("%d ", arr[i]);  }  void rvereseArray(int arr[], int start, int end)  {      int temp;      while (start < end)      {          temp = arr[start];          arr[start] = arr[end];          arr[end] = temp;          start++;          end--;      }  }  int main()  {      int arr[] = {1, 2, 3, 4, 5, 6, 7};      int n = sizeof(arr)/sizeof(arr[0]);      int d = 2;      leftRotate(arr, d, n);      printArray(arr, n);      return 0;  } | | 10 |
| B4a)ii) | Given an array of positive integers. The task is to print the minimum product of any two numbers of the given array.  Input:  The first line of input contains an integer T denoting the number of test cases. Then T test cases follow. Each test case consists of two lines. First line of each test case contains a integer N and the second line contains N space separated array elements.  Output:  For each test case, print the minimum product of two numbers in new line.  Input:  2  4  2 7 3 4  4  5 3 6 4  Output:  6  12  ANSWER  #include<stdio.h>  int main()  {  int test;  scanf("%d",&test);  while(test--)  {  int n , i;  scanf("%d",&n);  long int a[100005];  for(i=0;i<n;i++)  {  scanf("%ld",&a[i]);  }  //selection sort  int min , j , temp;  for(i=0;i<2;i++)  {  min = i;  for(j=i+1;j<n;j++)  {  if(a[min]>a[j])  {  min = j;  }  }  if(i != min)  {  temp = a[min];  a[min] = a[i];  a[i] = temp;  }  }  printf("%ld\n",a[0]\*a[1]);  }  return 0;  } | | 06 |
| (OR) | | | |
| B4b)i) | * 1. Find the order of growth of the given function f(n).   f(n)=[n(n log n + n2+3)+log 0.5n]/n2 +3.14.   * 1. Determine the asymptotic relationship between the two functions given below. * F(n)=log n2 g(n)=log n+5 * F(n)=log log n g(n)=log n * F(n)=n log n+n g(n)=log n * F(n)= 2n g(n)=3n   1. Find the Time complexity of the following functions.  |  |  | | --- | --- | | Function 1:  for(i=0;i<n;i++)  for(j=0;j<i\*i;j++)  for(k=0;k<j;k++) | Function 2:  for(i=1;i<n;i\*=2)  for(j=0;j<n;j++)  for(k=0;k<n;k+=2) |   ANSWER  a)  New Doc 2018-07-16_4  b)i) Theta Θ  ii) Oh O  iii) Omega Ώ  iv) Oh O  c)  O(N5)  O(N2(LOG N)) | | 06 |
| B4b)ii) | Given a NxN matrix. In the given matrix, you have to print the boundary elements of the matrix.    Input: First line consists of  the input contains an integer T denoting the number of  test cases. Then T test cases follow. First line of every test case consists of an integer N, denoting size of the Matrix. Second line of every test case consists of N x N spaced integers denoting elements of Matrix elements.  Output: For each test case output a single line output, print the boundary elements of the matrix.  Constraints: 1<=T<=100 1<=N<=100  Example: Input: 2//number of test cases 4 // size of matrix of test case 1 1 2 3 4  5 6 7 8  1 2 3 4  5 6 7 8 3 // size of matrix of test case 2 45 48 54  21 89 87  70 78 15 Output:  //Boundary Elements 1 2 3 4 5 8 1 4 5 6 7 8 45 48 54 21 87 70 78 15  ANSWER  #include<stdio.h>  void printBoundary(int a[100][100], int m, int n)  {  for (int i = 0; i < m; i++) {  for (int j = 0; j < n; j++) {  if(i==0 || j==0 || i == m-1 || j == n-1)  printf("%d\t",a[i][j]);  else  printf("\t");  }  printf("\n");  }  }  int main()  {  int a[4][100] = { { 1, 1, 1, 1 }, { 2, 2, 2, 2 }, { 3, 3, 3, 3 }, { 4, 4, 4, 4 } };  printBoundary(a, 4, 4);  return 0;  } | | 05 |
| B4b)iii) | Suppose any subset S of elements {1, ... , n}is represented in a “bit array” A of N elements in the usual way ie., bit array means contains only 1’s and 0’s. Specifically in the array if A[i] =1 means that i∈S and if A[i] =0 means i ∉ S. Design an algorithm to compute the size of S (the number of elements in S) from its bit array representation.  ANSWER   |  |  | | --- | --- | | #include <stdio.h>   |  | | --- | | void findCounts(int \*arr, int n)  {  int count=0;  for(int i=0;i<n;i++)  {  if(a[i]==1)  count++;  }}  printf(“Length of S is %d\n”,count);  }  int main()  {      int arr[] = {1,0,1,0,1,1,0,0,1,1,0,0};      findCounts(arr, sizeof(arr)/ sizeof(arr[0]));      return 0;  } | | | | 05 |
| B5a)i) | Andi and Bob were friends since childhood days. But, as they grew up Bob started behaving weird and this used to irritate Andi. Once, while Andi took a break after typing a large program Bob came from nowhere and swapped some alphabet keys on Andi's keyboard. Andi got very angry on seeing this and decided to end their friendship once forever. As we all know Bob is very good at heart and never does anything wrong intentionally. He decided to type the remaining program with the same keyboard configuration. Given the original fragment of the code that Bob needs to type, You need to tell Bob the code that he should type to get the original code as output. Help him saving his friendship with Andi. Write an algorithm for the given scenario.  INPUT :  First line of the input contains a single integer N denoting the number of swaps done by Bob. Next N lines contain a pair of characters A, B denoting the characters which are swapped by Bob (Note that Bob performs these swaps in the given order). From the very next line of input the remaining fragment of the code starts and ends with an end of file character.  OUTPUT: Print the fragment of the program that Bob types.  Note: code fragment contains uppercase and lowercase english alphabets only and space.  A, B belongs to Alphabet set (both upper case and lower case letters are included).  SAMPLE INPUT  1  W H  WelloHorld  SAMPLE OUTPUT  HelloWorld  Explanation  For the given test case:  Letter W is swapped with the letter H. So, to type WelloHorld Bob must type HelloWorld on Andi's keyboard.  ANSWER  #include <stdio.h>  #include <ctype.h>  #include <string.h>  int main(){  int read;  char line[1000006];  char H[300];  int n, i, p1, p2, j;  char a, b;  scanf("%d", &n);  for(i = 0; i < 26; ++i){  H[i] = (char)(i + 65);  }  for(j = 0; j < n; ++j){  scanf(" %c %c", &a, &b);  a = toupper(a);  b = toupper(b);  for(i = 0; i < 26; ++i)  if(H[i] == a){  p1 = i;  break;  }  for(i = 0; i < 26; ++i)  if(H[i] == b){  p2 = i;  break;  }  a = H[p1];  H[p1] = H[p2];  H[p2] = a;  }  while ((read = scanf("%s", line)) != -1) {  read = strlen(line);  for(i = 0; i < read; ++i){  a = line[i];  if(a > 'Z'){  a = toupper(a);  b = H[(int)a - 65];  b = tolower(b);  }else{  a = toupper(a);  b = H[(int)a - 65];  b = toupper(b);  }  line[i] = b;  }  printf("%s",line);  }  return 0;  } | | 08 |
| B5a)ii) | Given a string which comprises of lower case alphabets (a-z), upper case alphabets (A-Z), numbers, (0-9) and special characters like !,-.; etc. You are supposed to write an algorithm to find out which character occur the maximum number of times and the number of its occurrence, in the given string. If two characters occur equal number of times, you have to output the character with the lower ASCII value.  For example, if your string was: aaaaAAAA, your output would be: A 4, because A has lower ASCII value than a.  Input format: The input will contain a string.  Output format: You've to output two things which will be separated by a space: i) The character which occurs the maximum number of times.  ii) The number of its occurrence.  Constraints: The maximum length of the string can be 1000.  SAMPLE INPUT  Pulkit is a dog!!!!!!!!!!!!  SAMPLE OUTPUT  ! 12  ANSWER  #include <stdio.h>  #include<string.h>  #include<stdlib.h>  int main()  {  char str[1000];  int max=0, maxindex;  int \*count=(int\*)calloc(256,sizeof(int));  scanf("%[^\n]%\*c",str);    for(int i=0;\*(str+i);i++)  {  count[\*(str+i)]++;  }  for(int i=0;i<256;i++)  {  if(count[i]>max)  {  max=count[i];  maxindex=i;  }  }  printf("%c %d",(char)maxindex,count[maxindex]);  free(count);  return 0;  } | | 08 |
| (OR) | | | |
| B5b)i) | Roy wanted to increase his typing speed for programming contests. His friend suggested that he type the sentence "The quick brown fox jumps over the lazy dog" repeatedly. This sentence is known as a pangram because it contains every letter of the alphabet. After typing the sentence several times, Roy became bored with it so he started to look for other pangrams. Given a sentence, write an algorithm to determine whether it is a pangram or not.  **Input Format:** Input consists of a string s.  **Constraints**:  Length of s can be at most  103 (1 < |s| <= 103) and it may contain spaces, lower case and upper case letters. Lower-case and upper-case instances of a letter are considered the same.  Output Format: Output a line containing pangram if s is a pangram, otherwise output not pangram.  Sample Input & Output:   |  |  | | --- | --- | | Input #1  We promptly judged antique ivory  buckles for the next prize  Output #1  pangram | Input #2  We promptly judged antique ivory buckles for the prize  Output #2  Not pangram |   Explanation  In the first test case, the answer is pangram because the sentence contains all the letters of the English alphabet. The second sentence does not.  ANSWER  #include<stdio.h>  char st[100000];  int i,ind[1000];  int main()  {  while(gets(st))  {  for(i='A';i<='Z';i++)  ind[i]=0;  for(i=0;st[i];i++)  {  if(st[i]>='a' && st[i]<='z')  st[i]-=32;  ind[st[i]]++;  }  for(i='A';i<='Z';i++)  if(ind[i]==0)  break;  if(i=='Z'+1)  printf("pangram\n");  else  printf("not pangram\n");  }  return 0;  } | | 08 |
| B5b)ii) | Write the recurrence equation for the Tower of Hanoi algorithm. Solve it using substitution method.  ANSWER  WhatsApp Image 2018-07-16 at 12  WhatsApp Image 2018-07-16 at 12 | | 08 |
| Name & Signature of the Faculty In-Charge | | Signature of HoD/Director | |