KARPAGAM COLLEGE OF ENGINEERING, COIMBATORE-641032

B.E (CSE,ECE,EEE,EIE,ETE,MECH)/B.TECH IT SEMESTER : IV

15PE05 / 15FE05 – DESIGN AND ANALYSIS OF ALGORITHMS

CONTINUOUS INTERNAL ASSESSMENT: II Answer Key

DURATION : 3 HOURS DATE : 26.02.2018

MAXIMUM : 100 MARKS SESSION : FN

Answer All Questions PART- A (10 x 2 = 20 Marks)

A1 Given a binary search tree (BST) 'T' with integer key values, and let x and y be two integers with x ≤y. The goal is to design an algorithm to print all the key values v stored in T, that satisfy the condition x ≤ v ≤ y. Let h be the height of T, and k the number of values printed. What is the time complexity of the given problem?

Ans:

```
void Print(structnode *root, int x, int y)
{
   if( NULL == root )
     return;
   Print(root->left, x, y);
   if(root->data >= x &&  root->data <= y )
    printf("%d ", root->data );
   Print(root->right, x, y);
}
```

A2 Given an array of size 'n', suppose algorithm 'A' takes $8n^2$ time to process the array and algorithm 'B' takes 64nlogn to do the same job. For what values of 'n' does program 'A' outperforms algorithm 'B'?

Ans: $8n^2<=64nlogn$

$n^2<=8nlogn$ ➔ n<=8logn➔ On solving n-8logn = 0 you get

n = 43

A3 In a general case, binary search is more efficient than linear search. Give two example situations where linear search is more efficient than binary search.

Ans: (a) Array[] = {12,23,45,56,98} Search key: 12

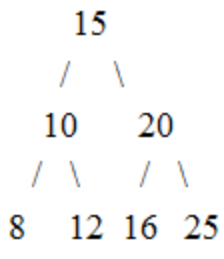
In linear search the key will be found in the first comparison but in binary search the key will be found in third comparison

(b) Array[] = {102,223,321,287,435,561,980,1020, 1128} Search key: 223

In linear search the key will be found in the second comparison but in binary search the key will be found in third comparison.

A4 What is the output of print (root, 3) where 'root' represents the root of the given Binary Search Tree?

```
int count = 0;
void print(struct node *root, int k)
{
  if (root != NULL && count <= k)
  {
    print(root->right, k);
    count++;
    if (count == k)
      printf("%d ", root->data);
    print(root->left, k);
  }
}
```



Ans: The function prints the kth largest element in the BST. The output is 16.

A5 The Fibonacci sequence can be abstractly defined as follows:

$$\begin{cases} F_0 = 1 \\ F_1 = 1 \\ F_n = F_{n-2} + F_{n-1} \quad \text{for } n \geq 2 \end{cases}$$

(This yields $1, 1, 2, 3, 5, 8, 13, 21, \ldots$)

In a couple of lines using your favorite programming language, write a recursive program to compute Fn given 'n', using the definition above. How many function calls will your recursive program perform to compute F10, F20 and F30?

Ans:
```
int fib(int n)
{
  if (n==0)
    return 0;
  Else if(n==1)
    Return 1;
  return fib(n-1) + fib(n-2);
}
```
F(10)=177
F(20)=21891
F(30)= 2692537

A6 Find the minimum number of comparisons required to merge five files A (with 10 records), B (with 20 records), C (with 15 records), D (with 5 records) and E (with 25 records). Arrange the sequence of merging operations using greedy technique and choose the one from the options given below.

A: 165 B: 90 C: 85 D: 65

Ans:

Arrange files in increasing order of records

D A C B E

5 10 15 20 25

75

30 45

15 15 (C) 20 (B) 25 (E)

5 (D) 10 (A)

No of movements= $15 + 30 + 45 + 75 = 165$

A7 A 3-ary max-heap is like a binary max-heap, but instead of 2 children, nodes have 3 children. A 3-ary heap can be represented by an array as follows: The root is stored in the first location, a[0], nodes in the next level, from left to right, is stored from a[1] to a[3] and so on. Which one of the following is a valid sequence of elements in an array representing 3-ary max heap?

(A)1, 3, 5, 6, 8, 9 (B)9, 6, 3, 1, 8, 5(C)9, 3, 6, 8, 5, 1(D)9, 5, 6, 8, 3, 1

Ans: (D)9, 5, 6, 8, 3, 1

A8 Given two arrays, A and B, of equal size n, the task is to write an algorithm to find the minimum value of A[0]*B[0]+A[1]*B[1] +…+ A[n-1]*B[n-1], where shuffling of

elements of arrays A and B is allowed.
Input:
2 //No of test cases
3 //TC1: number of elements
3 1 1 //Array A
6 5 4 // Array B
5//TC2: number of elements
6 1 9 5 4//Array A
3 4 8 2 4// Array B
Output:
23
80

Ans:
```
intminValue(intA[], intB[], intn)
{
sort(A, A + n);
sort(B, B + n);
intresult = 0;
for(inti = 0; i< n; i++)
result += (A[i] * B[n - i - 1]);
returnresult;
}
```

A9 Write an efficient divide-and-conquer algorithm for exponentiation problem for computing a^n where a > 0 and n is a positive integer. Set up and solve a recurrence relation for the number of times the basic operation is executed.

Ans:
```
double power(double x,int y){
if(y ==0)return1;
double d = power(x, y/2);
if(y%2==0)return d*d;
elsereturn x*d*d;}
```
T(n)=T(n/2)+2 T(0)=0
Time Complexity: O(log n)

A10 Find the asymptotic (big-O notation) running time of the following algorithms by writing its recurrence equation using Master theorem.

1. An algorithm solves problems by dividing a problem of size n into 3 sub-problems of one-fourth the size and recursively solves the smaller sub-problems. It takes constant time to combine the solutions of the sub-problems
2. An algorithm solves problems by dividing a problem of size n into 2n sub-problems of half the size and recursively solves the smaller sub-problems. It takes linear time to combine the solutions of the sub-problems.

Ans: 1) $T(n)=3T(n/4)+C = O(n^{\log_4 3})$
2) T(n)=2nT(n/2)+O(n) This cannot be solved by master theoren as 'a' is not a constant.

Answer All Questions PART- B (5 x 16 = 80 Marks)

B1a)i) Given two individually sorted arrays (A and B) containing n numbers with all unique values (i.e. no two numbers in A are equal, no two numbers in B are equal and no number in A equals any number in B). Design a divide and conquer based algorithm to find the median of Union (A,B). 10

- A[] = {1, 12, 15, 26, 38}
- B[] = {2, 13, 17, 30, 45}

a) Write the procedure for the given problem.
b) Show the sequence of median finding process.
c) Set and solve up the recurrence relation for the same and state its asymptotic time

complexity in Big O.

Ans: a) Pseudo Code:

```
intgetMedian(intar1[], intar2[], intn)
{
if(n <= 0)
return-1;
if(n == 1)
return(ar1[0] + ar2[0])/2;
if(n == 2)
return(max(ar1[0], ar2[0]) + min(ar1[1], ar2[1])) / 2;
intm1 = median(ar1, n);
intm2 = median(ar2, n);
if(m1 == m2)
returnm1;
if(m1 < m2)
{
if(n % 2 == 0)
returngetMedian(ar1 + n/2 - 1, ar2, n - n/2 +1);
returngetMedian(ar1 + n/2, ar2, n - n/2);
}
if(n % 2 == 0)
returngetMedian(ar2 + n/2 - 1, ar1, n - n/2 + 1);
returngetMedian(ar2 + n/2, ar1, n - n/2);
}
intmedian(intarr[], intn)
{
if(n%2 == 0)
return(arr[n/2] + arr[n/2-1])/2;
else
returnarr[n/2];
}
```

b) Median Finding Process:

ar1[] = {1, 12, 15, 26, 38}
ar2[] = {2, 13, 17, 30, 45}

For above two arrays m1 = 15 and m2 = 17

For the above ar1[] and ar2[], m1 is smaller than m2. So median is present in one of the following two subarrays.

[15, 26, 38] and [2, 13, 17]

Repeat the process for above two subarrays:

m1 = 26 m2 = 13.

m1 is greater than m2. So the subarrays become

[15, 26] and [13, 17]
Now size is 2, so median = (max(ar1[0], ar2[0]) + min(ar1[1], ar2[1]))/2
= (max(15, 13) + min(26, 17))/2
= (15 + 17)/2

= 16

c) Time Complexity: T(n) = T(n/2) + 1 = O(log n)

B1a)ii) Consider the following two functions: 06

```
int Fun1 (int n)
{
if (n == 1) return 0;    else return (1 + Fun1 (n/2));
}

int Fun2 (int n)
{
int L = 0;
while (n > 1) { n = n/2;  L++; }
return L;
}
```

(a) Trace the two functions for n = 1, 2, 4, 8. To do this, build a table containing (n), the returned value from each function for that (n), and the total number of integer arithmetic operations T(n) needed to achieve the result.

n	Fun1(n)	T1(n)	Fun2(n)	T2(n)

(c) What is the complexity (Big-O) of each function?

(d) What are the objectives of these two functions and in what way do they differ?

Ans:

n	Fun1(n)	T1(n)	Fun2(n)	T2(n)
1	0	0	0	0
2	1	1	1	1
4	2	2	2	2
8	3	3	3	3

c) Time Complexity:

Fun1() : O(n)

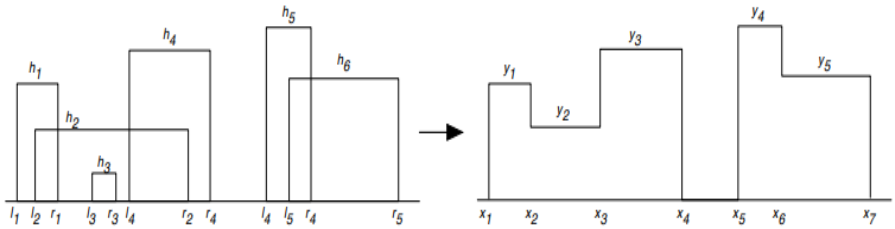
Fun2: O(n)

d) Objectives: Both the functions generate the logarithmic value of the given integer 'n'.

Difference:Fun1() is a recursive function, where as, Fun2() is an iterative one.

(OR)

B1b)i) Draw a two-dimensional skyline like the following: 06



Given a list of the building x-coordinates and their heights: (l1, h1, r1),(l2, h2, r2), . . . ,(ln, hn, rn). This list will be sorted in increasing order of left-most x-coordinate. Sketch

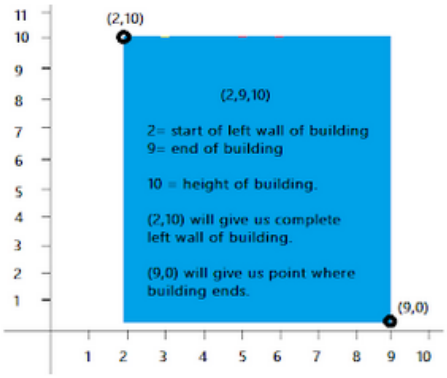
an O(n log n) algorithm to produce the skyline line to draw in the format: x1, y1, x2, y2, x3 . . . meaning that at x1 we draw a building at height y1 until x2 at which point we draw a line up or down to high y2 and then continue horizontally until x3 and so on. Note that for this problem, you need to sketch only the main ideas.

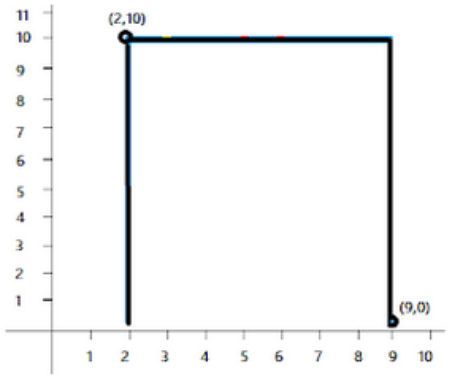
Ans:

STEP 1:

In Step 1, we will break the problem from multiple buildings to 1 building.

Lets see what is required to draw skyline if we have only 1 building.



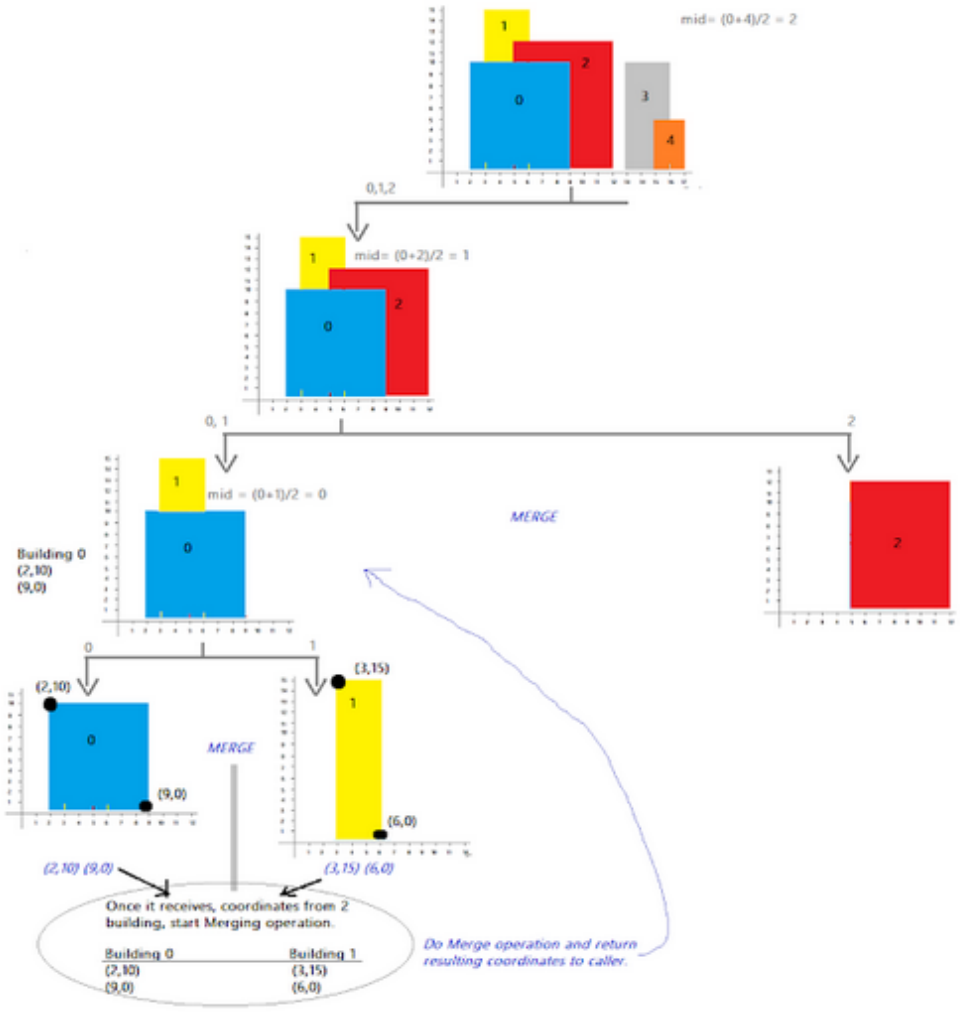


From above picture, we can see, if we have diagonal coordinates of a building then we can easily draw skyline of that building.

So what we will do in this step is break down multiple buildings recursively until only 1 building is left and identify diagonal coordinates of each building.

Breaking of building is done in same way as we break the array in merge sort.

Once we return back, we will have diagonal coordiantes from 2 buildings at that point we need to start merging. We will see that later. lets calculate diagonal coordinate and return.



STEP 2:

In step 2, start merging coordinates of building by applying below rule.

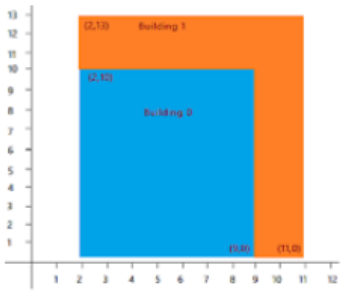
1. From 2 coordinates (x0, y0) of Building 0 and (x1, y1) of Building 1, compare x0 and x1, whichever is smaller among them that is whichever building appeared first among them will be part of output.
   Say x0 is smaller, So x0 from Building 0 will be part of output. What about the height coordinate?

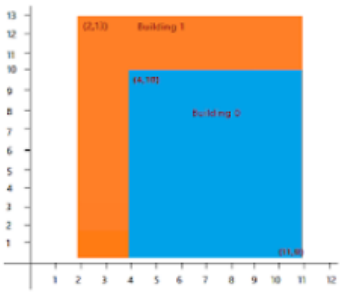
   For height coordinate, it can't simply pick y0 because y0 can be overlapped by some taller building, so for height coordinate it has to compare y0 of current Building 0 with previously encountered height from Building 1 and whichever is greater that will be height coordinate.

   Say height from Building 0 is greater so y0 will be height coordinate
   So output coordinate will be (x0, y0).

Corner Case:

There are some corner cases like if 2 building start from same coordinate, so what to do in that case,
Consider like example below,





In this case, we compare (2,10) with (2,13) and x coordinate of both is same that is 2.
It means both building start from same coordinate. So x coordinate is 2 in output. (2, ?)

For height we need to compare highest among current two building in comparison that is compare height 10 and 13,
Height 13 is greater *so resulting coordinate is (2, 13).* And increment pointers from both Building.

(Note: we are not comparing it with previous encountered building height because which building previous height we will compare Building 0 or Building 1 as we took x coordinate which is same for both Buildings.)

Pseudo Code:

```
SkyLine *findSkyline(Building arr[], int l, int h)
{
   if (l == h)
   {
SkyLine *res = new SkyLine(2);
       res->append(new Strip(arr[l].left, arr[l].ht));
       res->append(new Strip(arr[l].right, 0));
       return res;
   }
    int mid = (l + h)/2;
SkyLine *sl = findSkyline(arr, l, mid);
SkyLine *sr = findSkyline(arr, mid+1, h);
SkyLine *res = sl->Merge(sr);
   delete sl;
   delete sr;
 return res;
}
SkyLine *SkyLine::Merge(SkyLine *other)
{
SkyLine *res = new SkyLine(this->n + other->n);
    int h1 = 0, h2 = 0;
   int i = 0, j = 0;
   while (i< this->n && j < other->n)
   {
      if (this->arr[i].left < other->arr[j].left)
       {
          int x1 = this->arr[i].left;
          h1 = this->arr[i].ht;
          int maxh = max(h1, h2);
           res->append(new Strip(x1, maxh));
i++;
```

```
        }
        else
        {
            int x2 = other->arr[j].left;
            h2 = other->arr[j].ht;
            int maxh = max(h1, h2);
            res->append(new Strip(x2, maxh));
j++;
        }
    }
     while (i< this->n)
    {
        res->append(&arr[i]);
i++;
    }
    while (j < other->n)
    {
        res->append(&other->arr[j]);
j++;
    }
    return res;
}
```

B1b)ii) The product of two nXn matrices, A and B is a third nXn matrix Z, where 10
$Zij=\sum A_{ik}B_{kj}$ where k varies from 1 to n

a) A programmer directly implements this formula when writing a function to multiply two matrices. Write the brute force algorithm and state its asymptotic time complexity.

b) An alternative strategy to compute Z is to divide A and B into four n/2 X n/2 matrices. Computing Z then involves eight n/2 and n/2 matrix multiplications followed by a series of matrix additions. This approach is then applied recursively.

i) Identify the algorithmic strategy in use.

ii) Show the recurrence relation for the algorithm designed tocompute AxB.

iii) Solve the recurrence to ascertain the running time.

c) Compare the running time of Brute force and divide and conquer strategy.

Ans: a) Brute Force Algorithm:

```
voidmultiply(intA[][N], intB[][N], intC[][N])
{
for(inti = 0; i< N; i++)
{
for(intj = 0; j < N; j++)
{
C[i][j] = 0;
for(intk = 0; k < N; k++)
{
C[i][j] += A[i][k]*B[k][j];
}
}
}
}
```

b) (i) The algorithmic strategy used here is **Divide and Conquer Technique.**

Pseudo Code:

```
longprod(longu,longv)
{
 longx,y,w,z;
 longn,m,p,q,r;
  n=noOfDigit(u,v);
  if(u==0 || v==0)
 {
  return0;
 }
  elseif(n<=2)
 {
  return(u*v);
 }
 else
 {
  m=floor(n/2);
   w=u/pow(10,m);
  x=u%(int)pow(10,m);
  y=v/pow(10,m);
  z=v%(int)pow(10,m);
   p=prod(w,y);
  q=prod(x,z);
  r=prod(w+x,y+z);
 returnp * pow(10,2*m) + (r-p-q) * pow(10,m) + q;
 }
}
longnoOfDigit(longm,longn)
{
 longmax;
 intb=0;
 if(m>=n)
  max=m;
 else
  max=n;
 while(max>0)
 {
  max=max/10;
  b++;
 }
 returnb;
}
```

(ii) $T(N) = 7T(N/2) + O(N^2)$

(iii)

T(n) = $7T(n/2) + an^2$

According to Master`s Theorem, a=7 , b=2, k=2 and p=0

so, $a > b^k$ as $7 > 2^2$

Therefore, $T(n) = O(n^{\log_7 2})$

$= O(n^{\log_7 2})$

$= O(n^{2.80735})$

$= O(n^{2.81})$

$T(N) = O(N^{Log7})$ which is approximately $O(N^{2.8074})$

B2a)i) Given an n-element array of integers, A, and an integer, m, determine the maximum value of the sum of any of its subarrays modulo 'm'. This means that you must find the sum of each subarray modulo m, then print the maximum result of this modulo operation for any of the n(n+1)/2 possible sub-arrays. 08

Input Format:

The first line contains an integer, q, denoting the number of queries to perform. Each query is described over two lines:

- The first line contains two space-separated integers describing the respective n (the array length) and m (the right operand for modulo operations) values for the query.
- The second line contains n space-separated integers describing the respective elements of array A=a0, a1, a2,…..an-1 for that query.

Output Format:

For each query, print the maximum value of sub-array sum % m on a new line.

Sample Input

```
1
5 7
3 3 9 9 5
```

Sample Output

```
6
```

Ans:

```
int maxSubarray(int arr[], int n, int m)
{
int x, prefix = 0, maxim = 0;
set<int> S;
S.insert(0);
for (int i = 0; i< n; i++)
{
prefix = (prefix + arr[i])%m;
maxim = max(maxim, prefix);
 auto it = S.lower_bound(prefix+1);
   if (it != S.end())
  maxim = max(maxim, prefix - (*it) + m );
S.insert(prefix);
    }
  return maxim;
}
```

B2a)ii) Given a set of 'n' jobs where each job 'i' has a deadline and profit associated to it. Each job takes 1 unit of time to complete and only one job can be scheduled at a time. We earn the profit if and only if the job is completed by its deadline. The task is to write an algorithm to find the maximum profit and the number of jobs done. 08

Input:

The first line of input contains an integer T denoting the number of test cases. Each test case consist of an integer N denoting the number of jobs and the next line consist of Job id, Deadline and the Profit associated to that Job.

Output:

Output the number of jobs done and the maximum profit.

Example:

Input:

```
2
4
```

1 4 20 2 1 10 3 1 40 4 1 30
5
1 2 100 2 1 19 3 2 27 4 1 25 5 1 15
Output:
2 60
2 127

Ans:

```
voidprintJobScheduling(Job arr[], intn)
{
sort(arr, arr+n, comparison);
intresult[n];
boolslot[n];
for(inti=0; i<n; i++)
slot[i] = false;
for(inti=0; i<n; i++)
{
for(intj=min(n, arr[i].dead)-1; j>=0; j--)
{
if(slot[j]==false)
{
result[j] = i;
slot[j] = true;
break;
}
}
}
for(inti=0; i<n; i++)
if(slot[i])
print(arr[result[i]])
}
```

(OR)

B2b)i) You are the head of a firm and you have to assign jobs to people. You have N persons working under you and you have N jobs that are to be done by these persons. Each person has to do exactly one job and each job has to be done by exactly one person. Each person has his own capability (in terms of time taken) to do any particular job. Your task is to assign the jobs to the persons in such a way that the total time taken is minimum. A job can be assigned to only one person and a person can do only one job. 08

Input:

The first line of input contains an integer ‘T’ denoting the number of test cases. Then ‘T’ test cases follow. The first line of each test case contains an integer N, where N is the number of jobs and the number of persons under you. The next line contains N^2 positive integers. The first N of these integers denote the time taken by the first person to do the N jobs, the next N integers denote the time taken by the second person to do N jobs and so on till the N^{th} person.

Output:

For each test case in a new line, print the time taken in the best possible assignment that you can do.

Example:

Input:

2
2
3 5 10 1

3
2 1 2 9 8 1 1 1 1
Output:
4
3
Explanation:
The first person takes times 3 and 5 for jobs 1 and 2 respectively. The second person takes times 10 and 1 for jobs 1 and 2 respectively. We can see that the optimal assignment will be to give job 1 to person 1 and job 2 to person 2 for a total for 3 + 1 = 4.

Ans:

```
for i = 1 to n do
Set k = min(dmax, DEADLINE(i))
while k >= 1 do
if timeslot[k] is EMPTY then
timeslot[k] = job(i)
break
endif
Set k = k - 1
endwhile
endfor
void printschedue(struct Job
arr[], int n)
for (int j=min(n, arr[i].dead)-1; j>=0; j--)
{
if (!slot[j])
{{
sort(arr,n);
result[j] = i;
slot[j] = 1;
break;
}
}
}
slot[i] = 0;
for (int i=0; i<n; i++)
if (slot[i])
for (i=0; i<n; i++)
{
printf("%c", arr[result[i]].id);
}
```

B2b)ii) Mr. John has one long loaf of bread of length 1. He wants to cut it into as many little loaves as he can. But he wants to adhere to the following rule: At any moment, the length of the longest loaf which he possesses may not be larger than the length of shortest one, times some constant factor. Every time, he is only allowed to cut exactly one loaf into two shorter ones. 08

Input:
One floating-point number, $1 \leq k \leq 1.999$, meaning the stated constant factor. The number will have at most 3 digits after the decimal point.
Output:
First, you should output one number 'n', the maximal achievable number of loaves for the given value of the constant factor. Then, you should output any proof that this

number of loaves is in fact achievable: n-1 descriptions of cutting, using the following notation. At each step, you print two numbers: first, the index of the loaf that you want to cut into two parts; second, the length of the newly created loaf (cut off from the original one). It is assumed that the starting loaf has 0th index. Each newly created loaf will be given the lowest possible free integer index (so, at the ith step this will be i). Each time, the size of size of the original loaf will be decreased by the size of the newly created loaf.
Example:
Input:
1.5
Output:
4
0 0.4
0 0.3
1 0.2

Ans:

```
temp = 0.00000001;
scale-=temp;
logscale = log(scale);
log2 = log(2);
ans = (int)(logscale/(log2 - logscale))+1;
valc = 2;
vals[0] = 1;
vals[1] = 1;
sum = 2;
for (i=2; i<2*ans; i+=2)
{
next = 2/scale*vals[i-1];
vals[i] = next;
vals[i+1] = next;
sum = sum + 2*next;
valc = valc + 2;
}
for (i=0; i<valc; i++)
{
vals[i] = vals[i]/sum;
}
bot = 0;
top = valc;
while (top-bot>1)
{
vals[top++] = vals[bot] + vals[bot+1];
bot+=2;
}
numbering[top-1]=0;
printf("%d\n",valc);
number = 1;
for (i=0; i<valc-1; i++)
{
printf("%d %.15f\n",numbering[top-1],vals[bot-2]);
numbering[bot-1]=numbering[top-1];
numbering[bot-2]=number++;
bot-=2;
top--;
}
return 0;
}
```

B3a)i) Let S be a set of 'n' line segments in the plane. Give an algorithm to compute the convex hull of S. Analyze the time complexity of designed algorithm and argue it is correct. 04

Ans: Pseudo code:

```
void compute(struct P p[20],int n)
{
int i,j,k;
for(i=0;i<n;i++)
 {
  for(j=0;j<n;j++)
```

```
   {
    if(i==j)
    {
     continue;
    }
struct P ptI=p[i];
struct P ptJ=p[j];
    int right=1;
    for(k=0;k<n;k++)
     {
      if(k==i||k==j)
      {
       continue;
      }
      int d=whichside(ptI,ptJ,p[k]);
      if(d<0)
       {
       right=0;
       break;
       }
     }
   if(right)
printf("(%d %d),(%d %d)\n",ptI.x,ptI.y,ptJ.x,ptJ.y);
   }
 }
}
```

(OR)

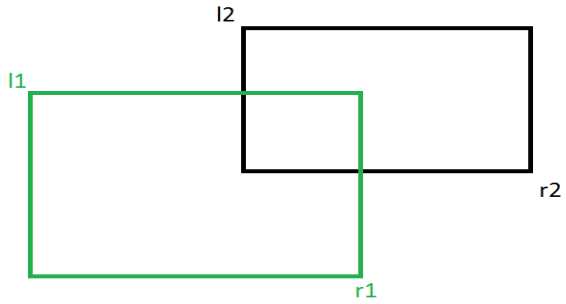
Algorithm SLOWCONVEXHULL(P)
Input. A set P of points in the plane.
Output. A list $\mathcal{L}$ containing the vertices of $CH(P)$ in clockwise order.

1. $E \leftarrow \emptyset$.
2. **for** all ordered pairs $(p,q) \in P \times P$ with p not equal to q
3. **do** $valid \leftarrow$ **true**
4. **for** all points $r \in P$ not equal to p or q
5. **do if** r lies left of the directed line from p to q
6. **then** $valid \leftarrow$ **false**
7. **if** $valid$ **then** Add the directed edge $\vec{pq}$ to E
8. From the set E of edges construct a list L of vertices of $CH(P)$, sorted in clockwise order.

Time Complexity: $O(n^3)$

B3a)ii) Given two rectangles, find if the given two rectangles overlap or not. Note that a rectangle can be represented by two coordinates, top left and bottom right. So mainly we are given following four coordinates. 04

l_1: Top Left coordinate of first rectangle.
r_1: Bottom Right coordinate of first rectangle.
l_2: Top Left coordinate of second rectangle.
r_2: Bottom Right coordinate of second rectangle.
We need to write a function bool doOverlap(l_1, r_1, l_2, r_2) that returns true if the two given rectangles overlap.



Ans:
```
structPoint
{
intx, y;
};
booldoOverlap(Point l1, Point r1, Point l2, Point r2)
{
if(l1.x > r2.x || l2.x > r1.x)
returnfalse;
if(l1.y < r2.y || l2.y < r1.y)
returnfalse;
returntrue;
}
```

B3a)iii) A Fox is looking to collect some coins placed in a line. The fox is very greedy and follows an unusual way of collecting coins. The Fox has decided that it will collect coins in an increasing order only since she always craves for more money and all the coins must be contiguous. It wants to maximize the amount of money it collects. Each coin has its value printed on it. 08

Unfortunately, the Fox is greedy but not intelligent enough to find the solution and asks you for help. Print the maximum amount of money it will take home. You are given the value of each coin in an array in the order in which the coins are placed.

Input:

The first line of input contains an integer T denoting the number of test cases. Each test case contains the number of elements in the array a[] as n and next line contains space separated n elements in the array.

Output:

Print an integer which is the required answer.

Example:

Input:

2
6
2 1 4 7 3 6
5
38 7 8 10 12

Output:

12
38

Ans:
```
long int temp=0,sum=0,psum=0,temp1;
for(int i=0;i<N;i++)
{
        temp1=temp;
        scanf("%ld",&temp);
```

```
        if(temp>temp1)
                sum=sum+temp;
                if(temp<=temp1)
                {
                                if(psum>sum)
                                psum=psum;
                                else
                                psum=sum;
                                sum=0+temp;
                }
        }
if(psum>=sum)
printf("%ld\n",psum);
else
printf("%ld\n",sum);
}
```

(OR)

B3b)i) 1) What is the brute force procedure to find the multiplication of two large integers A and B? 08

2) Design an algorithm for the same using divide and conquer technique.

3) State the recurrences to find the number of multiplications for computing A*B using brute force technique and also Divide and Conquer Technique.

4) Apply the algorithm for multiplying 2345 X 4567.

Ans: 1) Brute Force Procedure:

```
string multiply(string num1, string num2)
{
   int n1 = num1.size();
   int n2 = num2.size();
   if (n1 == 0 || n2 == 0)
     return "0";
   int i_n1 = 0;
   int i_n2 = 0;
   for (int i=n1-1; i>=0; i--)
   {
      int carry = 0;
      int n1 = num1[i] - '0';
      i_n2 = 0;
      for (int j=n2-1; j>=0; j--)
      {
         int n2 = num2[j] - '0';
         int sum = n1*n2 + result[i_n1 + i_n2] + carry;
         carry = sum/10;
         result[i_n1 + i_n2] = sum % 10;
         i_n2++;
      }
      if (carry > 0)
         result[i_n1 + i_n2] += carry;
      i_n1++;
   }
   int i = result.size() - 1;
   while (i>=0 && result[i] == 0)
```

```
i--;
    if (i == -1)
       return "0";
string s = "";
    while (i>= 0)
        s += std::to_string(result[i--]);
    return s;
}
```

Time Complexity: O(m*n)

2) Divide and Conquer Algorithm:

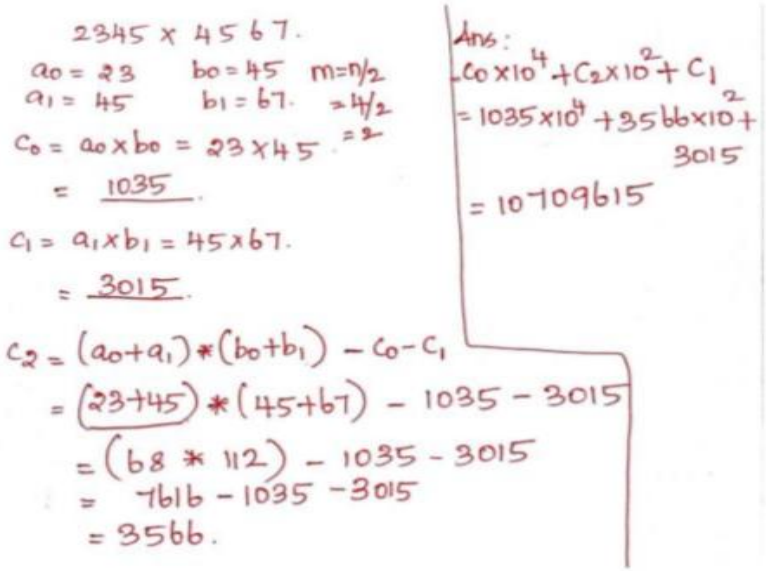
```
longprod(longu,longv)
{
 longx,y,w,z;
 longn,m,p,q,r;
  n=noOfDigit(u,v);
  if(u==0 || v==0)
 {
  return0;
 }
  elseif(n<=2)
 {
  return(u*v);
 }
 else
 {
  m=floor(n/2);
   w=u/pow(10,m);
  x=u%(int)pow(10,m);
  y=v/pow(10,m);
  z=v%(int)pow(10,m);
   p=prod(w,y);
  q=prod(x,z);
  r=prod(w+x,y+z);
   returnp * pow(10,2*m) + (r-p-q) * pow(10,m) + q;
 }
}
 longnoOfDigit(longm,longn)
{
 longmax;
 intb=0;
 if(m>=n)
  max=m;
 else
  max=n;
 while(max>0)
 {
  max=max/10;
  b++;
 }
 returnb;
```

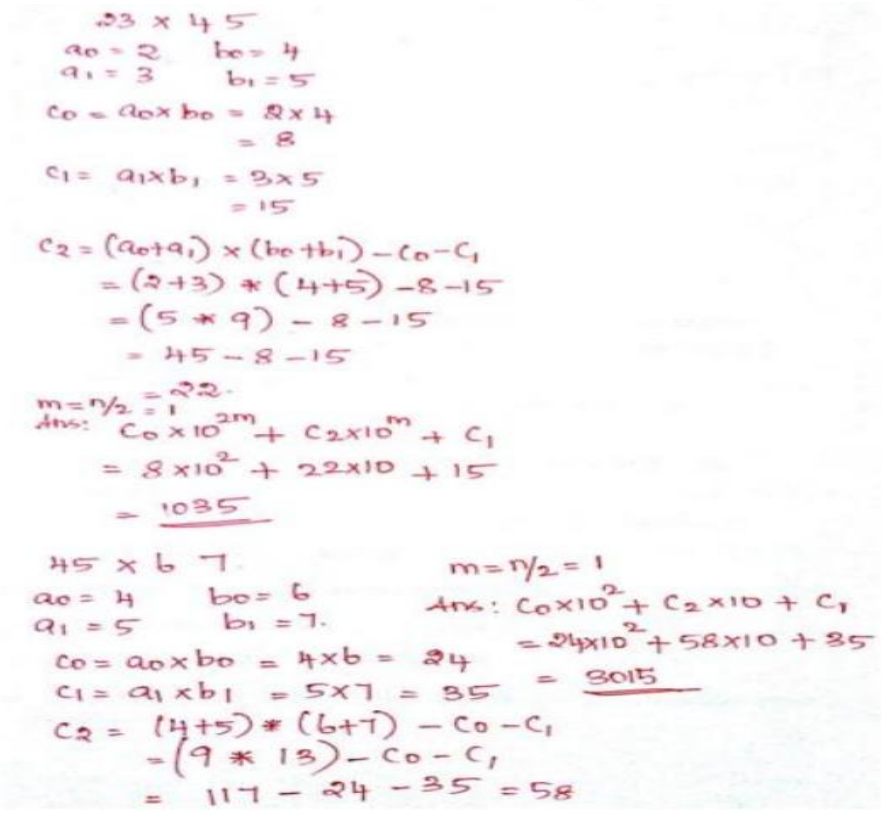
}

3) Recurrence Relation:

$$T(n) = 3T(n/2) + \Theta(n) \implies T(n) = \Theta(n^{\log_2 3}) = O(n^{1.585})$$

4)2345 * 4567:





B3b)ii) Indicate whether the following statements are true or false: 08

a. If e is a minimum-weight edge in a connected weighted graph, it must be among edges

of at least one minimum spanning tree of the graph.
b. If e is a minimum-weight edge in a connected weighted graph, it must be among edges of each minimum spanning tree of the graph.
c. If edge weights of a connected weighted graph are all distinct, the graph must have exactly one minimum spanning tree.
d. If edge weights of a connected weighted graph are not all distinct, the graph must have more than one minimum spanning tree.
e. Will either Kruskal's or Prim's algorithm work correctly on graphs that have negative edge weights?
f. Let T be a tree constructed by Dijkstra's algorithm in the process of solving the single-source shortest-path problem for a weighted connected graph G.
a) True or false: T is a spanning tree of G?
b) True or false: T is a minimum spanning tree of G?

Ans: a) True
b) False
c) True
d) False
e) True
f) a) True
b) False

B4a)i) Given an array of n real numbers, consider the problem of finding the maximum sum in any contiguous sub-vector of the input. For example, in the array {31, −41, 59, 26, −53, 58, 97, −93, −23, 84} the maximum is achieved by summing the third through seventh elements {59 + 26 + (−53) + 58 + 97} which is 187. When all numbers are positive, the entire array is the answer and when all numbers are negative the empty array maximizes the total at 0. 08

1) Give a simple brute force algorithm to find the maximum contiguous sub-vector and the sum. State its time complexity.
2) Design a Θ(n log n) time algorithm using divide-and-conquer strategy.
3) Compare the running time of divide and conquer algorithm with brute force technique.

Ans: <u>1)Pseudo code using Brute Force Technique:</u>

```
int max_sum(int arr[])  {
int N = arr.length, max = Integer.MIN_VALUE;
for (int i = 0; i< N; i++)
{
int sum = 0;
for (int j = i; j < N; j++)
{
sum += arr[j];
if (sum > max)
max = sum;
}
}
return max;
}
```

Time Complexity: $O(n^2)$

<u>2)Pseudo code using divide and conquer tecnique:</u>

```
intmax(inta, intb) { return(a > b)? a : b; }
intmax(inta, intb, intc) { returnmax(max(a, b), c); }
```

```
intmaxCrossingSum(intarr[], intl, intm, inth)
{
   intsum = 0;
   intleft_sum = INT_MIN;
   for(inti = m; i>= l; i--)
   {
      sum = sum + arr[i];
      if(sum >left_sum)
left_sum = sum;
   }
   sum = 0;
   intright_sum = INT_MIN;
   for(inti = m+1; i<= h; i++)
   {
      sum = sum + arr[i];
      if(sum >right_sum)
right_sum = sum;
   }
   returnleft_sum + right_sum;
}
intmaxSubArraySum(intarr[], intl, inth)
{
   if(l == h)
   returnarr[l];
   intm = (l + h)/2;
   returnmax(maxSubArraySum(arr, l, m),
maxSubArraySum(arr, m+1, h),
maxCrossingSum(arr, l, m, h));
}
```

3)

Divide and Conquer	Brute Force Method
$T(n)=2T(n/2)+\Theta(n)$ $= O(n \log n)$	$T(n) = O(n^2)$

B4a)ii) 1) What is the largest number of key comparisons made by binary search in searching for a key in the following array? 04

Array: {3, 14, 27, 31, 39, 42, 55, 70, 74, 81, 85, 93 and 98}

2) List all the keys of this array that will require the largest number of key comparisons when searched for by binary search.

Ans:

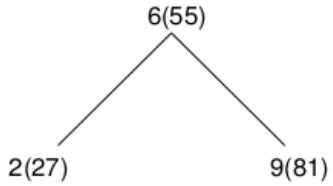
Ans. : The largest number of key comparisons means finding the worst case time complexity of binary search.

$$C_{worst} = \lfloor \log_2 n \rfloor + 1$$
$$= \lfloor \log_2 13 \rfloor + 1$$
$$= 3 + 1$$
$$= 4$$

That means at the most 4 comparisons are needed to search the desired element.

1. a. Take advantage of the formula that gives the immediate answer.

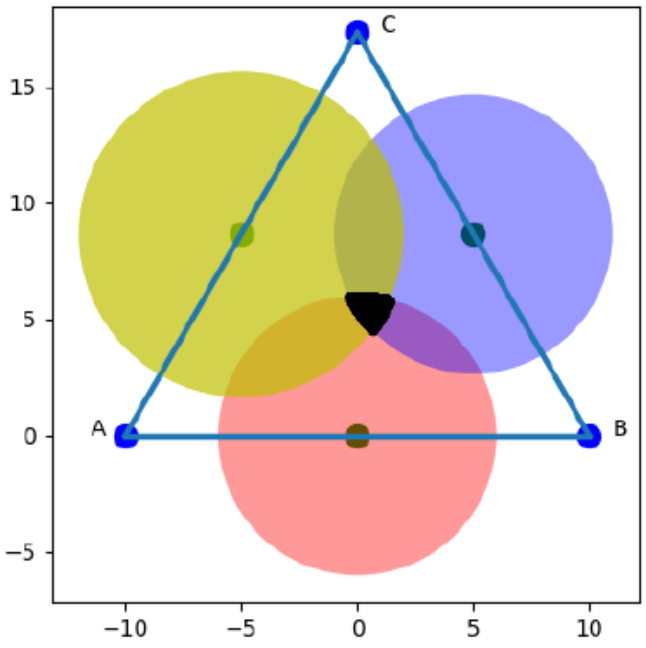
(b)–(d) The most efficient prop for answering such questions is a binary tree that mirrors the algorithm's operations in searching for an arbitrary search key. The first three nodes of such a tree for the instance in question will look as follows:



(The first number inside a node is the index m of the array's element being compared with a search key; the number in the parentheses is the value of the element itself, i.e., $A[m]$.)

B4a)iii) There is an equilateral triangle ABC with side length **2*a**. The coordinates of the triangle's vertices are A = **(-a, 0)**, B = **(a, 0)** and C = **(0, sqrt(3) * a)**. Next, there are three circles with centers at the midpoints of sides AB, BC and AC respectively and radii $\mathbf{r_1}$, $\mathbf{r_2}$ and $\mathbf{r_3}$ respectively. Compute the area of the intersection of the triangle and the three circles. 04

Ans: The black colored area shown in the figure is the intersection between the triangle and the three circles.



(OR)

B4b)i) Given an array of n elements, a majority element is any element appear in more than n/2 times position in A. Assume elements can only be compared for equality, they cannot be ordered or sorted. Design an efficient divide and conquer algorithm to find a majority element in A (or determine that no majority element exists) and analyze its running time. 08

Ans:
```
MajorityElement(A, startIndex, endIndex)
{
if (startIndex == endIndex)
      return A[startIndex];
x = MajorityElement(A, startIndex, (startIndex + endIndex - 1)/2);
y = MajorityElement(A, (startIndex + endIndex - 1)/2 + 1, endIndex);
if (x == null && y == null)
   return null;
else if (x == null && y != null)
   return y;
else if (x != null && y == null)
   return x;
else if (x != y)
   return null;
else return x
}
```

B4b)ii) Write and analyze a divide and conquer based MAXMIN algorithm that uses ((3n)/2)−2 comparisons for any n. 08

Ans: Pseudo Code:
```
MaxMin(i, j, max, min)
{
if (i=j) then max := min := a[i]; //Small(P)
else if (i=j-1) then // Another case of Small(P)
{
if (a[i] < a[j]) then max := a[j]; min := a[i];
else max := a[i]; min := a[j];
}
else
{
mid := ( i + j )/2;
MaxMin( i, mid, max, min );
MaxMin( mid+1, j, max1, min1 );
if (max < max1) then max := max1;
if (min > min1) then min := min1;
}
}
```

Complexity:

If T(n) represents this number, then the resulting recurrence relation is

$$T(n) = \begin{cases} 0 & n=1 \\ 1 & n=2 \\ T(n/2) + T(n/2) + 2 & n>2 \end{cases}$$

When n is a power of two, $n = 2^k$

-for some positive integer k, then

$T(n) = 2T(n/2) + 2$

$= 2(2T(n/4) + 2) + 2$

$= 4T(n/4) + 4 + 2$

.

.

$= 2^{k-1}T(2) + \sum(1 \leq i \leq k-1)\ 2^k$

$= 2^{k-1} + 2^k - 2$

$= 3n/2 - 2 = O(n)$

B5a)i) The characters 'a' to 'h' have the set of frequencies based on the first 8 Fibonacci numbers as follows. 08

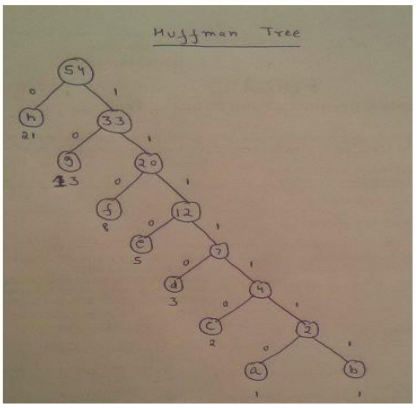
character	a	b	c	d	e	f	g	h
Frequency	1	1	2	3	5	8	13	21

A Huffman code is used to represent the characters.

a) Construct a Huffman Tree using greedy technique to encode the characters.
b) Find the codeword for each character.
c) Give the procedure to decode the characters from the codeword.
d) What is the sequence of characters corresponding to the following code? Code: 110111100111010
   A. fdheg B. ecgdf C. dchfg D. fehdg
e) How many bits are used if the same sequence of characters is represented using ASCII code? How many bits are saved?

Ans:

a)



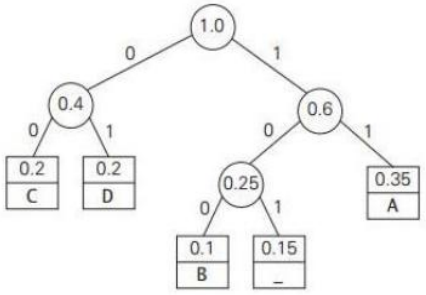
b)

a	1111110
b	1111111
c	111110
d	11110
e	1110
f	110
g	10
h	0

c) **Procedure for encoding and decoding:** A Huffman code is an optimal prefix-

free variable-length encoding scheme that assigns bit strings to symbols based on their frequencies in a given text. This is accomplished by a greedy construction of a binary tree whose leaves represent the alphabet symbols and whose edges are labeled with 0's for the left child and 1's for the right child.
For encoding the given character, traversal should be done from the root by listing allthe values of the edges from the root to the particular character.

Example:



For the character 'B' the code is: 100.
For Decoding the reverse process is applied.
Example: 000111
The decoded character is CDA
d) Given String can be decomposed as

110	11110	0	1110	10
f	d	h	e	g

e) No of bits saved:
Using ASCII code: (1+1+2+ 3+5+ 8+ 13+ 21) * 8 =432
Using Huffman tree: (1*7+1*7+2*6+3*5+5*4+8*3+13*2+21*1) =132
No of bits saved: 432-132 =300

B5a)ii) Given arrival and departure times of all trains that reach a railway station, find the minimum number of platforms required for the railway station so that no train waits. We are given two arrays which represent arrival and departure times of trains that stop. Write an algorithm to find the maximum number of trains that are there on the given railway station at a time. 08

Example:
Input: arr[] = {9:00, 9:40, 9:50, 11:00, 15:00, 18:00}
dep[] = {9:10, 12:00, 11:20, 11:30, 19:00, 20:00}
Output: 3
There are at-most three trains at a time (time between 11:00 to 11:20).

Ans:
```
intfindPlatform(intarr[], intdep[], intn)
{
  sort(arr, arr+n);
  sort(dep, dep+n);
  intplat_needed = 1, result = 1;
  inti = 1, j = 0;
  while(i< n && j < n)
  {
    if(arr[i] < dep[j])
    {
```

```
plat_needed++;
i++;
        if(plat_needed> result)
          result = plat_needed;
      }
      else
      {
plat_needed--;
j++;
      }
   }
   returnresult;
}
```

(OR)

B5b)i) There are two parallel roads, each containing N and M buckets, respectively. Each bucket may contain some balls. The buckets n both roads are kept in such a way that they are sorted according to the number of balls in them. George starts from the end of the road which has the bucket with lower number of balls (i.e. if buckets are sorted in increasing order, then George will start from the left side of road). The George can change the road only at the point of intersection (which means, buckets with the same number of balls on two roads). Now you need to help George to collect the maximum number of balls. 16

Input:

First line of input contains T denoting number of test cases. First line of each test case contains two integers N and M, denoting the number of buckets on road1 and road2 respectively. 2nd line of each test case contains N integers, number of balls in buckets on first road. 3rd line of each test case contains M integers, number of balls in buckets on second road.

Output:

For each test case output a single line containing the maximum possible balls that George can collect.

Example:

Input:

1
5 5
1 4 5 6 8
2 3 4 6 9

Output:

29

Explanation:

The path with maximum sum is (2,3,4)[5,6](9). Integers in [] are the buckets of first road and in () are the buckets of second road. So, max balls George can collect is 29.

Ans:

```
#include <stdio.h>
int main() {
        long long int t,arr1[5000],arr2[5000];
        scanf("%lld",&t);
        while(t--){
           long long int n,m,i;
           long long int j=0,sum1=0,sum2=0,total_sum=0;
        scanf("%lld%lld",&n,&m);
           for( i=0;i<500;i++){
```

```
        arr1[i]=0;
        arr2[i]=0;
    }
    for( i=0;i<n;i++){
scanf("%lld",&arr1[i]);
    }
    for( i=0;i<m;i++){
scanf("%lld",&arr2[i]);
    }
i=0;
    while(i<n&&j<m){
        if(arr1[i]==arr2[j]){
total_sum+=(sum1>sum2?sum1:sum2);
total_sum+=arr2[j];
            sum1=0;
            sum2=0;
i++;
j++;
        }
        else if(arr1[i]<arr2[j]){
            sum1+=arr1[i];
i++;
        }
        else{
            sum2+=arr2[j];
j++;
        }
    }
    if(i<n){
        while(i<n){
            sum1+=arr1[i];
i++;
        }
total_sum+=(sum1>sum2?sum1:sum2);
    }
    else{
        while(j<m){
            sum2+=arr2[j];
j++;
        }
total_sum+=(sum1>sum2?sum1:sum2);
    }
printf("%lld\n",total_sum);
}
return 0;
}
```