Ramaiah Institute of Technology

(Autonomous Institute, Affiliated to VTU)

**Department of Computer Science & Engineering**

**QUESTION BANK FOR IVSEMESTER (Term: Jan-May 2017)**

**Algorithms Laboratory (CSL1542)**

**I.A. Marks : 50 Exam Hours: 03**

**Credits: 0:0:1 Exam Marks: 50**

**Note: Programs need to be written using C / C++/ Python programming language.**

|  |  |
| --- | --- |
|  | a) Ramesh wants to arrange a block of unsorted numbers using an approach where the smallest element is first searched and swapped with element in the first position until the numbers are sorted in order.  Design and implement an algorithm to solve it using random numbers and also display the execution time.  b) Write the best-case, average-case and worst-case for the algorithm.  **Selection Sort**  import random  import time  n=int(input("enter the number of elements"))  aList = [random.randint(0,100) for i in range(n)]  print(aList)  start=time.clock()  for i in range(0, n-1):  least = i  for k in range( i + 1 , n ):  if aList[k] < aList[least]:  least = k  #swap  aList[least],aList[i]=aList[i],aList[least]  stop=time.clock()  print(aList)  print(stop-start)  **Time complexity:**  **average ,best, worst : O(n^2)**  //it’s the same |
|  | a) Given a set of men and women design and implement Gale- Shapely algorithm to determine the stable set of marriages among them.  **Assumptions:**  Men propose first according to their preference list.  Women can choose a better partner based on the preference.   |  |  | | --- | --- | | Men’s preference list | Women’s preference list | | |  |  |  |  | | --- | --- | --- | --- | | A | V | W | X | | B | W | V | X | | C | V | W | X | | |  |  |  |  | | --- | --- | --- | --- | | V | A | B | C | | W | B | C | A | | X | C | A | B | |   b) Write the time complexity of the above algorithm.  **Gale Shapely**  # include<stdio.h>  # define max 505  int men[max][max],women[max][max],mmark[max],mc[max],wc[max];  int wprefer(int w,int m,int m1,int n)  {  int i;  for(i=1;i<=n;i++)  {  if(women[w][i]==m)  return 1;  else if(women[w][i]==m1)  return 0;  }  }  void stable\_match(int n)  {  int i,freecount;  for(i=1;i<=max;i++)  {  wc[i]=-1;  mmark[i]=0;  }  freecount=n;  while(freecount>0)  {  int m;  for(m=1;m<=n;m++)  for(i=1;i<=n&&mmark[m]==0;i++)  {  int w=men[m][i];  if(wc[w]==-1)  {  wc[w]=m;  mmark[m]=1;  freecount--;  }  else  {  int m1=wc[w];  if(wprefer(w,m,m1,n))  {  wc[w]=m;  mmark[m1]=0;  mmark[m]=1;  }  }  }  }  }  int main()  {  int n,i,j;  printf("enter the number of men and women\n");  scanf("%d",&n);  printf("enterthe preference list of women\n");  for(i=1;i<=n;i++)  {  printf("enter the preference list of %d women:\n",i);  for(j=1;j<=n;j++)  {  scanf("%d",&women[i][j]);  }  }  printf("enter the preference list of men\n");  for(i=1;i<=n;i++)  {  printf("enter the preference list of %d men:\n",i);  for(j=1;j<=n;j++)  {  scanf("%d",&men[i][j]);  }  }    stable\_match(n);  for(i=1;i<=n;i++)  mc[wc[i]]=i;  printf("the stable matching is:\n");  for(i=1;i<=n;i++)  printf("%d woman with %d man\n",i,mc[i]);  return 0;  }  **Time complexity:** **O(n^2)** |
|  | a) In a shop, a set of jackets numbered according to their sizes are present where half of them are sorted and the rest are unsorted. Shopkeeper now takes the jacket from the unsorted part and put it into the sorted part until all the jackets are sorted according to their sizes.  Design and implement an algorithm to solve it using random numbers and also display the execution time.  b) Write the best-case, average-case and worst-case for the algorithm.  **Insertion Sort**  import random  import time  n=int(input("enter the number of elements"))  a = [random.randint(0,100) for i in range(n)]  print(a)  start=time.clock()  for i in range(1,n):  r=a[i]  j=i-1  while(j>=0 and a[j]>r):  a[j+1]=a[j]  j=j-1  a[j+1]=r  stop=time.clock()  print(a)  print(stop-start)  **Time complexity:**  Best case: **O(n)**  Avg. case :**O(n^2)**  Worst case: **O(n^2)** |
|  | a) A GPS navigation system needs an approach to discover the reachable areas in a given geographical region from a given source area.  Design and implement an algorithm to find which nodes can be reached from a given source node for the following graph.    b) Write the time-complexity of the above algorithm.  def bfs(s,ver) :  visited=[0 for i in range(ver+1)]  visited[s]=1  level=[[s]]  i=0  tree=[]  while level[i]:  level.append([])  for u in level[i]:  for v in a[u]:  if visited[v]==0:  visited[v]=1  tree.append((u,v))  level[i+1].append(v)  i=i+1  level.pop()  print("the tree has the following edges")  print(tree)  print("layers:")  print(level)  ver=int(input("enter the number of vertices"))  e=int(input("enter the number of edges"))  a={}  for i in range(ver+1):  a[i]=[]  for i in range(e):  u=int(input("enter u"))  v=int(input("enter v"))  a[u].append(v)  a[v].append(u)  #print (a)    for i in range(ver+1):  a[i].sort()  del a[0]  print (a)  x=int(input("enter the start vertex"))  bfs(x,ver)  **Time complexity: O(|V|+|E|)** |
|  | a) Design and implement merge sort algorithm that takes random number input and displays the execution time required.  b) State the design strategy used for sorting and time complexity of the same.  **Merge Sort**  import random  import time  n=int(input("Enter the number of elements:"))  a=[random.randint(0,100) for i in range(n)]  def merge(a,b):      i,j,c=0,0,[]      while((i<len(a)) and (j<len(b))):          if (a[i]<b[j]):              c.append(a[i])              i=i+1          else:              c.append(b[j])              j=j+1      while(i<len(a)):          c.append(a[i])          i=i+1      while(j<len(b)):          c.append(b[j])          j=j+1      return c    def merge\_sort(a):      if(len(a)>1):          a1=merge\_sort(a[0:len(a)//2])          b1=merge\_sort(a[len(a)//2:])          a=merge(a1,b1)      return a    print("The array is :\n",a)  start=time.clock()  print("The sorted array is :")  print(merge\_sort(a))  stop=time.clock()  print("time : ", stop-start)  **Time complexity: O(n log(n))** |
|  | a) Three users in an online music portal listen to a playlist of 8 songs that are numbered from 1 to 8 in a random order. Each user needs to be recommended to another user playlist’s order that has minimum number of inversions.  Design and implement an algorithm to determine the number of inversions.  b) State the design strategy used for counting the inversions and time complexity of the same.  **Counting Inversions**  import random  import time  count = 0  def merge\_sort(li):  if len(li) < 2: return li  m = len(li) // 2  return merge(merge\_sort(li[:m]), merge\_sort(li[m:]))  def merge(l, r):  global count  result = []  i = j = 0  while i < len(l) and j < len(r):  if l[i] < r[j]:  result.append(l[i])  i += 1  else:  result.append(r[j])  count = count + (len(l) - i)  j += 1  result.extend(l[i:])  result.extend(r[j:])  return result    n = int(input("Enter number of elements in the list"))  unsorted = [random.randint(0,100) for i in range(0,n)]  print("Unsorted list" , unsorted)  start = time.clock()  print ("Sorted List" , merge\_sort(unsorted))  stop = time.clock()  print ("Number of inversions = " ,count , "\nTime taken", stop-start )  **Time complexity: O(n log(n))** |
|  | a) In a database of numbers there is a table of unsorted numbers. The database admin now wants to sort these numbers using an approach wherein a pivot element is selected for sorting. At certain point, the first half elements are less than the pivot and right half elements are greater than the pivot.  Design and implement an algorithm to solve it using random numbers and also display the execution time.  b) State the design strategy used for sorting and time complexity of the same?  **Quick Sort**  import random  import time  def partition(l,r):  global a  pivot=a[l]    i=l+1  j=r    while True:  while (i<=j and a[i]<=pivot):  i=i+1  while (i<=j and a[j]>=pivot):  j=j-1  if i>j:  break  else:  a[i], a[j] = a[j], a[i]    a[l],a[j] = a[j], a[l]  return j    def quicksort(l,r):  global a  if l<r:  mid=partition(l,r)  quicksort(l,mid-1)  quicksort(mid+1,r)    m= int(input("Enter the number of elements in the list: "))  a = [random.randint(0,100) for i in range(0,m)]  print(a)  start=time.clock()  quicksort(0,len(a)-1)  stop =time.clock()  print(a)  print(stop-start)      **Time complexity:**  Best case : **O(n log(n))**  Avg case: **O(n log(n))**  Worst case: **O(n^2)** |
|  | a) A truck driver is given a set of locations to be covered with their distances by a company. The company strictly orders that truck should be started from a particular location.  Design and implement an algorithm that gives a greedy solution to the truck driver’s problem and display the shortest path for a given source location to all other locations.    b) What is the time complexity of the above algorithm?  **Dijkstras**  import heapq as hq  def Dijk(graph,start):  n = len(graph)  Q = [[0, start]]  #print(Q)  d = [999 for i in range(n)]  #print(d)  d[start]=0  while Q:  [length, u] = hq.heappop(Q)  for v in range(n):  if d[v] > d[u] + graph[u][v]:  d[v] = d[u] + graph[u][v]  hq.heappush(Q, [d[v], v])  return d  #graph = [[0, 5, 10, 999], [5, 0 ,4, 11],[10, 4, 0, 5], [999, 11, 5,0]]  graph = []  n = int(input("Enter number of nodes"))  print("Enter the weights of respective edges")  for i in range(0,n):  m= []  print( "Next edge . . .")  for k in range(0,n):  print( "From ", i+1 ," To ", k+1)  val= int(input())  m.append(val)    graph.append(m)  d = Dijk(graph,0)  print (d)  **Time complexity:**  **O(V^2)** //for adjacency matrix  **O(E log(V))** // for adjacency list |
|  | a) A car driver is given a set of locations to be covered with their distances by a company. Now the company gives a privilege for the car driver to start at any arbitrary location. But, the condition is the route chosen by the driver should be minimum i.e. the total cost of the entire driving should be minimum.  Design and implement an algorithm that gives a greedy solution to the car driver and display the minimum cost achieved.    b) What is the time complexity of the above algorithm?  **Prims**  #include<stdio.h>  int main()  { int i,j,n;  int cost[10][10];  int a,b,u,v,min,mincost=0,visited[10]={0},ne=1;  printf("enter the number of nodes\n");  scanf("%d",&n);  for(i=1;i<=n;i++)  for(j=1;j<=n;j++)  {printf("enter cost between %d and %d : \n",i,j);  scanf("%d",&cost[i][j]);  }  visited[1]=1;  while(ne<n)  {    min=999;    for(i=1;i<=n;i++)  for(j=1;j<=n;j++)  if(cost[i][j]<min)  if(visited[i]!=0)  {min=cost[i][j];  a=u=i;  b=v=j;}  if(visited[u]==0||visited[v]==0)  {printf("node %d and node %d : cost %d\n",a,b,min);  ne=ne+1;  mincost=mincost+min;  visited[b]=1;}  cost[a][b]=cost[b][a]=999;  }    printf("%d",mincost);  }  **Time complexity: O(E log(V))**  // for adjacency list |
|  | a) A phone company wants to lay lines for communication in a city. Different amounts are charged for connecting between a pair of cities.  Design and implement a greedy solution such that it forms a spanning tree with minimum cost.    b) What is the time complexity of the above algorithm?  **Kruskals**   #include<stdio.h>  int n,cost[10][10];  void kruskal();  int main()  {              int i,j;  **// Read the number of nodes**              printf("Enter the number of nodes\n");              scanf("%d",&n);              printf("Enter the cost matrix\n");                for(i=1;i<=n;i++)              {                          for(j=1;j<=n;j++)                          {                                      printf("Enter the cost between %d and %d\n",i,j);                                      scanf("%d",&cost[i][j]);                          }              }              kruskal();              return 0;  }    void kruskal()  {              int i,parent[10],j,ne=0,mincost=0,u,v,min,a,b;  **//Initialize the parent array.**              for(i=1;i<=n;i++)                          parent[i]=0;  **//Algorithm : Unless n-1 edges**              while(ne != n-1)              {min=999; **// Get the min from the cost matrix**                          for(i=1;i<=n;i++)                          {                                      for(j=1;j<=n;j++)                                      {                                                                if(cost[i][j]<min)                                                              {                                                                          min=cost[i][j];                                                                          a=u=i;                                                                          b=v=j;                                                              }                                         }                          } **//Find the parent of each 'u' and 'v'.**                          while(parent[u]!=0)                                      u=parent[u];                            while(parent[v]!=0)                                      v=parent[v];    if(u!=v) **// If u!=v then no cycle in the graph**                          {                                      printf("The next node from %d-------> is %d with cost %d\n",a,b,min);                                      parent[v]=u;                                      ne=ne+1;                                      mincost=mincost+min;                      }**//Mark cost of 'a' and 'b' to infinity.**                          cost[a][b]=cost[b][a]=999;              }**// end while**  printf("The mincost is %d\n",mincost);  }  **Time complexity: O(E log(V))** // for adjacency list |
|  | a) Alia is planning for a trekking expedition with a backpack that can hold 7kg. She needs to select the most valuable items from the following list that can be accommodated within the backpack.  Design and implement an algorithm that displays the most valuable items that can be carried by him using Dynamic programming principles.   |  |  |  | | --- | --- | --- | | **Items** | **Weight** | **Value** | | 1 | 3 | 10 | | 2 | 5 | 4 | | 3 | 6 | 9 | | 4 | 2 | 11 |   b) What is the recurrence relation used and time complexity of the above algorithm?  **Knapsack**  n=int(input("enter the number of items"))  W=int(input("enter the knapsack capacity"))  items=[0]  for i in range(1,n+1):  items.append(i)  weights=[0]  value=[0]  for i in range(1,n+1):  print("enter the weight for item ",i, " : ")  wi=int(input())  weights.append(wi)  print("enter the value for item ",i ," : ")  vi=int(input())  value.append(vi)  print(items)  print(weights)  print(value)  M = [[0 for i in range(W+1)] for j in range(n+1)]  i = 1  while i <= n:  x = 1  while x <= W:  if weights[i] > x:  M[i][x] = M[i-1][x]  else:  M[i][x] = max(M[i-1][x],value[i]+M[i-1][x-weights[i]])  x = x + 1  i = i + 1  print(M)  print("maximum value= ",M[n][W])  i=n  k=W  cont=[]  while i>0 and k>0:  if M[i][k]!=M[i-1][k]:  cont.append(i)  k=k-weights[i]  i=i-1  else:  i=i-1  print("items in knapsack are: ")  print(cont)  **Time complexity: O(nW)** |
|  | a) A drama venue needs to be allocated for different drama school requests such that maximum profit is obtained for the company owning the drama venue. The requests are shown in the table with start–time, finish-time and the amount affordable by the drama school.  Design and implement an algorithm such that maximum profit is obtained for the company owning the drama venue using Dynamic programming principles.   |  |  |  |  | | --- | --- | --- | --- | | **Drama**  **School** | **Start-**  **time** | **Finish-**  **time** | **Value** | | 1 | 1 | 2 | 100 | | 2 | 2 | 5 | 200 | | 3 | 3 | 6 | 300 | | 4 | 4 | 8 | 400 | | 5 | 5 | 9 | 500 | | 6 | 6 | 10 | 100 |   b) What is the recurrence relation used and time complexity of the above algorithm?  **Weighted interval scheduling**  M = []  Schedule = []  def maximum(a,b):  if a > b :  return a  else:  return b  def calculate\_predecessor(jobs,n):  p = [0 for i in range(n+1)]  cur\_job = n  chosen\_job = cur\_job - 1  while cur\_job > 1 :  if chosen\_job <= 0 :  p[cur\_job] = 0  cur\_job=cur\_job-1  chosen\_job=cur\_job-1  else:  if jobs[cur\_job][0] < jobs[chosen\_job][1]:  chosen\_job = chosen\_job - 1  else:  p[cur\_job] = chosen\_job  cur\_job = cur\_job-1  chosen\_job = cur\_job -1  return p  def opt(j,jobs,p):  global M  if j == 0:  return M[j]  elif j ==1:  M[j] = maximum(jobs[j][2],0)  return M[j]  else:  if M[j] == -1:  M[j] = maximum(opt(j-1,jobs,p),jobs[j][2]+opt(p[j],jobs,p))  return M[j]  def wis(jobs,n):  p = calculate\_predecessor(jobs,n)  value = opt(n,jobs,p)  return value,p  def find\_solution(j,jobs,p):  global M  global Schedule  if j > 0 :  if jobs[j][2] + M[p[j]] >= M[j-1]:  Schedule.append(j)  find\_solution(p[j],jobs,p)  else:  find\_solution(j-1,jobs,p)  return  def main():  n = int(input("Enter the number of jobs: "))  global M  M = [-1 for i in range(n+1)]  M[0] = 0  jobs = [0]  for i in range(n):  s = int(input("Start time: "))  f = int(input("Finish time: "))  v = int(input("Value: "))  jobs.append((s,f,v))  max\_value,p = wis(jobs,n)  print(M)  print(max\_value)  global Schedule  find\_solution(n,jobs,p)  print(Schedule)  return  main()  **Time complexity: O(n log(n))** |
|  | a) Design and implement Bellman ford algorithm to find the shortest path from a given source to all other nodes using dynamic programming.    b) What is the time complexity of the above algorithm?  **Bellman Ford**  def bf(adj\_list,n,s):  distance = [999 for i in range(n+1)]  distance[s] = 0  for i in range(n-1):  for pair in adj\_list:  distance[pair[1]] = min(distance[pair[1]],distance[pair[0]]+pair[2])  del(distance[0])  print(distance)    def main():  adj\_list= []  n=int(input("enter the number of vertices"))  e=int(input("enter the number of edges"))  for i in range(e):  u = int(input("Enter the vertex u: "))  v = int(input("Enter the vertex v: "))  w = int(input("Enter the corresponding weights: "))  l=[u,v,w]  adj\_list.append(l)  print(adj\_list)  source = int(input("Enter a source:"))  bf(adj\_list,n,source)  main()  **Time complexity: O(|V|\*|E|)** |
|  | a) Design and implement an algorithm for Travelling salesman problem using Branch and bound technique.    b) What is the time complexity of the above algorithm?  #include<stdio.h>    int matrix[25][25], visited\_cities[10], limit, cost = 0;    int tsp(int c)  {  int count, nearest\_city = 999;  int minimum = 999, temp;  for(count = 0; count < limit; count++)  {  if((matrix[c][count] != 0) && (visited\_cities[count] == 0))  {  if(matrix[c][count] < minimum)  {  minimum = matrix[count][0] + matrix[c][count];  }  temp = matrix[c][count];  nearest\_city = count;  }  }  if(minimum != 999)  {  cost = cost + temp;  }  return nearest\_city;  }    void minimum\_cost(int city)  {  int nearest\_city;  visited\_cities[city] = 1;  printf("%d ", city + 1);  nearest\_city = tsp(city);  if(nearest\_city == 999)  {  nearest\_city = 0;  printf("%d", nearest\_city + 1);  cost = cost + matrix[city][nearest\_city];  return;  }  minimum\_cost(nearest\_city);  }    int main()  {  int i, j;  printf("Enter Total Number of Cities:\t");  scanf("%d", &limit);  printf("\nEnter Cost Matrix\n");  for(i = 0; i < limit; i++)  {  printf("\nEnter %d Elements in Row[%d]\n", limit, i + 1);  for(j = 0; j < limit; j++)  {  scanf("%d", &matrix[i][j]);  }  visited\_cities[i] = 0;  }  printf("\nEntered Cost Matrix\n");  for(i = 0; i < limit; i++)  {  printf("\n");  for(j = 0; j < limit; j++)  {  printf("%d ", matrix[i][j]);  }  }  printf("\n\nPath:\t");  minimum\_cost(0);  printf("\n\nMinimum Cost: \t");  printf("%d\n", cost);  return 0;  }  **Time complexity: O((n^2)\*(2^n))** |

**Note : Execution : 35 Marks**

**Write up : 8 marks**

**Viva : 7 marks**

**Change of question : -10 marks**