National University of Computer & Emerging Sciences, Karachi
Fall-2020 Department of Computer Science
Final Exam
01 January 2021, 03:00 PM – 06:00 PM

Course Code: CS302	**Course Name:** Design and Analysis of Algorithms
Instructor Name / Names: Dr. Muhammad Atif Tahir, Dr. Fahad Sherwani, Zeshan Khan, Waqas Sheikh, Sohail Afzal	
Student Roll No:	**Section:**

Instructions:

- Return the question paper.
- Read each question completely before answering it. There are **11 questions** on **5 pages**.
- In case of any ambiguity, you may make assumption. But your assumption should not contradict any statement in the question paper.

Time: 180 minutes. **Max Marks: 50**

Question # 1 **[0.5*10 = 5 marks]**

Answer the following questions. You must explain in only 3-4 lines.

(a) What is meant by complexity of an algorithm?

(b) What is the difference between divide & conquer approach and Dynamic Programming?

(c) Write down the formal definition of Big-Theta Notation i.e. in terms of f(n) and g(n)

(d) Write down any one real life application of Maximum sub-array.

(e) When does the worst case of quick sort occur?

(f) Write down any one real life application of Dijkstra Algorithm.

(g) Which one is faster binary search or linear search? Why

(h) What is the difference between NP-hard and NP-complete problems?

(i) Give one example of P and NP problem.

(j) If one problem can be solved in polynomial time by non-deterministic algorithm and can be verified in polynomial time by deterministic algorithm, then in which class this problem lies?

Question # 2 **[0.125 each = 4.5 marks]**

Write time complexities of below mentioned algorithms.

Algorithms	Worst Case	Best Case	Average Case
Insertion Sort			
Merge Sort			
Quick Sort			
Counting Sort			
Radix Sort			
Graham Scan			
Package Wrap (Jarvis)			
Kruskal's algorithm for finding MST			
Shortest path by Dijkstra			
Shortest path by Bellman Ford			
Matrix Chain Multiplication by Dynamic Programming			
0/1 Knapsack by Dynamic Programming			

Question # 3 **[2.5*2 = 5 marks]**

Compute the time complexity for both algorithms one by one. Show all steps.

Algorithm 1:

```
Sum=0;
for(int i = 0; i < n; i++)
{
   for( int j = 0; j < n * n; j++)
      {
          for(int k = 0; k < j; k++)
          {
            sum++;
          }
      }
}
```

Algorithm 2:

```
Sum=0;
for(int i = 0; i < n; i++)
{
   for( int j = 0; j < 2*i; j++)
      {
          for(int k = 0; k > j; k--)
          {
            sum++;
          }
      }
}
```

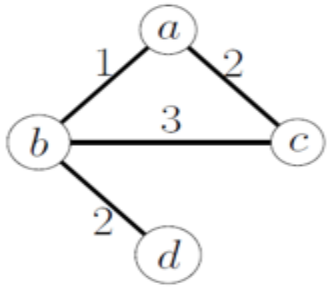
Question # 4 **[3 marks]**

Consider the following APPROX-VERTEX-COVER algorithm. Proof that this algorithm is 2-approximation method for VERTEX-COVER.

```
APPROX-VERTEX-COVER(G)
    C = ∅;
    E'=G.E;
    while(E' ≠ ∅ ){
        Randomly choose a edge (u,v) in E', put u and v into C;
        Remove all the edges that covered by u or v from E'
    }
    Return C;
```

Question # 5 **[2+3=5 marks]**

An edge coloring of an undirected graph *G* is an assignment of labels to the edges so that any two edges that share a vertex have different colors. The edges coloring problem is to color the edges of a graph with the fewest number of colors. Below is an example of an edge-coloring with 3 colors, which is optimum for this graph. The colors are labeled 1, 2, 3.



a) Show that if a graph *G* has a vertex with degree d, then at least d colors are necessary to edge-color *G*. Note in particular this shows that 3 colors is optimum for the example above: the vertex (which is the vertex of maximum degree) has degree 3. However, your proof should work for any graph *G*.

b) Vizing's theorem implies that there is a polynomial time routine, VIZINGALGO, that colors all edges of a graph with maximum degree Δ with at most $\Delta + 1$ colors. You do not need to derive how such an algorithm works.
Instead, use VIZINGALGO to give a 2-approximation to edge-coloring problem. That is, give an algorithm *A* such that the number of colors that it uses to edge-color a graph *G*,*A(G)*, always satisfies,

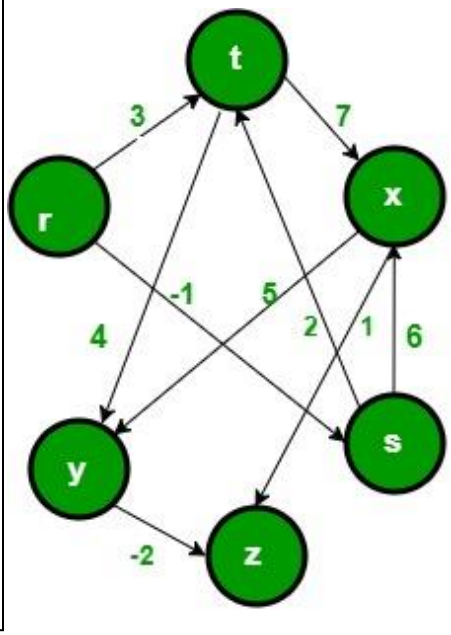
$$A(G) \leq 2.opt(G),$$

where $opt(G)$ is the minimum number of colors needed to edge-color *G*. You may assume that $\Delta \geq 1$, that is, the graph has at-least one edge.

Question # 6 **[5 marks]**

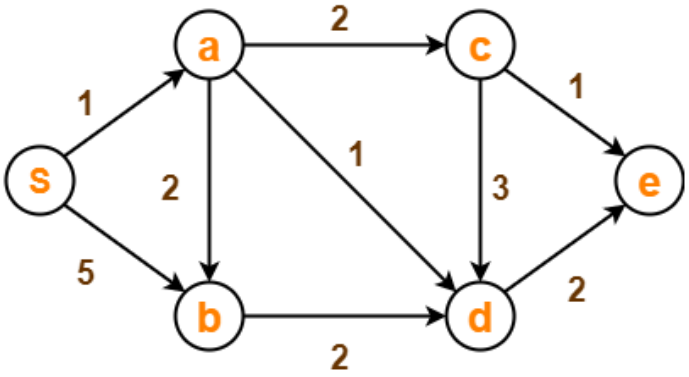
Apply the following algorithm on the provided directed graph to compute shortest path from node **"s"** to all other nodes.

```
def sort(v, visited, stack):
        visited[v] = True
        if v in graph:
                for node, weight in graph[v]:
                        if visited[node] == False:
                                sort(node, visited, stack)
        stack.append(v)
def path_optimal(s):
        visited = {False}
        stack =[]
        for i in range(V):
                if visited[i] == False:
                sort(s, visited, stack)
        dist = {∞}
        dist[s] = 0
        while (stack.isempty()==False):
                i = stack.pop()
                for node, weight in graph[i]:
                        if dist[node] > dist[i] + weight:
                                dist[node] = dist[i] + weight
        return dist
```



Question # 7 **[3+2=5 marks]**

a) Apply *Dijkstra algorithm* on given graph to find shortest path to every vertex starting from "S".



b) Suppose that we are given any weighted, directed graph *G=(V,E)* in which edges that leave the source vertex *"s"* may have negative weights, all other edge weights are non-negative, and there are no negative-weight cycles. Argue that *Dijkstra's algorithm* correctly finds shortest paths from *"s"* in this type of graph.

Question # 8 **[5 marks]**

We are required to place cell towers (Zong) in various cities to provide network coverage to whole country. For simplicity, we assume the cities in a row instead of two-dimensional plane. Each city has a tower cost and the coverage of that city with its both neighboring cities.
e.g. If we have cities {A, B, C, D, E, F, G, H}, a tower in city B can cover the city A, B and C. Design a linear time algorithm to compute the minimum cost necessary to provide coverage to all cities. *(Hint: Use Dynamic Programming)*

Question # 9 **[1+4 = 5 marks]**

Given a set of points $S = \{(1,2), (2,5), (3,4), (4,4), (5,7), (6,8)\}$. Find the pair of points in S that are closest together. You need to provide 2 solutions:

a) Solution in $O(n^2)$
b) Solution in $O(nlogn)$

Question # 10 **[1.5+1.5+2 = 5 marks]**

A) Given two convex polygons P1 and P2 with the convex hull of H1 and H2 ($H1 \neq H2$) respectively. Is it possible to have the convex hull H3 of $P1 \cup P2$ (All points of P1 and P2) same as H1? (Yes or No) Justify with example.
B) Given a convex polygon P (set of points) and a point Q inside convex hull. Design O(n) time algorithm to compute convex hull on $P \cup Q$ (All points of P and Q)
C) List two real life applications of convex hull.

Question # 11 **[2.5 marks]**

Given an array $A = \{576, 494, 194, 296, 278, 176, 954\}$, apply radix sort algorithm to sort these numbers. Show all iterations.

```
RADIX-SORT( array A, int n, int d)
1  for i ← 1 to d
2  do stably sort A w.r.t i^th lowest order digit
```

BEST OF LUCK

National University of Computer & Emerging Sciences, Karachi
Fall-2020 Department of Computer Science
Final Exam
01 January 2021, 03:00 PM – 06:00 PM

Course Code: CS302	**Course Name:** Design and Analysis of Algorithms
Instructor Name / Names: Dr. Muhammad Atif Tahir, Dr. Fahad Sherwani, Zeshan Khan, Waqas Sheikh, Sohail Afzal	
Student Roll No:	**Section:**

Instructions:

- Return the question paper.
- Read each question completely before answering it. There are **11 questions** on **5 pages**.
- In case of any ambiguity, you may make assumption. But your assumption should not contradict any statement in the question paper.

Time: 180 minutes. **Max Marks: 50**

Question # 1 **[0.5*10 = 5 marks]**

Answer the following questions. You must explain in only 3-4 lines.

(a) What do you mean by complexity of an algorithm?

a. In computer science, the complexity of an algorithm is a way to classify how efficient an algorithm is, compared to alternative ones. We determine efficiency by space and time complexity.

(b) Write any one difference between divide & conquer approach and Dynamic Programming?

a. In dynamic programming, we have overlapping sub-problems where results are not re-computed but rather stored while in divide and conquer, this is not the case.

(c) Write down the formal definition of Big-Theta Notation i.e. in terms of f(n) and g(n)

- **Definition:** $f(n) = \Theta(g(n))$ iff there are three positive constants c_1, c_2 and n_0 such that
  $c_1|g(n)| \leq |f(n)| \leq c_2|g(n)|$ for all $n \geq n_0$
- If $f(n)$ is nonnegative, we can simplify the last condition to
  $0 \leq c_1\, g(n) \leq f(n) \leq c_2\, g(n)$ for all $n \geq n_0$

(d) Write down any one real life application of Maximum sub-array.

a. In **computer vision**, **maximum-subarray** algorithms are used on bitmap images to detect the brightest area in an image.

(e) **When does the worst case of quick sort occur?**

a. The **worst case occurs** when the picked pivot is always an extreme (smallest or largest) element. This **happens** when input array is **sorted** or reverse **sorted** and either first or last element is picked as pivot.

(f) Write down any one real life application of Dijkstra Algorithm.

a. To find shortest path from one city to every other city

(g) Which is faster between binary search and linear search? Why

a. Binary search is faster with (logn) complexity while linear search has O(n) complexity. Binary search ignores half elements in every iteration that's why it takes less amount of time

(h) All NP-hard problems are also NP-complete or not? If no then what is the difference?

a. All NP-hard problems are not NP-complete. NP-complete are least hard problems of NP-hard. All NP-complete problems can be verified in polynomial time but all NP-hard problems cannot.

(i) Give one example of P and NP problem.

a. Finding if given number is prime because solution can be obtained and verified in polynomial time. But detecting hamiltonian cycle / solving sudoku game cannot be solved in polyomial time but can be verified in polynomial time so they are NP problems

(j) If one problem can be solved in polynomial time by non-deterministic algorithm and can be verified in polynomial time by deterministic algorithm then in which class this problem lies?

a. It is NP problem

Question # 2 **[0.125 each = 4.5 marks]**

Write time complexities of below mentioned algorithms.

Algorithms	Worst Case	Best Case	Average Case
Insertion Sort	$O(n^2)$	$O(n)$	$O(n^2)$
Merge Sort	$O(nlogn)$	$O(nlogn)$	$O(nlogn)$
Quick Sort	$O(n^2)$	$O(nlogn)$	$O(nlogn)$
Counting Sort	$O(n+k)$	$O(n+k)$	$O(n+k)$
Radix Sort	$O(nk)$ or $O(d(n+k))$	$O(nk)$ or $O(d(n+k))$	$O(nk)$ or $O(d(n+k))$
Graham Scan	$O(nlogn)$	$O(nlogn)$	$O(nlogn)$
Package Wrap (Jarvis)	$O(n^2)$	$O(nlogn)$	$O(nlogn)$
Kruskal's algorithm for finding MST	$ElogE$ or $ElogV$	$ElogE$ or $ElogV$	$ElogE$ or $ElogV$
Shortest path by Dijkstra	$O(V^2)$	$O(V+ElogV)$	$O(V+ElogV)$
Shortest path by Bellman Ford	$O(VE)$	$O(VE)$	$O(VE)$
Matrix Chain Multiplication by Dynamic Programming	$O(n^3)$	$O(n^3)$	$O(n^3)$
0/1 Knapsack by Dynamic Programming	$O(nW)$	$O(nW)$	$O(nW)$

Question # 3 **[2.5*2 = 5 marks]**

Compute the time complexity for both algorithms one by one. Show all steps.

Algorithm 1:

```
Sum=0;
for(int i = 0; i < n; i++)
{
   for( int j = 0; j < n * n; j++)
      {
          for(int k = 0; k < j; k++)
          {
            sum++;
          }
      }
}
```

Algorithm 2:

```
Sum=0;
for(int i = 0; i < n; i++)
{
   for( int j = 0; j < 2*i; j++)
      {
          for(int k = 0; k > j; k--)
          {
            sum++;
          }
      }
}
```

Solution # 3

Algorithm 1)

$$\sum_{i=1}^{n} \sum_{j=1}^{j^2} \sum_{k=1}^{j} 1$$

$$\sum_{i=1}^{n} \sum_{j=1}^{i^2} J$$

$$\sum_{i=1}^{n} (i^2(i^2+1))/2 \approx \theta(n^5)$$

Algorithm 2)

Inner loop= 1

Middle loop = $\sum_{j=1}^{2i}(1) = \sum_{j=1}^{2i} 1 = 2i$

Outer loop = $\sum_{i=1}^{n} 2i = \theta(n^2)$

Some constants are ignored for easy computation

Question # 4 **[3 marks]**

Consider the following APPROX-VERTEX-COVER algorithm. Proof that this algorithm is 2-approximation method for VERTEX-COVER.

```
APPROX-VERTEX-COVER(G)
    C = ∅;
    E'=G.E;
    while(E' ≠ ∅ ){
        Randomly choose a edge (u,v) in E', put u and v into C;
        Remove all the edges that covered by u or v from E'
    }
    Return C;
```

Solution # 4

APPROX-VERTEX-COVER is a polynomial-time 2-approximation algorithm.

Proof

The set C of vertices that is returned by APPROX-VERTEX-COVER is a vertex cover, since the algorithm loops until every edge in $G.E$ has been covered by some vertex in C.

To see that APPROX-VERTEX-COVER returns a vertex cover that is at most twice the size of an optimal cover, let A denote the set of edges that line 4 of APPROX-VERTEX-COVER picked. In order to cover the edges in A, any vertex cover—in particular, an optimal cover C^*—must include at least one endpoint of each edge in A. No two edges in A share an endpoint, since once an edge is picked in line 4, all other edges that are incident on its endpoints are deleted from E' in line 6. Thus, no two edges in A are covered by the same vertex from C^*, and we have the lower bound

$$|C^*| \geq |A| \tag{35.2}$$

on the size of an optimal vertex cover. Each execution of line 4 picks an edge for which neither of its endpoints is already in C, yielding an upper bound (an exact upper bound, in fact) on the size of the vertex cover returned:

$$|C| = 2|A| \ . \tag{35.3}$$

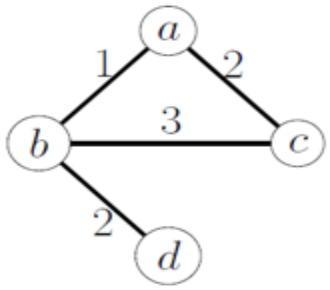
Combining equations (35.2) and (35.3), we obtain

$$\begin{aligned} |C| &= 2|A| \\ &\leq 2|C^*| \ , \end{aligned}$$

thereby proving the theorem. ■

Question # 5 **[2+3=5 marks]**

An edge coloring of an undirected graph *G* is an assignment of labels to the edges so that any two edges that share a vertex have different colors. The edges coloring problem is to color the edges of a graph with the fewest number of colors. Below is an example of an edge-coloring with 3 colors, which is optimum for this graph. The colors are labeled 1, 2, 3.



a) Show that if a graph *G* has a vertex with degree d, then at least d colors are necessary to edge-color *G*. Note in particular this shows that 3 colors is optimum for the example above: the vertex (which is the vertex of maximum degree) has degree 3. However, your proof should work for any graph *G*.

b) Vizing's theorem implies that there is a polynomial time routine, VIZINGALGO, that colors all edges of a graph with maximum degree Δ with at most $\Delta + 1$ colors. You do not need to derive how such an algorithm works.
Instead, use VIZINGALGO to give a 2-approximation to edge-coloring problem. That is, give an algorithm *A* such that the number of colors that it uses to edge-color a graph *G*, *A(G)*, always satisfies,

$$A(G) \leq 2.opt(G),$$

where $opt(G)$ is the minimum number of colors needed to edge-color *G*. You may assume that $\Delta \geq 1$, that is, the graph has at-least one edge.

Solution# 5

SOLUTION:

Given a vertex v in graph G with incident edges $e_1, ..., e_d$, then each edge $e_1, ..., e_d$ must be a different color in any valid edge-coloring. Therefore, there must be at least d colors used in any valid edge-coloring of G.

GRADING:

Points were given based on how along these lines the solutions are.

Run VIZINGALGO on the graph and output it's edge coloring.

From part a), we know that OPT is at least as large as the degree of any vertex in the graph, so we must have $OPT \geq \Delta$. Our output from VIZINGALGO uses at most $\Delta + 1$ colors, and $\Delta + 1 \leq 2\Delta$ when $\Delta \geq 1$, which was assumed. Therefore $\Delta + 1 \leq 2 \cdot OPT$, and VIZINGALGO is a 2-approximation algorithm.

GRADING:

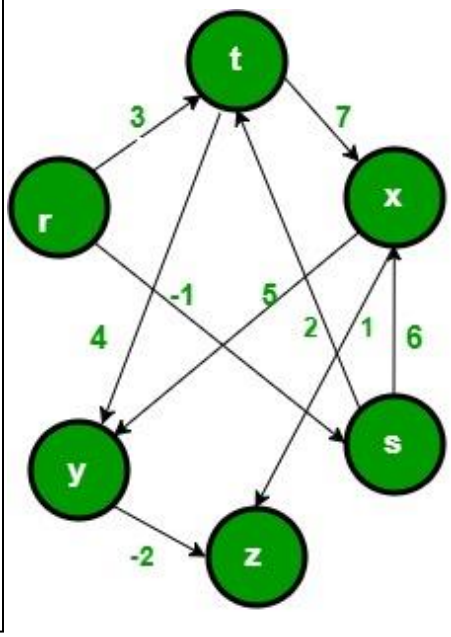
1 point for algorithm.

1 point for justification.

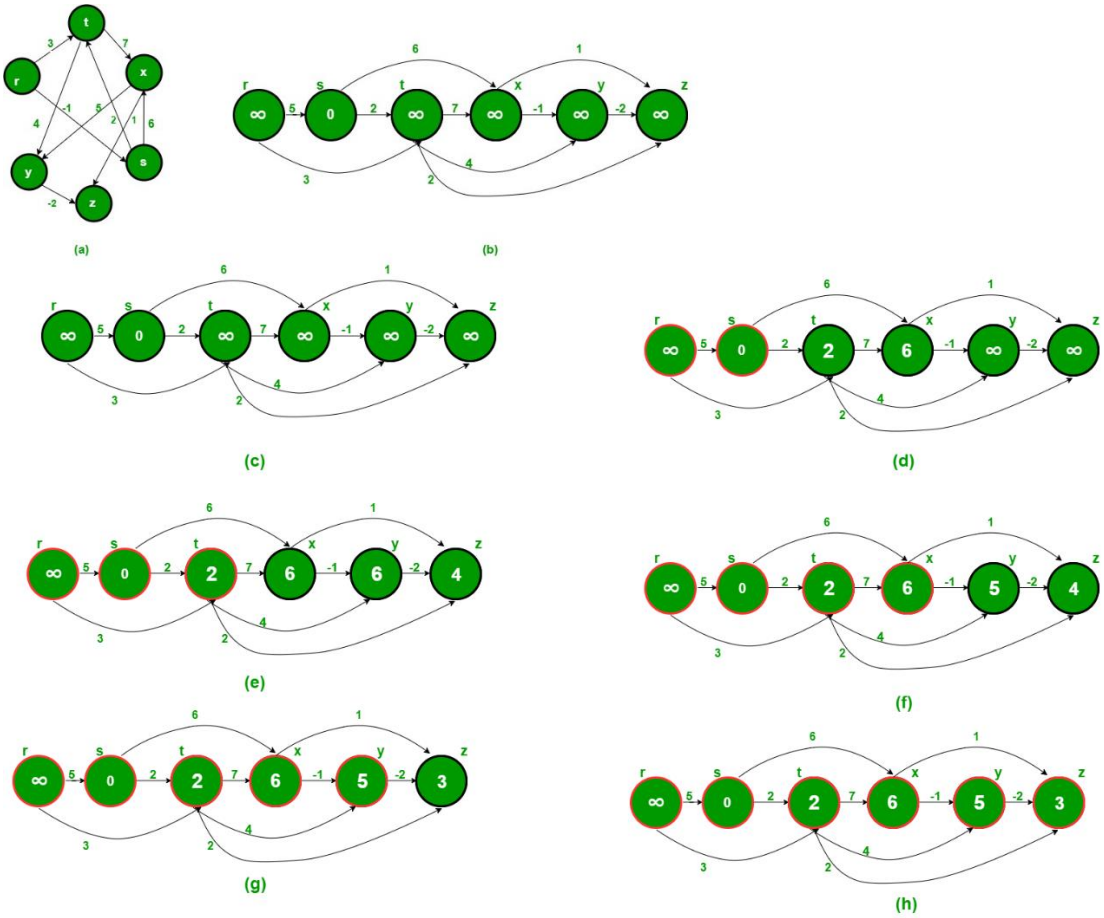
Question # 6 **[5 marks]**

Apply the following algorithm on the provided directed graph to compute shortest path from node **"s"** to all other nodes.

```
def sort(v, visited, stack):
        visited[v] = True
        if v in graph:
                for node, weight in graph[v]:
                        if visited[node] == False:
                                sort(node, visited, stack)
        stack.append(v)
def path_optimal(s):
        visited = {False}
        stack =[]
        for i in range(V):
                if visited[i] == False:
                sort(s, visited, stack)
        dist = {∞}
        dist[s] = 0
        while (stack.isempty()==False):
                i = stack.pop()
                for node, weight in graph[i]:
                        if dist[node] > dist[i] + weight:
                                dist[node] = dist[i] + weight
        return dist
```



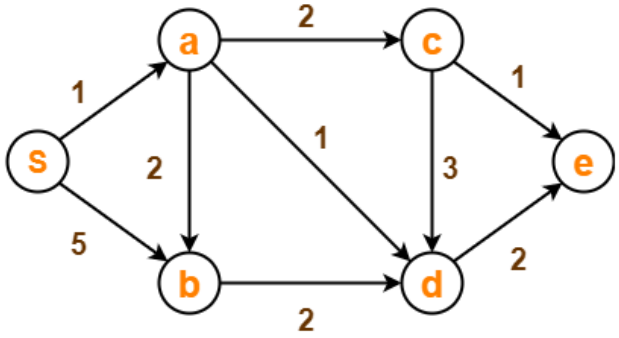
Solution # 6



dist[r,s,t,x,y,z]=dist[inf,0,2,6,5,3]

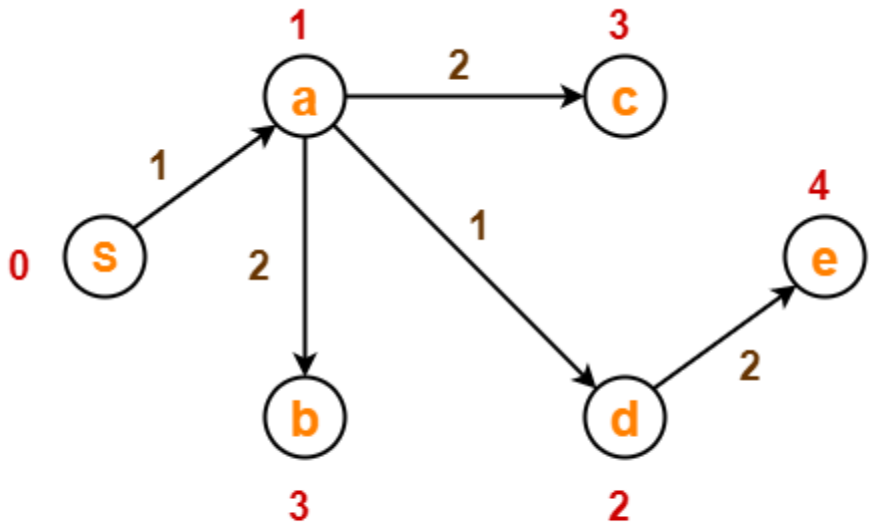
Question # 7 **[3+2=5 marks]**

a) Apply *Dijkstra algorithm* on given graph to find shortest path to every vertex starting from "S".



b) Suppose that we are given any weighted, directed graph *G=(V,E)* in which edges that leave the source vertex *"s"* may have negative weights, all other edge weights are non-negative, and there are no negative-weight cycles. Argue that *Dijkstra's algorithm* correctly finds shortest paths from *"s"* in this type of graph.

Solution # 7



Shortest Path Tree

The proof of correctness, Theorem 24.6, goes through exactly as stated in the text. The key fact was that $\delta(s, y) \leq \delta(s, u)$. It is claimed that this holds because there are no negative edge weights, but in fact that is stronger than is needed. This always holds if y occurs on a shortest path from s to u and $y \neq s$ because all edges on the path from y to u have nonnegative weight. If any had negative weight, this would imply that we had "gone back" to an edge incident with s, which implies that a cycle is involved in the path, which would only be the case if it were a negative-weight cycle. However, these are still forbidden.

Question # 8 **[5 marks]**

We are required to place cell towers (Zong) in various cities to provide network coverage to whole country. For simplicity, we assume the cities in a row instead of two-dimensional plane. Each city has a tower cost and the coverage of that city with its both neighboring cities.
E.g. If we have cities {A, B, C, D, E, F, G, H}, a tower in city B can cover the city A, B and C. Design a linear time algorithm to compute the minimum cost necessary to provide coverage to all cities. *(Hint: Use Dynamic Programming)*

Solution # 8

Recursive Relation: 2 Marks

$$\boldsymbol{f(n) = 0 \; if \; n == 0}$$
$$\boldsymbol{f(n) = c[n] \; if \; n == 1}$$
$$\boldsymbol{f(n) = min(C[n] + f(n-1), c[n] + f(n-2), c[n-1] + f(n-2))}$$

Recursive Algorithm: Memory based 1 Mark

```
Cost as C, Memory as Mem

F(n):
        If(n==0):
                Mem[n]=0
        If(n==1):
                Mem[n]=c[n]
        Else:
                if (Mem[n-1]==-1):
                        Mem[n-1]=f(n-1)
                If (Mem[n-2]==-1):
                        Mem[n-2]=f(n-2)
                Mem[n]=min(C[n]+Mem[n-1], C[n]+Mem[n-2], C[n-1]+Mem[n-2])
```

Iterative: 2 Marks

```
F(n):
        Mem={0}
        Mem[1]=c[1]
        For i in range(2,n):
                Mem[i]=min(c[i]+Mem[i-1], c[i]+Mem[i-2], c[i-1]+Mem[i-2])
        Return Mem[n]
```

Question # 9 **[1+4 = 5 marks]**

Given a set of points S = {(1,2),(2,5),(3,4),(4,4),(5,7), (6,8)}. Find the pair of points in S that are closest together. You need to provide 2 solutions :

a) Solution in $O(n^2)$
b) Solution in *O(nlogn)*

Solution # 9

A) O(n^2) = Brute Force [1 Point]
Distance from {1,2} to {2,5} = sqrt((2-1)^2 + (5-2)^2) = sqrt(10)
Shortest distance is sqrt((4-3)^2 + (4-4)^2) = 1
B) Merge Sort for O(nlogn)
Step 1: Sort According to x-axis
{(1,2),(2,5),(3,4),(4,4),(5,7), (6,8)}
Step 2: Find median of x-values which is 3.5 so
(1,2), (2,5), (3,4) on left hand side => Shortest distance b/w (1,2) and (3,4) which is sqrt(5) = 2.23
and
(4,4), (5,7). (6,8) on right hand side => Shortest distance b/w (4,4) and (5,7) which is sqrt(10) =
Step 3: Sort According to y-axis (to find elements in Combine)

P1: 1	**2**
P2: 3	4
P3: 4	4
2	5
5	7
6	9

We need to check distances of those which are 2 distance in sorted list so P1, P2 and P3 are within 2 distances accord to Y. So, (3,4) and (4,4) are shortest i.e. 1

Question # 10 **[1.5+1.5+2 = 5 marks]**

Solution # 10

A) Yes, when H2 is inside H1.
B) Constant time algorithm is return the current polygon.
C) Disease region of a country, ground region clustering

Question # 11 **[2.5 marks]**

Given an array $A = \{576, 494, 194, 296, 278, 176, 954\}$, apply radix sort algorithm to sort these numbers. Show all iterations.

Solution # 11

576		49[4]		9[5]4		[1]76		176
494		19[4]		5[7]6		[1]94		194
194		95[4]		1[7]6		[2]78		278
296	→	57[6]	→	2[7]8	→	[2]96	→	296
278		29[6]		4[9]4		[4]94		494
176		17[6]		1[9]4		[5]76		576
954		27[8]		2[9]6		[9]54		954

Here is the algorithm that sorts A[1..n] where each number is d digits long.

```
RADIX-SORT( array A, int n, int d)
1   for i ← 1 to d
2   do stably sort A w.r.t i^th lowest order digit
```

BEST OF LUCK