Design and Analysis of Algorithms Final Exam Solution

1.

Let the transmitted temp value be x.
Actual Temperature = y = x - 5
Now we have to search this y value in two arrays, which are ordered. We can first check which array
should we search by comparing y with maximum of both arrays. Then we can use binary search.
PS. We may have to search in both arrays, so it'll take 2lgn or O(lgn)

2.

a. The solution is just like applying the Shortest Job Next algorithm. By selecting the vehicle for unloading having the LOWEST unloading time. Therefore, if we sort all vehicle's unloading time in ascending order and select vehicles one by one in sequence, it will minimize the total time in the system for all vehicles.
b. Greedy algorithm
c. O(n log n) as the sorting time will dominate other operations.

3a.

i. Solution

	Problem 1	Problem 2	Problem 3	Problem 4	Problem 5
H_i	7	2	5	20	10
O_i	10	10	50	30	25

Given algorithm: H1+O2+H4+O5 = 7+10+20+25 = 62
Optimal solution: O1+O3+O5 = 10+50+25 = 85

ii. Solution:
Let Opt[i] denote the maximum obtainable marks for problems ito n.
Opt[n] = Max (H[n] , O[n]) = O[n] // initialization
Opt[n+1] = 0
For (i = n-1 down to 1)
Opt[i] = Max (H[i] +Opt[i+1] , O[i] + Opt[i+2])
The final answer will be in Opt[1]
Time Complexity = O(n)

3b.

i. Solution:
Total time = 180 minutes

	Problem 1	Problem 2	Problem 3	Problem 4	Problem 5
$time_i$	50	70	60	100	30
$marks_i$	100	110	50	130	30
marks/ time	2	1.6	0.8	1.3	1

Given algorithm solution: 100 + 110 + 30 = = 240 marks
Optimal solution = 100 + 110 + 50 = 260 marks
ii. Solution:
This problem can be mapped to binary knapsack problem as follows:
Items = Problems
Knapsack capacity = Total Time
Value of Item = marks of problem
Weight of Item = time required for problem
The time complexity is same as binary knapsack O(nT) where n is number of problems and T is total Time.

4.

Solution: Find the strongly connected components of the directed graph and compute the component graph G_c. If there is only one node in G_c then no new road is required. Otherwise,find the source and sink nodes of G_c. It is guaranteed that G_c will have only one source node (all places are reachable from Town Hall). Add a directed road from any vertex in sink to any vertex in the source node of G_c. Do this for each sink node. The minimum number of roads is same as the number of sink nodes.

5a.

a. In the following, let $d_A(x)$ be the shortest distance from A to a node x, and $d_B(x)$ the shortest distance from B to x.

Scheme:
Call Dijkstra on G with A as the source, mark all $d_A(.)$ values.
Call Dijkstra on G with B as the source, mark all $d_B(.)$ values.
Find the node h that minimizes $d_A(h) + d_B(h)$.

Pseudocode:

deliverToBob(G(V, E), A, B)

Create Arrays d_A and d_B
DijkstrawrtA(G, A) //this fills up d_A
DijkstrawrtB(G, B) //this fills up d_B

minSum ← INFINITY, minh ← 0
For each h in V (h!=A and h!=B)
IF(d_B[h]+ d_A[h] < minSum)
minSum ←d_A[h]+ d_B[h]
minh ← h

return h

Total time: O((|V|+|E|)lg|V|)

(b) In this case, let d(x) be the shortest distance from A to x. Let d_R(x) be the shortest distance from x to A (note this is the same as the shortest distance from A to x in the reverse graph G_R).

Scheme:

Call Dijkstra on G with A as the source, mark all d(.) values.
Compute G_R, the reverse graph of G.
Call Dijkstra on G_R with A as the source, mark all d_R(.) values.
Find the node h that minimizes d(h) + d_R(h).

Pseudocode:

dropAtHotel(G(V, E), A, B)

Create Arrays d and d_R to store distances and reverse distances with A as source
DijkstrawrtA(G, A) //this fills up d
G_R← ReverseGraph(G)
DijkstrawrtARev(G_R, A) //this fills up d_R
minSum ← INFINITY, minh ← 0
For each h in V (h!=A and h!=B)
IF(d[h]+ d_R[h] < minSum)
minSum ← d[h]+ d_R[h]
minh ← h
return h
Total time: O((|V|+|E|)lg|V|)

5b.

a. G= (V, E) is an undirected weighted graph. The destinations are the vertices in V. The edges are the connections, and their weights are the pair-wise cabling costs.
b. The most cost effective network must minimize the global cost of joining all the destinations. Hence, the network will be a minimum spanning tree, T. We can find it using Kruskal's algorithm in O(|E|lg|V|).
c. Removing an edge, e, from an MST, T, splits in into two sub-MST, T' and T'' respectively. From cut property, we know that e must be the lightest edge between T' and T''(although this makes intuitive sense, the proof needs a bit more detail). Now we will replace it with the second best option. While making sure that we do not create a cycle and only do linear work.

```
updateNetwork(T=(V, E), e=(a, b))
        //Separate T' and T'' by removing e=(a, b) from E
          //… and by marking the component numbers on the vertices in T' and
T''

Remove e from E

          markComponents (G)
//simply uses a single DFS call to mark the component numbers 1 and 2 on the
nodes of T' and T''

bestSoFar INF
bestEdge nil
        For each edge e'=(a', b') ∈ E
                IF(compnum(a')!= compnum(b'))
                        IF(w(a, b) <bestSoFar)
bestSoFarw(a, b)
bestEdge (a, b)

        Add bestEdge to E
```

Time taken: O(|V|+|E|)

d. We can adapt Prim's MST Algorithm to accomplish this task. All we need to do is start from the governor's office, g. After that we stop after Prim has added k-1 more vertices to the MST. This takes O(|V|+ (k+|E|)lg|V|), i.e. O(|E|)lg|V|) work. Below is the pseudo-code:

```
kMST(G=V, E), g)//where g is the starting point: the governor's office
  For each x in V
        cost(x)  INF
        par(x)   nil

 cost(g)  0
```

```
 H ← BuildMinHeap(V, cost)
  T ← {}
  Repeat k times
x ← H. removeMin()
        T.add(x)
For each {x, y} ∈E such that T.contains(y)==false
IF(w(x, y) < cost(y))
                        H.updateKey(y, w(x, y))
                        cost(y) ←w(x, y)
par(y) ← x
```

Note: Kruskal will not work in this case, as it does not grow a single MST. Prim will work because, inductively: for k=2, naturally it will give the correct MST by picking the node with the lightest edge from g.

From the lightest m-size tree that includes g, it will extend it to the lightest m+1 size tree (including g) by picking the next lightest connection.

In the best case scenario, all of the k lightest edges (cost $5, 000) are selected in the MST. In total an MST has |V| - 1 edges. k of these have cost 5000 and |V|-1-k have 1000 cost.
Total cost: (|V|-1-k)*10,000 + k*5000 = 10, 000|V| - 5000k - 10000