Question -1

Answer:

minimumCost(n,p,q)

start traversal from bottom

Calculate the cost for left

Cost (n-1,i) = distance(I,n-1) + p

Calculate the cost for right travsersal

Cost(n,j) = distance (j,n) + q

Total cost = Summation (cost of Left traversal + cost of right traversal)

End minimumCost

Question – 2

Answer:

$1 \leq j \leq n$, given array F[1….n], so array is in order. We need to find smallest element in the array. So, we can find smallest element of the array using Binary Search method.

Pseudo Code:

smallestElement (F[a….b])

IF a > b return F

Let m = [(a+b) / 2]

If F(m) > F(m-1) return smallestElement F[a…(m -1)]

Else if F[m] > F[m +1] return smallestElement (F [(m+1)…b])

Else return m

End smallest Element.

Now, let's find out runtime for this algorithm. Let, T(n) express the runtime for the above mentioned algorithm:

$$T(n) = \begin{cases} A & if\ n = 0 \\ T\left(\frac{n}{2}\right) + d & if\ n \geq 1 \end{cases}$$

Now, let's apply Master's theorem:

a = 1, b= 2 (a >= 1 & b > 1), We can apply Master's theorem
c = $\log_b a$ = $\log_2 1$ = 0 ; f(n) = d

Case – 1 : Not applicable

For, case 1, $n^{c-\epsilon}$ = $n^{-\epsilon}$.

So, if we apply limit test, this will grow infinitely with n -> ∞. So, case -1 is not correct.

Caae – 2

f(n) = d; g(n) = $n^c \ (\log n)^k$
Let's apply limit test for c = 0 & k = 0

$\lim_{n \to \infty} \frac{d}{n^0 \ (\log n)^0}$ = d (which is constant)

So, here case -2 works. So, T(n) ϵ Θ $(\log n)$. And as proved using Master's theorem (second case) O(log n).

Question -3

Answer:

We need to find path of highest bandwidth between given two vertices. So, we can find it using Dijkstra's algorithm as it is similar to problem of finding shortest distance between two nodes. We'll use max heap or max priority queue here instead of min priority queue used in traditional Dijkstra's algorithm. We also need to change relaxation step to insert minimum of the edge weights.

Using Dijkstra's

```
        graphBandwidth (G = (V, E), s, d))
                Let C = ∅
                Let P = ∅
        Assign s.bandwidth = ∞ and add s to C
                if d ∉ C
                        Let u ∈ P , and have the highest(estimate) bandwidth in P
                        Remove u from P
                        Add u to C
                        Relax (u)
                End if
                Return route(s,d)
        End graphBandwidth
Relax (u)
        For each vertex  v ∉ C and adjacent to u:
                Let estimate = min (u.bandwidth, bw(u,v))
```

If v ∉ P: Assign v.bandwidth = estimate and Assign v.previous = u and Add v to P
If v ∈ P and estimate > v.bandwidth: assign v.bandwidth = estimate
End for
End Relax

Proof: Consider an arbitrary weighted connected undirected graph G= (V,,w) with no negative weights and s∈$V$and d∈V.

Let S be a list of vertices and the path lengths assigned by Dijkstra's algorithm in the order the vertices are relaxed by Dijkstra's algorithm. Also, for paths in S = {s1,s2,….st}; from s0 to st is given by min{bw (v_i, v_{i+1}) | 0 ≤ $i < t$}.

Let O be a list of vertices with Optimal path with the highest minimum bandwidth.

Let v be the first vertex such that the estimate sv≠ov. Since O is optimal ov≤sv. Let C be the vertices prior to v upon which S and Oagree (this set might only include s).

Consider any path to v other than the one found by S.Any such path must start in C, leave C via another vertex u and then return to v.
Since u is connected to a vertex in C it must be a vertex for which we have an estimate. Since v is the vertex with minimum estimate, v must have a greater (or equal) estimate to u. So, estimated minimum bandwidth of v must be greater than or equal to estimated minimum bandwidth of u. Since, estimated bandwidth path is calculated considering all the edges, estimated minimum bandwidth with alternate route (v) will not increase estimated calculation done using path u. So, such alternate solutions doesn't exist.
Thus the optimal solution must agree with algorithm's estimate.

Question 04

Answer:

4(a) – Code is attached. Please see method '**MyersEditDistance'**

4 (b) – Code is attached. Please see method '**MillerEditDiatnce'**

Note: Above mentioned both methods are located at **\HW2\src\sample\Main.java**.

4(c) – Count of Comparisons performed by both algorithms:

Sr No.	# of Deletions	Edit Distance	E. Myers O(ND)	Wu, Manber, Myers, W. Miller O(NP)
1	10	1020	525730	15111
2	50	1100	610650	57551

3	100	1200	725800	115101
4	200	1400	986100	245201
5	400	1800	1626700	565401
6	600	2200	2427300	965601
7	800	2600	3387900	1445801
8	1000	3000	4508500	2006001

Question – 5

Here, a_i is the ith element of sequence A; and b_i is the ith element of sequence B. Once can receive payoff of $\sum_{i=1}^{n} a_i log b_i$. We need to design an algorithm which can give maximum payoff.

So, if we choose maximum value for both a_i and b_i, then we can get the maximum payoff.

Algorithm:

MaxPayOff(A[1,…n], B[1….n])
    Sort A in non decreasing order such that a_1 > a_2> a_3> a_4 > …. > a_n
    Sort B in non decreasing order such that b_1 > b_2> b_3> b_4 > …. > b_n
    Let maxA=a_i
    Let maxB=b_i
    Return $Payoff(\sum_{i=1}^{n} a_i log b_i)$
End MaxPayOff

Running time: If both A and B are already sorted than time complexity is **O(n).** And, if both A and B are not sorted, than we need to first sort them.Time complexity to sort will be **O(n log n).** So, total time taken is **O(nlog n + n).**
(As, summation and other operations can be done in O(n) time)

Proof:
Let's say, above solution is not optimal. And, solution S in an optimal solution. In S, a_1 is paired with b_t to get maximum payoff.
Now, consider another solution S'. In S', a_1 is paired with b_1. Other elements are same in both

Now, let's compare Payoff generated by both solutions

$\frac{Payoff\ (S)}{Payoff\ (s')} = \frac{\sum_{i=1}^{n} a_i log b_i}{\sum_{i=1}^{n} a_i log b_i} = \frac{a_1 \log b_t}{a_1\ \log b_1} = \frac{\log b_t}{\log b_1} \leq 1$ (as ($b_t < \ b_1$)).

This contradicts with the assumption that S is the optimal solution. Therefore, as mentioned in the algorithm above, sorting the A and B and then calculating payoff will maximize the payoff.

Ref: Discussed problems and possible approach with study group and fellow students. Also referred class slides, notes and textbooks/handbooks suggested as per syllabus.