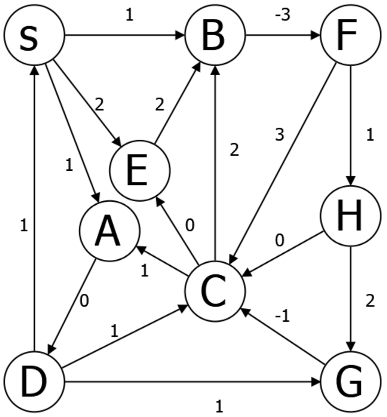
Question 1 (Shortest Paths, 30 points)**.** *Find the lengths of the shortest path from s to each other vertex in the graph below:*



We run Bellman-Ford on this graph and get the following output:

k	s	A	B	C	D	E	F	G	H
0	0	∞	∞	∞	∞	∞	∞	∞	∞
1	0	1	1	∞	∞	2	∞	∞	∞
2	0	1	1	∞	1	2	-2	∞	∞
3	0	1	1	1	1	2	-2	2	-1
4	0	1	1	-1	1	1	-2	1	-1
5	0	0	1	-1	1	-1	-2	1	-1
6	0	0	1	-1	0	-1	-2	1	-1
7	0	0	1	-1	0	-1	-2	1	-1

After this, things stabilize, so the correct distances are given by the last row.

Question 2 (Crossing Close Pair, 35 points)**.** *Give an algorithm that given two sets A and B each consisting of n real numbers finds the minimum distance between a point of A and a point of B. In particular, it computes the minimum possible value of $|a - b|$ over all pairs $a \in A$ and $b \in B$. For full credit, your algorithm should run in time $O(n \log(n))$.*

For example, if $A = \{0, 1, 8\}$ and $B = \{11, 3, 5\}$, then the closest pair would be 1 *and* 3 *with distance* 2*.*

The algorithm is the following:

```
Sort A U B keeping track of which point came from which set.
For each pair of consecutive elements x,y in the sorted list
  if x and y came from different sets, record |x-y|.
Return the smallest number recorded.
```

The runtime of the first step here is $O(n \log(n))$. The second and third steps are clearly linear time, so the full runtime is $O(n \log(n))$.

To show correctness, it is not hard to see that the closest such pair of points a, b will be consecutive points in the sorting of $A \cup B$. This is because otherwise there is some point in between them, either an $a' \in A$ or a $b' \in B$, and in that case either $|a' - b|$ or $|a - b'|$ will be less than $|a - b|$. Therefore, we only need to check all pairs of consecutive points, which we do.

Question 3 (Concert Tour, 35 points). *Sadie and her band are looking for gigs. There are k different venues willing to offer to let them play. Each venue has a schedule of performance openings for them, and each opening takes place at a specific time and will pay Sadie's band a specific amount of money. However, travelling between venues takes time and Sadie's band will not be able to play two different performances unless there is enough time to get from one venue to the other (they can play multiple performances at the same venue without any gap between them).*

Give an algorithm that given the schedule of n possible performances (along with times, payments and which venue it takes place in) along with the transit times between each pair of venues, computes the most amount of money that Sadie's band can make by performing. For full credit, your algorithm should run in time $O(kn\log(n))$.

For example, if venue A has performances at 9am (paying \$100*), 10am (paying* \$200*) and 12pm (paying* \$100*) and venue B has performances at 11am (paying* \$200*), and 4pm (paying* \$300*) and the two venues are 5 hours apart from each other, Sadie could play at 9am and 10am at venue A and at 4pm at venue B for a total of* \$600.

The algorithm is as follows:

```
Sort events at each location by time.
Create a directed graph G whose vertices are the events.
For each v in G, create edges to the earliest event at each location that can be reached from v.
  (You can find this using binary search)
Add vertices s and t and weight 0 edges from s to each other vertex and from each vertex to t.
Assign each edge a weight equal to minus the payout of the gig at the end of it,
  and edges ending in t get value 0.
Compute the shortest path from s to t using the shortest paths in DAGs algorithm
Return minus the length of this path.
```

To analyze the runtime, we can sort the events at each location in time $O(n\log(n))$ time. The graph G has $O(n)$ vertices and $O(nk)$ edges (k for each gig plus $O(n)$ from s and t). Each edge can be computed in $O(\log(n))$ time by using binary search to find the first event at the location whose time is at least the time of the previous gig plus transportation time. The shortest path in DAGs algorithm them runs in time $O(|V|+|E|)=O(kn)$ time.

To show correctness, we first note that G is a DAG. This is because each edge connects a gig only to other gigs that take place at later times. Next, we note firstly that every $s-t$ path in G corresponds to a schedule of gigs that Sadie's band can make. This is because each edge connects a gig to other gigs that can be reached from it. Thus, every $s-t$ path corresponds to the negative of a payout that is achievable from some valid schedule. We claim that the converse is nearly true. In particular, for every valid schedule, there is an $s-t$ path in G corresponding to a schedule that is at least as good. In particular, if this schedule visits gigs $g_1, g_2, \ldots, g_m$, the sequence $s, g_1, g_2, \ldots, g_m, t$ might not be a path in G but only if for some i g_{i+1} is later than the next gig at its location that can be reached from g_i. In this case, we can add extra gigs at g_{i+1}'s location between g_i and g_{i+1} and come up with a better schedule. This means that the best schedule will necessarily correspond to a path in G whose total weight will be negative that schedule's profit. Thus, the best schedule for the band corresponds to a shortest $s-t$ path in G with the total profit being the negative of the path length.

Note: It is actually possible to achieve runtime $O(kn+n\log(n))$ by using a more efficient algorithm to compute the edges for G. For any pair of locations ℓ_1 and ℓ_2 with transportation time t and with sets of gigs A and B (each sorted in increasing order of start time), we can compute all edges between A and B in the following way.

```
a = 1, b = 1
While(a < Length(A))
  If Time(B[b]) >= Time(A[a])+t
    Create edge from A[a] to B[b]
    a++
  Else
    b++
```

This correctly computes the edges in $O(|A| + |B|)$ time. Running this over all pairs of locations, we can compute all edges of G in $O(kn)$ time.