Ashlyn Cooper
CPSC 3120 Homework 3

Question 1

Dijkstra's algorithm is as follows:

- We assume that there is a path from the source vertex to every other vertex in the graph. We let the distance of the start vertex from the start vertex = 0. Then we let the distance of all other vertices from start be equal to infinity. Then we iteratively repeat through the following steps: 1) Visit the unvisited vertex with the smallest known distance from the source. 2) examine the unvisited neighbors of the current vertex. 3) For the current vertex calculate the distance of each neighbor from the source. 4) If the calculated distance is less than the known distance, we update the shortest distance and update the previous vertex for each updated distance. 5) We add the current vertex to a list of visited vertices. We do this repetition until all vertices are visited and we have a shortest path.

For this graph the steps to find the Shortest Path from Node 1 to Node 6 are:

1. Visit Node 1, the shortest distance from Node 1 to Node 1 is 0. All other distances are unknown, so we set them to infinity

Vertex	**Shortest Distance from Node 1**	**Previous Vertex**
1	0	
2	infinity	
3	infinity	
4	infinity	
5	infinity	
6	infinity	
7	infinity	
8	infinity	
9	infinity	
10	infinity	
11	infinity	

2. To fully start the algorithm, we visit the vertex with the lowest know distance, Node 1. Then we examine the neighbors of Node 1. Its unvisited neighbors are 2, 7, 8, 9, 10, 11. We then calculate the distance from each of these neighbors and if that distance is less than the known distance, we reset the known distance to be the new calculated distance.

Vertex	**Shortest Distance from Node 1**	**Previous Vertex**
1	0	
2	8	1
3	infinity	
4	infinity	

5	infinity	
6	infinity	
7	9	1
8	10	1
9	6	1
10	12	1
11	3	1

3. Go to next unvisited vertex with smallest know distance from Node 1. This is Node 11. Go to Node 11 and do same distance calculation and resetting as before with its neighbors, Nodes 2. 10, and 3.

Vertex	**Shortest Distance from Node 1**	**Previous Vertex**
1	0	
2	8	1
3	8	11
4	infinity	
5	infinity	
6	infinity	
7	9	1
8	10	1
9	6	1
10	11	11
11	3	1

4. Go to next unvisited vertex with shortest known distance from Node 1, which is Node 9. Go to Node 9 and do same distance calculation and resetting as before with its neighbors, Nodes 8 and 10.

Vertex	**Shortest Distance from Node 1**	**Previous Vertex**
1	0	
2	8	1
3	8	11
4	infinity	
5	infinity	
6	infinity	
7	9	1
8	9	9
9	6	1
10	11	11
11	3	1

5. Go to next unvisited vertex with shortest known distance from Node 1, which is Node 2. Go to Node 2 and do same distance calculation and resetting as before with its neighbors, Nodes 5 and 3.

Vertex	Shortest Distance from Node 1	Previous Vertex
1	0	
2	8	1
3	8	11
4	infinity	
5	10	2
6	infinity	
7	9	1
8	9	9
9	6	1
10	11	11
11	3	1

6. Go to next unvisited vertex with shortest known distance from Node 1, which is Node 3. Go to Node 3 and do same distance calculation and resetting as before with its unvisited neighbor Node 4.

Vertex	Shortest Distance from Node 1	Previous Vertex
1	0	
2	8	1
3	8	11
4	17	3
5	10	2
6	infinity	
7	9	1
8	9	9
9	6	1
10	11	11
11	3	1

7. Go to next unvisited vertex with shortest known distance from Node 1, which is Node 7. Go to Node 7 and do same distance calculation and resetting as before with its unvisited neighbors Node 8, 5, and 6.

Vertex	Shortest Distance from Node 1	Previous Vertex
1	0	
2	8	1
3	8	11
4	17	3
5	10	2
6	17	7
7	9	1
8	9	9

9	6	1
10	11	11
11	3	1

8. Go to next shortest unvisited Node which is Node 8. Go to Node 8’s unvisited neighbors. It has no unvisited neighbors, so the distances will not change.

Vertex	**Shortest Distance from Node 1**	**Previous Vertex**
1	0	
2	8	1
3	8	11
4	17	3
5	10	2
6	17	7
7	9	1
8	9	9
9	6	1
10	11	11
11	3	1

9. Go to next shortest unvisited Node, which is Node 5. Go to node 5’s unvisited neighbors, which are 6 and 4 and recalculate distances.

Vertex	**Shortest Distance from Node 1**	**Previous Vertex**
1	0	
2	8	1
3	8	11
4	17	3
5	10	2
6	17	7
7	9	1
8	9	9
9	6	1
10	11	11
11	3	1

10. Go to next shortest unvisited node, which is Node 10. Go to Node 10’s unvisited neighbor. It has none so the distance values will not change.

Vertex	**Shortest Distance from Node 1**	**Previous Vertex**
1	0	
2	8	1
3	8	11
4	17	3
5	10	2

6	17	7
7	9	1
8	9	9
9	6	1
10	11	11
11	3	1

11. Go to next shortest unvisited node, Node 4. Node 4 has 1 unvisited vertex, Node 6. Recalculate distance values with Node 6.

Vertex	**Shortest Distance from Node 1**	**Previous Vertex**
1	0	
2	8	1
3	8	11
4	17	3
5	10	2
6	17	7
7	9	1
8	9	9
9	6	1
10	11	11
11	3	1

12. Go to next shortest unvisited node, Node 6. Node 6 has no unvisited neighbors so there is nothing left to do. We have visited all nodes and can now put together out shortest path, tracing from Node 6 back to Node 1.

Vertex	**Shortest Distance from Node 1**	**Previous Vertex**
1	0	
2	8	1
3	8	11
4	17	3
5	10	2
6	17	7
7	9	1
8	9	9
9	6	1
10	11	11
11	3	1

Answer: The shortest path from Node 1 to Node 6 is 17 units long and goes from 1 – 7 – 6.

One conclusion that we can draw from this process, is that if we are using Dijkstra's algorithm to visit a specific target, we can stop the algorithm once we are using it as a picked minimum vertex. In my own workings, I did not pick node 6 before node 4 when they had the same distance, but had I picked node 6 first there would have been no need to visit node 4. This could limit the amount of time the algorithm takes, as it normally is used to find a general shortest distance to all vertices, not just a specific one.

Question 2

The weighted Interval Scheduling problem consists of the following:

- We have S = {1,2,…,n} which is the set of all activities that compete for a resource (time in this case). Each activity *i* has designated starting time, *s,* finishing time, *f,* and weight *v.* The goal of the problem is to find the maximum weight of mutually compatible activities. We say that two activities are compatible if their time periods are disjoint, i.e. they do not overlap. So, we want to select the largest set by weight of activities of compatible activities within a given time.

To solve the problem using dynamic programming we take a few steps:

- First, we sort our given arbitrary array in order of increasing finishing time so that $f_1 \leq f_2 \leq \cdots \leq f_n$. The sorting of this array will take O(nlogn) time, if we use an optimized sorting algorithm such as a merge, quick, or heap sort.
- Then we say that p(j) is the largest index i < j such that activity i is compatible with j from our given arbitrary array. We compute p(j) for all jobs in the given array. This takes O(nlogn) time via sorting by start time.
- Then we say that OPT(j) is the value of the optimal solution to the problem consisting of activities 1,2,…,j.
- Set OPT[0] = 0. The complexity at this step if O(1) because it is a singular operation.
- For each activity, j from 1 to n, set OPT[j] equal to the max of $v_j + OPT[p(j)]$ and $OPT[j-1]$. The complexity of this step is O(n) because we are only looping through all activities once.
- The value of our optimal solution to our problem is OPT[n].
- To find the actual solution with the previously determined optimal value, we make a second pass on OPT[] using a second function called FindSolution(j).
    - In FindSolution(j) if j = 0, we output nothing. If $v_j + OPT[p(j)] > OPT[j-1]$ we print j as one part of our solution and then call FindSolution on p(j). Otherwise, we call FindSolution on j-1. The number of recursive calls is $\leq$ n so the complexity is O(n).

Question 3

After sorting the given array by finish times, we have this table that shows us the values, v, at the new sorted locations and p(i).

Index, i	**v of i**	**p(i)**
1	2	0
2	4	0
3	4	1
4	7	0
5	2	3
6	1	4

This below table shows the changes in our array of OPT[].

Index ->	**0**	**1**	**2**	**3**	**4**	**5**	**6**
	0	2					
	0	2	4				
	0	2	4	6			
	0	2	4	6	7		
	0	2	4	6	7	8	
	0	2	4	6	7	8	8

The optimal value of our solution is 8. The items in the solution are the activities at indices 1, 3, and 5 in our array that is sorted by finish time or the items with starting times [1, 3.5, 7] and finishing times [3, 6, 12] respectively.