EE 367 Spring 2018
Homework 11 Total Points = 6
Submit hard copy in class for all problems except the last, where you upload the program bf.c

Problem 26.1-2 (CLRS p. 713, 1 pt)

Problem 26.2-6 (CLRS p. 730, 1 pt)

Problem 26.3-3 (CLRS p. 735, 1 pt)

Problem 26-1 (CLRS p. 760, 1 pt) (a), but don't do part (b) This is the Escape problem.

Problem A (1 pt). Consider Problem 4 of Midterm Exam 2. Recall that there is a file subseq.c is a program that will **find the longest increasing subsequence** of an array of integers val[ ]. It has a function 'subseq(val[ ], n)' which will return the length of the longest increasing subsequence and display the subsequence itself. The input values are in the array val[ ], and the array has length n. For example, if the randomly permuted array val[ ] was 0, 8, 4, 12, 2, 10, 6, 14, 1, 9, then subseq will return 4, which is the length of the longest increasing sequence, and display 0, 4, 6, 14 (Note that there are other longest increasing subsequences, and any of them could be displayed).

The following is a solution to the function 'subseq( )':

```
void subseq(int val[], int length)
{

int * dist = (int *) malloc(sizeof(int)*length);
int * pred = (int *) malloc(sizeof(int)*length);
int * path = (int *) malloc(sizeof(int)*length);

for (int i=0; i<length; i++) {
      dist[i] = 1;
      pred[i] = -1;  /* Indicates no predecessor */
}
for (int i=0; i<length; i++) {
      for (int j=0; j<i; j++) {
            if (val[j] < val[i]) {
                  if (dist[i] <= dist[j] + 1) {
                        dist[i] = dist[j]+1;
                        pred[i] = j;
                  }
            }
      }
}

int best_dist = 0;
int best_index = -1;
for (int i=0; i<length; i++) {
      if (best_dist < dist[i]) {
            best_dist = dist[i];
            best_index = i;
      }
}

printf("Length of the longest increasing subsequence: %d\n", best_dist);

printf("Longest increasing subsequence:\n");
int m = 0;
for (int i=best_index; i>-1; i=pred[i]) {
      path[m]=val[i];
      m++;
}
for (int i=m-1; i>=0; i--) {
      printf("%d ",path[i]);
}
printf("\n");

return;
}
```

It's time complexity is $O(n^2)$.

This algorithm can be improved so that it has a lower time complexity. Write an algorithm in pseudocode that finds the longest increasing subsequence and has time complexity $O(n \log n)$. Assume that the input to the algorithm is val[ ] and *n*. Explain why your algorithm is correct and its time complexity.

(Hint:

- Use the ideas in Chapter 14 Augmenting Data Structures.
- Recall that pseudo code doesn't require you to write out all the code. For example, in an algorithm, you can explain that you use a priority queue implemented by a heap; and that operations of insertion and removal are O(log n). Another example, is in the case of Kruskal's algorithm, where we state that we use mergesort to sort the edges. We don't explicitly write out mergesort.

)

Problem B (1 pt). (This is still under construction).
Attached is a program "bf.c" which computes shortest paths in a weighted undirected graph G. To run the program, do the following:

./a.out <input file>

The input file has the following format:

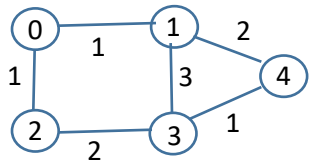
N
M
X0 Y0 W0
X1 Y1 W1
...

where N is the number of nodes, which are assumed to be 0, 1, ..., N-1; M is the number of undirected links; (X0, Y0) is the first link with weight W0; (X1, Y1) is the second link with weight W1; and so on. All the values are integer.

The program bf.c will display the shortest paths from the source, which is assumed to be 0, to all the other nodes. Thus, if N = 10, the program will display 9 shortest paths from source node 0 to the other nodes 1, 2, ..., 9.

For example, consider the following input file for the graph below:

```
5
6
0 1 1
0 2 1
1 3 3
2 3 2
3 4 1
1 4 2
```



Then function bf will compute the shortest paths and display them, e.g.,

Shortest paths:
0 -> 1
0 -> 2
0 -> 2 -> 3
0 -> 1 -> 4

There is a function bf(adjlist, n) in bf.c which computes the shortest paths using the Bellman Ford algorithm (note 'bf' means Bellman Ford). The function also displays the shortest paths. The input parameter 'adjlist' is the adjacency list of the graph and its link weights, and n is the number of nodes in the graph.

Currently bf( ) doesn't work, so you must make the changes so bf.c works properly. Upload your version of 'bf.c' in laulima. (Note: Also attached is bf.dat which is an example input file.)