1. (15 points) Suppose you’re implementing a graph algorithm that uses a heap as one of its primary data structures. The algorithm does at least $n$ insert and $n$ deletemin operations and at most $m^{2/3}$ insert and $m^{2/3}$ deletemin operations, when $m^{2/3} > n$. It also does at most $m$ changekey operations, all of which reduce the key value. If you implemented this algorithm using $d$-heaps, what value of $d$ would you use to get the lowest asymptotic running time. (Hint: break this into two cases; one for smaller values of $m$, one for larger values.)

What is the resulting asymptotic running time?

What is the resulting asymptotic running time if you used a Fibonacci heap, instead of a $d$-heap?

Which type of heap would you choose for this application? Why?
2. (15 points) In the Fibonacci heaps data structure, a cut between a vertex \( u \) and its parent \( v \) causes a cascading cut at \( v \) if \( v \) has already lost a child since it last became a child of some other vertex. Suppose we change this, so that a cascading cut is performed at \( v \) only if \( v \) has already lost two children. How does this change alter the lemma shown below (this lemma is from the analysis of the running time of Fibonacci heaps)? Explain your answer.

**Lemma.** Let \( x \) be any node in an F-heap. Let \( y_1, \ldots, y_r \) be the children of \( x \), in order of time in which they were linked to \( x \) (earliest to latest). Then, \( \text{rank}(y_i) \geq i - 2 \) for all \( i \).

Let \( S_k \) be the smallest possible number of descendants that a node of rank \( k \) has, in our modified version of Fibonacci heaps. Give a recursive lower bound on \( S_k \). That is, give an inequality of the form \( S_k \geq f(S_0, S_1, \ldots, S_{k-1}) \) where \( f \) is some function of the \( S_i \)'s for \( i < k \).

Use this to give a lower bound on the smallest number of descendants that a node with rank 7 can have.
3. (15 points) Suppose you are asked to write a program that responds to a series of queries about a given graph of the form “Will the graph have a negative cycle, if the weight of edge $(u,v)$ is changed to $x$?” You may assume that the graph has no negative cycles, although it does have negative edges. Describe (in words) an algorithm that responds to such queries in $O(m+n \log n)$ time. Your algorithm may pre-compute certain information before the first query is received, but the amount of additional information is limited to $O(n)$ new values. How much time does your algorithm need to initialize this extra information?

Show the auxiliary information your method would compute for the graph shown below.
4. (10 points). The figure below shows an intermediate state in the execution of Dijkstra’s algorithm. The bold edges in the graph are the edges defined by the parent pointers, and the numbers next to the vertices are the current distance values. Fill in the blanks (as appropriate) in the arrays that implement the $d$-heap (assume $d=2$).

Show how the heap content changes after the next iteration.
5. (15 points) Consider a binary search tree in which each vertex has an associated key and a cost. The vertices are ordered by the keys in the usual way (so the keys of the vertices in the left subtree of a given vertex \( x \) are strictly less than the key of \( x \), and so forth). The costs are represented using the differential representation we used for representing path sets. Complete the recursive function `search`, shown below, so that it returns the node with the smallest key from among those vertices with costs less than or equal to a given bound. The structure of the tree nodes is shown below also.

```c
class twoWayTrees {
    int n;       // trees defined on items {1,...,n}
    struct node {
        int k, Dc, Dm;   // key and differential cost fields
        int lc, rc;     // indices of left and right children
    } *vec;
    ...
};
#define left(x) (vec[x].lc) // you may assume similar declarations
// for right, key, Dcost, Dmin
int search(int t, int costBound, int dmSum) {
    // Return the index of the leftmost node in the subtree with
    // root t that has cost <= costBound. The variable dmSum is
    // the sum of the Dmin values for the proper ancestors of t.
    // Return Null, if there is no node with cost less than costbound.
```
6. (15 points) In this problem, you are to show that the general preflow-push algorithm takes $O(mn)$ time to find a maximum flow in a graph in which all edges have capacity 1. (Recall that the bound for general graphs is $O(mn^2)$.) How many steps add flow to an edge in the residual graph without saturating it? Why?

Explain why the time spent on relabeling steps is $O(mn)$.

Explain why the time spent on steps that saturate an edge in the residual graph is $O(mn)$.

Explain why the time spent finding admissible edges (using the $nextedge$ pointers) is $O(mn)$.
7. (15 points) Given a bipartite graph with edge weights.

Draw a picture of the min-cost flow graph that can be used to find a maximum matching in this graph.
Identify a minimum cost augmenting path in the flow graph. What is its cost? Draw the residual graph that results from saturating this path.

Identify a minimum cost augmenting path in the residual graph. What is its cost?

What is the matching corresponding to the flow that exists after flow is added to this augmenting path?
8. (10 points) The figure below show a possible intermediate stage in the execution of the Edmonds-Karp algorithm for finding a maximum size matching. The states of the vertices are indicated by the plus and minus signs (for even and odd), the arrows represent the parent pointers and the edges adjacent to certain vertices correspond to their bridge values. The partition data structure is shown at right and the queue of edges to be processed is at the top. In the diagram at left, draw a closed curve around the sets of vertices that have been condensed into a single vertex in the current shrunken graph. If any of these sets contain subsets, corresponding to smaller blossoms, circle the vertices in those smaller blossoms, as well.

queue: ae, ih, la, kd, ln

![Graph Diagram]

What happens when the next edge in the queue is processed by the algorithm? Explain.

When the next edge, [i,h] is processed, an augmenting path is found. What is that augmenting path?