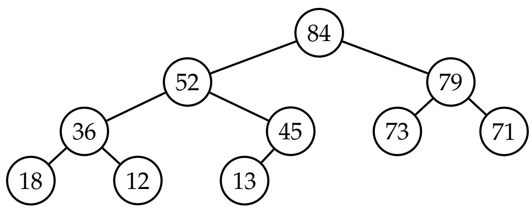
Last name(s) **Name** **ID**

Final EDA Exam **Length: 3 hours** **09/06/2017**

- *The exam has 4 sheets, 8 sides and 4 problems.*
- *Write your full name and ID on every sheet.*
- *Write your answers within the reserved space in the exam sheet.*
- *Unless otherwise specified, your answers must be justified.*

Problem 1 **(2 pts.)**

(a) (0.5 pts.) Given the following max-heap:



draw the max-heap resulting from adding 55 and deleting the maximum element (in this order). You do not need to justify anything.

(b) (0.5 pts.) Using that $P \subseteq NP$, prove that $P \subseteq co\text{-}NP$.

(c) (0.5 pts.) Consider the following function:

```
int f(const vector<int>& v, int i, int j) {
  // Precondition: 0 ≤ i ≤ j < v.size()
  if (i == j) return v[i];
  int p = (j − i + 1)/3;
  int m1 = i+p;
  int m2 = m1+p;
  return f(v, i, m1) + f(v, m1+1, m2) + f(v, m2+1, j);
}
```

Compute the asymptotic cost in time of f as a function of $n = j - i + 1$.

(d) (0.5 pts.) Consider the following statement:

The function n^n grows asymptotically faster than any other function.

Is it true? If so, justify it. Otherwise give a counterexample.

Last name(s) **Name** **ID**

Problem 2 **(2 pts.)**

Given a vector A with n different integers and another vector B with m different integers, we want to compute a vector (of different integers) which is the intersection of the two (that is, that contains the common elements).

For example, if $A = (3, 1, 6, 0)$ and $B = (4, 6, 1, 2, 7)$, then any of the two vectors $(1, 6)$, $(6, 1)$ would be a valid answer.

Let us assume that A and B **are not necessarily** sorted. Give a high-level description of how you would implement a function

vector <**int**> *intersection* (**const** *vector* <**int**>& A, **const** *vector*<**int**>& B);

that returns the intersection of A and B with a cost in time of $O(n + m)$ **in the average case**. Justify the cost.

This side would be intentionally blank if it were not for this note.

Last name(s)	Name	ID

Problem 3 **(3 pts.)**

Given a directed acyclic graph (DAG) G, the *level* of its vertices is defined inductively as follows:

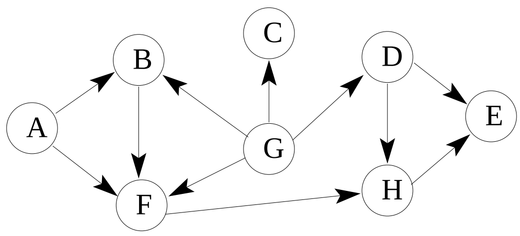
- if v is a root of G (a vertex without predecessors) then $\text{level}(v) = 0$
- otherwise,

$$\text{level}(v) = 1 + \max\{\text{level}(u) \mid u \text{ is a predecessor of } v\}$$

Moreover, the *depth* of G is the largest level of any vertex:

$$\text{depth}(G) = \max\{\text{level}(v) \mid v \text{ vertex of } G\}$$

(a) (0.9 pts.) Fill out the following table pointing out, for each vertex of the given DAG, its level. What is the depth of the DAG? You do not need to justify anything.



level :

A	B	C	D	E	F	G	H

depth : $\square$

(b) (0.4 pts.) For each of the following statements, mark with an X the corresponding cell depending on whether it is true or false. You do not need to justify anything.

Note: Every right answer will add 0.1 points; every wrong answer will subtract 0.1 points, except when there are more wrong answers than right ones, in which case the grade of the exercise will be 0.

(1) For any vertex u of a DAG G, if u is a leaf (vertex without successors) then $\text{level}(u) = \text{depth}(G)$.

(2) For any vertex u of a DAG G, if $\text{level}(u) = \text{depth}(G)$ then u is a leaf.

(3) The depth of a DAG with n vertices is $O(n)$.

(4) The depth of a DAG with n vertices is $\Omega(\log n)$.

	(1)	(2)	(3)	(4)
TRUE				
FALSE				

(c) (1.7 pts.) Here it is assumed that graphs are represented with adjacency lists, and that vertices are identified with consecutive natural numbers 0, 1, etc.

Fill the gaps of the following function:

```
vector<int> levels(const vector<vector<int>>& G);
```

which, given a DAG $G = (V, E)$, returns a vector that, for each vertex $u \in V$, contains the value level(u) in position u. Give and justify the cost in time in the worst case in terms of $n = |V|$ and $m = |E|$.

```
vector<int> levels(const vector<vector<int>>& G) {
  int n = G.size();
  vector<int> lvl(n, −1), pred(n, 0);

  for (int u = 0; u < n; ++u)
    for (int v : G[u])
      ++pred[v];

  queue<int> Q;
  for (int u = 0; u < n; ++u)
    if (pred[u] == 0) {
      Q.push(u);
      [                              ]
    }

  while (not Q.empty()) {
    int u = Q.front(); Q.pop();
    for (int v : G[u]) {
      −−pred[v];
      [                              ]
      if (pred[v] == 0) Q.push(v);
  } }
  return lvl;
}
```

Cost and justification:

Last name(s)	Name	ID

Problem 4 **(3 pts.)**

Given $n > 0$, a *derangement* (of size n) is a permutation of $\{0, \ldots, n-1\}$ without fixed points; that is, π is a derangement if and only if $\pi(i) \neq i$ for all i, $0 \leq i < n$. For example $\pi = (2, 0, 3, 1)$ is a derangement, but $\pi' = (1, 3, 2, 0)$ is not (since $\pi'(2) = 2$).

Fill out the following C++ code for generating all derangements of size n. You do not need to justify anything.

```
void write(const vector<int>& p, int n) {
  for (int k = 0; k < n; ++k) cout << " " << p[k];
  cout << endl;
}

void generate(int k, int n, vector<int>& p, vector<bool>& used) {
      if (                              ) write(p, n);
      else {
        for (int i = 0; i < n; ++i)
          if (                              ) {
            
            
            generate(k+1, n, p, used);
            
          }
      }
};

void generate_all (int n) {
  vector<int> p(n);
  vector<bool> used(n, false);
  generate (                              , n, p, used);
}
```

This side would be intentionally blank if it were not for this note.