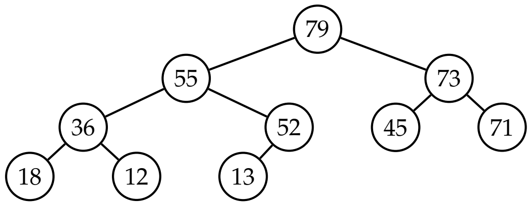
Solution of the Final EDA Exam — 09/06/2017

Proposed solution to problem 1

(a) The resulting max-heap is:



(b) To prove that P $\subseteq$ co-NP we only need to see that if L is a problem from class P, then its complement $\overline{L}$ belongs to NP. But if L is from P then there is a deterministic polynomial algorithm A that decides L. Let us consider now the algorithm $\overline{A}$ that does the same as A, but returns 1 when A returns 0, and returns 1 when A returns 0. Then $\overline{A}$ decides $\overline{L}$, and since $\overline{A}$ takes polynomial time, we have $\overline{L} \in \text{P} \subseteq \text{NP}$.

(c) The cost $C(n)$ of f in function of n follows the recurrence

$$C(n) = 3C(n/3) + \Theta(1)$$

as there are 3 recursive calls on subvectors of size $n/3$ and additionally operations of constant cost are performed. By the Master Theorem of Divisive Recurrences, the solution to the recurrence is $C(n) = \Theta(n)$.

(d) It is not true. For example, $n^{2n} = (n^n)^2$ grows asymptotically faster than n^n.

Proposed solution to problem 2

We must use a dictionary with integer keys implemented with a hash table, and a vector which is initially empty that will contain the intersection.
First of all we pass over A and add all its elements to the dictionary. We make n insertions to the dictionary, each of which takes time $O(1)$ on average. So the first pass costs $O(n)$ on average.
In the second place we pass over B and, for each of its elements, we check if it already belongs to the dictionary. If so, the element is added to the vector of the intersection with a *push_back*. Otherwise nothing is done. Hence we make m lookups to the dictionary, each of which takes time $O(1)$ on average. Since each *push_back* takes constant time, the cost of the second pass is $O(m)$ on average.
In total, the cost is $O(n + m)$ on average.

Proposed solution to problem 3

(a) The filled table is:

level :

A	B	C	D	E	F	G	H
0	1	1	1	4	2	0	3

depth : $\boxed{4}$

(b) The filled table is:

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

(c) A possible solution:

```
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);
      lvl[u] = 0;
    }

  while (not Q.empty()) {
    int u = Q.front(); Q.pop();
    for (int v : G[u]) {
      --pred[v];
      lvl[v] = max(lvl[v], lvl[u]+1);
      if (pred[v] == 0) Q.push(v);
  } }
  return lvl;
}
```

The construction of the vectors has cost $\Theta(n)$. The first loop has cost $\Theta(n+m)$. The second loop has cost $\Theta(n)$. The third loop has cost $\Theta(n+m)$. In total the cost is $\Theta(n+m)$.

Proposed solution to problem 4

```
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 (k == n) write(p, n);
      else {
```

```
            for (int i = 0; i < n; ++i)
                if (not used[i] and k ≠ i) {
                    used[i] = true;
                    p[k] = i;
                    generate(k+1, n, p, used);
                    used[i] = false;
            }
        }
};

void generate_all (int n) {
    vector<int> p(n);
    vector<bool> used(n, false);
    generate (0, n, p, used);
}
```