Sample Final. 2020-12-16.

This sample (unlike the actual exam) is NOT scaled to fit in 120 minutes.

Summary. This is a sample exam, it only gives an approximate idea regarding the question types and their difficulty. The actual exam may cover some other topics (it may also include things taught before the midterm exam).

Final exam emphasizes certain things that are important for further IT subjects (such as Time Complexity, Graph Algorithms) and also the knowledge used in practical programming (using standard libraries and ensuring integrity of data structures and their representation invariants).

Question 1 (Time Complexity).

(A) We need a sorting algorithm that would order all pairs of integers (x, y) by the first number x (and by the second number y, if the first numbers in the pair are equal). For example $(1, 100) < (2, 10) < (2, 20) < \ldots$. Which comparison function can be passed as a parameter to the sorting method `sort(...)` on Line 38.

(B) As we know, we can define various ways how compare objects. Are there any mathematical properties that a comparison algorithm `Comp(...)` should satisfy to be passed as a parameter to a sorting algorithm like the one above? (Please list just the most general properties that do not depend on the object types to be sorted and the particular ordering that you need.)

(C) What is the time complexity of the sorting algorithm shown on Line 18–32 of the code? Express your answer as $O(g(n))$, where n is the length of the input array.
(Optionally, you can also name the algorithm, if you recognize it.)

```
 1  struct Pair { int x; int y;};
 2
 3  bool CompA(Pair p1, Pair p2) {
 4    return (p1.x <= p2.x) ||
 5      (p1.y < p2.y);
 6  }
 7
 8  bool CompB(Pair p1, Pair p2) {
 9    return (p1.x < p2.x) ||
10      (p1.y < p2.y);
11  }
12
13  bool CompC(Pair p1, Pair p2) {
14    return (p1.x < p2.x) ||
15      (p1.x == p2.x && p1.y < p2.y);
16  }
17
18  void sort(Pair arr[], int n,
19      bool (*f)(Pair, Pair)) {
20    int i, j;
21    Pair key;
22    for (i = 1; i < n; i++) {
23      key = arr[i];
24      j = i - 1;
25      while (j >= 0 &&
26          (*f)(key, arr[j])) {
27        arr[j + 1] = arr[j];
28        j = j - 1;
29      }
30      arr[j + 1] = key;
31    }
32  }
33
34  int main() {
35    Pair arr[] = {{12,2},
36      {6,1},{13,1},{5,100},{6,1}};
37    int n=sizeof(arr)/sizeof(arr[0]);
38    sort(arr,n,CompA);
39  }
```

Question 2 (Hashing).
Assume that you have to create an unordered set of very large integers. You want to store them in a hashtable with exactly 10 buckets. You can use either of the following two hash functions:

$$h_1(n) = (17 \cdot n) \bmod 10,$$

$$h_2(n) = n^4 \bmod 10.$$

Assume that the numbers n_i that are stored in the unordered set (and serve as inputs to the hash function h_1 and h_2) are uniformly distributed in the interval $[1; 10^{100}]$ and you insert a large number of values.

(A) How many hash buckets will receive values in case of $h_1(n)$? In case of $h_2(n)$?

(B) Which hash buckets will receive most of the values? For both $h_1(n)$ and $h_2(n)$ specify, which hash bucket is the fullest and specify the percentage of inserted values that will arrive to that bucket. (If several hash buckets are expected to be equally full, you can specify any one of them.)

(C) Which hash function would be more efficient (ability to find set elements faster)?

Question 3 (Representation Invariants).
(*Note.* For most data structures covered in this course we defined the representation invariants: Properties that need to be preserved as we manipulate and modify the data. Programmers may inadvertently break the representation invariants leading to unpredictable behavior of some built-in data structures and standard libraries.)

```
#include <iostream>
#include <vector>
#include <map>
#include <string>

using namespace std;
int main()
{
    vector<int> alice{1, 2, 4};
    vector<int> bob{7, 8, 9, 10};
    vector<int> eve{1, 2, 3};

    map<vector<int>,string> m;
    m.insert({alice, "alice"});
    m.insert({bob, "bob"});
    m.insert({eve, "eve"});
    cout<<"size="<<m.size()<<endl;
    cout<<alice.at(2)<<endl;
    eve.at(2) = 5;
    cout<<alice.at(2)<<endl;
    cout<<(m.at(alice))<<endl;
    cout<<(m.at(eve))<<endl;
    cout<<"size2="<<m.size()<<endl;
}
```

(A) What operations should be supported by the datatype `vector` so that it can serve as keys in a map `m`?

(B) What causes the program to crash?

Question 4 (Graph theory task).
Definition. An undirected graph is called *bipartite*, iff the set of vertices V can be divided into two disjoint sets V_1 and V_2 so that every edge connects a vertex in V_1 to a vertex in V_2 (and there are no edges from V_1 to V_1 or from V_2 to V_2).
Equivalently, a bipartite graph is a graph that does not contain any odd-length cycles. See `https://bit.ly/2JZrvek`.

There is an (undirected) graph with vertices $\{A, B, C, D, E, F, G, H\}$ represented by the following adjacency matrix:

$$\begin{array}{c|cccccccc} & A & B & C & D & E & F & G & H \\ \hline A & - & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ B & 0 & - & 0 & 0 & 0 & 1 & 1 & 0 \\ C & 0 & 0 & - & 0 & 1 & 0 & 0 & 1 \\ D & 0 & 0 & 0 & - & 0 & 0 & 1 & 0 \\ E & 0 & 0 & 1 & 0 & - & 1 & 0 & 1 \\ F & 1 & 1 & 0 & 0 & 1 & - & 0 & 0 \\ G & 0 & 1 & 0 & 1 & 0 & 0 & - & 0 \\ H & 1 & 0 & 1 & 0 & 1 & 0 & 0 & - \end{array}$$

(A) Draw the visual representation of this graph (vertices are circles labeled with letters $A \ldots H$); edges are segments connecting these vertices.
(B) Run the BFS traversal on this graph starting in vertex A. Order the vertices by their level in the BFS tree. In every level visit the child vertices alphabetically. Mark the BFS tree edges bold (or in different color), and also draw the cross edges (not bold or dashed).
(C) Check, if the graph is bipartite. You can use the levels and discovery/cross edges created during the BFS traversal, if necesssary.

Question 5 (Sequences, MST, etc.).
Consider the following subset of airports and travel times.

Origin	**Destination**	**Travel Time**
Riga	Frankfurt	120 min
Frankfurt	Riga	120 min
Frankfurt	London	60 min
London	Frankfurt	60 min
London	Edinburgh	45 min
Edinburgh	London	45 min
London	Paris	90 min
Paris	London	90 min
Paris	Frankfurt	120 min
Frankfurt	Paris	120 min

Passengers are divided into three groups E, F, B (Economy class, First class and Business class respectively). These enter and exit the plane in this order: first B, then F, then E. Within each group passengers enter and exit in the last-in-first-out manner.
Consider the following example. 5 passengers arrive to the check-in desk of a flight in the following order:

$$E_1, B_1, E_2, F_1, F_2.$$

In this case they are boarded in the plane like this:

$$B_1, F_1, F_2, E_1, E_2.$$

They exit the plane like this:

$$B_1, F_2, F_1, E_2, E_1.$$

If several passengers have flight with several connections, assume that they check in at the next flight in the same order in which they exited the previous one.

(A) Create an undirected graph G representing all the
Find all the minimum spanning trees of G. If the length of the flight is the weight of each edge, find the minimum spanning tree with least weight.

(B) Consider the following order how passengers arrive in Riga check-in desk:

$$B_1, B_2, F_1, E_1, F_2, B_3, E_2, F_3, E_3.$$

They all go to Edinburgh (and change flights at Frankfurt and London). London to Edinburgh flight has no First class (Business and

First class passengers are boarded together in whichever order they arrive at the check-in desk.)
Write the passenger queues for the following:

1. Boarding order in Riga
2. Exit order in Frankfurt
3. Boarding order in Frankfurt
4. Exit order in London
5. Boarding order in London
6. Exit order in Edinburgh

(C) Using Stack and/or Queue ADT methods (push, pop, enqueue, dequeue) write pseudocode to get the exit order in Frankfurt (from the original order how the passengers arrive at the check-in desk in Riga).

(D) Now change travel time from Edinburgh back to London: it is now 50 minutes (instead of 45 minutes because of strong wind). All the other travel times remain the same. Give the adjacency matrix of the graph G of all the flights. A vertex is a city and a directed edge $c_1 \to c_2$ represents a flight from city c_1 to city c_2.

(E) Using the `<map>` STL, write the contents of the `int main() { ... }` function that does the following things:

- Creates an empty map called `Europe`, with strings as keys and integers as values.
- Adds 5 keys, the first letter of every city, all with value 1.
- Outputs the dictionary as a sequence of `key:value` separated by a space
- Changes the value of key `L` to 0.
- Changes the value of key `E` to 0.
- Outputs the dictionary as a sequence of `key:value` separated by a space