AIP: Tutorial 1

1.

   (a) With $h(S) = 0$, A* search will be optimal because this never over-estimates the cost for the remainder of the plan, meaning the heuristic considered is the length of the path so far (i.e. breadth first search without expanding the goal - the result is still the same).

   (b) With $h(S) = 0$ for goal states and $h(S) = rand(0, 100)$ for non-goal states, A* will not be optimal because a random number could easily over-estimate distance to a goal, and also will probably not provide a consisten heuristic.

   (c) With $h(S) = \#unsatGoalConds$, A* is not necessarily optimal, because a given action may satisfy multiple conditions at once, bringing it closed to the goal despite having more unsatisfied conditions than another given state that is further from the goal (on a path that satisfies conditions "slowly").

   (d) With $h(S) = \#unsatGoalConds/mostGoadsAchievedByOneAction$, A* will not be optimal because the heuristic is admissible but not consistent.

2. A static fact is one that does not change (e.g. 'There is a path from A to B'). These can save memory, because they do not need to be stored in any state and can be stored once, because they will be consistent for every state.

3.

   (a) A directed graph shows the entire possible search space, including many useless actions like going back on a given state. A search tree, on the other hand, avoids this by considering the closed list of states that have been visited before, and merging duplicate states.

   (b) The closed list could be represented by a HashSet of State objects. The hash code for state objects could be comprised of the hash of the trees within them, which in turn could be the combined hash of the facts within it in some deterministic order. This would be fast, as the hash is generated only at insertion time for a given state in the closed list, and the implementation of a `HashSet` efficiently checks whether the set contains a given value.

```
class State {

        TreeSet<String> facts;

        // ...

        @Override
        public int hashCode() {
                int prime = 31;
                int result = 1;
                for (String fact : facts) {
                        result = result * prime + fact.hashCode();
                }
                return result;
        }

}
```

```
HashSet<State> closedList = new HashSet<>();
```

```
public boolean isInClosedList(State s) {
        return closedList.contains(s);
}
```