ALG2 Assignment D

Answer the questions in the document!

1. Imagine we have an algorithm with 2nn2 time complexity and a machine that can handle inputs of size 30 but not larger. Now we have found a 10 times faster algorithm for the same problem, thus with complexity 2nn2 /10. What is the maximal input size that we can handle with this new algorithm? 33 And with a 100 times faster algorithm? 36
2. Under which condition a “brute force” algorithm is still acceptable:
   1. Not much time to design and program another algorithm (Yes/Maybe/No)
   2. Instances are small (e.g. <20) (Yes/Maybe/No)
   3. Need to solve the problem only once (Yes/Maybe/No)
3. A problem can be solved with a brute forces algorithm with 2n complexity. Reason about the different situations where you can or cannot apply this algorithm depending on the number of instances you need to solve and the size of the instances, as shown below in the table:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | size of instance | | | | |
| number of instances to solve |  | <10 | <20 | <30 | >100 |
| 1 | y/m/n | y/m/n | y/m/n | y/m/n |
| 100 | y/m/n | y/m/n | y/m/n | y/m/n |
| 10000 | y/m/n | y/m/n | y/m/n | y/m/n |

1. For n=50 what is the speedup of an algorithm with complexity vs
   1. more than million
   2. more than a billion
   3. more than a trillion
2. For n=50 what is the speedup of an algorithm with complexity vs
   1. more than million
   2. more than a billion
   3. more than a trillion
3. You have discussed a “bounded search tree” method for finding a vertex cover of a graph by which in each iteration an edge e with vertices A and B is selected and the tree then “branches” in three further cases: 1. A is in the vertex cover and B is not, 2. B is in the vertex cover but A is not, 3. Both A and B are in the vertex cover. Reason about the following statements:
   1. the search tree based algorithm always finds the smallest vertex cover
   2. in some special cases the search tree will not give the correct answer
   3. at each level of the search tree it determines the assignment of at least one vertex
   4. it is possible to construct a search tree that determines the assignments of at least two vertices at each level
4. In the same search tree method as in question 6, assume that the assignment of at least 2 vertices is determined in each level of the tree, starting with 0 assignment at the root of the tree. The level of the root is 0 and the level of the leaves of the tree is n/2. In every leaf of the tree all vertices have been assigned (in the vertex cover or not), thus each leaf is a possible vertex cover for the given graph. How many leaves, and thus different assignments for the vertices of the graph, are at the last level of the search tree?
   1. 2n
   2. 3n/2
   3. 3n
   4. 23n
   5. 2n/2 · 3
5. You are thinking about developing a “bounded search tree” algorithm for independent set of a graph. As usual every vertex is either in the set (is assigned “1”) or is not in the set (is assigned “0”). For an edge e with vertices A and B you are reasoning what a valid assignments of A and B could be with respect to an independent set. Which of the following is true:
   1. At most one of A and B can be assigned “1”
   2. Both A and B can be assigned “1”
   3. At least one of A and B must be assigned “1”
   4. Both A and B could be assigned “0”
   5. Both A and B must be assigned “0”
6. You are thinking further about the “bounded search tree” algorithm for independent set you want to develop. Again you consider an edge e with vertices A and B. At certain level of the tree either A or B has been assigned a value (“1” or “0”). How can you deal in this situation regarding the further branching of the tree:
   1. Modify the search tree
   2. Initiate a brute force search
   3. Whit a short analysis it becomes obvious to you what to do next

????????

1. What is the complexity of the best known (exact) algorithms for maximum independent set in a (general) graph? Are there polynomial algorithms for some special cases? Please add reference to the sources you use for this question.

???????

1. What is the complexity of the best known (exact) algorithm for the 3SAT problem? Please add reference to the sources you use for this question.
2. What is the complexity of the best known algorithm for the traveling salesman problem? Are there polynomial algorithms for some special case(s)? Please add reference to the sources you use for this question.

???????

1. Imagine you have a computer that can handle at most 1 billion different solutions (different assignments of the vertices in a graph or variables in a Boolean formula). Now consider your answers of questions 10, 11 and 12. What is the maximum size (n, where n is the number of vertices in the graph problems, and the number of Boolean variables in the 3SAT problem) that your computer can handle with these algorithms. Imagine you have an algorithm for independent set with time complexity of 1.1n. What is the maximum size you can handle with this algorithm? 119