WARNING: Drafts of slides are made available prior to lecture for your convenience. After lecture, slides will be updated to reflect material taught. Check the date on this page to make sure you have the correct, updated version.

WARNING: Slides do not include all class material; if you have missed a lecture, make sure to find out from a classmate what material was presented verbally or on the board.
Topics requested

- FPT
- Ratio bounds
- Randomized sorting and selection
- Adversary lower bounds, including A3 W3
- Reductions, including A3 W2
FPT review: parameter choice

Question: How can a parameter limit the size of an instance, since it depends on the instance itself?

Answer: A better wording would have been the following: “A constant bound on the size of the parameter should not result in an instance of constant size.”

Examples for vertex cover:

- Parameter $k$ is fine. It does not imply a graph of a particular size.
- Parameter maximum degree of any vertex in the graph is fine. It will limit the number of edges with respect to the number of vertices, but does not limit the number of vertices.
- Parameter number of vertices is not fine. It results in a graph of constant size.
FPT review: branching

We create a search tree (like for backtracking) in which the number of children of each node is a constant $c$ and the depth is at most $k$, so that the number of nodes is in $O(c^k)$.

The total cost is the product of the number of nodes and the cost of processing a node.

When the cost of processing a node can be expressed as $O(f(k)n^{O(1)})$, then the total cost is $O(c^kf(k)n^{O(1)})$. Here we can “split” the costs into the $O(c^kf(k))$ part (a function only of $k$) and the $O(n^{O(1)})$ part (a function only of $n$).

Notes:

- This is just one of many paradigms used to form an FPT algorithm.
- The analysis we gave in class is just one of the ways to ensure fixed-parameter running time.
FPT review: Vertex cover algorithm

- Example of algorithm
- Running time analysis
- Correctness
A ratio bound gives an upper bound on how bad the output of an approximation algorithm can be compared to the optimal solution. The bound will hold for any input to the algorithm.

For a specific instance, you can find the ratio between the approximate and optimal solutions without comparing both to some other measure.

We can show that a bound doesn’t hold by finding a counterexample.
Randomized selection and sorting review

Algorithm:

• Choose a pivot by random selection.
• Check if the pivot is “good” by comparing it to all the other values.
• If the pivot is good, solve the problem recursively on the smaller input.
• If the pivot is bad, try again to select a good pivot.

Analysis:

• The probability of finding a “good” pivot is 1/2.
• The expected number of tries needed to get a good pivot is 2. (You don’t need to know this.)
Why a $3/4$ split is good enough

Recall Master method:

- $T(n) = aT(n/b) + f(n)$
- Compare $f(n)$ and $x = n^{\log_b a}$
- $T(n)$ is $\Theta$ of the “bigger one”, or $\Theta(x \log n)$ if they are the “same size”

Reduce by $3/4$ instead of $1/2$ for search: $T(n) \leq T(\lceil 3n/4 \rceil) + \Theta(1)$

$a = 1$, $b = 4/3$, $f(n) \in \Theta(1)$, $n^{\log_{4/3} 1} = n^0 = 1$

For sorting, we need to solve a more complicated recurrence, such as $T(n) \leq T(\lceil 3n/4 \rceil) + T(\lfloor n/4 \rfloor) + \Theta(n)$, which is outside the scope of the course.
Adversary lower bound review

Goals:

- View the process of finding a worst-case input for an algorithm as the interactions between the algorithm and an adversary.
- The algorithm’s job can be seen as identifying what the input is (and hence what the correct output should be).
- Define a strategy for the adversary based on a key operation but no other details of the algorithm.
- Prove a lower bound by showing that if the algorithm (that is, any algorithm) takes too few steps, there will still be two possible inputs consistent with the answers of the adversary, each leading to a different output.
In this question you will prove an adversary lower bound on the problem of determining if a graph has a *triangle*, that is, a set of three vertices \( x, y, \) and \( z \) such that there is an edge between each pair of vertices in the set. For your bound, you will consider only algorithms in which the key operation (and the only operation on graphs) is asking whether or not two vertices are adjacent. You may assume that you have a list of IDs of all the vertices in the graph.

Possible adversary approaches:

- Always say “yes”. But then a clever algorithm can find an algorithm after only three questions.
- Always say “no” - but modified so that the answer can be “yes” or “no” to prove a bound.
Reduction review

From A3 W2

**Partial Managing Set Search**

**Input:** A graph $G$ and a subset $A \subseteq V(G)$

**Output:** A set of vertices $S \subseteq V(G)$ of minimum size such that $S$ manages $A$

**Partial Managing Set Decision**

**Input:** A graph $G$, a subset $A \subseteq V(G)$, and an integer $k$

**Output:** Yes or no, answering “Does $V(G)$ contain a subset $S$ of size at most $k$ such that $S$ manages $A$?”