**Known Runtimes from Lectures**

These runtimes below are worst case runtimes that can be used in the class without additional justification in the NP section of the course. Note in some cases these are not actual real-world worst-case runtimes.[[1]](#footnote-1) But please use these runtimes unless you want to lose marks.

|  |  |  |
| --- | --- | --- |
| Operation | Run Time | Proof |
| Verify clique | O(n2) | NP-1-Fall24 |
| Set: Add node | O(1) | NP-2-Fall24 |
| Set:  Find/Delete 1 node  Iterate through all nodes  Check size | O(n) | NP-2-Fall24 |
| Any Hash Map operation | O(n) | NP-3-Fall24 |
| Quicksort | O(n2) | NP-4-Fall24 |

**Known NP-Complete Problems from Lectures**

**Credit to the person who posted on ED**

[**SAT**](https://en.wikipedia.org/wiki/Boolean_satisfiability_problem)

* **Input: boolean formula f in CNF with n-variables and m-clauses**
* **Output: Satisfying assignment S where each clause evaluates to True. Returns 'NO' when no such S exists.**

***SAT Variations***

* **k-SAT is SAT with an additional input constraint k so that each clause has at most k literals. To be NP-C, k > 2**

[**Independent-Set (Search)**](https://en.wikipedia.org/wiki/Independent_set_(graph_theory))

* **Input: Graph G = (V, E) and goal g**
* **Output: Subset S in V if there are no edges between every vertex in S and |S|>=g. Returns 'NO' when no such S exists.**

[**Clique (*Search*)**](https://en.wikipedia.org/wiki/Clique_problem#Definitions)

* **Input: Undirected Graph G = (V, E) *and goal g***
* **Output: subset S in V where there are edges between every pair of vertices in S and |S|>=g. Returns 'NO' when no such S exists.**

[**Vertex Cover (Search)**](https://en.wikipedia.org/wiki/Vertex_cover)

* **Input: Undirected Graph G = (V, E) and budget, b**
* **Output: Subset S in V, if every e=(x,y) has either x in S or y in S and |S|<=b. Returns 'NO' when no such S exists.**

[**K-Colorings**](https://en.wikipedia.org/wiki/Graph_coloring)

* **Input: Undirected Graph G=(V,E) & integer k > 0**
* **Output: assign each vertex a color in {1, 2, ..., k} so that adjacent vertices get different colors and NO if no such k-coloring exists**
* ***Note:* for k=2, K-Colorings∈P because it can be solved with modified BFS/DFS to find a bipartite graph**

[**Knapsack (Search)**](https://en.wikipedia.org/wiki/Knapsack_problem)

* **Input: n-items with integer weights: w\_1,...,w\_n; integer values v\_1,...,v+n; capacity B; and goal g**
* **Output: subset S items where the total value V is >=g and total weight W is <= B. Returns 'NO' when no such S exists.**

[**Subset Sum (Search)**](https://en.wikipedia.org/wiki/Subset_sum_problem)

* **Input: n positive integers a\_1, ..., a\_n and a target, t**
* **Output: Subset S of [1, ...., n] where the sum of a\_i for all i in S = t, if such a subset exists. Returns 'NO' when no such S exists.**

**Template for NP Complete Written Problems**

**Section (A) Polynomial Time**

* Refer to chart of known operations runtime. If the runtime is not of a known operation, then you gotta justify it
* Verification runtime doesn’t have to be optimal… it just has to be poly time.
* Mention all the runtimes of the steps used.
* Explicitly state that it is poly time for the verification at the end

**SAT → Unknown Problem**

We demonstrate that unknown problem is at least as hard as a problem known to be NP-Complete by reducing the **SAT** → **Unknown Problem**

* Reuse this statement word for word and just switch “SAT” and “Unknown Problem” for whatever problems you are using.
* The known NP-Complete algo is always on the left side
* Keep the 2 problems in the same domain (graph problems / truth literal stuff / etc).

**Section (B) Input Transformation**

* Explicitly state that it is poly time for the transformation at the end (and also mention the runtimes of all the steps used)
* When you are passing data to the other algo… remember for graphs there is usually a number you have to pass. Say you are doing a clique > unknown problem reduction. And the unknown problem requires a number N. Say something like: “Let k be the size of the clique we are searching for. We set equal to k and pass N along with the transformed graph to the unknown problem.” It is important to define the inputs of the known NPC problem and pass them accordingly.

**Section (C) Output Transformation**

Return no if the algorithm returns no.

* Always remember you need to “return no” if the other algo returns no. so just keep the
* Return whatever the objective is if you have whatever it is you need.
* Explicitly state that it is poly time for any transformation at the end (and also mention the runtimes of all the steps used).

**Section (D) Correctness Proof**

* You’re on your own, kid.

1. <https://medium.com/@yashodhara.chowkar/internal-working-of-hashmap-in-java-and-performance-improvement-in-java-8-a28ee1660cda> + proof NP-3-Fall24 [↑](#footnote-ref-1)