Exercise 0 Exercise 0.1 from Chapter 0 of Dasgupta. You are not asked to formally prove this from the definition. Just use your intuition to tell which is true $O()$, $\Omega()$ or both (and see if you were correct from the answers provided later.)

Solution: We make use of the following:

(i) $n^a = O(n^b)$ for $a \leq b$. (See question (b))

(ii) $n^a = \Omega(\log n)^b$ for any $a > 0$ and $b > 0$. (See question (g))

(iii) $\log_a n = \Theta(\log_b n)$ for $a, b > 1$. (See question (p)).

(iv) If $f(n) = O(g(n)$ then $h(n)f(n) = O(h(n)g(n))$. (See question (i))

(v) If $f(n) = O(g(n))$ then $g(n) = \Omega(f(n))$ and vice versa.

(a) $f(n) = n - 100$ and $g(n) = n - 200$
$f = \Theta(g)$.

(b) $f(n) = n^{1/2}$ and $g(n) = n^{2/3}$
$f = O(g)$.

(c) $f(n) = 100n + \log n$ and $g(n) = n + (\log n)^2$
$f = \Theta(g)$.

(d) $f(n) = n \log n$ and $g(n) = 10n \log(10n)$
$f = \Theta(g)$. Note, $10n \log(10n) = 10n(\log n + \log 10)$

(e) $f(n) = \log(2n)$ and $g(n) = \log(3n)$
$f = \Theta(g)$. Note, $\log(2n) = \log n + \log 2$. and $\log(3n) = \log n + \log 3$.

(f) $f(n) = 10 \log n$ and $g(n) = \log(n^2)$
$f = \Theta(g)$. Note, $\log(n^2) = 2 \log n$.

(g) $f(n) = n^{1.01}$ and $g(n) = n \log^2 n$
$f = \Omega(g)$. Note $n^{1.01} = n \cdot n^{0.01}$ and $n^{0.01} = \Omega(\log^2 n)$.

(h) $f(n) = n^2 / \log n$ and $g(n) = n(\log n)^2$
$f = \Omega(g)$. Note $n^2 = \Omega(n(\log n)^3)$. No divide both sides by $\log n$.

(i) $f(n) = n^{0.1}$ and $g(n) = (\log n)^{10}$
$f = \Omega(g)$.

(j) $f(n) = (\log n)^{\log n}$ and $g(n) = n/(\log n)$
$f = \Omega(g)$. Note $(\log n)^{\log n} = \Omega(2^{\log n}) = \Omega(n)$

(k) $f(n) = \sqrt{n}$ and $g(n) = (\log n)^3$
$f = \Omega(g)$.

(l) $f(n) = n^{1/2}$ and $g(n) = 5^{\log_2 n}$
$f = O(g)$. Note $n^{1/2} \leq n = 2^{\log_2 n}$

(m) $f(n) = n2^n$ and $g(n) = 3^n$
$f = O(g)$. Note $n = O(1.5^n)$ so $n2^n = O(1.5^n 2^n) = O(3^n)$.

(n) $f(n) = 2^n$ and $g(n) = 2^{n+1}$
$f = \Theta(g)$. Note $2^{n+1} = 2(2^n)$.

(o) $f(n) = n!$ and $g(n) = 2^n$
$f = \Omega(g)$. If $n \geq 4$ then $n! \geq n \cdot (n-1) \cdots 4 \geq 4^{n-4} = 4^{-4}4^n$.
(So $f = \Omega(g)$ and $f(n) \neq O(g(n))$.)

(p) $f(n) = (\log n)^{\log n}$ and $g(n) = 2^{(\log_2 n)^2}$
$f = O(g)$. Note $2^{(\log_2 n)^2} = (2^{\log_2 n})^{\log_2 n} = n^{\log_2 n}$

(q) $f(n) = \sum_{i=1}^{n} i^k$ and $g(n) = n^{k+1}$
$f = \Theta(g)$. Clearly, $f(n) < n(n^k) = n^{k+1}$. For the other direction, assume that n is even. Then,
$\sum_{i=1}^{n} i^k > \sum_{i=n/2+1}^{n} i^k > \sum_{i=n/2+1}^{n} (n/2)^k = (n/2)(n/2)^k = \Omega(n^{k+1})$.
(To see the last step, note that k is not a variable. So, $(1/2)^k$ is a constant.) For odd n, the notation is a bit different but the proof is similar.

Exercise 1 Exercise 8.1 from Dasgupta. Remember that all distance are assumed integer.

Solution: Let us first see why the easy approach is not a polynomial reduction. The easy approach is to try values $b = 1, 2, 3, \ldots$ until the TSP-algorithm finds a feasible solution. The number of times we apply the TSP algorithm is exactly the length of the shortest tour. This is not polynomial in the input size. The input size of a number d is $O(\log_2 d)$. Putting it the other way around, the *value of an integer number is exponential in its input*

size. (To illustrate this assume that there are only 3 points and all 3 distances are 2^{20}. Then the input size is about 60 but the number of iterations of this approach is more than 3 million.) See Dasgupta page 242 for more on this.

The efficient (polynomial time) approach is to apply binary search. Let L be a lower bound on the optimal value and U an upper bound. Initially, take $L = 0$ and let U be an upper bound on the optimal value, in this case we could take $U = \sum_{i,j} d(i,j)$, i.e., the sum of all distances. In each iteration we check if there is a valid solution, i.e, a tour of length at most b and then adjust L or U. We maintain the invariant that there is valid solution of value at most U and there is no solution of value $L - 1$.

```
def BinarySearch(L, U):
    while L ≠ U:
        b = ⌊(L + U)/2⌋
        if Valid(b):
            U = b
        else:
            L = b + 1
    return U
```

The number of times that we check for a valid solution of value at most b is only $O(\log(\sum_{i,j} d(i,j)))$ which is polynomial in the input size.

Exercise 2 Exercise 8.2 from Dasgupta.

Solution: Let $G = (V, E)$ be the graph. Order the edges in an arbitrary order $e_1, e_2, \ldots, e_m$. Assume we have a function that tells if a graph G has a Rudrata Path (RP)

If $G = (V, E)$ has no RP then we can stop immediately. Otherwise we iterate as follows. We try the edges one by one. If there is still a RP after deleting an edge e_i then we delete it. Otherwise we keep it.

```
F = E
for i = 1 ... m:
    if G = (V, F \ {e_i}) has a RP then:
        F = F \ {e_i}
```

The final set F is a Rudrata path. To see this note that we only keep an edge e_i if $G = (V, F)$ has a RP while $G = (V, F \setminus \{e_i\})$ has no RP. But then e_i must be in *every* RP in $G = (V, F)$. Hence, all edges in the final set F are in every RP in $G = (V, F)$. That means, F itself is a RP.

Exercise 3 Exercise 8.10 from Dasgupta. Hint: The following problems were mentioned as NP-complete in the chapter and they provide the answer to subquestions (a) - (g) (in some order). See Chapter 8 for their definitions. Rudrata Path, Rudrata Cycle, Clique, Vertex Cover, SAT, Independent Set.

(a) SUBGRAPH ISOMORPHISM is a generalization of CLIQUE.
Assume we want to find a clique of size k in a graph $G = (V, E)$. Let $H = K_k$. There is a subgraph of G that is isomorph to H if and only if G has a clique of size k.

(b) LONGEST PATH is a generalization of RUDRATA PATH
Assume we want to find a RUDRATA PATH. There is a path of length $n - 1$ if and only if there is a Rudrata path. (Note, a RP has $n - 1$ edges.)

(c) MAX SAT is a generalization of SAT
Assume we want to find a satisfying truth assignment for a SAT instance. Let m be the number of clauses. There is a truth assignment that satisfies at least m clauses if and only if there is a truth assignment that satisfies all clauses.

(d) DENSE SUBGRAPH is a generalization of CLIQUE
Assume we want to find a clique of size k. Let $a = k$ and $b = a(a-1)/2$. There is a set of a vertices with at least b edges between them if and only if there is a clique of size k.

(e) SPARSE SUBGRAPH is a generalization of INDEPENDENT SET
Assume we want to find an independent set of size k. Let $a = k$ and $b = 0$. There is a set of a vertices with at most b edges between them if and only if there is an independent set of size k.

(f) SET COVER is a generalization of VERTEX COVER
Assume we want to find a vertex cover of size k in a graph $G = (V, E)$. Define the following Set Cover instance. Let $B = E$ be the set of

elements and for each $i \in V$ let S_i be the set of edges that are incident with (i.e., attached to) vertex i. There is a set cover of size k if and only if G has a vertex cover of size k.

(g) RELIABLE NETWORK is a generalization of RUDRATA CYCLE Assume we want to find a Rudrata cycle in a graph $G = (V, E)$. Define $d_{ij} = 1$ for all $(i, j) \in E$ and $d_{ij} = 2$ if $(i, j) \notin E$. Let $b = n$ and $r_{ij} = 2$ for all pairs of points i, j. If there is a Rudrata cycle C in G then C satisfies conditions (1) and (2). Vice versa, if there is a set of edges C that satsfies conditions (1) and (2) then this must be a Rudrata cycle.

Exercise 4 Exercise 8.12 from Dasgupta.

Solution: (a) To show that it is a search problem we must show that there is a polynomial time algorithm that, given a candidate solution, verifies correctly that it is a valid solution. Given a graph we should verify that (1) it is a spanning tree and (2) all degrees are at most k. The verify (1) it is enough to verify that it has $n - 1$ edges and that it connects all vertices. The latter can be done by breath- or depth-first search. To verify (2) we only need to check for each vertex that it has at most k neighbors. Clearly, this can be done in polynomial time.

(b) The k-SPANNING TREE is a generalization of RUDRATA PATH. Assume we want to find a RUDRATA PATH. Let $k = 2$. There is spanning tree in which each note has a degree of at most 2 if and only of there is a Rudrata path.

Exercise 5 Exercise 8.13 from Dasgupta.

Solution:

(a) In **P**.
Assume that there exists a spanning tree T such that its set of leaves includes L. If we delete the vertices L from the tree then it remains connected. Hence, if we delete L and all its incident edges from G then the graph remains connected. Now the following algorithm finds a spanning tree T such that its set of leaves includes L if such a tree exists.
(1) Delete L and all its incident edges from G. Let this be G'.
(2) Find a spanning tree T' in G'.
(3) For each vertex v in L, add an edge that connects v to T'.

(b) **NP**-complete.
This problem is a generalization of RUDRATA (s,t)-PATH. Assume we want to find a Rudrata path from $s \in V$ to $t \in V$. Let $L = \{s,t\}$. Then there is a spanning tree with set of leaves exactly L if and only if there is a Rudrata path from s to t.

(c) **NP**-complete.
This problem is a generalization of RUDRATA (s,t)-PATH. Assume we want to find a Rudrata path from $s \in V$ to $t \in V$. Let $L = \{s,t\}$. Then there is a spanning tree with set of leaves *included* in L if and only if there is a Rudrata path from s to t.
(Note that we could use here the same reduction as in (b) since each tree has at least two leaves: If the set of leaves is included in $L = \{s,t\}$ it must be exactly L.)

(d) **NP**-complete.
This problem is a generalization of RUDRATA PATH. Assume we want to find a Rudrata path. Let $k = 2$. Then there is a spanning tree with at most k leaves if and only if there is a Rudrata path.

(e) **NP**-complete. *(This exercise turns out to be much more complictaed than the other exercises. Skip this one if you like.)*
This problem is a generalization of 3D-MATCHING. Assume we are given an instance of 3D-matching in the form of a set W of m elements and a set V of triples of W. An example is given in Figure 1. Without loss of generality, each element of W appears in at least one triple (since otherwise no 3DM exists.) Now construct the following bipartite graph. We define a vertex for each element in W and V. (For simplicity we denote the vertex sets also by V and W.) There is an edge (v,w) between $v \in V$ and $w \in W$ if w is in triple v. Further, there is an edge between any two vertices of V. Let $k = n + m - m/3 = n + 2m/3$.
If there is 3DM $S \subset V$ then construct the following tree. Connect the vertices S by a path and connect all other vertices directly to S. The number of leaves is exactly $n + m - |S| = k$.
Now assume there is a spanning tree with k leaves. Each vertex in V is neighbor of at most 3 vertices in W. If x is the number of leaves in W then there are at most $n - x/3$ leaves in V. So we must have $n + 2x/3 \geq k = n + 2m/3$ which implies $x \geq m$. But also $x \leq |W| = m$. So $x = m$. All vertices in W must be leaves and the $m/3$ non-leaves in V define a 3DM.

Conclusion: There is a 3DM if and only if there is a tree with (at least) $k = n + 2m/3$ leaves.

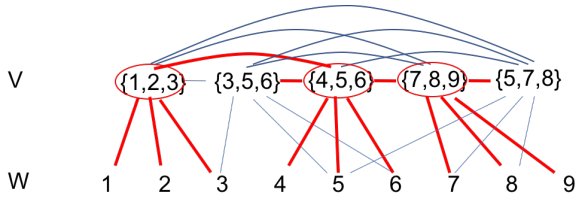

Figure 1: Illustrating the reduction for (e). There is a 3DM and a corresponding tree with $k = n + 2m/3 = 11$ leaves is shown in red.

(f) **NP**-complete.
This problem is a generalization of RUDRATA PATH. Assume we want to find a Rudrata path. Let $k = 2$. Then there is a spanning tree with exactly k leaves if and only if there is a Rudrata path.

Exercise 6 Exercise 8.16 from Dasgupta. Hint: you do not need to know or use the definition of 3SAT to answer this question. It satisfies to know that it is an NP-complete problem.

Solution: First we show that: EXPERIMENTAL CUISINE is a generalization of INDEPENDENT SET.
Assume we want to find an independent set of size at least k in a graph $G = (V, E)$. Then construct the following instance of the EXPERIMENTAL CUISINE problem with $n = |V|$ ingredients. Take penalty $p = 0$ and let the discord between ingredient i and j be 1 if $(i, j) \in E$ and let it be zero otherwise. We can choose k ingredients with total penalty $p = 0$ if and only if there is an independent set of size k.

From the above: If EXPERIMENTAL CUISINE is solvable in polynomial time then INDEPENDENT SET is solvable in polynomial time. We know that INDEPENDENT SET is NP-complete. So if INDEPENDENT SET is solvable in polynomial time then every problem in NP is solvable in polynomial time, including 3SAT.

Exercise 7 Exercise 8.21 from Dasgupta.

Solution: (a) The reconstruction problem reduces to DIRECTED RUDRATA PATH. The hint gives the reduction. Assume that for a given multi set $\Gamma(x)$

of k-mers we want to reconstruct a corresponding string x. Construct a directed graph with one node for each k-mer, and with an edge from a to b if the last $k-1$ characters of a match the first $k-1$ characters of b.

Figure 2: Example illustrating the reduction (a). Given a multi set of k-mers (left, $k = 3$) we construct a directed graph as described. Any directed Rudrata path in the graph corresponds with a reconstruction of the multi set and vice versa.

(b) The reconstruction problem reduces to DIRECTED EULER PATH. Assume that for a given multi set $\Gamma(x)$ of k-mers we want to reconstruct a corresponding string x. Construct the following digraph. For each substring of $k-1$ characters that appears in $\Gamma(x)$ we define a vertex. For each k-mer there is an arc from a vertex a to a vertex b where a is the vertex that corresponds with the first $k-1$ characters of the k-mer and b corresponds with the last $k-1$ characters.

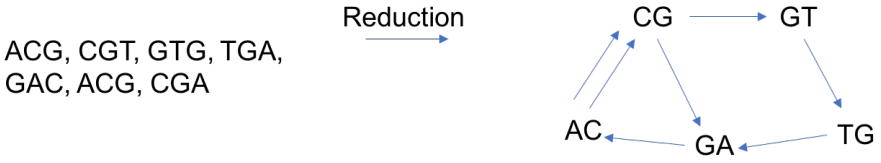

Figure 3: Example illustrating the reduction (b). Given a multi set of k-mers (left, $k = 3$) we construct a directed graph as described. Any directed Euler path in the graph corresponds with a reconstruction of the multi set and vice versa.