1 Question 1

In increasing order of growth, we have: $100n^2, n^{10}, (log\ n)^{log\ n}, n^{log\ n}, (n^2)^{log\ n}, (1.1)^n$. No two have the same order of growth.

2 Question 2

First an informal description of the simulation: we note that a two-side unbounded Turing machine will only ever have examined a finite portion of its tapes. This suggests that we could simulate a two-side unbounded Turing machine with a standard Turing machine by reproducing it's actions, but only ever storing the visited window of the simulated machine's tape. If we try to move to the right of the window, it can easily be extended, and if we try to move to the left of the window, we can move the window one cell over, then extend it by one cell on the left, and continue from there.
More precisely, to simulate a k-tape two-side unbounded Turing machine M with a k-tape standard Turing machine M_S, make M_S follow the following algorithm:

1: **machine** M_S **on input** w:
2: Place a start marker $\vdash$ before w on the first tape, and an end marker $\dashv$ after w on the first tape, placing the read head over the first symbol in w. If $w = \epsilon$, treat the first tape like the others, as described in the next step.
3: On all other tapes, place the symbols: $\vdash \Box \dashv$, where $\Box$ is the empty cell symbol, and put the read head on the $\Box$.
4: Now, begin simulating M on these tapes, until any read head sees either $\vdash$ or $\dashv$, or M halts.
5: **if** any read head sees a $\dashv$ **then**
6:     Write $\Box$ over the $\dashv$.
7:     Move the read head one cell to the right.
8:     Write $\dashv$ here.
9:     Move the read head one cell to the left.
10:     Go back to step 4, continuing the simulation.
11: **else if** any read head sees a $\vdash$ **then**
12:     Move the contents of the read head's tape between $\vdash$ and $\dashv$ inclusively one cell to the right, returning the read head to right over $\vdash$.
13:     Write $\Box$ over the $\vdash$.
14:     Move the read head one cell to the left.

15: Write $\vdash$.
16: Move the read head one cell to the right.
17: Go back to step 4, continuing the simulation.
18: **else if** M has halted and accepted **then**
19: **accept**
20: **else if** M has halted and rejected **then**
21: **reject**
22: **end if**

3 Question 3

In all the questions below, I assume that the total Turing-machines M_1 and M_2 decide languages L_1 and L_2; for each of the required sets I will construct a total Turing-machine M that will decide the set:

3.1 Part (i)

1: **machine** M **on input** s:
2: Simulate M_1 on s
3: Simulate M_2 on s
4: **if** any accepted **then**
5: **accept**
6: **else**
7: **reject**
8: **end if**

Theorem 1. *M is total.*

Proof. As M_1 and M_2 are total, the simulations in steps 2 and 3 must finish whatever the value of s. As all the other steps obviously finish, and as M accepts or rejects whatever the result of the test in step 4, M will either accept or reject for all values of s. So M is total. □

Theorem 2. $L(M) = L(M_1) \cup L(M_2)$

Proof. By double inclusion:

- If $x \in L(M)$, then M must have accepted in step 5. But then, either M_1 or M_2 accepted x. So $x \in L(M_1) \cup L(M_2)$.
- If $x \in L(M_1) \cup L(M_2)$, then x is accepted by one of M_1 or M_2. Thus, as M cannot possibly reject at line 7, and as M is total and thus must either accept or reject, M must accept. So $x \in L(M)$.

□

3.2 Part (ii)

```
1: machine M on input s:
2: Simulate M1 on s
3: Simulate M2 on s
4: if both accepted then
5:     accept
6: else
7:     reject
8: end if
```

Theorem 3. *M is total.*

Proof. Precisely as the proof of **Theorem 1** □

Theorem 4. $L(M) = L(M_1) \cap L(M_2)$

Proof. By double inclusion:

- If $x \in L(M)$, then M must have accepted in step 5. But then, both M_1 and M_2 accepted x. So $x \in L(M_1) \cap L(M_2)$.
- If $x \in L(M_1) \cap L(M_2)$, then x is accepted by both of M_1 and M_2. Thus, as M cannot possibly reject at line 7, and as M is total and thus must either accept or reject, M must accept. So $x \in L(M)$.

□

3.3 Part (iii)

1: **machine** M **on input** s:
2: Simulate M_1 on s
3: **if** it accepted **then**
4:     **reject**
5: **else**
6:     **accept**
7: **end if**

Theorem 5. *M is total.*

Proof. Since M_1 is total, step 1 finishes. Since all other steps obviously finish, and M accepts or rejects regardless of the result of the test in step 3, M will either accept or reject regardless of the value of s. So M is total. □

Theorem 6. $L(M) = \overline{L(M_1)}$

Proof. By double inclusion:

- If $x \in L(M)$, then the test in line 3 must fail. Thus, M_1 must fail to accept x, and so $x \notin L(M_1)$ i.e. $x \in \overline{L(M_1)}$.
- If $x \in \overline{L(M_1)}$, then $x \notin L(M_1)$, and so M_1 fails to accept x. This means that M cannot possibly reject on line 4, and since M is total ant thus must either accept or reject, M must accept. So $x \in L(M)$.

□

3.4 Part (iv)

1: **machine** M **on input** s:
2: **for** $(p, q) \in \{(x, y) | x \in L_1, y \in L_2, xy = s\}$ **do**
3:     Simulate M_1 on p
4:     Simulate M_2 on q
5:     **if** Both accepted **then**
6:         **accept**
7:     **end if**
8: **end for**

9: **reject**

Theorem 7. *M is total.*

Proof. M is total as:

- The **for** on step 2 runs for at most $|s| + 1$ iterations i.e. it runs for a finite number of iterations.
- Steps 3 and 4 must finish, since they consist of simulating total Turing-machines.
- Steps 5, 6, 9 finish.

And, regardless of the number of iterations of the **for**, or of the result of the test on step 5, M must either accept or reject (due to step 9), for any value of s. So M is total. $\square$

Theorem 8. $L(M) = L(M_1); L(M_2)$

Proof. By double inclusion:

- If $s \in L(M)$, then at some point M must accept on step 5. This implies that, for some (p, q), we have that $pq = s$, M_1 accepts p, and M_2 accepts q. This implies, by definition, that $s \in L(M_1); L(M_2)$.
- If $s \in L(M_1); L(M_2)$, then $s = pq$ for some $p \in L(M_1), q \in L(M_2)$. But then, M cannot possibly reject s, as by the time M would potentially reject s,M should already have tried (p, q) in the **for**-loop, and have accepted. So, as M is total, and thus must either accept or reject any input, M must accept s. So $s \in L(M)$.

$\square$

3.5 Part (v)

machine M **on input** s:
if $s = \epsilon$ **then**
    accept.
end if
for $(w_1, ..., w_k) \in \{(s_1, ..., s_k) : s_1...s_k = s, s_i \in \Sigma^+, i = 1...k\}$ **do**
    Simulate M_1 on $w_1, ..., w_k$
    if all accepted **then**

8: **accept**
9: **end if**
10: **end for**
11: **reject**

Theorem 9. *M is total.*

Proof. M is total as:

- The **for** on step 5 runs for at most $2^{|s|-1}$ iterations i.e. it runs for a finite number of iterations.
- Step 6 must finish, since it consists of simulating total Turing-machines.
- All other steps finish.

And, regardless of the number of iterations of the **for**, or of the result of the tests on steps 2 and 7, M must either accept or reject (due to step 10), for any value of s. So M is total. □

Theorem 10. $L(M) = L(M_1)^*$

Proof. By double inclusion:

- If $x \in L(M)$, then M either accepts on line 3 or accepts on line 7. So, either $x = \epsilon$, or, for some values of $w_1, ..., w_k \in \Sigma^+$ we have that $w_1...w_k = x$ and M_1 accepts $w_1, ..., w_k$ i.e. $w_1, ..., w_k \in L(M_1)$. But in either case $x \in L(M_1)^*$.
- If $x \in L(M_1)^*$, then either $x = \epsilon$ or $x = w_1...w_k$ for some $w_1, ..., w_k \in L(M_1)$. But this means that M cannot possibly reject x, as by the time it would have rejected x, it either would have needed to previously accept x on line 3 (if $x = \epsilon$), or on line 8 (if $x = w_1...w_k$). So, as M is total, M must accept x, and so, $x \in L(M)$.

□

4 Question 4

Theorem 11. *Given a machine M_H that decides* HALTING*, and a machine M that accepts L, then a machine M_T can be constructed that decides L (this directly implies that* HALTING *is R.E. complete).*

Proof. Consider the following machine:

1: **machine** M_T **on input** s:
2: Simulate M_H on $\langle M, s\rangle$.
3: **if** M_H accepted **then**
4:     Simulate M on s.
5:     Accept/reject if and only if M accepted/rejected.
6: **else**
7:     **reject**
8: **end if**

Now note that M_T is total, as:

- Step 2 must finish, as M_H is a decider, and thus total.
- Step 4 must finish if executed, as it is only executed if M_H accepts $\langle M, s\rangle$ i.e. if M will halt on input s.
- All other steps finish.
- Regardless of the result of the test on line 3, M will certainly either accept or reject.

Also note that $L(M_T) = L(M)$, as:

- If $s \in L(M_T)$, then M_T accepted s on line 5, and so M accepts s. So $s \in L(M)$.
- If $s \notin L(M_T)$, then, as M_T is total, M_T must reject s. But then, either M rejects s (if M_T rejected on line 5), or M_H rejects $\langle M, s\rangle$ i.e. M does not halt on s (if M_T rejected on line 7), so in either case, M does not accept s, and so $s \notin L(M)$.

So M_T is the desired machine. $\square$

5 Question 5

Theorem 12. EMPTINESS *is undecidable.*

Proof. To show this, I construct a mapping reduction from HALTING to $\overline{\text{EMPTINESS}}$. First, for each Turing-machine M and string x, consider the machine $N_{\langle M,x\rangle}$, defined as follows:

1: **machine** $N_{\langle M,x\rangle}$ **on input** s:

2: Simulate M on x for $|s|$ steps.
3: **if** M has halted within these steps **then**
4:     **accept**
5: **end if**
6: **reject**

Note that M will halt on input x if and only if $N_{\langle M,x\rangle}$ has a non-empty language (*), as:

- If M halts on input x, it must do so after some number of steps n. This implies that any string of at least n characters is accepted by $N_{\langle M,x\rangle}$, and thus this machine has non-empty language.
- If $N_{\langle M,x\rangle}$ accepts some string s, then it must accept s on line 4. This implies that M halts on input x after at most $|s|$ steps.

Let M_0 be a Turing-machine that accepts no strings. Now define $f : \Sigma^* \to \Sigma^*$ by:

$$f(w) = \begin{cases} \langle N_{\langle M,x\rangle}\rangle & \text{if } w = \langle M, x\rangle \text{ for some Turing-machine } M \text{ and string } x \\ \langle M_0\rangle & \text{otherwise} \end{cases} \tag{1}$$

Note that $w \in \textsc{Halting} \iff f(w) \in \overline{\textsc{Emptiness}}$, as:

- If $w \in \textsc{Halting}$, then $w = \langle M, x\rangle$ for some Turing-machine M and some string x on which M halts. So by definition $f(w) = \langle N_{\langle M,x\rangle}\rangle$. By (*), as M halts on w, we have that $N_{\langle M,x\rangle}$ has non-empty language. So $f(w) = \langle N_{\langle M,x\rangle}\rangle \in \overline{\textsc{Emptiness}}$.
- If $w \notin \textsc{Halting}$, there are two cases:
  - If $w = \langle M, x\rangle$ for some Turing-machine M and some string x on which M does not halt, then $f(w) = \langle N_{\langle M,x\rangle}\rangle$. By (*), as M does not halt on x, $N_{\langle M,x\rangle}$ has $\emptyset$ as its language. So $f(w) = \langle N_{\langle M,x\rangle}\rangle \in \textsc{Emptiness}$, and thus $f(w) \notin \overline{\textsc{Emptiness}}$.
  - Otherwise, w is not of the form $\langle M, x\rangle$ for any Turing-machine M and string x, and so $f(w) = \langle M_0\rangle$. As M_0 has empty language by definition, $f(w) = \langle M_0\rangle \in \textsc{Emptiness}$, and thus $f(w) \notin \overline{\textsc{Emptiness}}$.

So, as f is computable, $\textsc{Halting} \leq_m \overline{\textsc{Emptiness}}$. This implies that, as $\textsc{Halting}$ is undecideable, so is $\overline{\textsc{Emptiness}}$. By the contrapositive of part (iii) of question 3, $\textsc{Emptiness}$ is also undecidable. $\square$

6 Question 6

I assume that this question refers to one-tape Turing machines, as it talks about a the Turing machine's "tape", as opposed to "tapes" – however, the argument generalises to multi-tape Turing machines.

Lemma 1. *A (one-tape) polynomially bounded Turing-machine M run on a string w has a finite number of possible configurations.*

Proof. Note that a configuration for a one-tape Turing machine consists of the tape contents, the current head position, and the current state. Now, supposing that M is polynomially bounded by f, has a tape alphabet Γ, and state set Q, we note that:

- The tape contents belong to $\{s \in \Gamma^* : |s| \leq f(|w|)\}$, and this set is finite; in particular it has size $\sum_{i=0}^{f(|w|)} |\Gamma|^i$.
- The state belongs to the finite set Q.
- The tape head position belongs to the finite set $\{1, 2, ..., f(|w|)\}$.

This, overall, implies that the Turing machine's configurations belong to a finite set, of size $(\sum_{i=0}^{f(|w|)} |\Gamma|^i) * |Q| * f(|w|)$. □

Theorem 13. *Suppose M is a (one-tape) polynomially bounded Turing-machine, which uses no more than $f(|w|)$ cells on its tape on input w, for some polynomial f. Then we can decide if M halts.*

Proof. Consider the following Turing-machine:

```
machine M_H on input ⟨M, x⟩, where M is polynomially bounded:
Initialize a simulation S of M on x without running it.
repeat
    Store a finite representation of the configuration of S [1].
    Advance S by a step.
    if S has halted then
        accept
    else if S has entered a configuration it has already been in before then
        reject
```

[1]This is possible as S's configuration is characterised by a finite amount of information: the part of the tape currently written on, the tape position, and the current state

10: **end if**
11: **until** forever

First: M_H is total. To see why, suppose that M_H were to run forever on some input $\langle M, x\rangle$. This implies that M will not halt when run on x (as M_H never accepts on line 7), and that M never enters the same configuration twice (as M_H never rejects on line 9). Thus, when run on x, M will enter an infinite number of different configurations. But this contradicts the previous lemma about polynomially bounded Turing-machines. So, our supposition is false, and M_H is total.

Now I show that M_H accepts $\langle M, x\rangle$ if and only if M halts when run on x, where M is a polynomially bounded Turing machine and x a string:

- If M_H accepts $\langle M, x\rangle$, then the condition on line 5 must at some point be satisfied. Thus M must halt when run on x.
- If M halts when run on x, then M can never enter the same configuration twice (as this would imply an infinite loop), and thus M_H can never reject on line 9. As M_H is total, it must thus accept $\langle M, x\rangle$.

So, M_H decides the halting problem for polynomially bounded Turing machines. $\square$

7 Question 7

Theorem 14. *If* CLIQUE *can be solved in time $T(n)$, where n is the number of nodes of the input graph, then* OPT-CLIQUE *can be solved in $O(n^c\, T(n))$ for some $c \in \mathbb{N}$.*

Proof. Supposing that G's nodes are taken from the set $\{1, ..., n\}$, consider the following algorithm for OPT-CLIQUE:

function OPT-CLIQUE(G)
  $i \leftarrow 1$
  while G has a clique of size at least $i+1$ **do** ▷ checked using CLIQUE
    $i \leftarrow i+1$
  end while
  for $j \leftarrow 1, 2, ..., n$ **do**
    $G' \leftarrow G \setminus \{j\}$ ▷ i.e. G but without node j.
    if G' has clique of size at least i **then** ▷ checked using CLIQUE
      $G \leftarrow G'$

end if

end for

return G

end function

To show correctness:

- Let G_0 denote the initial value of G.
- After the **while** loop, i will equal the size of the maximal clique of $G = G_0$. This can be shown using the following invariant: "G contains a clique of size at least i". Let m denote the size of this maximal clique.
- As G is initially a subgraph of G_0 (as they are initially equal), and is only ever assigned subgraphs of itself, G will always be a subgraph of G_0.
- As G initially contains a clique of size at least m, and is only ever assigned graphs that contain cliques of size at least m $(= i)$, G will always contains a clique of size at least m.
- As G initially has no cliques of size larger than m, and removing nodes from G cannot possibly introduce any new cliques, G can never have a clique larger than m.
- From the previous two results, the size of the maximal clique of G is m at all times during execution.
- I want to show that after the **for** loop, G will contain at most m nodes. Let G_1 be the value of G at the end of the **for** loop. Suppose, for contradiction, that it contains strictly more than m nodes. We know that the size of G_1's maximal clique is m; now, let K be one such clique. Since K is a subgraph of G_1, K has m nodes, and G_1 has more than m nodes, thus there must exist some node x that belongs to G_1 but not to K. However, x being in G_1 contradicts the construction of the algorithm, because x should have been removed by the **for** loop when $j = x$ (since, at this point in the **for** loop, removing j keeps K in G, so at this point G' contains K i.e. a clique of size i $(= m)$). This contradiction implies that our supposition is false i.e. that G_1 contains at most m nodes.
- So, overall, at the point where we return G, G is a subgraph of G_0, G contains a clique of size at least m, and G has at most m nodes. This implies that G is

a clique of G_0 of size m. Since the size of the maximal clique of G_0 is m, this implies that G is a maximal clique of G_0. So it is appropriate to return G.

To find the complexity of the algorithm, note:

- The **while** loop's number of iterations is bounded above by the size of the maximal clique of G_0. Also, the size of the maximal clique of G_0 is at most the number of nodes of G_0.. Since G_0 has n nodes, this means that the **while** loop runs at most n times.
- The **for** loop runs n times.
- Line 2 is done in $O(1)$.
- Line 3 is done in $O(T(n))$, and is repeated at most n times. It thus costs $O(nT(n))$ overall.
- Line 4 is done in $O(1)$ and is repeated at most n times. It thus costs $O(n)$ overall.
- Line 7 can be implemented in $O(n^2)$ in the worst case, for common representations of graphs (i.e. adjacency lists or adjacency matrices), and is repeated n times. It thus costs $O(n^3)$ overall.
- Line 8 is done in $O(T(n))$ and is repeated n times. It thus costs $O(nT(n))$ overall.
- Line 9 can be done in $O(1)$, and is repeated n times at most. It thus costs $O(n)$ overall.

So overall we have cost $O(n^3 + nT(n))$, but, since both terms of this sum are asymptotically bounded above by $n^2T(n)$ (as $T(n) \geq n \geq 0$ and so $n^2 * T(n) \geq n^2 * n = n^3$ and $n^2 * T(n) \geq n * T(n)$), the overall cost is also $O(n^2T(n))$, as desired. $\square$

NB: The linear search on line 3 can be replaced by a binary search, but, due to the check on line 8, this does not improve the complexity of the algorithm.