1. Question 35-1

Part (b)

Consider any packing of n object with sizes $s_1, s_2, \ldots, s_n$ into b bins. Let $S = \sum_{i=1}^{n} s_i$. Let $e_1, e_2, \ldots, e_b$ be the unused space in each bin and let $e = \sum_{i=1}^{b} e_i$.
Therefore,

$$\begin{aligned} b &= \frac{\text{total space in bins}}{\text{size of each bin}} \\ &= \frac{\text{space used in bins} + \text{space unused in bins}}{1} \\ &= S + e \\ &\geq S \end{aligned}$$

(since $e \geq 0$). Since b is an integer number, $b \geq \lceil S \rceil$. This is true in particular for the optimal solution: $b^* \geq \lceil S \rceil$.

Part (c)

The First-Fit heuristic places each object in the first available bin. Suppose by contradiction that two bins b_i and b_j are half full and suppose $i < j$. All objects in b_j would have fit into b_i, since both b_i and b_j are half full. Therefore, the First-Fit heuristic would have placed these objects in b_i. Therefore, it is not possible that two bins may be half empty under the First-Fit heuristic.

Part (d)

Consider a bin-packing solution provided by the First-Fit heuristic. Using similar notation as in part (b), we get

$$b = S + e = S + \sum_{i=1}^{b} e_i \tag{1}$$

We can assume without loss of generality that $b \geq 2$, for the case where $b = 1$ is easily satisfied.
If all the bins are at least half full, then $b = S + e \leq S + \frac{b}{2}$, which implies $b \leq 2S \leq \lceil 2S \rceil$.

By part (c), it remains to consider the case in which exactly one bin is less than half full. Let j be the index of the half full bin. For any other index $j' \neq j$, by the way the First-Fit heuristic works, $e_{j'} + e_j < 1$, for otherwise both contents would have been combined. Take $j' \neq j$. Thus,

$$e = \sum_{i=1}^{b} e_i = (\sum_{i=1 i\neq j,j'}^{b} e_i) + e_j + e_{j'} < (\sum_{i=1 i\neq j,j'}^{b} \frac{1}{2}) + 1 = \frac{b-2}{2} + 1 = \frac{b}{2} \quad (2)$$

Substituting (2) in (1), we get

$$b = S + e < S + \frac{b}{2}$$

So, $2b < 2S + b$, which implies $b < 2S \leq \lceil 2S \rceil$.

Part (e)

Let b be the number of bins used by the First-Fit heuristic and let b^* be the optimal number of bins.
By part (d), $b \leq \lceil 2S \rceil$.
By part (b), $\lceil S \rceil \leq b^*$.
Combining both inequalities, we get

$$b \leq \lceil 2S \rceil \leq 2\lceil S \rceil \leq 2b^*$$

Therefore, $\frac{b}{b^*} \leq 2$. That is, the approximation ratio is 2.

Part (f)

```
Algorithm First-Fit(n, s_1, s_2, ..., s_n):
    b ← 0
    Initialize data structures
    for i = 1 to n do
        Let j be first bin that can fit object i with size s_i (*)
        if j exists then
            Insert i at the list L[j] (list of objects in bin j)
            Update data structures I
        else
            b ← b + 1
            Insert i at the list L[b]
            Update data structures II
        endif
```

```
endfor
```

Depending on the data structure employed, the algorithm may be less or more efficient.

- The straightforward implementation would keep an array $B[j]$ to store the space left at bin j. The maximum size of the array is n. The initialization only needs to declare the array.

`Update data structures I:`	$B[j] \leftarrow B[j] - s_i$
`Update data structures II:`	$B[b] \leftarrow 1 - s_i$

  Step (*) can be done in $O(b)$ time. The total running time of this algorithm is $O(b \times n) \subseteq O(n^2)$.

- A more careful implementation takes $O(n\log b) \subseteq O(n\log n)$.

  We use a balanced binary search tree (such as an AVL tree, or a red-black tree) to store the free space in each of the bins. The key for the balanced binary search tree (BBST) is (e, i), where e is the empty space in bin i. If two bins have the same empty space, the one with the smallest index will be considered smalles.

  If the tree contains b elements, the following operations can be performed in $O(\log b)$ time:

  1. `insert(`e`, `k`)` = insert key (e, k) into the BBST
  2. `delete(`s`, `i`)` = delete and returns the smallest key larger than (s, i). It returns fail (say $(-\infty, 0)$) if there is no key larger than (s, i).

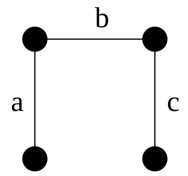
  To calculate an empty BBST takes constant time. The initialization only needs to create an empty BBST B.

  1. Step (*) can be accomplished by the following operation: (e, j) `=` B`.delete(`s_i`, 0)`
  2. `Update data structures I:` B`.insert(`$e - s_i$`, `j`)`
  3. `Update data structures II:` B`.insert(`$1 - s_i$`, `b`)`

Each of the above steps can be done in $O(\log b)$ basic steps. Therefore, the total running time for this algorithm is $O(n \times \log b) \subseteq O(n \log n)$.

Question 35-4

Part (a)



A maximal matching: $\{b\}$.
A maximum matching: $\{a, c\}$.

Part(b)
The trick of this solution is to make the complexity independent on the number of vertices, and also ensure that our algorithm does not run in $O(|E|^2)$ by carefully selecting the data structures. Suppose that we keep the number of edges constant and the number of vertices goes to infinity (adding a bunch of isolated vertices). The solution presented here would run in constant time. Another solution that we accepted for full marks runs in $O(|E| + |V|)$, but it does not strictly comply with the complexity requested. This alternative solution is also included.
The data structure used in this algorithm consists of:

- number of vertices and edges (2 integer numbers)
- an array of edges (each containing a pair of vertices)
- an array of vertices, each pointing to a list of edges incident to this vertex

```
Algorithm Greedy-Matching(G = (V, E)):
    for each e ∈ E do
        M(e) ← 1
    for each e ∈ E do
        if (M(e) = 1) then
            let v, w be the endpoints of e
```

```
            for all edges e' incident to v do
                if (e' ≠ e) then M(e') ← 0
            for all edges e' incident to w do
                if (e' ≠ e) then M(e') ← 0
        endif
    S ← ∅
    for all e ∈ E do
        if (M(e) = 1) then S ← S ∪ {e}
    return S
```

Note that under the RAM model, if the graph is given using the data structure above, the time complexity of the algorithm is independent on the number of vertices.

Justification that the algorithm runs in $O(|E|)$:

1. Loops 1-2, 11-12 clearly consist of $O(|E|)$ steps, each using constant time.

2. A rough look at loop 3-10 might lead us to conclude that it runs in $O(|E|^2)$. However, analyzing more carefully, we can reach a tighter bound. We claim that over all iterations of 3-10, lines 7 and 9 get executed $O(|E|)$ times. Let $e = \{v, w\}$ in step 4. Indeed, when $M(e) = 1$, we only have to go through the list of edges incident to vertices v and w once, since the first time we do that, v or w will not be incident to another edge e' such that $M(e') = 1$. So, each edge in E may be examined at most three times: once when it is tested in line 4, and possibly twice more if it is set to 0 at steps 7 and 9, if its endpoints are both incident to an edge in the matching. Overall, we have time $O(3|E|) = O(|E|)$ for steps 3-10, which concludes the proof of the $O(|E|)$ running time.

Note: Full justification should be given for the algorithm running in $O(|E|)$, since the obvious implementation runs in $O(|E|^2)$.

The following solution, which runs in $O(|V| + |E|)$, was also accepted for full marks: `Algorithm Greedy-Matching2(`$G = (V, E)$`):`

```
    declare an array A[|V|]
    for i = 1 to |V| do
        A[i] ← 0
```

```
M ← ∅
for each edge e = {u, v} ∈ E
    if (A[u] == 0 and A[v] == 0) then
        M ← M ∪ {e}
        A[u] ← 1
        A[v] ← 1
    endif
```

Part (c)
Let M be a matching and C be a vertex cover. Every edge of M must be covered by some vertex in C. Every vertex of C can cover at most one edge of M, since no two edges of M can be incident to C. Therefore, $|C| \geq |M|$.

Part (d)
Let $T' = V \setminus T$, and let $G' = (T', E')$ be the subgraph of G induced by T', i.e. the subgraph of G with vertex set T' and edge set E' being all edges in E with both ends in T'.
Then G' must be a set of isolated vertices, or more precisely, $E' = \emptyset$, by the following reason: if there were an edge $e \in E'$, e would connect $v_1, v_2 \in T'$, but then e could be added to M to form a larger matching. This is impossible, so $E' = \emptyset$.

Part (e)
First, note that T is a vertex cover. Indeed, by part (d) we know that $E' = \emptyset$, which means that for all $e \in E$, e is incident to a vertex in T.
Second, note that $|T| = 2|M|$. Indeed, for each $e_i = \{v_i, w_i\} \in M$, we have $v_i, w_i \in T$ and no two edges in M share a vertex, so the v_is and w_is are all distinct, and each edge in M "contributes" with 2 of them towards T.

Part (f)
The greedy algorithm in part (b) computes S, a maximal matching. Let M^* be a maximum matching.
By part (e), there exists a vertex cover C with $|C| = 2|S|$. On the other hand, part (c) implies that $|M^*| \leq |C|$. So $|M^*| \leq |C| = 2|S|$. Therefore, $\frac{|M^*|}{|S|} \leq 2$, i.e. the approximation ratio of the algorithm is 2.