Q1

a).

```
COMPARE(a,b)

  if SEGMENTS-INTERSECT(x and y coordinates of end points of a and b)then
    za, zb = z coordinates of the intersection point
    if za > zb then
      return ABOVE
    else
      return BELOW
  else
    return UNRELATED
```

b). Create a directed graph with vertices corresponding to the sticks. For each stick couple (*a, b*), if *a* is above *b* add an edge from *a* to *b*, if *a* is below *b* add an edge from *b* to *a*. Check if the graph is cyclic. If so, we cannot pick all of the sticks. If not, pick up the sticks in topological sorted order.

Q2

A 2-CNF formula can be converted into a directed graph $G = (V, E)$ as:

- For the variables in the formula $x_0, \dots, x_n$, denote vertex v_j for x_j, denote vertex $\overline{v_j}$ for $\neg x_j$
- For each clause, two edges can be constructed because the logical operation transition in the hints.

We claim that this formula is satisfiable if and only if no pair of complimentary literals are in the same strongly connected component of G. If there are paths from u to v and vice versa, then in any truth assignment the corresponding literals must have the same value since a path is a chain of implications.

Conversely, suppose no pair of complementary literals are in the same strongly connected component. Consider the DAG obtained by contracting each strongly connected component to a single vertex. This DAG induces a partial order, which we then extend to a total order using topological sort. For each x_i, if the component of v_i precedes the component of $\overline{v_i}$, set $x_i = 0$ else set $x_i = 1$. We claim that this is a valid truth assignment, i.e. that (1) all literals in the same component are assigned the same values and (2) if a component B is reachable from A then A, B can't be assigned 1, 0.

We first prove (1). Assume for the contrary that two literals l_1 and l_2 are in the same strongly connected component S but the strongly connected component containing $\neg l_1$ precedes S in the total order and the component containing $\neg l_2$ is preceded by S. Since l_1 and l_2 are in the same component $l_1 \rightarrow l_2$ and $l_2 \rightarrow l_1$. It also follows that the clauses $(l_1 \vee \neg l_2)$ and $(\neg l_1 \vee l_2)$ can be obtained. Hence, there must be a path from $\neg l_2$ to $\neg l_1$. This contradicts the total order.

We then prove (2). Assume for contradiction that there are two connected components A and B so that B is reachable from A, but our algorithm assigns 1 and 0 to A and B. Let l_a and l_b be literals in A and B respectively. Note that there must be a path from $\neg l_b$ to $\neg l_a$. Let $\bar{B}$ and $\bar{A}$ be the component of $\neg l_b$ and $\neg l_a$. Clearly, $\bar{B}$ has value 1 and $\bar{A}$ has value 0. In the total order B preceded $\bar{B}$ and $\bar{A}$ preceded A. This implies that there is a cycle in the total order.

It is obvious that this algorithm runs in polynomial time.

Q3

a). The decision problem can be summarized as: Given a graph $G = (V, E)$ and an integer K, is there a subset V' of V with K vertices so that V' is an independent set?

To prove it is NP complete, we first show that it is in NP. The verifier function is chosen as checking whether $|V'| = K$ adnd $V' \subseteq V$. For each edge $e = (u, v) \subseteq E$, it check that at most one of u and v is in V'. This can be done by checking that there is no edge between any pair of vertices in V'. It takes $O({V'}^2)$ time.

Then we prove it is NP hard by reducing clique to this problem. Given an undirected graph $G = (V, E)$, we can define the complement of G as $\bar{G} = (V, \bar{E})$, where $\bar{E} = \{(u, v) | (u, v) \notin E\}$. The following procedures are very similar to the proof of NP completeness of Vertex Cover problem in Theorem 34.12 of the textbook.

b). The algorithm is straightforward: K start from $|V|$, check whether K satisfies the problem by calling the black box. If not, decreasing K by 1 then repeat the previous step. If yes, done. It runs in $O(V)$ time.

c). It is easy to observe that if each vertex in G has degree 2, G is a simple cycle. Then an independent set of max size can be formed by selecting every other vertex along the cycle starting from any vertex until the next vertex to select is or is the neighbor of the start point. The size of this set is $\lfloor |V|/2 \rfloor$. The running time is $O(V)$.

d). For the case that G is a bipartite, the independent size of max size is the set of the side with the larger number of vertices in the bipartite. The correctness can be easily proved by contradiction. The running time is $O(V)$.

Q4

a). The decision problem of this scheduling problem is like: given an integer K, does there exist a scheduling of the tasks that gets profit no less than K?

b). Firstly, we can find a verifier F(x,y) in polynomial time. Given an instance $x = (p[1, \ldots, n], d[1, \ldots, n], t[1, \ldots, n])$, and a schedule $y = (s[1, \ldots, n])$,where task i is scheduled at time $s[1]$. It can be easily verified if they are not overlapping and the profit is at least K in $O(n^2)$ time.

Then we can reduce Subset Sum problem to this scheduling problem. For an instance $X = (x[1, \dots, n], K)$, we construct an scheduling instance $Y = (x[1, \dots, n], d[1, \dots, n], x[1, \dots, n])$, and $\forall i, d[i] = K$. This construction procedure takes time $O(n)$. Suppose there is a subset of elements of X such that the sum is exactly K, then there is a corresponding schedule of Y. As well, if there is a schedule in Y, then for the subset S of tasks that meets deadline, $\sum_{i \in S} x[i] \leq K$ because we know all tasks meets the deadlines and no tasks overlap. Then we can also have $\sum_{i \in S} x[i] \geq K$ because we have the profit limitation. Thus $\sum_{i \in S} x[i] = K$.

c). An easy solution of this problem is constructing the table in subproblem (d). Then you can choose the highest value in the row K.

d). we need to figure out is what our optimal subproblems are. We do this by considering an optimal schedule over jobs $a[1, \dots, n]$ that runs until time t. The optimal subproblems can be generated as: we order the jobs by y increasing deadline. When we consider job $a[i]$ finishing at any time t (we assume these are now in sorted order so it has the ith deadline) we can simply look back at the optimal way to schedule the i − 1 jobs and whether or not we add $a[i]$ to the schedule. We will also make the additional assumption that we leave no time gaps between the jobs. It is easy to argue that if we have a schedule with time gaps between jobs, we can also do the jobs in the same order with no time gaps and receive the same profit and possibly more. The actual algorithm is as follows. We keep a grid of the n jobs versus the time, which can run up till d[n] (since the jobs are now sorted by deadline). Notice that since the processing times are integers between 1 and n, the maximum time taken to complete all the jobs is at most n^2 . So, if we have any deadline which exceeds n^2, we can simply replace it by n^2. . Thus, our table is at most $n \times n^2$. Each cell in the table i, j will represent the maximum profit possible for scheduling the first I jobs in time exactly j, assuming that we have no gaps between our jobs.

The recursion form of the algorithm is like:

$$T[i,t] = max \begin{cases} T[i-1,t] & \\ T\big[i-1, t-t[i]\big] + p[i] & if\ t \leq d[i] \\ T[i-1, t-t[i]] & if\ t > d[i] \end{cases}$$

And the base case(the first row of the table) is:

$$T[1,t] = max \begin{cases} 0 & if\ t \neq t[1] \\ p[1] & if\ t = t[1] \leq d[1] \\ 0 & if\ t = t[1] > d[1] \end{cases}$$

After filling the whole table, we can search through the last row for the highest profit. The schedule can be traced back from the cell of the last row by maintaining pointers pointing to predecessors. The running time of this algorithm $O(n^3)$.

Q5

a). In order to prove NP-hardness of Bin-Packing, it is sufficient to reduce an NP-complete problem to Bin-Packing. Although it is hinted to reduce from subset-sum problem, it is much easier to reduce from Partition problem.

Partition problem is defined as follows: Given a set of numbers A={a_1, a_2,…,a_n }. Is there a subset of A, B, such that the sum of elements in B is equal to the sum of elements in A-B.

This problem can be reduced to Bin-Packing as follows:

Let $sumA = \sum_{i=1}^{n} a_i$.

Let we have a set of numbers {s_1, s_2,…,s_n } such that $s_i = 2*a_i/sumA$ for $1 \le i \le n$.

Then if {s_1, s_2,…,s_n } can be packed into 2 bins, A can be partitioned into 2.

b). Suppose we have a solution that with k bins where $k < \lceil S \rceil$. Since each bin can hold the objects with a total size not greater than 1, the total size of all objects is less than or equal to k. This is a contradiction. Therefore, the number of bins cannot be lower than$\lceil S \rceil$.

c). Suppose we have two bins with the size less than or equal to 0.5. Since first-fit puts an object to a bin if it fits to the remaining area of the bin, the objects of the second of these two bins could be put into the first bin while performing the heuristic. This is a contradiction.

d). Let the number of bins be k. We know that at least k-1 bins filled more than half. The last bin is greater than the empty space of all these k-1 bins. So if we put the last bin over one of those k-1 bins the total size of these two bins would be greater than 1. Therefore we have this equation: $(k-2)*0.5 + 1 < S$ which yields $k<2S\le\lceil 2S \rceil$.

e). From b and d, we have $\lceil S \rceil \le optimal \le first\_fit < \lceil 2S \rceil \le 2\lceil S \rceil$. Therefore first-fit solution cannot have 2 times more bins than the optimal solution.

f). We can use max winner tree to find the first fit and update the tree in O(lg$n$) time for each item. This makes the running time O($n$lg$n$).