CSCI 6470
Homework 2
Jiakuan Li

1.

The number of distinct outcome is $n + 1$ in the worst case. Therefore, the decision tree has $n + 1$ leaves. Moreover, there are at most 3 outcome per tree level. $(>, <, =)$. So the tree height will be $\log_3 n$. Since $\log_3 n$ is in $\Omega(\log_2 n)$, is the lower bound is $\Omega(\log_2 n)$.

2.

Algorithm 1 SORTBINARYBIT $A(n)$

```
$j = n - 1$
$i = 0$
**while** $j < i$ **do**
  **if** $A[i] = 0$ **then**
    $i + +$
  **else**
    $swap(A[i], A[j])$
    $j - -$
  **end if**
**end while**
```

(1) The worst case if the list is in reverse order. The distance i and j moves is equal to n, which is the length of the list. So the upper bound for this algorithm is $O(n)$.

(2) The lower bound of this algorithm is $\Omega(n)$.
This is the optimal solution for the problem.
The loop in this algorithm moves i or j per execution. Therefore, in order to get out from the loop, there are 3 solutions.

  a Move i for n times. j remains at the origin position. (best case)

  b Move j for n times. i remains at the origin position. (worst case)

  c Move i for k times and j for $(n - k)$ times. (average case).

No matter which condition was met. The total moves is n. For this algorithm, the upper bound is equal to lower bound, which is $\Omega(n)$.

3.

Build a binary sear tree to sort the list.
STEP 1:
Since only k distinct element in the list, so the tree has only k nodes.

Therefore, the insertion/search takes $O(\log_2 k)$.
Then we need to maintain a counter for those k elements in the list to count the appearance of each distinct element in the list.
To walk through the tree takes $O(n)$, and build the tree takes $O(\log_2 k)$.

STEP 2:
To print out the sorted list, we need to travel through the tree and combine the list with the counter.
The tree traversal takes $O(k)$, the combination takes $O(n)$.

The total time is $O(n \log_2 k) + O(n) + O(k) = O(n \log_2 k)$ since $k < n$.

4.

(1)

Algorithm 2 MINIMUMVC2 $G < V, E >$, *Result*

```
if $|G| \leq 1$ then
  return $\emptyset$
end if
select a random vertex
add the selected vertex to Result
remove the vertex and its edge(s) from the graph
return MINIMUMVC2($G$, Result)
```

(2)

Algorithm 3 MINIMUMVC2 $G < V, E >$, *Result*

```
select a vertex with most degrees.
add the selected vertex to Result
remove the vertex and its edge(s) from the graph
remove the vertex with 0 degree from the graph.
return MINIMUMVC2($G$, Result)
```

5.

(1) I will maintain a array to store the second fastest.
The base case is $sf[1]$ since $sf[0]$ does not have the "second fastest" path.

$$sf[1] = min(sf[0] + s_1 - switch, sf[0] + s_2 - switch)$$

This is actually the objective function for this algorithm.

(2) To find the top k path, the time complexity is $O(n)$ since we just need to walk through the station once.

6.

(1) Property of optimal substructure:
For each roll, select the greatest probability to roll the i^{th} number.

Property of overlapping subproblem:

For the i^{th} roll, the probability is based on the previous roll. The previous roll might use loaded/normal dice. While computing the i^{th} roll, we need to compare switching dice and not switching dice. This cause a overlapping on previous roll.

(2) $p_1(i) = max(p_2(i-1) \cdot probabilitty - fair \cdot switch - chance, p_1(i-2) * probablity_fair \cdot not - switch - chance)$
$p_2(i) = max(p_2(i-1) \cdot probability - loaded \cdot not - switch - chance, p_1(i-1) \cdot switch - chance \cdot probablity - loaded)$

p_1, p_2 represent the fair dice and loaded dice.

7.

Algorithm 4 MAXPROBABILITY $T[i,2]$, $A[n]$

```
(1) T[0,0] = chance of rolling A[0] for fair dice
    T[0,1] = chance of rolling A[0] for loaded dice
    for i = 1:Len(A) do
      p1 = chance of rolling $A[i]$ on fair dice
      p2 = chance of rolling $A[i]$ on loaded dice
      $T[i,0] = max(T[i-1,0] \cdot p1 \cdot not - switch - chance, T[i-1,1] \cdot switch - chance \cdot p1)$
      $T[i,1] = max(T[i-1,0] \cdot switch - chance \cdot p2, T[i-1,1] \cdot switch - chance \cdot p2)$
    end for
```

Algorithm 5 PRINTOUT $T[i,2]$, $A[n]$

```
(2) Res[n]
    for i = Len(A):0 do
      if $T[i,0] > T[i,1]$ then
        Res[i] = 0
      else
        Res[i] = 1
      end if
    end for
```