CS 515
Tenghuan Li; 933638707; liten@oregonstate.edu
Liangjie Zheng 934056203; zhengli@oregonstate.edu
Xingjian Su; 933647032; sux@oregonstate.edu

Problem 1:

For discarded bolt, this algorithm discards k-1 bolt when the k^{th} largest bolt is selected.
And the probability of selecting each bolt is 1/n.
Thus the expected number of discarded bolts is:

E[B] = 0*1/n + 1*1/n + 2*1/n + …… + (n-1)*1/n = (n-1)/2

For discarded nuts, if the k^{th} nut is the last nut to be discarded, in this case, the largest bolt should not be chosen, and the probability of this case is 1-1/n = (n-1)/n. For k is the k^{th} nut which is to be discarded(0<=k<(n-1)), the probability it being discarded is $1/n^2$.

Thus, the expected number of discarded nuts is:

E[D] = $0*1/n^2 + 1*1/n^2 + 2*1/n^2$ + …… + $(n-1)*1/n^2 + n*((n-1)/n)$
= $1/n^2$ *(0+1+2+…+n-1) + (n-1)
= (n-1)/2n + (n-1)
= $(2n^2-n-1)/2n$

Thus, the exact expected number of nut-bolt tests performed by this algorithm is:

E[test] = E[B+D] = E[B] + E[D] = $(3n^2-2n-1)/2n$

Prove:
In this algorithm, until the largest bolt and nut is found, one bolt or nut must be discarded for each test. Therefore, the sum of the expected number of discarded bolts and the expected number of discarded nuts is the expected number of tests. For bolts, because it is randomly selected every time, the probability of discarding k-1 bolts is 1/n. For nuts, the k^{th} nut is the last nut to be discarded only when the largest bolt is not selected, the probability of all other cases is equal, all are $1/n^2$.

Thus E[test] = E[B+D] = E[B] + E[D] = $(3n^2-2n-1)/2n$
Is correct.

Problem 2:

a.
As the problem, we can know that Dirk and Death play with a complete binary tree with 4^n leaves, each colored either black or white. This game will end after $2n$ moves, and Dirk moves first, the Death get the last turn.
When Dirk's turn he will use a fair coin to choose moves to left or right, so we can know that $P(head\ to\ left) = P(tails\ to\ right) = \frac{1}{2}$.
When Death's turn will choose the maximizes the probability that Dirk loses the game.
Because the Death will know that which subtree is the max probability that Dirk will loses this game, we can know only the two children of tree are white can let Dirk live.
So, we can get three case:

Case1: when two children of tree are white, return 1, otherwise return 0.
Case2: Dirk's turn use coin to pick go right or left.
Case3: Death will choose the higher probability let Dirk lose side of tree. It same meaning, Death will choose the lower probability let Dirk win this game.

Input a Tree, and the height is $2n$.

```
DT(Tree, 2n);
DT(Tree, height){
    if(height == 1){
        if(This node's two child are white)
            return 1;
        else
            return 0;
    }else{
        if(height %2 ==0){//Because Dirk move first
            return (DT(Tree->left, height-1) + DT(Tree->right, height-1)) * 1/2;
        }else{//now is height %2 !=0, so is Death turn to move.
            // find the subtree has lower probability of Dirk wins
            return min(DT(Tree->left, height-1), DT(Tree->right, height-1));
        }
    }
}
```

b.

In this problem, we can know the Death is use same way as Dirk to move.

So, the basic case is different from a problem:

Case1: if the two subtrees are white, return 1, otherwise return 0.

Case2: When Death or Dirk turn, they use same way to choose how to move, to use a fair coin.

Input a Tree, the height is 2n.

```
DT(Tree, 2n);
DT(Tree, height){
    if(height == 1){
        if(This node's two child are white)
            return 1;
        else
            return 0;
    }else{
        return (DT(Tree->left, height-1) + DT(Tree->right, height-1)) * 1/2;
    }
}
```

Problem 3:

a.

As the skip list have l layers, we can know the next layer with probability is 1/2.

Assume the expected number is N, and the basic layer has n node. So, we can know the next layer have $\frac{1}{2} * n$ nodes. Thus, for all layer the expected number of nodes is N.

So, $N \leq n + \frac{1}{2} * n + \frac{1}{4} * n + \cdots$. When each layer has the half number of nodes as previous layer, we know this skip list have l layers.

$$N \leq n * \left(1 + \frac{1}{2} + \frac{1}{4} + \cdots\right) = n * \left(\frac{1-\left(\frac{1}{2}\right)^{l}}{1-\frac{1}{2}}\right) = 2n * (1 - \left(\frac{1}{2}\right)^{l}).$$

Then, we can now the expected number of nodes is $O(n)$.

b.

Assume the high probability is t, we can know $0 < t < 1$. N is the expected number of all node, So, the basic layer has n node, on the l layer the $N(l) = n * t^l$.
As a part, we can know that:

$N = \max N(l) \leq N(1) + N(2) + N(3) + \cdots + N(l)$.

$N \leq \sum_{i=1}^{l} N(i) = n * (t^1 + t^2 + \cdots + t^l)$.

$N \leq n * \frac{t(1-t^l)}{1-t}$.

Since $0 < t < 1$, we can know $\frac{t(1-t^l)}{1-t}$ is constant. Then, we can know the expected number of nodes is $O(n)$ with high probability.

c.

For this question, we need to insert a new key in skip list, call the new key X. That meaning we need to find all key before this new key is $< X$, and all key after this new key is $> X$.
This insert we need to start with the Top layer and go through all layer to the original list. Assume a new key can be insert in skip list is $O(k)$.(k is an assume number we don't know)

Assume $L(x)$ be the number of levels of the skip list that contains the search key X. If the search key X appears on the level l. We can get $\Pr[L(x) \geq l] = \frac{1}{2^l}$. So, this skip list has the least l levels if and only if $L(x) \geq l$ for at least of the n search keys. We can get:

$$\Pr[L \geq l] = \Pr[(L(1) \geq l) \vee (L(2) \geq l) \vee \ldots \vee (L(n) \geq l)]$$

$$\Rightarrow \Pr[L \geq l] \leq \sum_{x=1}^{n} \Pr[L(x) \geq l] = n * \Pr[L(x) \geq l] = \frac{n}{2^l}$$

When any constant c>1, we can get that $\Pr[L > c \log n] \leq \frac{1}{n^{c-1}}$. That is the high probability with a skip list has $O(\log n)$ $levels$. So, we can know the $k = \log n$.
When we insert a new key in a skip list is $O(\log n)$ time with high probability.

Problem 4:

a.

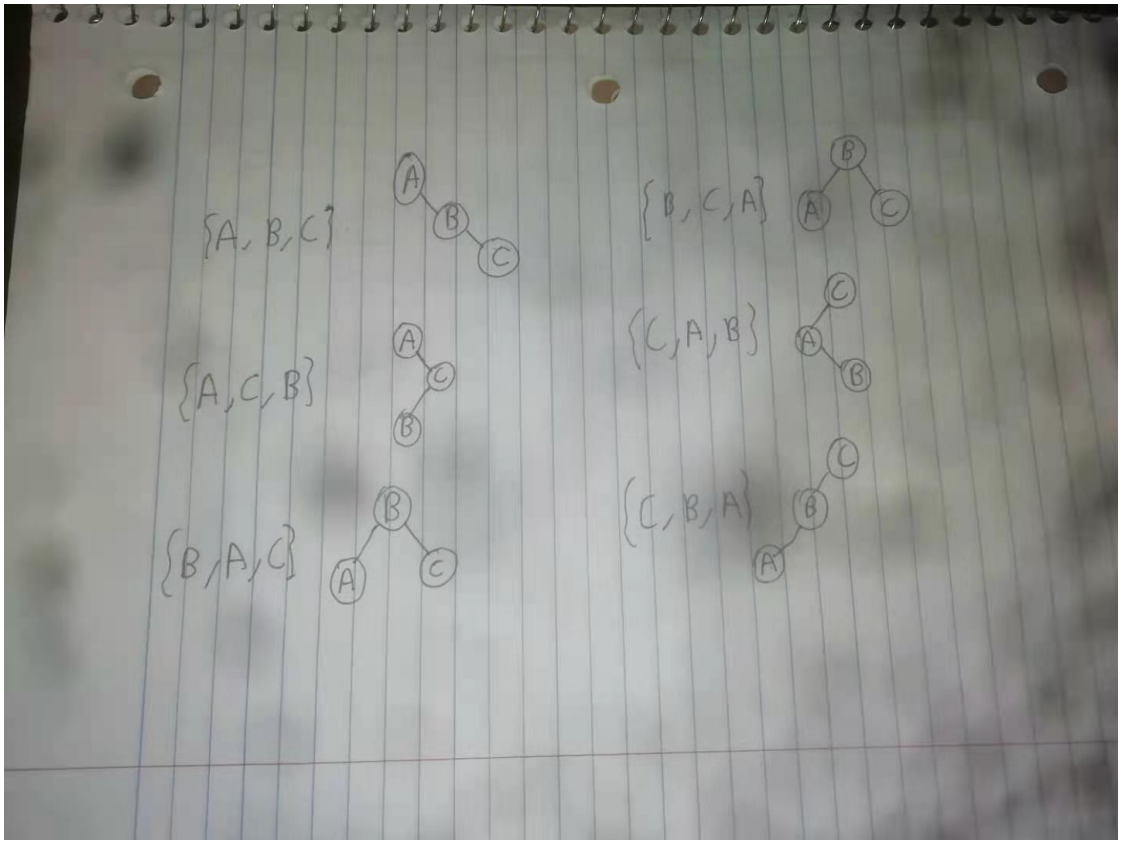
Assume that T is a binary tree with n nodes, because each node has 2degree of freedom to connect children, there are at most 2n degree of freedom. And because each edge need to connect two nodes, there are n-1 edges for this binary tree.

Thus, for T we have 2n - (n-1) = n+1 degree of freedom.

When the number of nodes in T increases from n to n+1, there are n+1 possibilities for this node, and there is only one case in the spine, that is, the new node is at the bottom left of T.
Thus: E[F(n+1)] = E[F(n)] + 1/(n+1)
We know for n=1, E[F(1)] = 1, for n=2, E[F(2)] = 1+1/2; (base case)
So, E[F(n)] = 1 + 1/2 + 1/3 +.....+ 1/n

b.

For three nodes with keys A, B, C, we can distribute these three nodes to a treap, and get 6 layouts. They are {A, B, C}, {A, C, B}, {B, A, C}, {B, C, A}, {C, A, B}, {C, B, A}.



For the 6 layout, we get P(B is leaf) = 2/6, P(A is leaf) = 1/2, P(C is leaf) = 1/2.

Suppose the K is the key of one node, and K-1 and K+1 are two nodes with closest keys.
If K-1 is Null, when K+1 has higher priority, K is a leaf;
If K+1 is Null, when K-1 has higher priority, K is a leaf;
In other case, when K-1 and K+1 has higher priority, K is a leaf.

These cases can be generalized to treap with more than three nodes, just insert a new node in the set {A, B, C}(random), no matter how many nodes are between K-1 and K, ones between K and K+1, this will not affect the judgement.

Thus, for a set of nodes with keys {K1, K2…..Kn}, the probability of each node to be a leaf is {1/2, 1/3, 1/3,….,1/3,1/2}. The expectation of the leaves in a treap is

$$1/2 + \sum_{2}^{n-1} 1/3 + 1/2 = \begin{cases} 1, & n = 1 \\ 1, & n = 2 \\ (n+1)/3, & n < 2 \end{cases}$$

c.
This question is similar to question b.
Suppose the K is the key of one node, and K-1 and K+1 are two nodes with closest keys. If K+1 and K-1 are existing, when K-1 and K+1 has lower priority, K has two children.

These cases also can be generalized to treaps with more than three nodes, just insert a new node in the set {A, B, C}, no matter how many nodes are between K-1 and K, ones between K and K+1, this will not affect the judgement.

And for the 6 layout about {A,B,C}(random), we get P(B has 2 children) = 2/6, P(A has 2 children) =0, P(C has 2 children) = 0.

Thus, for a set of nodes with keys {K1, K2…..Kn}, the probability of each node to be a leaf is {0, 1/3, 1/3,….,1/3,0}. The expectation of the leaves in a treap is

$$\sum_{2}^{n-1} 1/3 = \begin{cases} 0, & n = 1 \\ 0, & n = 2 \\ (n-2)/3, & n < 2 \end{cases}$$