Homework 4 Hints

1. Ingenuous Cubrency:

   ```
   //Counts the ways of making amount using
   //Cubes x^3 with x in the range [1,last]
   static long count(int amount, int last);
   static long count(int amount) { return count(amount,21); }
   ```

2. Diving For Gold:

   ```
   //ds: The depths of each treasure
   //vs: The values of each treasure
   //w: The constant from the problem (trip i costs 3*w*ds[i] air)
   //Computes the maximum value you can achieve with treasures
   //in the interval [pos,length(ds)-1] and t air left.
   static int solve(int[] ds, int[] vs, int w, int pos, int t);
   ```

   To reconstruct the choices made, we can use the solve function itself:

   ```
   static int buildOut(int[] ds, int[] vs, int w, int pos,
                            int t, StringBuilder sb)
   {
     if (pos == ds.length) return 0;
     int left = t-3*w*ds[pos];
     if (left >= 0 &&
     solve(ds,vs,w,pos+1,left)+vs[pos] == solve(ds,vs,w,pos,t))
     {
       sb.append(ds[pos]).append('␣')
         .append(vs[pos]).append('\n');
       return buildOut(ds,vs,w,pos+1,left,sb)+1;
     } else return buildOut(ds,vs,w,pos+1,t,sb);
   }
   ```

   Then you can just call:

   ```
   System.out.println(solve(ds,vs,w,0,t));
   StringBuilder sb = new StringBuilder();
   System.out.println(buildOut(ds,vs,w,0,t,sb));
   System.out.print(sb);
   ```

3. Garbage Heap: We can break this problem down into the following tasks. If the heap is stored in an **int** [][][] heap lets consider the first dimension to denote the height, and the remaining two to denote the length and width.

(a) Construct a separate array **int** [][][] sums. At each fixed height, we must use sums to compute the sum over a rectangle at that height in $O(1)$ time.

(b) Using 4 for loops, loop over all possible opposing corners of base rectangles.

(c) Then use a 5th loop to implement Kadane's algorithm in the height direction on the rectangular pillar with base determined by your first 4 loops.

4. Chest of Drawers:

```
//Assume higher drawers are labeled with higher indices.
//L: 1 or 0 determining if region above drawer n is
   locked/safe or not.
//Counts the number of valid configurations using drawers [1,n]
//and requiring exactly s safe drawers with L as above.
static long solve(int L, int n, int s);
```