Recursive Algorithm Analysis Worksheet

Name:_____

1. Solve each of the following recurrence relations

   (a) $T(n) = 2T(n/2) + 1$, $T(1) = 1$.

   (b) $T(n) = 2T(n-1) + n$, $T(1) = 1$.

   (c) $T(n) = 2T(n/2) + n^2$, $T(1) = 1$.

2. Find and prove a tight bound on the worst-case running time of each of the following algorithms (most or all of which should be familiar to you). Some will require a recurrence relation, and others will not. For those that require a recurrence relation, check with me before you solve it to make sure it is correct.

   (a) Give the running time of `BinarySearch(A,0,n-1,value)`.

```
int BinarySearch(const int *A,int First,int Last,int Value) {
     if(Last>=First) {
         int mid=(Last+First)/2;
         if(Value==A[mid])      return mid;
         else if(Value<A[mid])  return BinarySearch(A,First,mid-1,Value);
         else                   return BinarySearch(A,mid+1,Last,Value);
     }
     else { return -1; }
}
```

   (b) Give the running time of `Solve_Hanoi(n,0,1,2)`. *Note:* The final three parameters are not important when performing your analysis, but are very important to the correctness of the algorithm.

```
// Moves the top N discs from peg source to peg destination, using
// peg spare as needed.  (This is not a full implementation, but it
// has the same complexity as the full version.)
void Solve_Hanoi(int N,int source,int destination,int spare) {
    if(N==1) {
       Move_Disc(source,destination); // Takes constant time.
    } else {
       Solve_Hanoi(N-1,source,spare,destination);
       Solve_Hanoi(1,source,destination,spare);
       Solve_Hanoi(N-1,spare,destination,source);
    }
```

(c) Give the running time of `merge(A,0,(n-1)/2,n-1)`.

```
void merge(int A[],int L,int m,int R) {
    int size=R-L+1;
    int mid=m-L+1;
    int *B=new int[size];
    for(int i=0;i<mid;i++)
        B[i]=A[L+i];
    for(int j=mid;j<size;j++)
        B[j]=A[R-j+mid];
    int i=0;
    int j=size-1;
    for(int k=L;k<=R;k++)
       if(B[i]<B[j]) {
           A[k]=B[i];
           i++;
       } else {
           A[k]=B[j];
           j--;
       }
    delete []B;
}
```

(d) Give the running time of `mergesort(A,0,n-1)`.

```
void mergesort(int A[],int L,int R) {
   if(L<R) {
       int M=(R+L)/2;
       mergesort(A,L,M);
       mergesort(A,M+1,R);
       merge(A,L,M,R); // See (b) above for complexity of this
   }
}
```

(e) Give the running time of `stoogeSort(A,0,n-1)`.

```
void stoogeSort(int[] A,int L,int R){
    if(A[R-1]<A[L]) {  // Swap first and last element
        int temp=A[L]; // if they are out of order
        A[L]=A[R-1];
        A[R-1]=temp;
    }
    if(R-L>1){ // If the list has at least 2 elements
        int third=len/3;
        stoogeSort(A,L,R-third); // Sort first two-thirds
        stoogeSort(A,L+third,R); // Sort last two-thirds
        stoogeSort(A,L,R-third); // Sort first two-thirds again
    }
}
```

3. Give and solve a recurrence relation for the description of the following algorithm: Given an input array of size n, create 5 separate arrays, each one third of the size of the original array. This takes linear time in n to accomplish. Then call the same algorithm on each of the 5 arrays.

4. Solve the recurrence relation $T(n) = T(n/2) + T(n/3) + n$, $T(0) = 1$.