Analysis of Algorithms            Comprehensive Examination, Fall 2001

Sign the exam with your student number - not your name    _______________________

Answer all questions to the best of your ability.

1. (30 pts) The function addodds() receives a start index (min), an end index (max), and a data array (ele) and sums up all odd numbers in the array between the start and end indices. The function returns the resulting sum. Provide a time complexity analysis of the addodds() algorithm and a big-O approximation of its asymptotic growth. Assume the initial call is: addodds(0, n-1, ele); and that n = 2^p for some power p.

\[
\begin{align*}
\text{Add Odds} 1) &\equiv \\
\text{int addodds(int min, int max, int *ele) \{} \\
& \quad \text{int x,y;}
& \quad \text{if (min==max) \{} \quad \text{/* Base case*/}
& \quad & \quad \text{if ((1==ele[\text{min}]\%2)) \{} \quad \text{/* Checking for odd number */}
& \quad & \quad & \quad \text{return ele[\text{min}];} \quad \text{/* Return odds*/}
& \quad & \quad \text{\}}
& \quad & \quad \text{\} \quad \text{/* Return 0's if even*/}
& \quad \text{else \{} \quad \text{/* Recursive case*/}
& \quad & \quad \text{x=addodds(min,(min+max)/2,ele);} \quad \text{/* Recursive call 1st half*/}
& \quad & \quad \text{y=addodds(1+(min+max)/2,max,ele);} \quad \text{/* 2nd half*/}
& \quad & \quad \text{return x+y;} \quad \text{/* Return the sum*/}
& \quad \text{\}}
\end{align*}
\]
2. (30 pts) Here’s a snippet of code from Knuth’s Stanford GraphBase:

```c
2  (Knuth Snippet 2)
    nn[0]=nn[1]=1;
    for (k=2;k<=n;k++) { nn[k]=0; }
    for (j=2;j<=max_height;j++) {
        for (k=n-1;k>0;k--) {
            for (s=0,i=k;i>=0;i--) { s+=nn[i]*nn[k-i]; } /* overflow impossible */
            nn[k+1]=s;
        }
    }
    nverts=nn[n];
```

Express the time complexity of the code as a function of $n$ and $\text{max\_height}$. 
3. (30 pts) Solve the recurrence relation

\[ T(n) = T(n - 1) + 2^n + n + 1 \quad T(0) = 0. \]
4. (10 pts) Suppose an array $X[0..n-1]$ has been sprinkled with random real numbers chosen uniformly over the range $[0, 1]$, and consider the code fragment:

```java
float max = X[1];
for (int i = 2; i < n; i++) {
    if (max < X[i]) {
        max = X[i];
    }
}
```

What is the expected number of times that the variable `max` will be re-set? That is, what is the average time complexity of the statement `max = X[i]` that is inside the `for` loop?