ECE G205 Fundamentals of Computer Engineering
Fall 2004

Exercises in Preparation to the Midterm

- **Problem # 1**. Write two functions *summa* and *summaRec*, one iterative and one recursive, that given as input an array of N integer numbers returns the sum of the "even elements" of the array (i.e., the elements whose position is even, such as the 0th element of the array, the 2nd, the 4th and so on).

  Possible solutions are given by the following function:

```
int summa( int E[], int N ) {
  int sum = E[ 0 ];
  for( int i = 2; i < N; i += 2 )
    sum += E[ i ];
  return sum;
}

int summaRec( int E[], int i ) {
  if ( i == 0 )
    return E[ i ];
  else
    return E[ i ] + summaRec( E, i - 2 );
}
```

  In the case of the recursive version the call from `main` should be either `summaRec( A, n - 1 )` (n odd) or `summaRec( A, n - 2 )` (n even), where A is the given array and n its size.

- **Problem # 2**. *True or False.* Write either **T** or **F** at the right of each expression.

  1. $n+3 \in \Omega(n)$ **T**
  2. $n+3 \in O(n^2)$. **T**
  3. $2^{n+1} \in O(n+1)$. **F**
  4. $2^{n+1} \in \Theta(2^n)$. **T**
  5. $\sum_{i=1}^{n} i^h \in O(n^{h+1})$, $h \geq 0$. **T**

  The first four questions can be proven by definition, i.e., producing the required constants. The fifth one can be proven by induction on h. The induction base case ($h = 0$) is easily verified: $\sum_{i=1}^{n} i^0 = \sum_{i=1}^{n} 1 = n \in O(n^1) = O(n)$. Let us assume that the limitation holds for a generic $h > 0$, and let us consider a summation up to $h+1$:

$$\sum_{i=1}^{n} i^{h+1} = \sum_{i=1}^{n} i^h i \leq \sum_{i=1}^{n} i^h n = n \sum_{i=1}^{n} i^h.$$

  Since, by inductive hypothesis, is $\sum_{i=1}^{n} i^h \in O(n^{h+1})$ we have that $\sum_{i=1}^{n} i^{h+1} \in O(n^{h+2})$.

- **Problem # 3**. Write a recursive C++ procedure to compute the maximum among the n elements of an array of integers according to a "balanced" divide and conquer technique, i.e., at any recursive call the search range in the array should be halved.

  The following is a possible solution. (The solution makes use of the C++ function `max` from the library `algo.h` that returns the maximum between two numbers.)

```
int maxDC( int l, int r, int A[] ) {
  if ( r - l <= 1 )
    return max( A[ l ], A[ r ] );
  else
    return max( maxDC( l, ( ( l + r ) / 2 ), A ),
                maxDC( ( ( l + r ) / 2 ) + 1, r, A ) );
}
```

  The time complexity of the function is expressed by the following recursive equation.

$$T(n) = 2T(\frac{n}{2}) + 1$$

  with base case $T(1) = 1$. This recursive equation can be solved by repeated substitutions.

$$\begin{aligned} T(n) &= 2T(\frac{n}{2}) + 1 \\ &= 2(2T(\frac{n}{2^2}) + 1) + 1 \\ &= 2^2 T(\frac{n}{2^2}) + 2 + 1 \\ &= 2^2(2T(\frac{n}{2^3}) + 1) + 2 + 1 \\ &= 2^3 T(\frac{n}{2^3}) + 2^2 + 2 + 1 \\ &= \dots \\ &= 2^k T(\frac{n}{2^k}) + 2^{k-1} + \dots + 2^2 + 2 + 1 \end{aligned}$$

  The sum $2^{k-1} + \dots + 2^2 + 2 + 1$ is a particular case of the geometric series

$$\sum_{i=0}^{m} q^i = \frac{q^{m+1} - 1}{q - 1}.$$

  Hence, when $q = 2$ it is $2^{k-1} + \dots + 2^2 + 2 + 1 = 2^k - 1$, and $T(n) = 2^k T(\frac{n}{2^k}) + 2^k - 1$. When $k = \log n$, remembering that $T(1) = 1$, we have:

$$\begin{aligned} T(n) &= 2^k T(\frac{n}{2^k}) + 2^k - 1 \\ &= 2^{\log n} T(\frac{n}{2^{\log n}}) + 2^{\log n} - 1 \\ &= nT(1) + n - 1 \in O(n). \end{aligned}$$

- **Problem # 4**. Write an *optimal* C++ function that, given an integer $n \geq 0$, prints out all powers of 5 up to n.

  A lower bound to the problem is given by the number of powers of 5 that are printed out (any algorithm that solves this problem must print them). Given $n \geq 0$, the powers of 5 that are $\leq n$ are:

  $$1, 5, 5^2, \ldots, 5^k \leq n,$$

  and they are $k \leq \log_5 n$. Thus, a lower bound for the problem is $\Omega(\log n)$.

  The following function solves the problem.

```
void fifthPower( int n ) {
  int power = 1;
  while ( power < n + 1 ) {
    cout << power << " ";
    power *= 5;
  }
}
```

  Since the while loop is executed $\Theta(\log n)$ times, the given function is optimal.

- **Problem # 5**. Prove that $\lceil \log n \rceil \in O(n)$.

  The proof proceeds by definition, i.e., we have to show that there exist two constant $c > 0$ and $n_0 > 0$ such that $\lceil \log n \rceil \leq cn$ for each $n \geq n_0$. By looking at $\lceil \log n \rceil$ for small values of n it appears that for all $n \geq 1$, $\lceil \log n \rceil \leq n$. The proof that this holds for every $n \geq 1$ is by induction on n. The claim is certainly true for $n = 1$, $\lceil \log 1 \rceil = 0 < 1$. Now, suppose that $n > 1$, and that (induction hypothesis) $\lceil \log(n-1) \rceil \leq n - 1$. Then,

  $$\begin{aligned} \lceil \log n \rceil &\leq \lceil \log(n-1) \rceil + 1 \\ &\leq (n-1) + 1 \text{(by inductive hypothesis)} \\ &= n. \end{aligned}$$

  Hence, we can take $c = 1$ and $n_0 = 1$.