CSC 212: Data Structures and Abstractions
Spring 2019
University of Rhode Island

Weekly Problem Set #5

Due Thursday 3/28 at the beginning of class. Please turn in neat, and organized, answers hand-written on standard-sized paper **without any fringe**. At the top of each sheet you hand in, please write your name, and ID.

1 Recurrences

1. Find a closed-form equivalent of the following recurrences:

   (a) The Towers of Hanoi:

$$T(0) = 0; T(n) = 2T(n-1) + 1$$

$$\begin{aligned} T(n) &= 2T(n-1) + 1 \\ &= 2(2T(n-2) + 1) + 1 \\ &= 4T(n-2) + 3 \\ &= 4(2T(n-3) + 1) + 3 \\ &= 8T(n-3) + 7 \\ &= \ldots \end{aligned} \tag{1}$$

   This pattern can be written as follows:

$$T(n) = 2^k T(n-k) + (2^k - 1)$$

   Unrolling n times would yield: $T(n) = 2^n T(0) + (2^n - 1)$ Plugging in the base case $T(0) = 0$ gives us $T(n) = 2^n - 1$

   (b) Merge Sort:

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

$$\begin{aligned} T(n) &= 2T(\frac{n}{2}) + n \\ &= 2(2T(\frac{n}{4}) + \frac{n}{2}) + n \\ &= 4T(\frac{n}{4}) + 2n \\ &= 8T(\frac{n}{8}) + 3n \\ &= \ldots \end{aligned} \tag{2}$$

   This pattern can be written as follows:

$$T(n) = 2^k T(\frac{n}{2}) + kn$$

   Becoming trivial when $\frac{n}{2^k} = 1$ or $k = \log_2 n$ Putting it all together:

$$T(n) = nT(1) + n \log_2 n = n \log_2 n + n$$

(c) Generic:

$$T(0) = 1; T(n) = T(n-1) + 2^n$$

$$\begin{aligned}
T(n) &= T(n-1) + 2^n \\
T(n-1) &= T(n-2) + 2^{n-1} + 2^n \\
&= T(n-3) + 2^{n-2} + 2^{n-1} + 2^n \\
&= T(n-4) + 2^{n-3} + 2^{n-2} + 2^{n-1} + 2^n \\
&= \dots \\
&= T(n-k) + \sum_{i=n-k+1}^{n} (2^i) \\
&= T(n-n) + \sum_{i=1}^{n} (2^i) \\
&= 1 + 2^{n+1} - 2 = 2^{n+1} - 1
\end{aligned} \tag{3}$$

(d) Generic:

$$T(1) = 1; T(n) = T(\frac{n}{3}) + 1$$

$$\begin{aligned}
T(n) &= T(\frac{n}{3}) + 1 \\
T(\frac{n}{3}) &= T((\frac{n}{3})/3) + 1 + 1 = T(\frac{n}{9}) + 2 \\
T(\frac{n}{9}) &= T(\frac{n}{27}) + 3 \\
T(\frac{n}{27}) &= T(\frac{n}{81}) + 4 \\
&= T(\frac{n}{3^k}) + k \\
\text{Finding constants: } & \frac{n}{3^k} = 1 \\
& n = 3^k \\
& k = log_3 n \\
& T(n) = 1 + \log_3 n
\end{aligned} \tag{4}$$

2 Merge Sort

1. Determine the running-time of merge sort for a) sorted input; b) reverse-ordered input; c) random input; d) all identical input. Justify your answers.
   Merge Sort is guaranteed $O(n \log n)$ for all cases. The natural variant supports $O(n)$ for already sorted inputs.

2. Show the steps to sort the following array using Merge Sort: 6 1 7 11 4 10 2 5 9 3 8

```
[6 1 7 11 4 10] [2 5 9 3 8]
[6 1 7] [11 4 10] [2 5 9] [3 8]
[6 1] [7] [11 4] [10] [2 5] [9] [3] [8]
```

```
[1] [6] [7] [11] [4] [10] [2] [5] [9] [3 8]
[1 6] [7] [4 11] [10] [2 5] [9] [3 8]
[1 6 7] [4 10 11] [2 5 9] [3 8]
[1 4 6 7 10 11] [2 3 5 8 9]
[1 2 3 4 5 6 7 8 9 10 11]
```

3 Unimodal Array

1. Write a recursive algorithm to find the maximum of a weakly unimodal array of integers given the array and its start and end indices.

```
int searchUnimodal(int* array, unsigned start, unsigned end)
{
    if(start - end == 0)
        return array[start];
    unsigned i = (start+end)/2;
    if(array[i - 1] < array[i] && array[i + 1] > array[i])
    {
        return array[i];
    }
    else
    {
        int max1 = searchUnimodal(array, start, start > i-1 ? start : i-1);
        int max2 = searchUnimodal(array, end < i + 1 ? end : i+1, end);
        int maxSub = max1 > max2 ? max1 : max2;
        return maxSub > array[i] ? maxSub : array[i];
    }
}
```

2. For the following recursive function, determine the number of times foo is run given an initial call of foo(3) and foo(5). What is the general formula for the number of calls; foo(n)? (Remember to count the initial call of foo also)

```
void foo(unsigned n) {
    if(n > 0) {
        foo(n/2);
        foo(n/2);
        foo(n/2);
    }
}
```

$$\sum_{i=0}^{\log_2 n} 3^i$$

`drawing out a call tree can show that this is true, so long as we take the ceiling of log and not the floor eg.`

```
                          foo(3)
        foo(1)                  foo(1)                      foo(1)
foo(0)  foo(0) foo(0)   foo(0)  foo(0)  foo(0)     foo(0)  foo(0)  foo(0)
```