CSCI-2725 Homework 2

Yitao Tian

March 6, 2023

Extra credit: There is 5 percentage extra credit if you don't submit hand-written homework. You can use LaTeXor any other tool to write your homework.

1. (5 points each) Look at the pseudocode below, and find the Big-O estimate or worst case complexity of the following recursive functions.

   Hint: Write recurrence relation for the functions and solve it using Master's Theorem.

```
void fun(int n, int m) {          // T(l) =
  if (m > n) return;
  System.out.print("m = " + m); // 1+
  fun(n, m + 2);                  // T(l − 2)
}
```

Thus, the recurrence relation for the above function is

$$T(l) = 1 + T(l-2)(\text{ letting } l := m - n - 1), \quad T(0) = 1.$$

Fitting this to the hypothesis of the *Subtract and Conquer* version of the Master Theorem, we have $a = 1$, $b = 2$, and $f(l) \in O(1) = O(l^d)$, with $d = 0$.

By the Subtract and Conquer version, we have that:

$$\text{Since } a = 1, \text{ then } T(l) \in O(l^{d+1}) = O(l). \qquad \blacksquare$$

```c
float fun(int a[], int i) {                    // T(i) =
  if (i == 0) {
    return a[0];
  }
  if (i > 0) {
    return (i * fun(a, i - 1) + a[i])/(i + 1); // T(i − 1)
  }
}
```

Thus, the recurrence relation for the above function is

$$T(i) = T(i-1), \quad T(0) = 1.$$

Putting this into the hypothesis of the S&C version, we have $a = 1$, $b = 1$, and $f(i) \in O(i^d)$, with $d = 0$.

Applying the S&C version, we have that

$$\text{Since } a = 1, \text{ then } T(i) \in O(n^{d+1}) = O(i). \qquad \blacksquare$$

Assume that the function "random" has a complexity of $O(n)$.

```c
void fun(int array[], int a, int b) { // T(n) =
  if (a > b) return;
  mid = (a + b)/2;                     // 1+
  fun(array, a, mid);                  // T(n/2)+
  fun(array, mid + 1, b);              // T(n/2)+
  random(array, a, mid, b);            // g(n)
}
```

Thus, the recurrence relation for the above is

$$T(n) = 2T(n/2)+f(n)(\text{ letting } n := a-b-1 \text{ and } f(n) := 1+g(n), \text{ with } g(n) \in O(n)), T(0) = 1.$$

We cannot use the S&C version of the Master Theorem, but instead the *Divide and Conquer* version. Putting it into the hypothesis of the D&C, we have that $a = 2$, $b = 2$, and $f(n) \in \Theta(n^1) = \Theta(n^{\log_b(a)})$.

Applying the D&C, we have

$$\text{Since } f(n) \in \Theta(n^1) = \Theta(n^{\log_b a}), T(n) \in \Theta(n \log n). \qquad ■$$

```
int fun(int x, int n) {                          // T(n) =
  if (n == 0) return 1;
  if (x == 0) return 0;
  if (n == 1) return x;
  if (n % 2 == 0) return fun(x * x, n / 2);
  else return x * fun(x * x, n / 2);             // T(n/2)
}
```

Thus, the recurrence relation for the above function is

$$T(n) = T(n/2), \quad T(0) = 1.$$

In terms of the hypothesis of the D&C version, we have $a = 1$, $b = 2$, and $f(n) \in O(1) = O(n^{\log_b a - \varepsilon})$, for all $\varepsilon = 0$.

Applying the D&C, we have

$$\text{Since } f(n) \in O(n^{\log_b a-\varepsilon}), \text{ then } T(n) \in \Theta(n^0).$$

2. (10 points) Consider a recurrence relation given by following.

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

Show that $T(n) = n2^n$ is the solution of the above equation. If you need a base case, use $T(0) = 0$.

Answer: Using backward substitution, we have the following:

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

Undertaking a proof by induction, let k be the numbering of Ps as follows:

(a) $P(0)$ is the statement that $T(n)$ equals an expression in terms of $T(n-1)$, namely, $T(n) = 2T(n-1) + 2^n$;

(b) $P(k)$ is the statement that $T(n)$ equals some expression in terms of $T(n-1-k)$, namely, $T(n) = 2(2(\ldots 2T(n-1-k) + 2^n \ldots) + 2^n) + 2^n$;

(c) and of course, $P(k+1)$ is one in terms of $T(n-1-(k+1))$.

From the recurrence relation derived from the source code above, $P(0)$ follows.

Since $T(n)$ represents a recursion, we have that: For all $n_0 \in [0, n]$, $T(n_0) = 2T(n_0 - 1) + 2^n$. Therefore, $T(n_0) = 2T(n_0 - 1) + 2^n$. Noting the obvious, if $n_0 = n - 1 - k$, then $n_0 - 1 = n - 1 - (k+1)$. $P(k) \rightarrow P(k+1)$ follows immediately.

Thus, by induction, we have that $P(k)$ is true for all $k \in [0, n-1]$.

From the definition of $P(k)$, we have

$$\begin{aligned}
T(n) &= 2(2(\ldots 2T(n-1-a) + 2^n \ldots) + 2^n) + 2^n \\
&= 2^a T(n-1-a) + \underbrace{2^{n+a-1} + 2^{n+a-2} + \cdots + 2^{n+(a-(a-2))-1} + 2^{n+(a-(a-1))-1}}_{a} \\
&= 2^{n-1}T(0) + \underbrace{2^{n+(n-1)-1} + 2^{n+(n-1)-2} + \cdots + 2^{n+((n-1)-(n-1-1))-1}}_{n-1} && \boxed{\text{let } a = n-1} \\
&= \underbrace{2^{2n-2} + 2^{2n-3} + \cdots + 2^{2n-n}}_{n-1} && \boxed{T(0) = 0} \\
&= \sum_{i=0}^{n-2} 2^{2n-2} 2^{-i}.
\end{aligned}$$

The sum is that of a geometric progression, with $a = 2^{n-2}$ and $r = 0.5$. Generally, the sum of a geometric progression up to the k-th term is given as:

$$S_k = \frac{ar^{k+1} - a}{r-1}.$$

Here, $k = n-2$, so

$$\begin{aligned}
\sum_{i=0}^{n-2} 2^{2n-2} 2^{-i} &= \frac{2^{2n-2}(0.5)^{n-2+1} - 2^{2n-2}}{0.5-1} \\
&= \frac{2^{2n-2} - 2^{2n-2}2^{-n+1}}{0.5} \\
&= \frac{2^{n-1}\left(2^{n-1} - 1\right)}{0.5}
\end{aligned}$$

3. (10 points) Below is the code for Fibonacci sequence. **Write** a recurrence relation for this code and solve it using substitution (forward or backward) method.

   Note: There are two recurrence relations $T(n-1)$ and $T(n-2)$ and to simplify your calculation you can assume that $T(n-1) = T(n-2)$ as they are almost of same size.

```
int Fibonacci(int N) {                                  // T(n) =
  if (n == 0 || n == 1) {
    return 1;
  }
  else {
    return Fibonacci(n - 1) + Fibonacci(n - 2); //
    ↪  T(n-1) + T(n-2)
  }
}
```

Since we are given that $T(n-1) \approx T(n-2)$, the recurrence relation for the above then becomes

$$T(n) = 2T(n-1), \quad T(0) = 1.$$

Answer: Using forward substitution, we have the following:

$$\begin{aligned} T(1) &= 2T(0) \\ \therefore T(2) &= 2(2T(0)) \\ &\vdots \\ T(n) &= 2^n T(0) \\ \therefore T(n) &= 2^n. \end{aligned} \qquad \boxed{T(0) = 1}$$