Examples from class
September 4, 2018

Ex 2.3-3. We prove that when n is an exact power of 2, then the solution to the recurrence

$$T(n) = \begin{cases} 2 & \text{if } n = 2 \\ 2T(\frac{n}{2}) + n & \text{if } n = 2^k, \text{ for } k > 2 \end{cases}$$

Proof. By induction on k

Suppose $k = 1$, and so $n = 2$. Then $T(n) = 2 = 2 \cdot 1 = 2 \cdot \lg 2$.

Next, suppose that for some $k \geq 1$ and $n = 2^k$, $T(n) = n \lg n$. Then

$$\begin{aligned} T(2n) &= 2T(\tfrac{2n}{2}) + 2n \\ &= 2T(n) + 2n \\ &= 2n \lg n + 2n \\ &= 2n(\lg n + 1) \\ &= 2n \lg 2n. \quad \square \end{aligned}$$

2.3-7 (complete). Code solution:

```python
# Find a pair in a set that sums to a given number, if any.
# s - the sequence we're searching
# x - the sum we want to find two addends of
# returns a tuple with the values in the set that sums to x
def findPairSum(s, x):
    s.sort()
    # i and j are the inclusive endpoints of the range we're searching
    i = 0
    j = len(s) - 1
    while i <= j and s[i] + s[j] != x :
        if s[i] + s[j] < x :
            i += 1
        else :
            assert s[i] + s[j] > x
            j -= 1
    if i <= j :
        return (s[i], s[j])
    else :
        return None
```

Invariant (Loop of `findPairSum`). *After $k \in \mathbb{W}$ iterations,*

(a) $\forall\ a \in [0, i), s[a] + s[j] < x$

(b) $\forall\ b \in (j, n), s[i] + s[b] > x$

(c) $j - i = n - k - 1$

Correctness Claim (`findPairSum`). *The method `findPairSum` returns two values in the given sequence that sum to x, if any exist.*

Proof. By induction on k, the number of iterations.

Initialization. Suppose $k = 0$ (before the loop starts). $i = 0$ and $j = n - 1$. The two ranges $[0, i)$ and (j, n) are empty, and so clauses (a) and (b) are vacuously true. Moreover, $j - i = n - 1 - 0 = n - 0 - 1 = n - k - 1$.

Maintenance. Suppose the invariant is true after k iterations, for some $k \geq 0$. Suppose a k+1st iteration occurs. By the guard (which must have been true), either $S[i]+S[j] < x$ or $S[i] + S[j] > x$.

Suppose $S[i] + S[j] < x$. By the inductive hypothesis, for all $a \in [0, i)$, $S[a] + S[j] < x$. Hence for all $a \in [0, i+1)$, $S[a] + S[j] < x$. The invariant is maintained after i is incremented.

The situation is similar if $S[i] + S[j] > x$.

Additionally, either i is incremented or j is decremented. In either case $j_{\text{new}} - i_{\text{new}} = (j_{\text{old}} - i_{\text{old}}) - 1 = n - k - 1 - 1 = n - (k+1) - 1$.

Hence the invariant holds after $k + 1$ iterations.

Termination. After n iterations, $j - i = -1$ so $i > j$. Hence the loop will terminate after at most n iterations.

After the loop terminates, either $i > j$ or $S[i] + S[j] = x$.

Suppose $i > j$. Then, by the loop invariant, no elements exist that sum to x, and the algorithm correctly returns `None`.

On the other hand, suppose $S[i] + S[j] = x$. Then the algorithm correctly returns $S[i]$ and $S[j]$. □

For the analysis, here' s the code reproduce with anotations.

```
def findPairSum(s, x):
    s.sort()      # c_0 + c_1 n + c_2 n lg n

    i = 0
    j = len(s) - 1
    while i <= j and s[i] + s[j] != x :    #c_3(n + 1)
        if s[i] + s[j] < x :     #c_4 n
            i += 1
        else :
            assert s[i] + s[j] > x
            j -= 1
    if i <= j :          #c_5
        return (s[i], s[j])
    else :
        return None
```

Renaming constants, the worst-case running time is in the form

$$T(n) = d_0 + d_1 n + d_2 n \lg n$$

Which is $\Theta(n \lg n)$.

2-3.c. Be careful. What is the induction variable? Not i. Look at the proposed invariant again. The induction variable is actually the number of iterations which is $n-i$. That will make the math a little messier.

Proof. By induction on the number of iterations.

Init. After 0 iterations, $y=0$, $i=n$ by assignment. So

$$\sum_{k=0}^{n-(i+1)} a_{k+i+1} = \sum_{k=0}^{-1} a_{k+i+1}x^k = 0 = y$$

Maint. Now, suppose this holds true after N iterations, that is

$$y_{\text{old}} = \sum_{k=0}^{n-(i_{\text{old}}+1)} a_{k+i_{\text{old}}+1}x^k$$

where y_{old} and i_{old} are y and i after N iterations. Likewise, let y_{new} and i_{new} be the values after $N+1$ iterations.

By assignment $i_{\text{new}} = i_{\text{old}} - 1$. Then

$$\begin{aligned}
y_{\text{new}} &= a_{i_{\text{old}}} + x \cdot y_{\text{old}} && \text{by assignment} \\
&= a_{i_{\text{old}}} + x \cdot \textstyle\sum_{k=0}^{n-(i_{\text{old}}+1)} a_{k+i_{\text{old}}+1}x^k \\
&= a_{i_{\text{new}}-1} + x \cdot \textstyle\sum_{k=0}^{n-(i_{\text{new}}+2)} a_{k+i_{\text{new}}}x^k && \text{by substitution} \\
&= a_{i_{\text{new}}-1} + \textstyle\sum_{k=0}^{n-(i_{\text{new}}+2)} a_{k+i_{\text{new}}}x^{k+1} && \text{by distribution} \\
&= a_{i_{\text{new}}-1} + \textstyle\sum_{k=1}^{n-(i_{\text{new}}+1)} a_{k+i_{\text{new}}+1}x^k && \text{by change of variables} \\
&= a_{0+i_{\text{new}}-1}x^0 + \textstyle\sum_{k=1}^{n-(i_{\text{new}}+1)} a_{k+i_{\text{new}}+1}x^k \\
&= \textstyle\sum_{k=0}^{n-(i_{\text{new}}+1)} a_{k+i_{\text{new}}+1}x^k && \square
\end{aligned}$$