Algorithms Homework Assignment 1

Andrew Osborne

January 28, 2019

Conventions When I refer to $\mathbb{N}$, I speak of

$$\mathbb{N} = \{1, 2, 3, \ldots\}$$

Problem 1

$$T(n) = \left\{ \begin{array}{ll} 2 & : n = 1 \\ T(n-1) + 2 & : n \geq 2 \end{array} \right\}$$

In the interest of simplicity, I will define $t_n = T(n) \, \forall \, n \in \mathbb{N}$. Then, quite obviously, $t_n = t_{n-1} + 2$ and indexing each subscript one time, we have

$$\begin{aligned} t_n &= t_{n-1} + 2 \\ &= (t_{n-2} + 2) + 2 \\ &= t_{n-2} + 2 \times 2 \\ &= (t_{n-3} + 2) + 2 \times 2 \\ &= t_{n-3} + 3 \times 2 \\ &= (t_{n-4} + 2) + 3 \times 2 \\ &= t_{n-4} + 4 \times 2 \\ &. \\ &. \\ &. \\ &= t_{n-(n-1)} + (n-1) \times 2 \\ &= t_1 + (n-1) \times 2 \\ &= 2 + (n-1) \times 2 \\ &= n \times 2 \end{aligned} \tag{1}$$

So, after all of that, explicitly,

$$T(n) = 2 \times n$$

Really this is quite clear from the closure of $2\mathbb{Z}$ under addition. We alse could have defined $S(n) = \frac{1}{2} T(n)$ and seen that we were explicitly incrementing an index.

Problem 2

$$T(n) = \left\{ \begin{array}{ll} 2 & : n = 1 \\ T(n-1) + 4n - 3 & : n \geq 2 \end{array} \right\}$$

First, notice that we may rewrite this as

$$T(n) = \left\{ \begin{array}{ll} 2 & : n = 1 \\ T(n-1) + 4(n-1) + 1 & : n \geq 2 \end{array} \right\}$$

Then, we may define the difference between consecutive terms (because it is so clearly presented) as

$$\Delta_n = T(n) \,-\, T(n-1) \,:\, n \geq 2 \,\&\, n \in \mathbb{N}$$

Then $\Delta_n = 4(n-1)+1$ and

$$\begin{aligned} T(n) &= T(1) \,+\, \sum_{i=2}^{n} \Delta_i \\ &= 2 + \sum_{i=2}^{n} 4(i-1) + \sum_{i=2}^{n} 1 \\ &= 2 + 4\sum_{i=1}^{n-1} i + \sum_{i=1}^{n-1} 1 \\ &= 2 + n - 1 + 4\sum_{i=1}^{n-1} i \\ &= n + 1 + 4(\frac{1}{2}n(n-1)) \\ &= n + 1 + 2n^2 - 2n \\ &= 2n^2 - n + 1 \end{aligned} \tag{2}$$

Problem 3

$$T(n) \;=\; \left\{ \begin{array}{ll} 2 & : n = 1 \\ 2T(n \,-\, 1) - 1 & : n \geq 2 \end{array} \right\}$$

This problem can be solved in general with

$$T(n) \;=\; \left\{ \begin{array}{ll} T_1 & : n = 1 \\ m\,T(n \,-\, 1) - k & : n \geq 2 \end{array} \right\}$$

The solution method here relies heavily on one's intuition of Horner's method which can be used to cheaply evaluate polynomials. where $m, k \in \mathbb{Z}$.

$$\begin{aligned} T(1) &= T_1 \\ T(2) &= mT_1 - k \\ T(3) &= mT(2) - k \\ &= m(mT_1 - k) - k \\ &= m^2T_1 - mk - k \\ &. \\ &. \\ &. \\ T(n) &= m^{n-1}T_1 - k\sum_{i=0}^{n-2} m^i \\ &= m^{n-1}T_1 - k\frac{1-m^{n-1}}{1-m} \end{aligned} \tag{3}$$

And in the case of $m = 2$, $k = 1$, this reduces to

$$T(n) = 2^{n-1} + 1 \,:\, n \in \mathbb{N}$$

Problem 4

$$T(m) \;=\; \left\{ \begin{array}{ll} 0 & : m=1 \\ 2\,T(m\;-\;1)+m-1 & : m>1 \end{array} \right\}$$

Because I am lazy and not feeling like being particularly rigorous here, I will do this particular problem like an idiot. For the record, it is 3:00 AM right now.

$$\begin{aligned} T(1) &= 0 \\ T(2) &= 1 \\ T(3) &= 4 \\ T(4) &= 11 \\ T(5) &= 26 \\ T(6) &= 57 \end{aligned} \tag{4}$$

And then notice that

$$\begin{aligned} T(2)-T(1) &= 1 = 2^1-1 \\ T(3)-T(2) &= 3 = 2^2-1 \\ T(4)-T(3) &= 7 = 2^3-1 \\ T(5)-T(4) &= 15 = 2^4-1 \\ T(6)-T(5) &= 31 = 2^5-1 \end{aligned} \tag{5}$$

So apparently, the difference between adjacent terms in the sequence is 2^n-1, to that $T(m) = T(m-1)+2^{m-1}-1$. Then if we sum all of these differences, we get

$$T(n) = \sum_{i=0}^{m}(2^i-1)$$

which, from the last problem we know is

$$\frac{1-2^m}{1-2} - m = 2^m - 1 - m$$

Problem 5

Now rewriting the solution, which explicitly contains $n = 2^m - 1$, so that $T(n) = n - m$, but $n+1 = 2^m \implies m = \log_2(n+1)$ so then $T(n) = n + \log_2(n+1)$

Problem 2-1 (pg. 39 in the text)

Part a

From page 38 of the text, the worst case running time of insertion sort on an array of k elements is $\Theta(k^2)$. Which is to say that the runtime of insertion sort on k elements, $T(k)$ satisfies

$$\exists \quad c,d>0 \,\in\, \mathbb{R}\,\&\,n_0\in\mathbb{N} \,:\, 0\le c\,k^2 \;\le T(k)\;\le d\,k^2 \quad \forall\, n\ge n_0 \tag{6}$$

Then, in the worst case of insertion sort on $\frac{n}{k}$ arrays with k elements each, is simply $\frac{n}{k}\times T(k)$. n and k are supposed to be non–negative integers, so $\frac{n}{k}>0$ and clearly, from equation (6) we now have that

$$\exists \quad c,d>0 \,\in\, \mathbb{R}\,\&\,n_0\in\mathbb{N} \,:\, 0\le c\,\frac{n}{k}k^2 \;\le \frac{n}{k}T(k)\;\le d\,\frac{n}{k}k^2 \quad \forall\, n\ge n_0 \tag{7}$$

Hence, $\frac{n}{k}T(k) = \Theta(nk)$, which is in fact the worst case running time of insertion sort on $\frac{n}{k}$ arrays of length k.

Part b

Recall that, in our original analysis of Merge sort, we were working with the code

```
MERGE-SORT(A,p,r)
  if p < r
    q = floor((p+r)/2)
    MERGE-SORT(A,p,q)
    MERGE-SORT(A,q+1,r)
    MERGE(A,p,q,r)
```

and our modified pseudocode with courseness k is

```
MERGE-SORT-C(A,p,r,k)
  if k >= r - p
    // uses insertion sort to sort A[p..r]
    INSERTION-SORT(A,p,r)
  else
    q = floor((p+r)/2)
    MERGE-SORT-C(A,p,q,k)
    MERGE-SORT-C(A,q+1,r,k)
    MERGE(A,p,q,r)
```

which yields a cost function

$$T(m) = \left\{ \begin{array}{ll} c\,n & : n \leq k \\ 2\,T(\frac{n}{2}) + c\,n & : n > 1 \end{array} \right\}$$

which can be solved in much the same way as our original merge-sort analysis. On the other hand, it is clear that, if $n = 2^l\,k$ for some $l \in \mathbb{N}$, then we will have l recursive calls of MERGE, each of which has $\Theta(n)$ runtime, so our runtime for complete merge is $n\Theta(l) = \Theta(n\,l)$ but, solving for l in terms of n, we have that $l = \log_2(\frac{n}{k})$, and the worst–case runtime of our merge is then clearly $\Theta(n\log_2(\frac{n}{k}))$

Part c

The worst–case runtime of Merge sort is $\Theta(n\log(n))$. The question is this: what is the function $k(n)$ with maximal asymptotic growth for which $\Theta(n\log(n)) = \Theta(n\log(\frac{n}{k(n)}) + n\,k(n))$. If $k(n)$ grows faster than $\log(n)$, then this cannot be the case because of the $n\,k$ term, and we know that $n\log(\log(n)) + n\log(n)$ fits nicely inside of $\Theta(n\log(n))$. Being, rigorous, it can be shown that $\exists \quad n_0 \in \mathbb{N} \quad : \quad c\,n\log n \geq n\log(\log(n)) \quad \forall \quad n \geq n_0 \quad \& \quad \forall\ c > 0$, so, strictly speaking, if $k(n) = \Theta(\log(n))$, then our asymptotic runtimes are not equal, but we are free to toss away smaller terms in our runtime as is done frequently, so that we might say that $k(n) = \Theta\log(n)$ meets our needs. More explicitly, if we disregard the $-n\,\log(\frac{n}{k(n)})$ term, then we have satisfied our goal. If throwing away that term is unacceptable, then we are restricted to $k(n) = \Theta(1)$

Part d As in many cases, our best way to find an appropriate k for any given n, we should just do emperical tests. If you would prefer a mathematical answer, we could fix some $n \in \mathbb{N}$ and use a numerical method like gradient descent to optimize the difference between the runtime of Merge sort, and our coarsened merge sort. We could also probably just use calculus, though I am not particularly hopeful that it would work.

References

I used this website to look–up some series identities that were used in solving recursion equations

`https://en.wikipedia.org/wiki/List_of_mathematical_series`