1 Plug & Chug

Problem 1. Suppose you put $1000 in a bank account. At the end of each month, you earn 1% interest and then you immediately withdraw $5. Let $M_n$ be the amount of money in the account after $n$ months.

(a) Express the amount in the account after $n$ months with a recurrence and base case.

Solution.

\[ M_0 = 1000 \]
\[ M_n = 1.01M_{n-1} - 5 \quad (n \geq 1) \]

(b) Now we’re going to find a closed form for $M_n$ using the “plug and chug” method. Rewrite $M_n$ in terms of smaller and smaller $M_i$ by applying the recurrence equation over and over. Stop when you uncover a pattern. Simplify enough to keep the expressions managable, but not so much that you destroy the pattern!

Solution.

\[
M_n = 1.01M_{n-1} - 5 \\
= 1.01(1.01M_{n-2} - 5) - 5 \\
= 1.01^2M_{n-2} - 5 \cdot 1.01 - 5 \\
= 1.01^2(1.01M_{n-3} - 5) - 5 \cdot 1.01 - 5 \\
= 1.01^3M_{n-3} - 5 \cdot 1.01^2 - 5 \cdot 1.01 - 5 \\
= 1.01^3(1.01M_{n-4} - 5) - 5 \cdot 1.01^2 - 5 \cdot 1.01 - 5 \\
= 1.01^4M_{n-4} - 5 \cdot 1.01^3 - 5 \cdot 1.01^2 - 5 \cdot 1.01 - 5 \\
\]

(c) Based on the pattern you observed, what expression would you have after $k$ rounds of plug-and-chug?

Solution.

\[ M_n = 1.01^kM_{n-k} - 5 \cdot 1.01^{k-1} - \cdots - 5 \cdot 1.01 - 5 \]
(d) Use your expression in terms of $k$ to write $M_n$ entirely in terms of the base cases. (Set $k$ to around $n$ or so.)

**Solution.** Choosing $k = n$ gives:

$$M_n = 1.01^n M_0 - 5 \cdot 1.01^{n-1} - 5 \cdot 1.01^{n-2} - \cdots - 5 \cdot 1.01 - 5$$

(e) Find a closed-form for $M_n$ by applying summation techniques to your expression and substituting in base cases.

**Solution.** We use the formula for the sum of a geometric series.

$$M_n = 1.01^n M_0 - 5 \cdot \frac{1 - 1.01^n}{1 - 1.01}$$

$$= 1.01^n (1000) - 500 (1 - 1.01^n)$$

$$= 500 (1.01^n) + 500$$
2 The Akra-Bazzi Theorem

Theorem 1 (Akra-Bazzi, strong form). Suppose that:

\[ T(x) = \begin{cases} 
\text{is finite} & \text{for } 0 \leq x \leq x_0 \\
\sum_{i=1}^{k} a_i T(b_i x + h_i(x)) + g(x) & \text{for } x > x_0 
\end{cases} \]

where:

- \( a_1, \ldots, a_k \) are positive constants
- \( b_1, \ldots, b_k \) are constants between 0 and 1
- \( x_0 \) is “large enough” in a technical sense we leave unspecified
- \( |g'(x)| = O(x^c) \) for some \( c \in \mathbb{N} \)
- \( |h_i(x)| = O(x/\log^2 x) \)

Then:

\[ T(x) = \Theta \left( x^p \left(1 + \int_1^x \frac{g(u)}{u^{p+1}} \, du \right) \right) \]

where \( p \) satisfies the equation \( \sum_{i=1}^{k} a_i b_i^p = 1 \).

Notice that the functions \( h_i(x) \) appear in the statement of the recurrence, but nowhere in the solution! In algorithmic terms, this means that small changes in the size of subproblems have no impact on the asymptotic running time. (Here small means \( O(x/\log^2 x) \).) In particular, floor and ceiling operators have only a small impact on the size of a subproblem. For example:

\[ \left\lceil \frac{x}{4} \right\rceil = \frac{x}{4} + \left( \left\lceil \frac{x}{4} \right\rceil - \frac{x}{4} \right) \]

between 0 and 1

In this case, the addition of the ceiling operator changes the value of \( x/4 \) by at most 1, which is easily \( O(x/\log^2 x) \). So the ceiling operator has no impact on the solution to the recurrence.
3 Divide & Conquer

Problem 2. We have devised an error-tolerant version of MergeSort. We call our exciting new algorithm OverSort.

Here is how the new algorithm works. The input is a list of \( n \) distinct numbers. If the list contains a single number, then there is nothing to do. If the list contains two numbers, then we sort them with a single comparison. If the list contains more than two numbers, then we perform the following sequence of steps.

- We make a list containing the first \( \frac{2}{3}n \) numbers and sort it recursively.
- We make a list containing the last \( \frac{2}{3}n \) numbers and sort it recursively.
- We make a list containing the first \( \frac{1}{3}n \) numbers and the last \( \frac{1}{3}n \) numbers and sort it recursively.
- We merge the first and second lists, throwing out duplicates.
- We merge this combined list with the third list, again throwing out duplicates.

The final, merged list is the output. What’s great is that even if the sorter occasionally forgets about a number, the OverSort algorithm still outputs a complete, sorted list!

(a) Let \( T(n) \) be the maximum number of comparisons that OverSort could use to sort a list of \( n \) distinct numbers, assuming the sorter never forgets a number and \( n \) is a power of 3. What is \( T(3) \)? Write a recurrence relation for \( T(n) \). (Hint: Merging a list of \( j \) distinct numbers and a list of \( k \) distinct numbers, and throwing out duplicates of numbers that appear in both lists, requires \( j + k - d \) comparisons, when \( d > 0 \) is the number of duplicates.)

Solution. When \( n = 3 \), we begin with a list \((a, b, c)\) of 3 distinct numbers. OverSort starts by forming the lists \((a, b), (b, c)\), and sorts each of them. Sorting a list of length two takes 1 comparison, so the total number of comparisons required to sort both lists is 2. Next it merges these two lists. There must be exactly one duplicate in the two lists, so, using the Hint, we conclude that this takes \( 2 + 2 - 1 = 3 \) more comparisons and yields a sorted length three list of all the elements.

OverSort then forms the list \((a, c)\), uses 1 comparison to sort it, and merges this sorted list of length two with the previous length three list. However, merging these two lists takes only \( 2 + 3 - 2 = 3 \) comparisons, because there are two duplicates. So \( T(3) \), the worst case number of comparisons, is \( 2 + 3 + 1 + 3 = 9 \).

Now for \( n > 3 \), OverSort will form and then sort three different lists of length \((2/3)n\). The first two overlap by \((1/3)n\), so merging them takes \((2/3)n + (2/3)n - (1/3)n = n \) comparisons and yields a list of length \( n \). This list and the remaining
Recitation 13

list of length \( \frac{2}{3}n \) overlap by \( \frac{2}{3}n \), so merging them requires only another \( n \) comparisons. So

\[
T(n) = 3T\left(\frac{2}{3}n\right) + 2n.
\]

(b) Now we’re going to apply the Akra-Bazzi Theorem to find a \( \Theta \) bound on \( T(n) \). Begin by identifying the following constants and functions:

- The constant \( k \).
  Solution. \( k = 1 \)
- The constants \( a_i \).
  Solution. \( a_1 = 3 \)
- The constants \( b_i \).
  Solution. \( b_1 = \frac{2}{3} \)
- The functions \( h_i \).
  Solution. \( h_1(x) = 0 \)
- The function \( g \).
  Solution. \( g(x) = 2x \)
- The constant \( p \). You can leave \( p \) in terms of logarithms, but you’ll need a rough estimate of its value later on.
  Solution.

\[
3 \cdot \left(\frac{2}{3}\right)^p = 1
\]

\[
\ln 3 + p(\ln 2 - \ln 3) = 0
\]

\[
p = \frac{\ln 3}{\ln 3 - \ln 2} = 2.7095 \ldots
\]

(c) Does the condition \( |g'(x)| = O(x^c) \) for some \( c \in \mathbb{N} \) hold?
Solution. Yes. \( g'(x) = 2 = O(x^0) \).

(d) Does the condition \( |h_i(x)| = O(x/\log^2 x) \) hold?
Solution. Yes. \( 0 = O(x/\log^2 x) \).

(e) Determine a \( \Theta \) bound on \( T(n) \) by integration.
Solution.

\[
T(n) = \Theta \left(n^p \cdot \left(1 + \int_1^n \frac{2u}{u^{1+p}} \, du\right)\right)
\]

\[
= \Theta \left(n^p \cdot \left(1 + 2 \int_1^n \frac{1}{u^p} \, du\right)\right)
\]

\[
= \Theta(n^p)
\]

\[
= \Theta \left(n^{2.7095\ldots}\right)
\]
Recitation 13

Note that since \( p > 1 \), the integral nonnegative and bounded above by a constant, no matter how large \( n \) grows. Thus, everything except \( n^p \) is absorbed by the \( \Theta \).

**Problem 3.** Find \( \Theta \) bounds for the following divide-and-conquer recurrences. Assume \( T(1) = 1 \) in all cases.

(a) 
\[
T(n) = 3T\left(\left\lfloor \frac{n}{3} \right\rfloor\right) + n
\]

**Solution.** \( a_1 = 3, b_1 = 1/3, h_1(n) = \left\lfloor n/3 \right\rfloor - n/3, g(n) = n, p = 1, \)
\[
T(n) = \Theta(n(1 + \int_{1}^{n} \frac{u}{u^2} du)) = \Theta(n \log n).
\]

(b) 
\[
T(n) = 4T\left(\left\lfloor \frac{n}{3} \right\rfloor\right) + n^2
\]

**Solution.** \( a_1 = 4, b_1 = 1/3, h_1(n) = \left\lfloor n/3 \right\rfloor - n/3, g(n) = n^2, p = \log_3 4, \)
\[
T(n) = \Theta(n^{\log_3 4}(1 + \int_{1}^{n} \frac{u^2}{u^{\log_3 4}} du)) = \Theta(n^{\log_3 4}(1 + \int_{1}^{n} u^{1-\log_3 4} du)) = \Theta(n^2).
\]

(c) 
\[
T(n) = T\left(\left\lceil \frac{n}{4} \right\rceil\right) + T\left(\left\lfloor \frac{3n}{4} \right\rfloor\right) + n
\]

**Solution.** \( a_1 = 1, a_2 = 1, b_1 = 1/4, b_2 = 3/4, h_1(n) = \left\lceil n/4 \right\rceil - n/4, h_2(n) = \left\lfloor 3n/4 \right\rfloor - 3n/4, g(n) = n, p = 1, \)
\[
T(n) = \Theta(n(1 + \int_{1}^{n} \frac{u}{u^2} du)) = \Theta(n \log n).
\]