Assignment 1 Solutions

1 Asymptotics [10 marks]

Prove or disprove each of the following statements.

(a) For any constant $c > 0$, the function $f : n \mapsto 1 + b + b^2 + b^3 + \cdots + b^n$ satisfies

$$f(n) = \begin{cases} \Theta(b^n) & \text{if } b > 1 \\ \Theta(1) & \text{if } b \leq 1. \end{cases}$$

Solution. The statement is false.

Proof. When $b = 1$, then $f(n) = 1 + 1 + \cdots + 1 = n$. And for every constant $c > 0$, whenever $n > c$ then $f(n) > c$ so $f \neq O(1)$ and, therefore, $f \neq \Theta(1)$. The correct asymptotic bound for the case where $b = 1$ is $f = \Theta(n)$. □

(b) For every pair of functions $f, g : \mathbb{Z}^+ \to \mathbb{R}^+$ that satisfy $f = \Theta(g)$, the functions $F : n \mapsto 2^{f(n)}$ and $G : n \mapsto 2^{g(n)}$ also satisfy $F = \Theta(G)$.

Solution. This statement is false.

Proof. Consider the functions $f : n \mapsto n$ and $g : n \mapsto 2n$. Then $f = \Theta(g)$ and the functions F, G are defined by $F : n \mapsto 2^n$ and $G : n \mapsto 2^{2n}$. For every constant $c > 0$, when $n > \log \frac{1}{c}$ then $1 < c\,2^n$ so $F(n) = 2^n < c \cdot 2^{2n} = c\,G(n)$. Therefore, $F \neq \Omega(G)$ so $F \neq \Theta(G)$. □

(c) For every pair of functions $f, g : \mathbb{Z}^+ \to \mathbb{R}^{\geq 1}$ that satisfy $f = o(g)$, the functions $F : n \mapsto 2^{f(n)}$ and $G : n \mapsto 2^{g(n)}$ also satisfy $F = o(G)$.

Solution. This statement is true.

Proof. Since $f = o(g)$, there exists n_0 such that for all $n \geq n_0$, $f(n) < \frac{1}{2}g(n)$. For every $n \geq n_0$, we then also have $F(n) = 2^{f(n)} < 2^{\frac{1}{2}g(n)} = \sqrt{G(n)}$ and so

$$\lim_{n\to\infty} \frac{F(n)}{G(n)} < \lim_{n\to\infty} \frac{\sqrt{G(n)}}{G(n)} = \lim_{n\to\infty} \frac{1}{\sqrt{G(n)}}.$$

Since $f = o(g)$ and $f(n) \geq 1$, for every constant $c > 0$ there exists n_0 such that for every $n \geq n_0$, $1 \leq f(n) < \frac{1}{c}g(n)$ or, equivalently, $g(n) > c \Leftrightarrow 2^{-g(n)/2} < 2^{-c/2}$. Therefore,

$$\lim_{n\to\infty} \frac{1}{\sqrt{G(n)}} = \lim_{n\to\infty} 2^{-g(n)/2} = 0$$

and $F = o(G)$. □

2 Solving recurrences [10 marks]

Solve the following recurrence relations to obtain a closed-form big-Θ expression for $T(n)$. In each question, assume $T(c)$ is bounded by a constant for any small constant c.

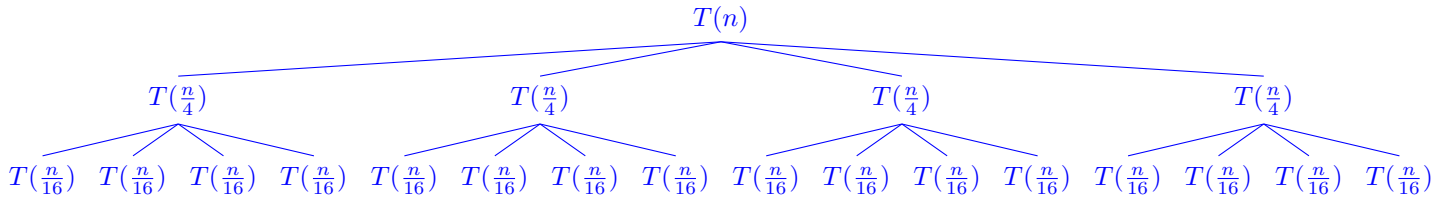
(a) $T(n) \leq 9\,T(\frac{n}{3}) + n^2$

Solution. $T(n) = \Theta(n^2 \log n)$ by the Master Theorem with parameters $a = 9$, $b = 3$, and $c = 2$. (We use the case where $\log_b a = \log_3 9 = 2 = c$.)

(b) $T(n) \leq 4\,T(\frac{n}{4}) + n \log n$

Solution. $T(n) = \Theta(n \log^2 n)$.

The Master Theorem does *not* apply in this case since the last term is not of the form $\Theta(n^c)$ for any constant c. But we can draw the recursion tree



⋮

We then observe that at depth d of this tree, a total amount of

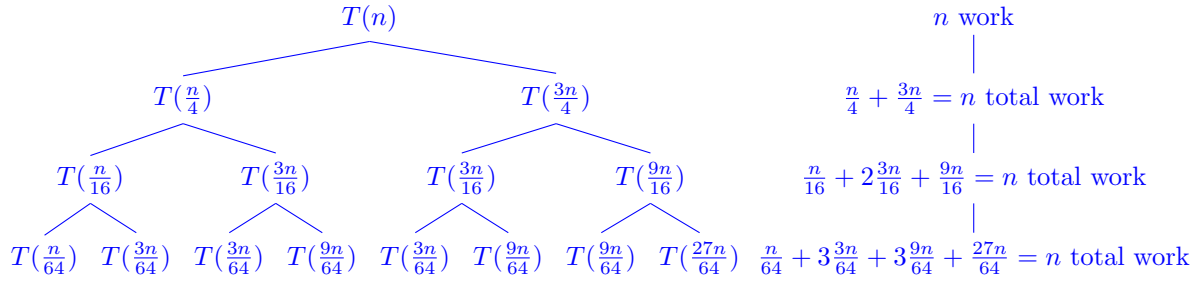
$$4^d \cdot \tfrac{n}{4^d} \log(\tfrac{n}{4^d}) = n \log(\tfrac{n}{4^d}) = n \log n - 2nd$$

work is done. The depth of the tree is $\log_4 n$, so that the total amount of work performed in the recursion tree is $\Theta(n \log n \cdot \log n) = \Theta(n \log^2 n)$.

(c) $T(n) \leq T(\frac{n}{4}) + T(\frac{3n}{4}) + n$

Solution. $T(n) = \Theta(n \log n)$.

Once again the Master Theorem does not apply but we can draw the recursion tree and compute the amount of work on every level.

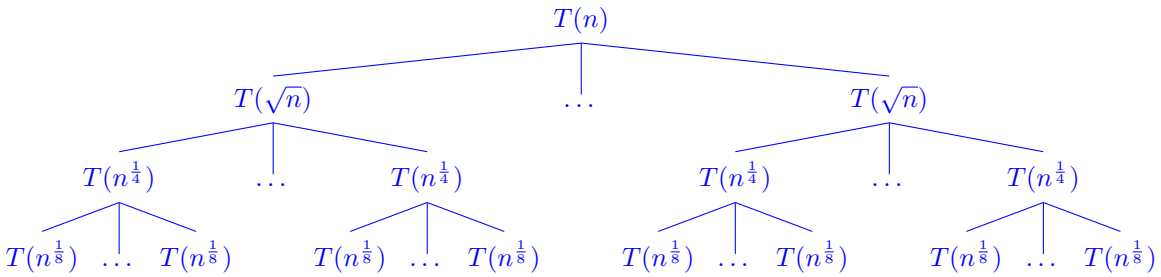


For each level of the recursion tree that is complete, n work total is completed. There are $\lfloor \log_4 n \rfloor = \Theta(\log n)$ such levels. And the remaining levels all contain at most n total work each. There are at most $\lceil \log_{4/3} n \rceil = \Theta(\log n)$ levels in total. So the total amount of work performed is $n \cdot \Theta(\log n) = \Theta(n \log n)$.

(d) $T(n) \leq \sqrt{n} \cdot T(\sqrt{n}) + n$
Hint. The correct expression is somewhere between $\Omega(n)$ and $O(n \log n)$.

Solution. $T(n) = \Theta(n \log \log n)$.

Let's again draw out the recursion tree.



At level 0 of the tree, there is 1 node performing n amount of work.

At level 1, there are $\sqrt{n}$ nodes performing $\sqrt{n}$ work each, for a total of $\sqrt{n} \cdot \sqrt{n} = n$ work.

At level 2, there are $\sqrt{n} \cdot \sqrt{\sqrt{n}} = n^{3/4}$ nodes performing $n^{1/4}$ work each, for a total of n work once again!

In general, at level ℓ there are $n^{\frac{1}{2}} \cdot n^{\frac{1}{4}} \cdot \cdots \cdot n^{\frac{1}{2^\ell}} = n^{1-1/2^\ell}$ nodes performing $n^{1/2^\ell}$ work each, for a total of n work.

So the closed form expression for $T(n)$ is n times the depth of this recursion tree. What is the largest value of ℓ that satisfies $n^{1/2^\ell} \geq 2$? We have $n^{1/2^\ell} = 2^{\log n / 2^\ell}$ so it is the maximum value of ℓ for which $\log n \geq 2^\ell \Leftrightarrow \log \log n \geq \ell$. Therefore, the depth of the recursion tree is $\Theta(\log \log n)$ and $T(n) = \Theta(n \log \log n)$.

3 Testing primality [10 marks]

Analyze the time complexity in terms of n of the following pseudocode using big-O notation. For this analysis, each operation on integers (including multiplication and squaring) takes constant time.

Algorithm 1: IsPrime(n)

```
1 j ← 2;
2 while j^2 ≤ n do
3     k ← 2;
4     while j * k ≤ n do
5         if j * k = n then
6             return 0;
7         k ← k + 1;
8     j ← j + 1;
9 return 1;
```

Solution. The IsPrime algorithm has time complexity $O(n \log n)$.

Proof. For each value $j = 2, 3, \ldots, \sqrt{n}$, the inner while loop is run $\lceil n/j \rceil = O(n/j)$ times, so the total number of iterations of this inner loop is

$$\left\lceil \frac{n}{2} \right\rceil + \left\lceil \frac{n}{3} \right\rceil + \left\lceil \frac{n}{4} \right\rceil + \cdots + \left\lceil \frac{n}{\lfloor \sqrt{n} \rfloor} \right\rceil = \Theta\Big(n \cdot (\tfrac{1}{2} + \tfrac{1}{3} + \tfrac{1}{4} + \cdots + \tfrac{1}{\sqrt{n}})\Big).$$

The series inside the big-O is the harmonic series and satisfies

$$\frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{\sqrt{n}} = O(\log \sqrt{n})$$

so the total number of iterations of the inner while loop is

$$O(n \log \sqrt{n}) = O(n \cdot \tfrac{1}{2} \log n) = O(n \log n).$$

Since each iteration of that inner while loop performs a constant number of elementary operations, the total runtime of the algorithm is also $O(n \log n)$. □

4 Common sum [10 marks]

In the Common Sum problem, we are given two arrays A and B of length n containing non-negative (not necessarily distinct) integers, and we must determine whether there are indices $i_1, i_2, j_1, j_2 \in \{1, 2, \ldots, n\}$ for which

$$A[i_1] + A[i_2] = B[j_1] + B[j_2].$$

Design an algorithm that solves the Common Sum problem and has time complexity $O(n^2 \log n)$ in the setting where operations on individual integers take constant time.

Your solution must include a description of the algorithm in words, the pseudocode for the algorithm, a proof of its correctness, and an analysis of its time complexity in big-Θ notation.

Solution. We reduce the Common Sum problem to a slight variant of the 2Sum problem we saw in class. Let C and D be arrays of n^2 non-negative integers each where we define

$$C[(i-1)n+j] = A[i] + A[j] \qquad \text{and} \qquad D[(i-1)n+j] = B[i] + B[j]$$

for every $i, j \in \{1, 2, \ldots, n\}$. Then the solution to the Common Sum problem is True if and only if there are indices $k, \ell \in \{1, 2, \ldots, n^2\}$ such that $C[k] = D[\ell]$. So to solve Common Sum, we build the arrays C and D and run the intersection test (which is a slight variant of 2Sum) on those arrays. The algorithms are as follow.

Algorithm 2: Intersect($C[1..m], D[1..m]$)

```
1  Sort(C); Sort(D);
2  i ← 1; j ← 1;
3  while i ≤ m and j ≤ m do
4      if C[i] = D[j] then
5          return True;
6      else if C[i] < D[j] then
7          i ← i + 1;
8      else
9          j ← j + 1;
10 return False;
```

Algorithm 3: CommonSumAlg($A[1..n], B[1..n]$)

```
1 for i = 1, ..., n do
2     for j = 1, ..., n do
3         C[(i − 1) + j] ← A[i] + A[j];
4         D[(i − 1) + j] ← B[i] + B[j];
5 return Intersect(C,D);
```

Theorem 1. *The* CommonSumAlg *solves the* Common Sum *problem.*

Proof. We first need to prove that the Intersect algorithm returns True if and only if C and D have an element in common. If C and D contain no common element, then the condition on line 4 is never satisfied so the algorithm always returns False. Otherwise, let i^*, j^* be the minimal indices such that $C[i^*] = D[j^*]$. We want to show that the while loop has indices $i = i^*$ and $j = j^*$ at some point. Initially, $i \leq i^*$ and $j \leq j^*$. If $i^* = 1$ and $j^* = 1$, then he while loop accepts on the

first iteration. Otherwise, one index is increased on every iteration of the loop, so we eventually reach a point where either $i = i^*$ and $j < j^*$, or $i < i^*$ and $j = j^*$.

Case 1: $i = i^*$ and $j < j^*$. Then $C[i^*] = D[j^*] > D[j]$ so j is increased.

Case 2: $i < i^*$ and $j = j^*$. Then $C[i] < C[i^*] = D[j^*]$ so i is increased.

In both cases, the indices are increased until we have $i = i^*$ and $j = j^*$, at which point the while loop returns True.

We are now ready to complete the proof of correctness for the CommonSumAlg algorithm. If there exist $i_1, i_2, j_1, j_2 \in \{1, 2, \ldots, n\}$ for which $A[i_1] + A[i_2] = B[j_1] + B[j_2]$, then for the values $i^* = (i_1 - 1)n + i_2$ and $j^* = (j_1 - 1)n + j_2$ we have $C[i^*] = D[j^*]$ so CommonSumAlg correctly returns True.

Conversely, if for every $i_1, i_2, j_1, j_2 \in \{1, 2, \ldots, n\}$ we have $A[i_1] + A[i_2] \neq B[j_1] + B[j_2]$ then also for every $i^* = (i_1 - 1)n + i_2$ and $j^* = (j_1 - 1)n + j_2$ we also have $C[i^*] \neq D[j^*]$ so CommonSumAlg correctly returns False. □

Theorem 2. *The* CommonSumAlg *has time complexity* $\Theta(n^2 \log n)$.

Proof. The time complexity of the Intersect algorithm is $\Theta(m \log m)$: that is the time required to sort the input arrays C and D, and the rest of the algorithm has time complexity only $\Theta(m)$.

The time complexity of CommonSumAlg is the sum of the time complexity for creating C and D and the time complexity of the call to Intersect. The creation of C and D take time $\Theta(n^2)$ each, and the call to Intersect has time complexity $\Theta(n^2 \log(n^2)) = \Theta(2n^2 \log n) = \Theta(n^2 \log n)$ since the algorithm is called with input arrays of length $m = n^2$. □

5 Programming question [10 marks]

Implement the algorithm you obtained for the Common Sum problem in Question 4.

Input and output. The input consists of $n + 1$ lines. The first line contains an integer value $n \geq 1$. The following n lines have two integer values each: $A[i]$ and $B[i]$ for $i = 1, 2, \ldots, n$. We will not be testing whether your program detects input errors. However, our tests will check that your algorithm is fast enough.

The output must have 1 line that contains `TRUE` or `FALSE`.

Sample input 1

```
4
1 2
7 8
5 6
3 4
```

Sample output 1

```
TRUE
```

Sample input 2

```
7
5  16
15  6
25 21
20 31
20 16
10 11
0   1
```

Sample output 2

```
FALSE
```

The valid solution for the first instance is `TRUE` since $A[2] + A[4] = B[1] + B[2] = 10$.

The valid solution for the second instance is `FALSE` because there are no values of $i_1, i_2, j_1, j_2 \in \{1, 2, \ldots, n\}$ for which $A[i_1] + A[i_2] = B[j_1] + B[j_2]$.