Algorithms and Data Structures 2014/15
Solutions for Coursework 1

1. (10 marks in total) This question is about the optimum buy-sell of an array \( A \), defined to be the value
\[
\max_{i,j:1 \leq i \leq j \leq n} \{(A[j].val - A[i].val)\}.
\]

(a) First an example of a small array in which the optimum buy-sell neither involves the maximum value in the array, nor the minimum value in the array. [2 marks]

Answer: There are many examples of this, but one example is the array of 4 elements which has the following key values (in this order):

\[213, 55, 200, 6.\]


Marking: 2 marks for any correct example.

Just as extra information (not for the solution itself) note that although the example I gave above was set up with the trick of putting the maximum value in the earliest cell, and the minimum value in the latest cell. That might suggest that after stripping off the first and last cell we would be left with a subarray \( A' \) where the optimum buy-sell would use either the maximum or minimum value. However, we can build more complex examples where even this fails to be true, for example:

\[213, 85, 205, 15, 200, 4, 180, 6.\]

(b) Next students were asked to give a simple \( \mathcal{O}(n^2) \) algorithm to find the optimum buy-sell of a given array. [2 marks]

Answer:

Algorithm optBuySell(A)

i. \( n \leftarrow \text{length}(A) \).
ii. \( \text{opt} \leftarrow 0 \)
iii. for \( i \leftarrow 1 \) to \( n \)
iv. \hspace{1em} for \( j \leftarrow (i + 1) \) to \( n \)
v. \hspace{2em} if \( (A[j].val - A[i].val) > \text{opt} \)
vi. \hspace{3em} \text{opt} \leftarrow A[j].val - A[i].val
vii. return \text{opt}

Marking: 2 marks if the algorithm is correct and well described (pseudocode or text). Students will lose 0.5 marks if they don’t refer to the .val extension of the cell entries.
(c) Give a Divide-and-Conquer algorithm which finds the optimum buy-sell of a given array in \(O(n \lg(n))\) time.

**answer:** The question was phrased wrt the \(O(n \lg(n))\) bound; however there is also a \(O(n)\) solution. The full marks will be going for either solution, as long as the details and the argument of correctness are good, and the \(O(\cdot)\) bound is justified carefully.

Here is the \(O(n \lg(n))\) algorithm in pseudocode:

**Algorithm optBuySellDC** \((A)\)

i. \(n \leftarrow \text{length}(A)\).
ii. **if** \(n < 2\)
iii. **return** 0
iv. **else**
  v. \(\text{opt} \leftarrow 0\)
  vi. \(\text{min} \leftarrow A[0]\)
  vii. \(\text{max} \leftarrow A[0]\)
  viii. **for** \(i \leftarrow 1\) **to** \(\lfloor n/2 \rfloor\)
    ix. **if** \((A[i].\text{val} < \text{min})\)
     x. \(\text{min} \leftarrow A[i].\text{val}\)
  xi. **for** \(i \leftarrow \lfloor n/2 \rfloor + 1\) **to** \(n\)
   xii. **if** \((A[i].\text{val} > \text{max})\)
    xiii. \(\text{max} \leftarrow A[i].\text{val}\)
 xiv. \(\text{opt} \leftarrow \text{max} - \text{min}\)
  xv. \(\text{optL} \leftarrow \text{optBuySellDC}(A[1 \ldots \lfloor n/2 \rfloor])\)
  xvi. \(\text{optR} \leftarrow \text{optBuySellDC}(A[\lfloor n/2 \rfloor + 1 \ldots n])\)
  xvii. **return** \(\max\{\text{opt}, \text{optL}, \text{optR}\}\)

Students were asked to reason that their algorithm is correct. The proof of correctness goes as follows: clearly if there is just one element in the array \(A\) the only option is to buy *and* sell at the same time, and hence have the value 0. Alternatively, \(n > 1\) and the array has more than one element. In this case consider the landscape of possible pairs \((i, j)\) to be considered, with respect to the midway index \(\lfloor n/2 \rfloor\). By \(i \in \{1, \ldots, n\}\), we *either* have \(i \in \{1, \ldots, \lfloor n/2 \rfloor\}\) or \(i \in \{\lceil n/2 \rceil + 1, \ldots, n\}\); and similarly for \(j\). However, by the constraint \(i \leq j\), we know that if we are considering \(i \in \{\lceil n/2 \rceil + 1, \ldots, n\}\), the partner indices \(j\) must satisfy \(j \in \{\lceil n/2 \rceil + 1, \ldots, n\}\). Therefore in evaluating the optimum buy-sell of an array, the pairs of indices \((i, j)\) we need to consider can be grouped into three classes:

\[ i, j \in \{1, \ldots, \lfloor n/2 \rfloor\}, i < j \]
\[ i, j \in \{\lceil n/2 \rceil + 1, \ldots, n\}, i < j \]
\[ i \in \{1, \ldots, \lfloor n/2 \rfloor\}, j \in \{\lceil n/2 \rceil + 1, \ldots, n\} \]

with the value of the optimum buy-sell for \(A\) being the value \(A[j].\text{val} - A[i].\text{val}\) for whichever pair (from any of the three categories) maximizes the value of this
expression. Let $\text{opt}(1, \ldots, \lfloor n/2 \rfloor)$ denote the maximum $A[j].val - A[i].val$ over all $i, j$ pairs in the first class, and let $\text{opt}(\lfloor n/2 \rfloor + 1, \ldots, n)$ denote the maximum $A[j].val - A[i].val$ over all $i, j$ pairs in the second class. Let $\hat{\text{opt}}(\lfloor n/2 \rfloor)$ denote the maximum $A[j].val - A[i].val$ over all $i, j$ pairs in the third class. By our grouping into the three classes, we know that

$$\text{opt}(1, \ldots, n) = \max \{ \text{opt}(1, \ldots, \lfloor n/2 \rfloor), \text{opt}(\lfloor n/2 \rfloor + 1, \ldots, n), \hat{\text{opt}}(\lfloor n/2 \rfloor) \}.$$ 

Observe that $\text{opt}(1, \ldots, \lfloor n/2 \rfloor)$ is the value computed by the recursive call on line xv. and $\text{opt}(\lfloor n/2 \rfloor + 1, \ldots)$ is the value computed by the recursive call on line xvi. Finally, observe that $\hat{\text{opt}}(\lfloor n/2 \rfloor)$ (the optimum for pairs where $i \leq \lfloor n/2 \rfloor$ and $j > \lfloor n/2 \rfloor$) is equal to $\max - \min$, where $\min$ is the smallest value in $A[1, \ldots, \lfloor n/2 \rfloor]$ and $\max$ is the largest value in $A[\lfloor n/2 \rfloor + 1, \ldots, n]$, and this is the value computed in line xiv. Hence the value computed in xvii. and return-ed on that line is the correct value of the optimum buy-sell.

Students were also asked to prove their algorithm had running time $\Theta(n \lg(n))$. To show this, we note that in the case of $n = 1$, we only carry out lines i. – iii., taking 3 steps at most. Otherwise for $n \geq 2$, we carry out lines v. – vii. in 3 steps, then the two for-loops are carried out, taking $\Theta(n)$ time each, then after that we have two recursive calls of size $\lfloor n/2 \rfloor$ and $\lceil n/2 \rceil$ and 2 extra steps; hence we have the following recurrence:

$$T(n) = \begin{cases} 
\Theta(1) & n = 1 \\
T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + \Theta(n) & n > 1 
\end{cases}$$ 

Considering the recurrence with respect to the Master Theorem, we have $a = 2$, $b = 2$ and $k = 1$. The critical exponent is $c = \log_2(2) = 1$, hence $c = k$ and we are in the middle case of the Master Theorem, which tells us the running-time is $\Theta(n \lg(n))$.

**Marking:** 2 marks for the algorithm, 2 marks for the argument of correctness, 2 marks for the proof of running time, with the possibility of some trade-off of these individual parts (if a particular aspect is extra good, even though some other part is missing). Students will also get (of course) a similar allocation of marks for a $\Theta(n)$ solution, as sketched below.

**Common mistakes:** Most students got (a) and (b) correct.

For (c), by far the most common error was not justifying the correctness of the algorithm. Many students wrote a paragraph or two saying what their algorithm did - but did not give proper justification of the fact that $\text{opt}(1 \ldots n)$ must be the best of the three candidates computed ... On the other hand, in a couple of cases, students tried to do a massive-scale proof by induction which was not what I was looking for at all.

$\Theta(n)$ solution: The $O(n)$ algorithm for Optimum Buy-Sell requires the main part of the algorithm to be set-up to return a triplet of values, the first entry being the
optimum buy-sell of the array, the second entry being the minimum element of the array, and the third element being the maximum element of the array. With that alteration, notice now that the lines viii. — x. can be omitted from the algorithm (instead we just extract the “min” entry from the triplet returned by the recursive call on $A[1\ldots,\lfloor n/2\rfloor]$), as can lines xi. — xiii. (instead we extract the “max” entry from the triplet returned by the recursive call on $A[\lceil n/2\rceil + 1\ldots n]$). Hence we lose the two sections of code which caused $\Theta(n)$ to be added to our recurrence above, but are still able to compute $\text{opt}$ regardless. Hence the recurrence for the $n > 1$ case will instead look like $T(n) = T(\lfloor n/2\rfloor) + T(\lceil n/2\rceil) + \Theta(1)$; hence we will have $k = 0$ this time, and hence we will be in the $c > k$ case, giving running-time $\Theta(n)$ time overall.

Of course, we also need to compute the min-value and the max-value of $A[1\ldots n]$ in order to have all parts of the triplet to be returned; however it is clear that the minimum of the array will be the minimum of the middle values from the triplets returned by the two recursive calls; similarly the maximum of the entire array will be the maximum of the values in the third position of the two triplets returned by the recursive calls. This calculation can be performed in $O(1)$ so we still have the same recurrence as mentioned in the previous paragraph, and the same overall running time $\Theta(n)$. 
2. In this question we prove a fragment of the Master theorem. Suppose we are given a recurrence of the following form:

\[
T(n) = \begin{cases} 
1 & n = 1 \\
αT(\lfloor n/2 \rfloor) + n^k & n \geq 2 ,
\end{cases}
\]

(1)

for some fixed \(α, k \in \mathbb{N}, α \geq 1, k \geq 1\), where \(αT(\lfloor n/2 \rfloor)\) is written to indicate a recursive call, each being either size \([n/2]\) or \([n/2]\).

In this question we will prove the \(O(\cdot)\) bound of the Master Theorem for this recurrence (for the cases \(\log (α) > k\) and \(\log (α) = k\)).

(a) Prove by induction that \(T(n) < T(m)\) for all \(n, m \in \mathbb{N}, n < m\) (this shows that \(T(\cdots)\) is monotonically increasing).

answer: We will decide to prove the result as follows: we will first prove by induction that we have \(T(n) < T(n+1)\) for every \(n \in \mathbb{N}\). Note that once this proof is done, then transitivity implies that \(T(n) < T(m)\) for all \(n, m \in \mathbb{N}, n < m\).

Our goal is to prove that for all \(n \in \mathbb{N}\) we have \(T(n) < T(n+1)\). We will formulate our Inductive Step in terms of the following Induction Hypothesis (IH):

Induction Hypothesis (IH): for every \(n \in \mathbb{N}\) such that \(n < ℓ\), \(T(n) < T(n+1)\).

Proof by induction.

Base case (\(ℓ = 2\)):

\(T(1)\) is defined as \(1\). The definition of \(T(2)\) includes a \(+n = +2\) term; also \(T(\lfloor 2/2 \rfloor) = T(1)\) and \(T(\lceil 2/2 \rceil) = T(1)\). Hence \(T(2) = αT(1) + 2 > 2 > 1\). Hence we have \(T(1) < T(2)\), as required.

Induction Step: Suppose the (IH) is true for some value of \(ℓ\), and now consider \(ℓ+1\).

We need to show that for all \(n < ℓ+1\), we have \(T(n) < T(n+1)\). First note that \(T(n) < T(n+1)\) follows directly from the (IH) for \(ℓ\) if \(n < ℓ\). So the only extra case we need to consider now for \(ℓ+1\) is the case of \(n = ℓ\).

To compare \(T(ℓ)\) with \(T(ℓ+1)\), we apply the recurrence to both \(T(ℓ)\) and \(T(ℓ+1)\). We can do this because we know \(ℓ \geq 2\) (\(ℓ = 1\) is the base case). Applying the recurrence to \(T(ℓ)\) and \(T(ℓ+1)\), we get

\[
T(ℓ) = αT(\lfloor ℓ/2 \rfloor) + ℓ^k \\
T(ℓ + 1) = αT(\lceil (ℓ + 1)/2 \rceil) + (ℓ + 1)^k
\]

We know that \(αT(\lfloor \cdot \rfloor)\) might be made up of any combination of \(T(\lfloor \cdot \rfloor)\) and \(T(\lceil \cdot \rceil)\) terms. Suppose we have \(α_1\) \(T(\lfloor \cdot \rfloor)\) terms and \(α_2\) \(T(\lceil \cdot \rceil)\) terms, for \(α = α_1 + α_2\). But observe now that for any \(ℓ\), \(\lceil (ℓ + 1)/2 \rceil\) is either \(\lfloor ℓ/2 \rfloor\) or \(\lceil ℓ/2 \rceil + 1\). In the case when
Now observe that the extra term satisfies $\ell^k < (\ell + 1)^k$; hence we have $T(\ell) < T(\ell + 1)$, as required.

By induction, we have $T(n) < T(n + 1)$ for all $n \in \mathbb{N}$.

**marking:**

The key ideas are “getting the (IH) right”, doing the base case, applying the recurrence to $\ell$ and $\ell + 1$, and using the (IH) correctly (observing that $\lceil \ell/2 \rceil$ and $\lfloor \ell/2 \rfloor$ are both <; and that $\lfloor (\ell + 1)/2 \rfloor$ is either $\lceil \ell/2 \rceil$ or $\lfloor \ell/2 \rfloor + 1$, and similarly for $\lfloor \cdot \rfloor$). For a reasonable attempt which contains at least two of these ideas, give up to 2 marks.

An attempt with all these ideas and good detail/explanation gets 4 marks.

There is an alternative way of solving this question (see below) which will get 4 marks distributed in the same proportions.

**Alternative proof:** An alternative proof would do the induction proof directly with respect to $n, m$ for $n < m$. This approach has many of the same steps as the one above. Perhaps the hardest part of doing the proof directly is coming up with the (IH) formulation. The formulation you need is:

**Induction Hypothesis (IH):** For any $n \in \mathbb{N}$, it is the case that for all $m$, $m > n$, we have $T(n) < T(m)$.

A direct proof will (of course) get equal marks to the one shown above, with similar breakdown of marks.

**common errors:** Not getting the (IH) written-down in a way that made it clear what parameter was the *induction parameter*. Now for the $T(n) < T(n + 1)$ proof I gave, the induction parameter is clear, but for the alternative version, the (IH) needs to be written in such a way as to make it clear which of $n, m$ is the parameter of induction (usually it is $n$. However it can be done with $m$, and in one proof I read this was the case). Apart from problems with the details of the (IH), in a number of cases this was missing entirely.

Second common error was omitting to state that $\lfloor n/2 \rfloor, \lceil n/2 \rceil$ are both *strictly* less than $n$ (for $n \geq 2$). This must hold, if you are to apply the (IH) to these values.

Third common error was a smaller one - just working with $T(\lfloor n/2 \rfloor)$ versus $T(\lceil (n + 1)/2 \rceil)$, rather than versus $T(\lfloor n/2 \rfloor + 1)$ (or in some cases, is equal to $T(\lfloor n/2 \rfloor)$ itself in which case the (IH) doesn’t need to be used).
(b) Now we prove that for every \( n \) which is a power of 2, that
\[
T(n) = n \log(a) + n^k \sum_{j=0}^{\log(n)-1} \left( \frac{a}{2^k} \right)^j.
\]

**Answer**: The proof is by induction.

We first formulate our **induction hypothesis** for the proof, which will be

**Induction Hypothesis (IH)**: The equation (*) holds for all powers-of-2 \( n = 2^p \) such that \( p \leq q \).

Base case: The first power of 2 is \( 1 = 2^0 \). If \( n = 1 \), then \( T(n) = 1 \). But if \( n = 1 \), then \( n \log(a) = 1 \). Also the value of \( \log(1) \) is 0, meaning that \( \log(1) - 1 < 0 \), and hence the range for \( j \) in the summation is empty. Hence the expression above works out as 1 overall, which is equal to \( T(1) \), as required.

Induction Step: We will show that when the (IH) holds, that (*) also holds for \( 2^q+1 \) (and hence we obtain the (IH) for \( q + 1 \)).

First note that the base case shows the (IH) for \( q = 0 \). Hence for the induction step we have \( q + 1 \geq 1 \) and \( 2^{q+1} \geq 2 \).

For our induction step, note that when proving the (IH) for \( q + 1 \), the (IH) for \( q \) will already give the expression (*) for \( 2^3, \ldots, 2^q \), so the only case we need to check is \( n = 2^{q+1} \). By \( n = 2^{q+1} \geq 2 \), we can apply the given recurrence to infer

\[
T(n) = aT(\lfloor n/2 \rfloor) + n^k.
\]

In this expression, we have \( a_1 T(\lfloor n/2 \rfloor) \) terms and \( a_2 T(\lceil n/2 \rceil) \) terms on the right-hand side; however since \( n = 2^{q+1} \), all these terms are \( T(n/2) \) terms, \( n/2 \) also being a power of 2 (the power \( 2^q \)). So we can apply the (IH) to \( 2^q \), and have the following derivations:

\[
T(n) = aT(n/2) + n^k
= n^k + a \cdot \left[ (n/2)^{\log(a)} + (n/2)^k \sum_{j=0}^{\log(n/2)-1} \left( \frac{a}{2^k} \right)^j \right] \quad \text{by the (IH)}
= n^k + a(n/2)^{\log(a)} + a \cdot (n/2)^k \sum_{j=0}^{\log(n/2)-2} \left( \frac{a}{2^k} \right)^j 
= a(n^{\log(a)}/a) + n^k + a \cdot (n/2)^k \sum_{j=0}^{\log(n/2)-2} \left( \frac{a}{2^k} \right)^j \quad \text{properties of \( \log \)}
\]
\[
= a \left( n^{\log(a)/a} \right) + n^k + a \cdot \left( n/2 \right)^k \sum_{j=0}^{\log(n)-2} \left( \frac{a}{2^k} \right)^j
\]

\[
= n^{\log(a)} + n^k \left[ 1 + \left( \frac{a/2^k}{2^k} \right) \sum_{j=0}^{\log(n)-2} \left( \frac{a}{2^k} \right)^j \right]
\]

as required. Hence by induction, (*) holds for all powers of 2 \( n \).

**marking:**

1 mark for doing the base case.
3 marks for the Induction step and its details.

**common errors:** Solutions for this were mostly good. The most common problem would have been the lack of explanations for steps along the way.

(c) Now we prove that when \( n \) is a power of 2, and when \( \log(a) = k \), that

\[
\text{(*) holds for all powers of } 2 \text{.}
\]

\[ T(n) = n^{\log(a)}(\log(n) + 1). \]

**answer:** We will use the result of (b) in solving this part of the question. From (b), we know that for every power-of-2 \( n \), we have

\[
T(n) = n^{\log(a)} + n^k \sum_{j=0}^{\log(n)-1} \left( \frac{a}{2^k} \right)^j.
\]

Applying the fact that \( \log(a) = k \), we find that in this case, we have

\[
T(n) = n^{\log(a)} + n^{\log(a)} \sum_{j=0}^{\log(n)-1} \left( \frac{a}{2^{\log(a)}} \right)^j
\]

\[
= n^{\log(a)} + n^{\log(a)} \sum_{j=0}^{\log(n)-1} \left( \frac{a}{2^{\log(a)}} \right)^j
\]

\[
= n^{\log(a)} + n^{\log(a)} \cdot \log(n)
\]

\[
= n^{\log(a)}(\log(n) + 1).
\]
marking: 1 mark goes for substituting in the $k \leftarrow \lg(a)$ into the expression $(*)$; another mark for noticing that $2^{\lg(a)} = a$ and stating that you are using that; another mark for noticing the sum evaluates to $n$, and 1 for finishing off.

common errors: solutions to this were mostly good. The most common error was omitting to mention you were applying $2^{\lg(a)} = a$ result. In couple of cases, the student used induction to prove this result - while that’s not an “error”, it makes the proof a lot bigger than this way.

(d) Next we prove that $T(n) = O(n^{\lg(a) \cdot \lg(n)}) = O(n^k \cdot \lg(n))$ for the case of $\lg(a) = k$ for all $n \in \mathbb{N}$ from first principles.

answer: We start with the expression derived for the $\lg(a) = k$ power-of-2 case in (c). This tells us that $T(n) = n^{\lg(a)}(n + 1)$ whenever $n$ is a power of 2.

Now suppose that $n \in \mathbb{N}$ is any natural number, not necessarily a power of 2. We define $\hat{n} = 2^{\lfloor \lg(n) \rfloor}$, and note that $\hat{n}$ is the smallest power of 2 which satisfies $n \leq \hat{n}$; hence we have $n \leq \hat{n} < 2n$.

Now we can upper bound $T(n)$, for any $n \in \mathbb{N}$, as follows:

$$
T(n) \leq T(\hat{n}) \quad \text{by part (a)}
= \hat{n}^{\lg(a)}(\lg(\hat{n}) + 1) \quad \text{by part (c)}
< (2n)^{\lg(a)}(\lg(2n) + 1) \quad \text{by } \hat{n} < 2n
= 2^{\lg(a)}n^{\lg(a)}(\lg(n) + 2)
$$

by properties of $\lg$ = $a \cdot n^{\lg(a)}(\lg(n) + 2)$.

Now note that if $n \geq 4$, then $\lg(n) \geq 2$, hence $\lg(n) + 2 \leq 2\lg(n)$. Therefore, for $n \geq 4$, we have $T(n) \leq 2a \cdot n^{\lg(a)} \cdot \lg(n)$, and this satisfies the definition of $O(n^{\lg(a) \cdot \lg(n)}) = O(n^k \cdot \lg(n))$ with $c = 2a$ and $n_0 = 4$.

marking: 1 mark for the definition of $\hat{n}$, 1 mark for applying (c) to $T(\hat{n})$, 1 mark for substituting $\hat{n} < 2n$ and working with this formula, 1 mark for working down to get $c$, $n_0$. Will be expecting students to directly refer to their use of (a) (at the start) and (c).

common errors: Some students only showed that $T(n) = O(\hat{n}^{\lg(a)} \cdot \lg(\hat{n}))$, when I wanted a bound in terms of $n$.

Of the students who did do the bound in terms of $n$, some missed the dependence on $a$ of the constant term $c$ of the $O(\cdot)$.

(e) Now we prove that when $n$ is a power of 2, and when $\lg(a) > k$, that $[4$ marks$]$
answer: We start with the expression proved in part (b) for powers of 2, which was

\[ T(n) = n^{\log(a)} + n^k \sum_{j=0}^{\log(n)-1} \left( \frac{a}{2^k} \right)^j. \]  

Our first step is to apply the formula for the sum of a geometric series (we can apply this because \( a \neq 2^k \), by \( \log(a) > k \)):}

\[
T(n) = n^{\log(a)} + n^k \left( \frac{\frac{a}{2^k} \log(n)}{a - 2^k} - 1 \right) \frac{2^k}{a - 2^k}
\]

\[
= n^{\log(a)} + n^k \left( \frac{a \log(n)}{2^k \log(n)} - 1 \right) \frac{2^k}{a - 2^k}
\]

\[
= n^{\log(a)} + n^k \left( \frac{n^{\log(a)}}{n^k} - 1 \right) \frac{2^k}{a - 2^k}
\]

\[
= n^{\log(a)} + (n^{\log(a)} - n^k) \frac{2^k}{a - 2^k}
\]

\[
= \left( n^{\log(a)} \frac{a - 2^k}{2^k} + n^{\log(a)} - n^k \right) \frac{2^k}{a - 2^k}
\]

\[
= \left( n^{\log(a)} \left( \frac{a}{2^k} - 1 \right) + n^{\log(a)} - n^k \right) \frac{2^k}{a - 2^k}
\]

\[
= \left( n^{\log(a)} \frac{a}{2^k} - n^k \right) \frac{2^k}{a - 2^k},
\]

as required.

marking: 1 mark goes for applying the result of (b), 1 mark for applying the geometric series formula, and the other 2 marks for the level of detail in working down to the desired expression. I’d hope to see students noting \( x^{\log(y)} = y^{\log(x)} \) rather than just applying it.

common errors: The main common error was using the formula for a geometric sum without stating its use (and without stating that the base \( a/2^k \) is different-from-1, which is needed for the application of the formula). Some students proved this result by induction - this is not an error, but makes the proof longer.

(f) Finally we show that \( T(n) = O(n^{\log(a)}) \) for the case of \( \log(a) > k \) for all \( n \in \mathbb{N} \). [4 marks]

answer: We start as in part (d), by defining \( \hat{n} = 2^{\lceil \log(n) \rceil} \), the smallest power of 2 which satisfies \( n \leq \hat{n} \); we have \( n \leq \hat{n} < 2n \).
We next apply (a) to observe that \( T(n) \leq T(\hat{n}) \). Next, as \( \hat{n} \) is a power of 2, and we are assuming \( \log(a) > k \), we can apply part (e), and continue the derivation from
there:

\[
T(n) \leq T(\hat{n}) \\
= \frac{2^k}{a - 2^k} \left( \frac{a}{2^k} \right) \hat{n}^{\lg(a)} - \hat{n}^k \\
\leq \frac{2^k}{a - 2^k} \left( \frac{a}{2^k} \right) \hat{n}^{\lg(a)} \\
< \frac{2^k}{a - 2^k} \left( \frac{a}{2^k} \right) (2\hat{n})^{\lg(a)} \\
= \frac{2^k}{a - 2^k} \frac{a}{2^k} \cdot \hat{n}^{\lg(a)} \\
= \frac{a^2}{a - 2^k} \hat{n}^{\lg(a)}
\]

Then this satisfies the conditions of \(O(\hat{n}^{\lg(a)})\) with \(c = \frac{a^2}{a - 2^k}\) and \(n_0 = 1\).

**marking:** 1 mark for defining the \(\hat{n}\) and noting \(\hat{n} < 2n\); 1 mark for applying (e) to \(T(\hat{n})\); 1 mark for developing this to a manageable expression; 1 mark for the \(c, n_0\).

**common errors:** The most common step missed, was failing to notice that the \(-\hat{n}^k\) could be dropped from the start, since we are proving an upper bound. This is not necessarily a mistake, but it makes the rest of the proof messier; and in particular, it made it very likely that the student would next apply \(\hat{n} < 2\hat{n}\) across the expression, and this is an error. Since the expression includes negative and positive \(\hat{n}\) terms, we can’t just do that, need to first factor out as

\[
\frac{2^k}{a - 2^k} \hat{n}^k \left( \frac{a}{2^k} \right) \hat{n}^{\lg(a)-k} - 1
\]

and then note that \(\lg(a) - k \geq 0\) before applying \(\hat{n} < 2\hat{n}\).

Other common error was having trouble getting the \(c, n_0\) correct at the end; or just failing to do this, and instead waffling a bit. There were also a few solutions which only showed \(O(\hat{n}^{\lg(a)})\).
3. (16 marks total) Question on Average sorting - or k-sorting.

Suppose that, instead of sorting an array completely, we only require that the elements increase on average. Formally, for any natural number \( k \geq 1 \), we say that the \( n \)-element array \( A \) is k-sorted if, for every index \( i = 1, 2, \ldots, n - k \), we have:

\[
\frac{\sum_{j=i}^{i+k-1} A[j]}{k} \leq \frac{\sum_{j=i+1}^{i+k} A[j]}{k}.
\]

(a) An array is 1-sorted iff for every \( i = 1, \ldots, n - 1 \) we have \( A[i] \leq A[i+1] \). Therefore an array is 1-sorted iff it is sorted in the traditional sense of sorting.

Marking (a): One mark for right answer.

(b) One example of a 2-sorted array (that is not sorted) is 5, 1, 6, 2, 7, 3, 8, 4, 9, 10. In general any permutation in which \( A[1], A[3], A[5], A[7], A[9] \) is sorted and also \( A[2], A[4], A[6], A[8], A[10] \) is sorted, will be a correct example.

Marking for (b): Three marks for a right answer.

(c) Prove that an array \( A \) is k-sorted if and only if \( A[i] \leq A[i + k] \) for all \( i = 1, 2, \ldots, n - k \).

Proof: Consider the definition of k-sortedness. The defining inequality (for all \( i = 1, \ldots, n - k \)) can be simplified by multiplying by \( k \) to give an equivalent condition:

\[
\sum_{j=i}^{i+k-1} A[j] \leq \sum_{j=i+1}^{i+k} A[j],
\]

for all \( i = 1, \ldots, n - k \).

Now observe that for any \( i \in 1, \ldots, n - k \), we have

\[
\sum_{j=i}^{i+k-1} A[j] \leq \sum_{j=i+1}^{i+k} A[j] \iff A[i] \leq A[i + k]
\]

Therefore an array is k-sorted if and only if \( A[i] \leq A[i + k] \) for all \( i = 1, \ldots, n - k \).

Marking (c): Give 4 marks for a nice correct argument in as much detail as above. The arguments should be presented in terms of \( \iff \), I want both directions - take at least a mark off if this was not done. Also I want the proof done with respect to the condition \( i = 1, \ldots, n - k \).

Common errors: Only doing a proof in the forwards direction (\( \Rightarrow \)). The other common error was mentioning \( i = 1, \ldots, n - k \) at the start and then forgetting all about it. Both of these were very common.
(d) We can k-sort an n-element array in \(O(n \lg(n/k))\) time as follows: [4 marks]

We use the fact from (c). The array is k-sorted if and only if \(A[i] \leq A[i + k]\) for all \(i = 1, \ldots, n - k\).

So we partition the array into k smaller arrays as follows:

For each \(j = 1, \ldots, k\), make a small subarray \(B_j\) of length \(1 + \left\lfloor \frac{(n - j)}{k} \right\rfloor\) containing the elements \(A[j], A[j + k], A[j + 2k], \ldots\).

So we have k arrays of length \(\leq \left\lceil \frac{n}{k} \right\rceil\).

Use an \(O(n \lg n)\) sorting algorithm to sort each of the k arrays. It takes \(O\left(\left(\frac{n}{k}\right) \lg\left(\frac{n}{k}\right)\right)\) time to sort each of the arrays, \(O(n \lg(n/k))\) time overall. Then, for each \(j = 1, \ldots, k\) transfer the elements of the sorted array \(B_j\) into \(A[j], A[j + k], \ldots\) in their sorted order. This takes \(O(n)\) time in total for all \(B_j\) arrays.

Now the if and only if condition of (c) is satisfied, so \(A\) is k-sorted.

Total run time is \(O(n \lg(n/k))\).

**Marking of (d):** 1 mark for details of partitioning.
+1 mark for the individual sorts.
+1 mark for using time for sorting one array as \(O\left(\left(\frac{n}{k}\right) \lg\left(\frac{n}{k}\right)\right)\).
+1 mark for multiplying by \(k\) and adding to \(\Theta(n)\) to get \(O(n \lg(n/k))\).

(e) Show that when \(k\) is a constant, it takes \(\Omega(n \lg n)\) time to k-sort an n-element array. [4 marks]

**answer:** Given: an n-element array which is already k-sorted can be sorted in \(O(n \lg k)\) time.

So here is an alternative algorithm for carrying out a sort (in the traditional sense) of an array \(A\):

1. First k-sort the array \(A\).
2. Then apply the “k-sorted \(\rightarrow\) sorted” algorithm to sort the array.

Suppose we take \(k\) to be some constant. Then \(O(n \lg k)\) is \(O(n)\). So the run time of our alternative algorithm is:

“time to k-sort \(n\) elements” + \(O(n)\).

We know that we have a lower bound for general sorting, of the form \(\Omega(n \lg n)\). Therefore this lower bound must apply to “time to k-sort \(n\) elements” + \(O(n)\).

Therefore, since \(c\) is not \(\Omega(n \lg n)\) for any constant \(c\), the time to k-sort \(n\)-elements (for constant \(k\)) must be \(\Omega(n \lg n)\).

*Can alternatively analyse the decision tree model directly. However the number of necessary permutations at the leaves is \([\left(\frac{n!}{k!}\right)^k]\), not the \(n!\) of 1-sorting. Alternatively they could work with \(\frac{n!}{k!}\) permutations (looking at just sorting one of the k subarrays). Either of these, if done correctly, would get 4 marks.*

**Marking of (e):** Between 0-4 marks depending on quality of answer. Give 1 mark at least if they mention that sorted can be implemented as k-sorting + “sorting a k-sorted array” (even if this doesn’t lead to any lower bound).
**Common errors:** The most common error was assuming that I was asking for a proof that the student’s k-Sort algorithm of (d) was $\Omega(n \lg(n))$. Now - of course it will be (since all algorithms to k-sort for constant k will be), but the question is asking about the *problem of k-sorting*, not a particular algorithm.

Second most common error was showing a decision-tree which modelled the set of 1-sorting algorithms which consist of
1. first k-sorting the array;
2. Then 1-sorting the already k-sorted array.
While this is a legitimate way to show the result we want, a few of these solutions had no introduction/scenario-setting, so weren’t really complete answers.

Third most common error for (e) was just waffling.

Mary Cryan