CS3230 Design & Analysis of Algorithms **5 October, 2021**

Midterm Solution Sketches

Prepared By:

Q1

a.i.

True. RHS $= \Theta(\lg n)$. LHS $= \frac{\lg n}{\log_2 100}$. Pick $c_1 = c_2 = \frac{1}{\log_2 100}, n = 1$.

a.ii.

False. $(\frac{3}{2})^n$ is unbounded. For any choice of c, there must exist some n such that $(\frac{3}{2})^n > c$.

a.iii.

True. LHS $= 2^{\frac{(n)(n+1)}{2}}$. We thus get $2^{\frac{n^2+n}{2}} < cn^{n^2}$, and then $n^2 \lg(\frac{\sqrt{2}}{n}) + \frac{n}{2}\lg(2) < \lg c$. One can observe that the LHS is negative-unbounded due to the first term as n increases, which means that for any chosen c, there must exist some n such that inequality holds.

a.iv.

True. $\lg n$ is unbounded. For any choice of c, there surely exists some n such that $c < \lg n$.

Grading Scheme: 0 or 2.5 for each wrong/correct answer.

b

Want to show that for $n \geq n_0, T(n) \leq c_1 \lg n - c_2$.
Let q be the maximum among $T(2)$, $T(3)$, $T(4)$, $T(5)$. Pick $n_0 = 2, c_1 = q + 4, c_2 = 4$.

Base case ($2 \leq n \leq 5$):
$T(2) \leq q \leq (q+4)\lg 2 - 4 = q$
Check easily that inequality also holds for $T(3), T(4), T(5)$.

Recursive case ($n > 5$):
Assume $T(k) \leq c_1 \lg k - c_2$ for $n > k \geq 2$.
$T(n) = 2T(\sqrt{n}) + 4 \leq 2(c_1 \lg \sqrt{n} - c_2) + 4 = c_1 \lg n + (4 - 2c_2) = c_1 \lg n - c_2$
Shown that for $n \geq n_0, T(n) \leq c_1 \lg n - c_2$, therefore $T(n)$ is $O(\lg n)$.

Grading Scheme:

- **Base case:**

  Mention c_1 large enough or otherwise correctly justify, 2 marks.

If c_1 is set to be equal to a single value of $T(k)$ where $2 \leq k \leq 5$, 2 marks. This shows understanding that c_1 needs to be "big enough" depending on the first few values of $T(n)$.

If c_1 is set to be a constant, or the student mentions $T(n) = O(1)$ for $2 \leq n \leq 5$ only, 0 marks.

- **Induction hypothesis:**

  If $T(n) = c_1 \lg n - c_2$ for some $c_2 \geq 4$, 4 marks.

  If $T(n) = c_1 \lg n - c_2$ for some $0 < c_2 < 4$, 2 marks.

  If $T(n) = c \lg n$, 1 mark.

  If $T(n) = O(\lg n)$, 0 marks.

- **Inductive step:**

  If the argument is correct, 4 marks.

  If the student assumed $T(n) = c \lg n$, and the argument is correct otherwise, 4 marks. This is up to the marker's decision, but generally students might attempt to write $O(\lg n)$ as the conclusion, forcibly remove the $+4$ term or leave the inductive conclusion as $c \lg n + 4$.

  If the student made a calculation error while writing their argument, 2 marks.

  If the student used $T(n) = O(\lg n)$ as inductive hypothesis, capped at 2 marks since the argument might introduce a dependency of c on n.

  Deduct up to 1 mark for major presentation issues or severe typos.

- Deduct up to one mark for improper presentation of induction argument.

- **Correct argument, but not using substitution method** (at most 3 marks)

c

Suppose there exists some deterministic algorithm for computing B that skips some bit, say x_1. Such an algorithm will output the same answer for $1, 1, 0, 0$ and also $0, 1, 0, 0$, since it does not query x_1. However, $B(1, 1, 0, 0) = 1$, $B(0, 1, 0, 0) = 0$, meaning this algorithm will get 1 out of the 2 cases wrong. The exact argument applies for x_2, x_3, x_4 due to symmetry.

Grading Scheme:

- Provide a pair of inputs where the algorithm fails along with a reasonable explanation (7 marks)

- Acknowledge symmetry of unqueried bit across x_1, x_2, x_3, x_4, OR directly show that argument applies to all 4 bits. (3 marks)

Q2

a) For the base case of $n = 2$, correctness follows from lines 2-3 of the algorithm. Assume the algorithm correctly sorts all arrays of size $k <= n - 1$. We want to argue that it then correctly sorts an array of size n. By applying the induction hypothesis, after line 8 is executed, the first two thirds of the array are in sorted order. For the whole array to be correctly sorted, we only need the elements in $A[\ell + 1..n]$ to be in sorted order, and for its elements to be larger then those in $A[1..\ell]$

After step 7, the elements from $A[\ell + 1..n]$ are in sorted order, and step 8 does not modify this part of the array, thus at the end of the algorithm, the elements of $A[\ell + 1..n]$ remain in sorted order. Now, consider the state of the array after step 7 has executed, before step 8 is executed, and consider the element $A[\ell+1]$. This element is larger than the elements in $A[n-\ell+1..\ell]$ by applying the induction hypothesis to step 7. If an element $A[k] \in A[n-\ell+1..\ell]$ was present in this same subarray after step 6 was executed, then we have that $A[\ell+1] > A[k] > A[1..n-\ell]$.

On the other hand, if no such element exists, observe that the number of elements in the upper third of the array $A[\ell+1..n]$ is $n-\ell$ and the number of elements in the middle third $A[n-\ell+1..\ell]$ is $\ell-(n-\ell+1)+1 = 2\ell-n$. We have that $2\ell - n = 2\lceil 2n/3\rceil - n \geq n/3 \geq n - \lceil 2n/3\rceil = n-\ell$, (i.e. the number of elements in the middle third, is larger or equal to the number of elements in the upper third).

Thus, if no element in $A[n-\ell+1..\ell]$ after step 7, was there after step 6, the only possibility is that $2\ell - n = n - \ell$ and all elements in $A[n-\ell+1..\ell]$ after step 6, are now in $A[\ell+1..n]$. These elements are greater then the elements in $A[1..n-\ell]$ given their positions after step 6, and they are also greater than the elements in $A[n-\ell+1..\ell]$ given their positions after step 7.

In either case, we have shown that the elements in $A[\ell+1..n]$ are larger then the elements in $A[1..\ell]$ We have already argued that $A[\ell+1..n]$ are in sorted order after step 7, thus it follows that the array is correctly sorted after step 8. $\square$

Note: Setting $\ell = \lfloor 2n/3\rfloor$ does not work. For instance, consider $n=4$. Then $\ell = \lfloor 2n/3\rfloor = 2$, and the algorithm would sort $A[1..2]$, $A[3..4]$, $A[1..2]$ in steps 6, 7 and 8. This sorting of disjoint halves will of course have inputs on which it fails.

Grading Scheme: 3+5+3+9=20.

- Correctly identify base case ($n = 2$) (3 marks). $n = 0, 1$ are not correct base cases.
- Correctly apply inductive hypothesis to steps 6, 7 and 8 (5 marks).
- Correctly justify the choice of ℓ (3 marks). In most cases, this component was awarded together with the next component.
- Provide a convincing argument for the correctness of the algorithm (9 marks). In most cases, 4 marks were awarded for this component if some understanding of the problem was demonstrated by making some arguments regarding rank of elements, but the justification is not complete.
- We did not deduct marks for not handling the ceiling/floor, i.e. the idea just needs to be correct for when n is a multiple of 3.

b) Let $T(n)$ denote the number of comparisons made by YETANOTHERSORT. We have that:

$$T(n) = \begin{cases} O(1) & \text{if } n = 2 \\ 3T(2n/3) + O(1) & \text{otherwise.} \end{cases}$$

By master theorem case 1, $T(n) = O(n^{\log_{1.5} 3})$ $\square$

Grading Scheme:

- State the base case (2 marks)
- Correctly justify the asymptotic complexity of the recurrence (4 marks for correct recurrence relation, 4 marks for correct asymptotic bound of the recurrence relation)

Q3

a) $O(nk + mk)$ solution: This comes from modifying the Karp-Rabin algorithm in the following manner. First, we hash the k patterns and the initial substring $T[1..m]$ in time $O(mk)$. Now, in the for loop, we check $T[i...i+m-1]$ against the k patterns in each step of the for loop, instead of checking against just a single pattern, which takes time $O(nk)$.

In the analysis, when we account for the chance of a false positive, this false positive can now occur against any of the k patterns. Let $E_{i,j}$ be the event that $T[i+1...i+m]$ is not equal to P_j but has the same hash. Set $K = 200mnk\ln(200mnk)$. Then for each P_j:

$$Pr[E_{i,j}] < m\frac{\ln(200mnk\ln(200mnk))}{200mnk\ln(200mnk)} < \frac{1}{200nk}(1 + \frac{ln(ln(200mnk))}{\ln(200mnk)}) < \frac{1}{100nk} \tag{1.1}$$

It follows by a union bound that $Pr[\cup_{i,j}E_{i,j}] \leq \sum_{i,j}[E_{i,j}] < \frac{1}{100}$.

b) $O(n+mk)$ solution: The improvement here is to use a hash table to store the hash values $h_p(P_1), ..., h_p(P_k)$. Using a hash function from a universal hash family, for any $1 \leq i \leq n-m$, the expected number of collisions between $h_p(T[i..i+m-1])$ and any of $h_p(P_1), \ldots h_p(P_k)$ is < 1 by choosing $M = k$. So, the expected time to query whether $h_p(T[i..i+m-1])$ is already stored in the hash table is $O(1)$. By linearity of expectations, the expected total time over all i is $O(n)$.

Let h_u denote the hash function from the universal hash family. In the pseudocode below, let the update in step 9 be as discussed in the lecture. Then the final pseudocode is as follows:

```
ModifiedKarpRabin
1  Pick a random prime p in range {1, ..., [200mnk ln(200mnk)]}
2  Pick a random hash function h_u from a universal hash family.
3  Initialise a table A of k slots
4  for i = 1 to k
5      Insert h_p(P_i) in slot h_u(h_p(P_i)) of A
6  if h_p(T[1..m]) is in slot h_u(h_p(T[1...m])) of A:
7      return True
8  for i = 1 to n - m
9      Update h_p(T[i...i+m-1]) to h_p(T[i+1...i+m]), using T[i], T[i+m], and pre-computed h_p(2^n)
10     if h_p(T[i+1...i+m]) is in slot h_u(h_p(T[i+1...i+m])) of A:
11         return True
12 return False
```

Grading Scheme:

- Correct algorithm with justification (10 marks)
- Correctly justify that the false positive happens with probability at most 1 per cent (10 marks)
- Correct analysis of the runtime, 2 marks for $O(nk+mk)$, 3 marks for the idea achieving $O(n+mk)$ without justification, remaining 5 marks for full justification.