# CS3230: Design and Analysis of Algorithms
# Semester 2, 2020-21, School of Computing, NUS

Midterm Test

Date: 6th March, 2021, Time Allowed: 2 hours

## Instructions

- This paper consists of FIVE questions and comprises of TWELVE (12) printed pages, including this page.
- Answer ALL the questions.
- Write ALL your answers in this examination book.
- This is an OPEN BOOK examination.
- Write your Student Number (that starts with "A") on the top of every page.

**Student Number:**

**Tutorial Group Number:**

| Question | Maximum | Score |
|---|---|---|
| Q1 | 5 | |
| Q2 | 15 | |
| Q3 | 25 | |
| Q4 | 15 | |
| Q5 | 20 | |
| **Total** | **80** | |

**Question 1 [5 marks]:** Rank the following functions in increasing order of growth; that is, the function $f(n)$ is before function $g(n)$ in your list, only if $f(n) = O(g(n))$.

- $n^{n+4} + n!$
- $n^{11\sqrt{n}}$
- $2^{4n \log n}$
- $4^{n^{1.1}}$
- $n^{11+\frac{\log n}{n}}$

To simplify notations, we write $f(n) < g(n)$ to mean $f(n) = o(g(n))$ and $f(n) \approx g(n)$ to mean $f(n) = \Theta(g(n))$. E.g., the four functions $n^2$, $n$, $2021n^2 + n$ and $n^3$ could be written in increasing order of growth as follows: $n < n^2 \approx (2021n^2 + n) < n^3$.

Note, all the logarithms are of base 2. No need to write down the proofs for this problem.

**Solution**

$$n^{11+\frac{\log n}{n}} < n^{11\sqrt{n}} < n^{n+4} + n! < 2^{4n \log n} < 4^{n^{1.1}}$$

**Grading Scheme** Marking is done by deduction, based on the following scheme:[1]

- If the operation is wrong (e.g.$f \approx g$ or $g < f$), 1 mark is deducted.
- If a function is wrong by $> 1$ out of place (e.g. $g < h < f$), 1.5 marks are deducted.
- If the function is not correctly written (e.g. $4n^{1.1}$ instead of $4^{n^{1.1}}$), 0.5 marks are deducted.

A few examples below help illustrate the above grading scheme. Suppose the correct series is $f < g < h < i < j$. Then,

- If "$f < h < i < j < g$" is written, 1.5 marks are deducted.
- If "$h < i < j < f < g$" is written, 3 marks are deducted ($2 \times 1.5$).
- If "$h < i < j < f \approx g$" is written, 4 marks are deducted.
- If "$j < i < h < g < f$" is written, 5 marks are deducted.

**Note:** Marks were not deducted for non-compliance with the notation given (e.g. use of << instead of $<$ and $\equiv$ instead of $\approx$).

[1] In the following, we take $f < g < h$ to be the correct series

**Question 2 [15 marks]:** Solve the following recurrence relations.

(a) $T(n) = T(n/10) + \log n$.

(b) $T(n) = 27T(n/3) + n^3/\log^2 n$.

(c) $T(n) = 3T(n/5) + \log^2 n$.

(d) $T(n) = 2T(n/3) + n\log n$.

(e) $T(n) = T(n-2) + \log n$.

Among the above recurrences, for each recurrence solvable by master theorem, please indicate which case it belongs to (either case 1, 2, or 3) with proper reasoning and state the (time) complexity. For the recurrences that are not solvable by master theorem, solve them using any of the methods taught in the class. (Express your answers using $\Theta()$ notation.)

**Solution**

(a) Note that

$$\log n = \Theta(n^{\log_{10} 1} \log n).$$

By Case 2 of Master theorem, we have

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

.

(b)

$$\begin{aligned}
T(n) &= 27T(\frac{n}{3}) + \frac{n^3}{\log^2 n} \\
\Rightarrow \frac{T(n)}{n^3} &= \frac{T(\frac{n}{3})}{(\frac{n}{3})^3} + \frac{1}{\log^2 n} \\
\frac{T(\frac{n}{3})}{(\frac{n}{3})^3} &= \frac{T(\frac{n}{9})}{(\frac{n}{9})^3} + \frac{1}{\log^2(\frac{n}{3})} \\
\frac{T(\frac{n}{9})}{(\frac{n}{9})^3} &= \frac{T(\frac{n}{27})}{(\frac{n}{27})^3} + \frac{1}{\log^2(\frac{n}{9})} \\
&\vdots \\
\frac{T(3)}{2^3} &= \frac{T(1)}{1^3} + \frac{1}{\log^2 3}
\end{aligned}$$

By Telescoping, we have

$$T(n) = \frac{1}{(\log_2 3)^2}[\frac{n^3}{(\log_3 n)^2} + \frac{n^3}{(\log_3 n - 1)^2} + \frac{n^3}{(\log_3 n - 2)^2} + \cdots + \frac{n^3}{1^2}] = \frac{n^3}{(\log_2 3)^2} \sum_{x=1}^{\log_3 n} \frac{1}{x^2}.$$

Observe that

$$\begin{aligned}
\sum_{x=1}^{\log_3 n} \frac{1}{x^2} &\le 1 + \sum_{x=2}^{\log_3 n} \frac{1}{(x-1)x} \\
&= 1 + \sum_{x=1}^{\log_3 n - 1} \frac{1}{x(x+1)} \\
&= 1 + \sum_{x=1}^{\log_3 n - 1} (\frac{1}{x} - \frac{1}{x+1}) \\
&= 1 + (1 - \frac{1}{\log_3 n}) \qquad \text{using telescoping series} \\
&\le 2
\end{aligned}$$

Furthermore,

$$\sum_{x=1}^{\log_3 n} \frac{1}{x^2} \geq 1.$$

Hence, we have

$$\sum_{x=1}^{\log_3 n} \frac{1}{x^2} = \Theta(1).$$

Therefore,

$$T(n) = \Theta(n^3).$$

(c) Note that

$$\log^2 n = O(n^{\log_5 3-\epsilon}) \text{ for some } \epsilon > 0.$$

By Case 1 of Master theorem, we have

$$T(n) = \Theta(n^{\log_5 3}).$$

(d) Note that

$$n \log n = \Omega(n^{\log_3 2+\epsilon}) \text{ for some } \epsilon > 0,$$

and

$$\frac{2}{3} n \log(\frac{n}{3}) \leq \frac{2}{3} n \log n.$$

By Case 3 of Master theorem, we have

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

(e) Since $T(n) = T(n-2) + \log n$, by recursion tree/telescoping method, we have

$$T(n) = \sum_{i=0}^{\lfloor \frac{n}{2} \rfloor} \log(n-2i) = \log \prod_{i=0}^{\lfloor \frac{n}{2} \rfloor} (n-2i).$$

By Stirling's approximation, we have

$$\log \prod_{i=0}^{\lfloor \frac{n}{2} \rfloor} (n-2i) = \Theta(n \log n).$$

Therefore,

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

**Grading Scheme**

(a) [2]

- 1 mark for stating Case 2
- 1 mark for final answer

(b) [5]

- 2 marks for using telescoping to obtain $T(n) = \frac{n^3}{(\log_2 3)^2} \sum_{x=1}^{\log_3 n} \frac{1}{x^2}$
- 2 mark for justifying $\sum_{x=1}^{\log_3 n} \frac{1}{x^2} = \Theta(1)$
- 1 mark for final answer

(c) [2]

- 1 mark for stating Case 1
- 1 mark for final answer

(d) [2]

- 1 mark for stating Case 3
- 1 mark for final answer

(e) [4]

- 2 mark for using recursion tree/telescoping to obtain $T(n) = \sum_{i=0}^{\lfloor \frac{n}{2} \rfloor} \log(n-2i)$
- 1 mark for justifying $\sum_{i=0}^{\lfloor \frac{n}{2} \rfloor} \log(n-2i) = \Theta(n \log n)$
- 1 mark for final answer

**Question 3 [7+8+10=25 marks]:** Consider a *circular array*, where the first and the last cells are neighbors. More specifically, in a circular array $A[1, \cdots, n]$ of size $n$, $A[1]$ and $A[n]$ are neighbors of each other. An element in the array is called a *peak* if it is greater than or equal to its neighbors. Now we would like to find a peak in a circular array. (Just to clarify, the array we consider in this question is one dimensional and unsorted.)

(a) Suppose we know the values of $A[1], A[i], A[i+1]$ and $A[n]$ for some $1 < i < n$. Consider the maximum value among these four cells. Then depending on the four possible maximum, in which of the following two circular sub-arrays $A[1, \cdots, i]$ and $A[i+1, \cdots, n]$, are you guaranteed to find a peak? (Note, you have to give answers for the four possible cases. Provide your answer with proper explanation.)

**Solution:** We consider 4 different cases depending on which cell $A[1], A[k], A[k+1], A[n]$ obtains the maximum value.

- $A[1] \geq A[k], A[1] \geq A[k+1], A[1] \geq A[n]$: Assume $A[m]$ is the maximum value in the sub-array $A[1, ..., k]$. If $A[m] = A[1]$, $A[1]$ is a peak, otherwise $1 < m < k$ and $A[m]$ is a peak. Thus a peak of $A$ will be in the sub-array $A[1, ..., k]$.
- $A[k] \geq A[1], A[k] \geq A[k+1], A[k] \geq A[n]$: Assume $A[m]$ is the maximum value in the sub-array $A[1, ..., k]$. If $A[m] = A[k]$, $A[k]$ is a peak, otherwise $1 < m < k$ and $A[m]$ is a peak. Thus a peak of $A$ will be in the sub-array $A[1, ..., k]$.
- $A[k+1] \geq A[1], A[k+1] \geq A[k], A[k+1] \geq A[n]$: Assume $A[m]$ is the maximum value in the sub-array $A[k+1, ..., n]$. If $A[m] = A[k+1]$, $A[k+1]$ is a peak, otherwise $k+1 < m < n$ and $A[m]$ is a peak. Thus a peak of $A$ will be in the sub-array $A[k+1, ..., n]$.
- $A[n] \geq A[1], A[n] \geq A[k], A[n] \geq A[k+1]$: Assume $A[m]$ is the maximum value in the sub-array $A[k+1, ..., n]$. If $A[m] = A[n]$, $A[n]$ is a peak, otherwise $k+1 < m < n$ and $A[m]$ is a peak. Thus a peak of $A$ will be in the sub-array $A[k+1, ..., n]$.

(b) Design a divide-and-conquer algorithm that finds a peak in a circular array of size $n$ in time $O(\log n)$. (Provide the running time analysis of your algorithm.)

**Solution:** Our algorithms for finding a peak in a circular array is a divide-and-conquer approach which is very similar to the algorithm taught in class for finding a peak in 1d-arrays. Using the result of previous part and setting $k = \lfloor \frac{n}{2} \rfloor$, by looking at value of $A[1], A[k], A[k+1]$ and $A[n]$, we can find a sub-array of $A$ (of size half of the size of $A$) that contains a peak of $A$. Moreover, the

stop rule for our recursion is the following: if the size of the circular array is less than 4, return the maximum element in the array (which is trivially a peak).

Let $T(n)$ denote the running time of our algorithm over a circular array of size $n$. Then we have the following recursion relation for $T$:

$$
\begin{aligned}
T(n) &= T(\frac{n}{2}) + 4c \quad \%c \text{ is a constant} \\
T(n) &= T(\frac{n}{4}) + 4c + 4c \\
&\vdots \\
T(n) &= T(\frac{n}{2^i}) + 4c + 4c + ... + 4c \quad \%\text{after } i \text{ iterations}
\end{aligned}
$$

Thus $T(n) = i \cdot 4c + T(\frac{n}{2^i})$ where $2^i \leq \frac{n}{3}$. Setting $i = \log(\frac{n}{3})$ and $T(3) = 4$, we have $T(n) = O(\log n)$.

**Grading Scheme**

(a) (7 marks)

- 1 mark for each case (4 marks for 4 cases). You have to give answers for 4 possible cases.
- 3 marks for a proper explanation of any case. If you only give the explanation of one case, you should mention that the explanations of different cases are similar.

(b) (8 marks)

- 5 marks for an algorithm finding a peak in time $O(\log n)$:
  - how to find a peak by dividing the array into sub-arrays (3 marks).
  - stop rules (2 marks)
- 3 mark for running time analysis:
  - recurrence relation (2 marks).
  - details about how to solve the recurrence relation (1 mark).

  If the algorithm is totally wrong, then there is no any mark for the running time analysis.

(c) Prove a lower bound of $\Omega(\log n)$ on the running time of any comparison-based algorithm that finds a peak in a circular array of size $n$. (Assume, each comparison among two numbers $x, y$ has two possible outcomes: Either $x \leq y$ or $x > y$.)

**Solution** Proving the $\Omega(\log n)$ lower bound for finding a peak in the array $A$ requires two steps:

(1) Showing that there are $n$ possible locations for the peak.

(2) Make an argument using a binary decision tree with at least $n$ leaves to prove the lower bound.

Following the hint, consider an array $A$ that has only *one* peak (so the peak is unique). The location of this peak could be any index $i$ between 0 and $n-1$ inclusive. However, we do not know which index $i$ corresponds to a peak, but we do know that there is exactly one such index $i$ for which $A[i-1] \leq A[i] \geq A[i+1]$. Therefore, there are $n$ possible locations for the peak.

We note that worst-case number of comparisons for a given comparison-based algorithm equals the height of its decision tree. Thus, it suffices to determine the height of a decision tree in which each possible location appears as a reachable leaf. As such, we consider a decision tree of height $h$ with $l$ leaves corresponding to a comparison-based algorithm that finds a peak in a circular array of size $n$. Now, each of the $n$ possible locations has to appear as some leaf, and so $l \geq n$. Since a binary tree of height $h$ has no more than $2^h$ leaves, we have:

$$\begin{aligned} 2^h &\geq l \\ &\geq n \end{aligned}$$

which by taking logarithms, implies $h \geq \Omega(\log n)$ as desired.

**Grading Scheme** Marks are assigned as follows:

(a) (4 marks) Showing $n$ possible locations for the peak
  (i) Consideration of array $A$ that has one peak (1 mark)
  (ii) Argument for number of possible configurations (which is $n$) (2 marks)[2]
  (iii) (Hence) Stating that there are $n$ possible peak locations (1 mark)

(b) (6 marks) Construction of binary decision tree
  (i) Consideration of binary decision tree + Stating each of the $n$ possible locations appear as some leaf ($1 + 2 = 3$ marks)[3]
  (ii) Mentioning association between root-leaf path of decision tree and execution of algorithm (1 mark)
  (iii) Mentioning decision tree is binary $\implies$ height is $\Omega(\log n)$ + Conclusion ($1 + 1 = 2$ marks)[4]

**Remarks:**

- Attempts which mention that total size of possible locations $n!$ will not receive the conclusion mark of 1.
- Attempts which were about the analysis of the algorithm in (b) that did not contain any key ideas listed above were given 0.

[2]Partial credit of 1 mark given for BOD cases
[3]Likewise, partial credit of 1 mark given for BOD cases
[4]Implicit mentioning of former still awards the 1 mark

**Question 4 [15 marks]:** Suppose you are given an array $A$ of $n$ pairs of positive integers $(p_i, q_i)$, where $p_i < n$ and $q_i < n^3$ for all $i \in \{1, 2, \cdots, n\}$. Let us define *r-value* of a pair $(p, q)$ as the real number $\sqrt{p} + q\sqrt{r}$. Unfortunately, you **cannot** (exactly) compute the $r$-value of an arbitrary pair (due to various reasons). Now, design an $O(n)$ time algorithm that sorts (in non-decreasing order) the pairs in $A$ by their $r$-values for $r = n$.

**Algorithm** Sort $A$ by $p_i$ using base-$n$ radix sort. Then sort the resulted array by $q_i$ using base-$n$ radix sort.

**Correctness** Claim 1: If $q_i < q_j$, then $\sqrt{p_i} + q_i\sqrt{n} < \sqrt{p_j} + q_j\sqrt{n}$.

Claim 2: If $q_i = q_j$, then $\sqrt{p_i} + q_i\sqrt{n} < \sqrt{p_j} + q_j\sqrt{n}$ iff $p_i < p_j$.

Similarly to the proof of radix sort, we can show that: for 2 pairs with different $q$, the pair with smaller $q$ precedes in the output array; for 2 pairs with the same $q$, the pair with smaller $p$ precedes in the output array. Together with claim 1 and claim 2, the output is in the correct order of *r-value*.

Time complexity: For all $i$, $p_i$ and $q_i$ have at most 1 and 3 digits base-$n$, respectively. Therefore, the algorithm takes $O(n)$ time.

**Grading Scheme** This is a rough guideline that covers the common cases. A correct algorithm outputs a correct answer and has the required time complexity. A correct proof needs to show the correctness of the output and analyze the time complexity. The proof can be brief/informal, but it must use valid arguments.

- Correct algorithm, with correct proof: 15
- Correct algorithm, with no proof: 10
- Correct algorithm, with bad proof: at most 12.
- Correct algorithm, with wrong proof: at most 9
- Almost correct algorithm, with proof of correctness: at most 10. One common example: Use counting sort (different from radix sort with counting sort as auxiliary) to sort by $q_i$, which results in $\Theta(n^3)$ complexity. Another common example: Sort by $q_i$, then for each subarray of pairs with equal $q_i$, use counting sort to sort by $p_i$. In the case where there are $n$ subarrays (when $q_i$s are distinct), each subarray takes $\Theta(n)$ due to counting sort; the algorithm takes $\Theta(n^2)$ time in total.
- Almost correct algorithm, with no/poor argument: at most 7
- Otherwise: at most 5

We recognized that many answers are similar to the solution of Question 1 from Practice Problem Set 2. Some even include the splitting & merging that is not necessary for this midterm question. In either case, the algorithm is typically correct. However, the proof is not satisfactory. Consider the statement "Since $\sqrt{p_i} < q_i\sqrt{n}$ for all $i$, sorting by $p_i$ followed by sorting stably by $q_i$ gives correct order of *r-value*." This statement is true; however, the argument form is invalid: the fact that the first term is always smaller than the second term is insufficient to show that the algorithm is correct. One can show the invalidity of the argument form by the following test:

Define *s-value* of a pair $(p, q)$ as the number $p + q$. Given a array $B$ of $n$ pairs of positive integers $(p_i, q_i)$, where $p_i < 10$ and $10 < q_i < n^3$ for all $i \in \{1, 2, \cdots, n\}$. By the same argument we get the statement "Since $p_i < q_i$ for all $i$, sorting by $p_i$ followed by sorting stably by $q_i$ gives correct order of *s-value*." This is wrong: with $B = [(9, 11); (1, 12)]$ satisfying $p_i < q_i$ for all $i$, the algorithm outputs $[(9, 11); (1, 12)]$ which is not the correct order of *s-value*.

**Question 5 [20 marks]:** Consider the insertion sort algorithm (perhaps, one of the first sorting algorithms we learned). The algorithm proceeds through the array and puts each element into place one at a time by comparing it to all the preceding items in the array. The details are not important for this question. However, for your reference, let us provide the pseudocode of the insertion sort algorithm. (Note, the pseudocode considers 0 as the starting index of an array. So the input array is $A[0, \cdots, n-1]$. However, for this question, it is not important.)

```
InsertionSort(Array A, integer n)
    for i=1 to (n-1) do
        item = A[i];
        int slot = i;
        while (slot > 0) and (A[slot] > item) do
             A[slot] = A[slot-1];
             slot = slot-1;
        A[slot] = item;
}
```

Figure 1: Pseudocode of the insertion sort algorithm

The key property that you need to consider is that the running time of InsertionSort depends on the number of *inversions* in the permutation being sorted. Given a sequence of distinct integers $\{b_1, b_2, \cdots, b_n\}$, an *inversion* is a pair $(b_i, b_j)$ such that $i < j$ but $b_i > b_j$. E.g., the sequence $\{2, 3, 8, 5, 4\}$ contains only three inversions $(8, 5), (8, 4)$ and $(5, 4)$.

Without loss of generality, assume that the input to the InsertionSort is an (arbitrary) permutation of $\{1, 2, \cdots, n\}$. The following result relates the running time of InsertionSort to the number of inversions in the input sequence: If a permutation $S$ of $\{1, 2, \cdots, n\}$ contains $k$ inversions, then InsertionSort$(S, n)$ runs in time $\Theta(n + k)$. (No need to prove this result. You can assume it.)

Now, analyze the average-case performance of InsertionSort. More specifically, let $S$ be a permutation of $\{1, 2, \cdots, n\}$ chosen uniformly at random from the set of all permutations of $\{1, 2, \cdots, n\}$. Show that the expected running time of InsertionSort on $S$ in $\Theta(n^2)$.

**Solution** Let $\{a_1, a_2, \cdots, a_n\}$ be the random input sequence. Fix two elements of the sequence $i$ and $j$. Without loss of generality, assume $i < j$. Since the input sequence is a random permutation. $\Pr[a_i > a_j] = 1/2$. That is, elements $i$ and $j$ create an inversion with probability $1/2$.

For every pair $(i, j)$ where $i < j$, let $X_{i,j}$ be the indicator random variable that equals 1 if $a_i > a_j$ and 0 otherwise. We know that $\Pr[X_{i,j}] = 1/2$, and hence $\mathbf{E}[X_{i,j}] = 1/2$.

Let $X$ be the total number of inversions in the random input sequence. Notice that $X = \sum_{i<j} X_{i,j}$. Thus:

$$\begin{aligned}
\mathbf{E}[X] &= \mathbf{E}\left[\sum_{i,j} X_{i,j}\right] \\
&= \sum_{i<j} \mathbf{E}[X_{i,j}] \\
&= \sum_{i<j} \frac{1}{2} \\
&= \frac{n(n-1)}{2} \cdot \frac{1}{2} \\
&= \frac{n(n-1)}{4}.
\end{aligned}$$

Thus, the expected number of inversions is $\Theta(n^2)$, and hence the expected running time of InsertionSort is $\Theta(n + n^2) = \Theta(n^2)$.

**Grading Scheme** Marks are assigned as follows:

(a) (8 marks) Showing that $\mathbf{E}[X_{i,j}] = 1/2$.

(b) (10 marks) Showing that $\mathbf{E}[X] = \sum_{i<j} \mathbf{E}[X_{i,j}] = \frac{n(n-1)}{4}$.

(c) (2 marks) Conclude that the expected running time is $\Theta(n + n^2) = \Theta(n^2)$.

- For part (a): if your answer is totally wrong, e.g., $\mathbf{E}[X_{i,j}] = 1/n$: then you get at most 2. If your answer is wrong with minor errors, e.g., $\mathbf{E}[X_{i,j}] = 1/2 - 1/n$: then you get at most 6.
- For part (a) and (b): if you only provide the proof of part (b) without showing $\mathbf{E}[X_{i,j}] = 1/2$, you get at most 6. If you give the correct idea for (a) and (b) with wrong answer, i.e., $\neq \frac{n(n-1)}{4}$, then you get at most 16.
- For part (c), if you only provide the conclusion, then you get 0. If you provide the correct proof for part (a),(b), and (c), you get 20.
- For answers that are not clearly written, you get at most 18.
- If you use $O$, $\Omega$ or $\approx$ for the final solution, you get at most 18.

—End of Paper—