THE UNIVERSITY OF BRITISH COLUMBIA
CPSC 320: MIDTERM EXAM #2 – Group – March 12, 2015

Important notes about this examination

1. **You have 50 minutes to complete this exam.**
2. You are allowed up to three textbooks and (the equivalent of) a 3" 3-ring binder of notes as references. Otherwise, no notes, aids, or electronic equipment are allowed.
3. Please see the regulations regarding student conduct during examinations on the opposite side of this page!
4. All students in your designated group collaborate to produce a single submission.
5. Good luck!

Full Name:_____

Signature:_____

UBC Student #: |_|_|_|_|_|_|_|_|_|

Full Name:_____

Signature:_____

UBC Student #: |_|_|_|_|_|_|_|_|_|

Full Name:_____

Signature:_____

UBC Student #: |_|_|_|_|_|_|_|_|_|

Full Name:_____

Signature:_____

UBC Student #: |_|_|_|_|_|_|_|_|_|

Full Name:_____

Signature:_____

UBC Student #: |_|_|_|_|_|_|_|_|_|

Please do not write in this space:

Question 1:	
Question 2:	
Question 3:	
Question 4:	
Question 5:	
Question 6:	

2 2 3 6 3 7 0 6 0 3 6 4 2

Student Conduct during Examinations

1. Each examination candidate must be prepared to produce, upon the request of the invigilator or examiner, his or her UBCcard for identification.
2. Examination candidates are not permitted to ask questions of the examiners or invigilators, except in cases of supposed errors or ambiguities in examination questions, illegible or missing material, or the like.
3. No examination candidate shall be permitted to enter the examination room after the expiration of one-half hour from the scheduled starting time, or to leave during the first half hour of the examination. Should the examination run forty-five (45) minutes or less, no examination candidate shall be permitted to enter the examination room once the examination has begun.
4. Examination candidates must conduct themselves honestly and in accordance with established rules for a given examination, which will be articulated by the examiner or invigilator prior to the examination commencing. Should dishonest behaviour be observed by the examiner(s) or invigilator(s), pleas of accident or forgetfulness shall not be received.
5. Examination candidates suspected of any of the following, or any other similar practices, may be immediately dismissed from the examination by the examiner/invigilator, and may be subject to disciplinary action:
    i. speaking or communicating with other examination candidates, unless otherwise authorized;
    ii. purposely exposing written papers to the view of other examination candidates or imaging devices;
    iii. purposely viewing the written papers of other examination candidates;
    iv. using or having visible at the place of writing any books, papers or other memory aid devices other than those authorized by the examiner(s); and,
    v. using or operating electronic devices including but not limited to telephones, calculators, computers, or similar devices other than those authorized by the examiner(s)— (electronic devices other than those authorized by the examiner(s) must be completely powered down if present at the place of writing).
6. Examination candidates must not destroy or damage any examination material, must hand in all examination papers, and must not take any examination material from the examination room without permission of the examiner or invigilator.
7. Notwithstanding the above, for any mode of examination that does not fall into the traditional, paper-based method, examination candidates shall adhere to any special rules for conduct as established and articulated by the examiner.
8. Examination candidates must follow any additional examination rules or directions communicated by the examiner(s) or invigilator(s).

- $\sum_{y=1}^{x} y = \frac{x(x+1)}{2}$, for $x \geq 0$.
- $\sum_{y=1}^{x} y^2 = \frac{x(x+1)(2x+1)}{6}$, for $x \geq 0$.
- $\sum_{y=0}^{x} 2^y = 2^{x+1} - 1$, for $x \geq 0$.

For a recurrence like $T(n) = aT(\frac{n}{b}) + f(n)$, where $a \geq 1$ and $b > 1$, the Master Theorem states three cases:

1. If $f(n) \in O(n^c)$ where $c < \log_b a$ then $T(n) \in \Theta(n^{\log_b a})$.
2. If for some constant $k \geq 0$, $f(n) \in \Theta(n^c (\log n)^k)$ where $c = \log_b a$, then $T(n) \in \Theta(n^c (\log n)^{k+1})$.
3. If $f(n) \in \Omega(n^c)$ where $c > \log_b a$ **and** $af(\frac{n}{b}) \leq kf(n)$ for some constant $k < 1$ and sufficiently large n, then $T(n) \in \Theta(f(n))$.

- $f(n) \in O(g(n))$ (big-O, that is) exactly when there is a positive real constant c and positive integer n_0 such that for all integers $n \geq n_0$, $f(n) \leq c \cdot g(n)$.
- $f(n) \in o(g(n))$ (little-o, that is) exactly when for all positive real constants c, there is a positive integer n_0 such that for all integers $n \geq n_0$, $f(n) \leq c \cdot g(n)$.
- $f(n) \in \Omega(g(n))$ exactly when $g(n) \in O(f(n))$.
- $f(n) \in \omega(g(n))$ exactly when $g(n) \in o(f(n))$.
- $f(n) \in \Theta(g(n))$ exactly when $f(n) \in O(g(n))$ and $f(n) \in \Omega(g(n))$.

1 Canopticon [8 marks]

VERBATIM COPY OF PRACTICE PROBLEM TEXT FROM HERE...

Classify each of the following recurrences (assumed to have base cases of $T(1) = T(0) = 1$) into one of the three cases of the Master Theorem—the cases in which **leaves** dominate, in which the **root** dominates, and in which the work is **balanced** across levels—or indicate that the Master Theorem **does not apply**.

...TO HERE. The individual recurrences are different.

1. $T(n) = 2T(\lceil n/4 \rceil) + 2^n$

   leaves **root** **balanced** **does not apply**

2. $T(n) = T(\lfloor n/100 \rfloor) + n^2$

   leaves **root** **balanced** **does not apply**

3. $T(n) = 16T(\lfloor n/4 \rfloor) + n^2$

   leaves **root** **balanced** **does not apply**

4. $T(n) = 81T(\lceil n/3 \rceil) + n^3$

   leaves **root** **balanced** **does not apply**

2 Doctoring the Master Theorem [10 marks]

Answer the questions below about this recurrence:

$$T(n) = \begin{cases} T(\lfloor n/2 \rfloor) + T(\lfloor 2n/3 \rfloor) + n\sqrt{n} & \text{when } n > 1 \\ 1 & \text{when } n \leq 1 \end{cases}$$

Throughout this problem, you may ignore floors and ceilings.

NOTE: the recurrence and some questions or parts of questions have been changed from practice version.

1. Draw a recursion tree for this recurrence. Include **at least two levels (root and children)**. Label your tree with: the **problem size** (in each node), and the **work per node** (next to each node). **[5 marks]**

2. Give the depth of the deepest leaf (ignoring floors and ceilings). **[2 marks]**

3. Use the Master Theorem to **give a Θ-bound on this related recurrence** $T_2(n) = 2T_2(n/2) + n\sqrt{n}$ (which gives an Ω-bound on $T(n)$). **Clearly indicate** a, b, and $f(n)$. **[2 marks]**

4. Use a similar technique to construct a modified recurrence $T_3(n)$ from which we could derive an O-bound on $T(n)$. (**DO NOT** give a bound on your recurrence.) **[1 mark]**

3 The High Price of Plausible Deniability [11 marks]

VERBATIM COPY OF PRACTICE PROBLEM TEXT FROM HERE...

You're solving the interval scheduling problem except **minimizing** the number of jobs performed rather than maximizing it. In particular, we define *the conflict set of a job* to be the set of all jobs that conflict with that job's time range. (Note that the conflict set of a job always includes the job itself.) Your solution should minimize the number of jobs performed while still performing exactly one job from each conflict set.

(Note: we consider two jobs' times to conflict even if the start time of one job is equal to the finish time of the other, i.e., they overlap at only one point.)

UNECESSARY FLAVOR TEXT: Your boss has just given you a list of jobs to perform. Each job has a start time and an end time. You can never do more than one job at a time. You're kind of tired; so, you'd like to do as few jobs as possible, but you can't just do **nothing** or you'll get fired. So, you want to find a list of the smallest number of jobs you can do so that every **other** job conflicts with (has times that overlap) at least one of the jobs you **are** doing.

...TO HERE. Subsequent parts are SUBSTANTIALLY DIFFERENT!

A friend suggests breaking the problem down into two cases: one where we **include** the first job in the solution and one where we **do not** include it.

Your friend's algorithm takes array `ByStart` as input. `ByStart` is sorted in order of increasing start time. Each array entry is an object with a start time field `start` and finish time field `finish`; so, `ByStart[1].start` is the first job's start time and `ByStart[1].finish` is its finish time.

Here is the algorithm:

```
LazyISP(ByStart):
  Return LazyISPHelper(ByStart, 1)


LazyISPHelper(ByStart, i):
  If i > length(ByStart):
    Return 0
  Else:
    Let j be the smallest index for which            // For part 3 below,
      ByStart[j].start > ByStart[i].finish           // assume these three
      or length(ByStart)+1 if no such index exists   // lines take O(1) time
    Let v1 be 1 + LazyISPHelper(ByStart, j)
    Let v2 be LazyISPHelper(ByStart, i+1)
    Return min(v1, v2)
```

1. Give an instance **with no more than 2 jobs** that shows that this algorithm is **not correct**. Indicate **both** what the algorithm returns and the correct answer. **[1 mark]**

2. Give a good Θ-bound on the worst-case runtime of the algorithm if we rewrite it to use dynamic programming (**without** trying to fix it). **Assume** that the three lines beginning at `Let j be the smallest` take constant time. **[2 marks]**

3. Rewrite `LazyISP` below to use dynamic programming (**without** trying to fix it): **[8 marks]**

```
LazyISP(ByStart):

  Let Soln be an array of length _________

  For i = ______________________________________:

    Let j be the smallest index for which
      ByStart[j].start > ByStart[i].finish
      or length(ByStart)+1 if no such index exists

    Let v1 be ____________________________________________________________

    Let v2 be ____________________________________________________________

    ____________________________________________________

  Return ______________________________

                // Feel free to write a helper function below if you need one!
```

4 I'm a k, You're $O(k)$ [13 marks]

NOTE: This problem deals with design of a divide-and-conquer algorithm and its analysis on two variables n and k. Otherwise, this is mostly unlike the practice problem of the same name.

Suppose that we are given an array `A` with n distinct elements and a guarantee that no element in `A` is more than $k-1$ indexes away from the index it belongs at in a sorted array (for some positive integer $k \leq n$), and we want to sort the array.

For example, the array `[15, 3, 19, 12, 16, 10, 21, 18]` in sorted order is: `[3, 10, 12, 15, 16, 18, 19, 21]`. `15` is therefore 3 indexes out of place (would be moved 3 places to the right to put it in the "correct" spot), `3` is only 1 index out of place (would be moved 1 place to the left), and `19` is 4 indexes out of place. Overall, the smallest k we could provide for this array is 5 (because $5-1=4$, and both `19` and `10` are 4 indexes out of place).

1. Complete this divide-and-conquer algorithm to sort an array of length n given the array `A` and the parameter k described above so that it is well-described by the recurrence:

$$T(n) = \begin{cases} 2T(n/2) + T(2k+1) & \text{when } n > 2k+1 \\ k \lg k & \text{when } n \leq 2k+1 \end{cases}$$

Do NOT assume (as in the previous part) that $k \leq n \leq 2k+1$. **[4 marks]**

HINT: Consider an element that belongs in the right half of the sorted array (e.g., something larger than the median). What is the smallest index at which you could find that element in `A`?

```
BoundedSort(A, k):
  BSHelper(A, k, 1, length(A))

BSHelper(A, k, lo, hi):
  If hi - lo + 1 <= 2k + 1:
    MergeSort the portion of A from lo up to hi
  Else:
    Let mid = lo + ceiling((hi - lo)/2)

    BSHelper(A, k, ________________, ________________)

    BSHelper(A, k, ________________, ________________)

    BSHelper(A, k, ________________, ________________)
```

2. Draw a recursion tree for the closely related recurrence relation below. Include **at least two levels (root and children) and one node from the next level**. Label your tree with: the **problem size** (in each node), the **work per node** (next to each node), the **total work per level**, and a **general form for the work per level** (labelling the level i to avoid confusion with k above). **[6 marks]**

   The recurrence is:

$$T(n) = \begin{cases} 2T(n/2) + k \lg k & \text{when } n > 2k+1 \\ k \lg k & \text{when } n \leq 2k+1 \end{cases}$$

3. Use your tree to show that the recurrence above is in $O(n \lg k)$. **Show your work.** You may (correctly!) assume that the height of the tree is $\lg(\frac{n}{k})$. *Reminder:* $\sum_{i=0}^{n} 2^i = 2^{n+1} - 1$. **[3 marks]**

5 I Want the Truth [8 marks]

VERBATIM COPY OF PRACTICE PROBLEM TEXT FROM HERE...

For each statement below, circle **one** answer to indicate whether the statement is **always** true, **never** true, or **sometimes** true for the circumstances indicated. So, if every possibility indicated causes the statement to be true, answer "always". If none causes the statement to be true, answer "never". If some cause the statement to be true and others cause it to be false, answer "sometimes".

...TO HERE. The individual questions are different.

1. Evaluate this statement over the possible input arrays of integers with length of at least 10 passed to the algorithm: The `RandomizedQuickSelect` algorithm picks the smallest element as its pivot.

   always true **never** true **sometimes** true

2. Evaluate this statement over the set of all realistic divide-and-conquer algorithms: Divide-and-conquer algorithms solve non-base-case inputs by: dividing the input into two subproblems, solving the two subproblems, and then computing a solution based on the subproblems' solutions.

   always true **never** true **sometimes** true

3. Evaluate this statement over the legal instances of the closest pair of points problem with at least four points: A pair of points is less than δ apart within the δ-wide strip on the **left side** of the dividing line (i.e., both points are on the left side) on the top-level recursive call to the divide-and-conquer closest pair of points algorithm.

   always true **never** true **sometimes** true

4. Evaluate this statement over the instances of weighted ISP (interval scheduling problem) in which **all weights are the same** (e.g., all 3 or all 7): Running the greedy algorithm for **unweighted** ISP on the instance (with the weights deleted) selects as a solution a set of jobs that would also be optimal if selected in the original instance.

   always true **never** true **sometimes** true

6 Bonus [Up to 4 Bonus Marks]

Bonus marks add to your exam total and also to your course bonus total. What course bonus points are worth **still** isn't clear, however! **WARNING**: These questions are too hard for their point values. We are free to mark these questions harshly. Finish the rest of the exam before attempting these questions. Do not **taunt** these questions.

1. In "I'm a k, You're $O(k)$", justify each of the following statements: **[2 marks]**

   (a) $k \leq n$

   (b) If $n \leq 2k + 1$, then using `MergeSort` to sort the whole array runs in $O(k \lg k)$ time. (Justify this **carefully** for credit!)

   (c) Give an example of a function $f(n)$ such that $f(n) \notin O(f(k))$ even when $n \in \Theta(k)$.

2. Imagine we have $n \geq 0$ indistinguishable (i.e., look-alike) balls and $k \geq 1$ indistinguishable jars. How many different ways are there to put the balls in the jars? Your solution must run in $O(kn)$ time. **[2 marks]**

   Explanation: Consider $n = 2, k = 2$, that is, two balls and two jars. What we mean by indistinguishable balls is that all you know about a jar is how many balls are in it, not which ones. So, if you have 2 balls and 2 jars, putting ball A in jar A and ball B in jar B does not count as a different way to organize the balls from putting ball B in jar A and ball A in jar B (because you cannot tell the balls apart). Furthermore, the jars are indistinguishable; so, putting both balls in jar A and none in jar B is not distinguishable from putting both balls in jar B and none in jar A (because you cannot tell the jars apart, either). So, with $n = 2, k = 2$, there would only be two possible ways to arrange the balls: both in one jar or one in each jar.

This page intentionally left (almost) blank.
If you write answers here, you must CLEARLY indicate on this page what question they belong with AND on the problem's page that you have answers here.

This page intentionally left (almost) blank.
If you write answers here, you must CLEARLY indicate on this page what question they belong with AND on the problem's page that you have answers here.