Instructions that will appear on the real exam

- **DO NOT OPEN THE EXAM UNTIL YOU ARE INSTRUCTED TO.**
- Answer all of the questions as well as you can. You have **one hour.**
- The exam is **non-collaborative**; you must complete it on your own. If you have any clarification questions, please ask the course staff. We cannot provide any hints or help.
- This exam is **closed-book**, except for **up to two double-sided sheets of paper** that you have prepared ahead of time. You can have anything you want written on these sheets of paper.
- **Please DO NOT separate pages of your exam**. The course staff is not responsible for finding lost pages, and you may not get credit for a problem if it goes missing.
- There are a few pages of extra paper at the back of the exam in case you run out of room on any problem. If you use them, please clearly indicate on the relevant problem page that you have used them, and please clearly label any work on the extra pages.
- Please make sure to sign out of the roster when handing in your completed exam to the teaching team.
- **Please do not discuss the exam until after solutions are posted!** (but of course it's fine to discuss this practice exam :))

NOTE for the practice exam: This exam may be less polished than a real exam would be. The goal is to give you a rough sense of the length and format for the real midterm.

General Advice also for the real exam

- If you get stuck on a question or a part, move on and come back to it later. The questions on this exam have a wide range of difficulty, and you can do well on the exam even if you don't get a few questions.
- Pay attention to the point values. Don't spend too much time on questions that are not worth a lot of points.
- There are **100** total points on this exam. There are **three problems** across **eight pages**.

Name and SUNet ID (please print clearly):

SOLUTION

This page intentionally blank. Please do not write anything you want graded here.

Honor Code

Note: Of course you can do anything you want on this practice exam. But this is the statement of the honor code and proctoring policy that will appear on the real exam.

The Honor Code is an undertaking of the Stanford academic community, individually and collectively. Its purpose is to uphold a culture of academic honesty. Students will support this culture of academic honesty by neither giving nor accepting unpermitted academic aid on this examination.

This course is participating in the proctoring pilot overseen by the Academic Integrity Working Group (AIWG), therefore proctors will be present in the exam room. The purpose of this pilot is to determine the efficacy of proctoring and develop effective practices for proctoring in-person exams at Stanford.

Unpermitted Aid on this exam includes but is not limited to the following: collaboration with anyone else; reference materials other than your cheat-sheet (see below); and internet access.

Permitted aid on this exam includes a "cheat-sheet:" two double-sided sheets of paper with anything written on them, which you have prepared yourself ahead of time.

I acknowledge and attest that I will abide by the Honor Code:

[signed] ____

Exam Break Sign-out

I pledge that during my exam break:

- I will not bring any paper, electronic devices (phone, smart watch, smart glasses, etc), or aid (permitted or unpermitted) *out of* or *into* the exam room.
- I will not communicate with anyone other than the course instructional staff about the content of the exam.

Signature Confirming Honor Code Pledge	Exit Time	Return Time	Proctor Initial	Length (min)

If you are feeling unwell and are not able to complete the exam, please connect with the proctor to discuss options.

Good Luck!

This page intentionally blank. Please do not write anything you want graded here.

1. **(45 pt.) [Multiple Choice!]** For each of the parts below, clearly **fill in the box** for **all** of the answers that are true.

   Grading note: In each part below, each of the first five answers are worth three points. The "None of the above" option on its own is not worth any points, it's just there so that you can register that you intentionally didn't fill in anything. If you fill in both "None of the above" and any other answer, we will ignore your "None of the above". If you don't fill in anything and also don't fill in "None of the above", we will assume you did not complete the problem and you will get a zero. Ambiguously filled-in answers will be marked as incorrect.

   [**We are expecting:** *For each part, just clearly fill in answers. No justification is required or will be considered when grading.*]

   (a) **(15 pt.)** Which of the following quantities are $\Omega(n^2)$?

   ☐ (A) $500n^3$
   ☐ (B) $n \log n$
   ☐ (C) The running time of MergeSort
   ☐ (D) The worst-case running time of QuickSort
   ☐ (E) $2^{\log^2 n}$
   ☐ None of the above.

   SOLUTION:
   A, D and E.

   (b) **(15 pt.)** Which of the following recurrence relations satisfy $T(n) = \Theta(n \log n)$?

   ☐ (A) $T(n) = 2T(\lfloor n/2 \rfloor) + 500n$ for $n > 2$; $T(0) = T(1) = 1$.
   ☐ (B) $T(n) = 2T(\lfloor n/2 \rfloor) + \sqrt{n}$ for $n > 100$; $T(n) = 1$ for $n \leq 100$.
   ☐ (C) $T(n) = T(\lfloor n/5 \rfloor) + T(\lfloor 7n/10 \rfloor) + n$ for $n > 10$; $T(n) = 1$ for $n \leq 10$.
   ☐ (D) $T(n) = T(\lfloor \sqrt{n} \rfloor) + n$ for $n > 4$; $T(n) = 1$ for $n \leq 4$.
   ☐ (E) $T(n) = 2T(\lfloor n/2 \rfloor) + \log n$ for $n > 4$; $T(n) = 1$ for $n \leq 4$.
   ☐ None of the above.

   SOLUTION:
   Only (A) (which you can see via the Master method; and we've seen this recurrence a bunch in class). (B) is $\Theta(n)$ (master method or tree method), (C) is $O(n)$ (as we saw in class when analyzing SELECT), (D) is $O(n)$ (you can see this via the tree method), (E) is $\Theta(n)$ (you can see this via the tree method; or you can see that it is $O(n)$ and thus not $\Theta(n \log n)$ by comparing it to (B)).

   (c) **(15 pt.)** Which of the following are true?

☐ (A) "Worst-Case Analysis" means that we look at the running time of an algorithm on the worst possible input.

☐ (B) "Worst-Case Analysis" *only* applies to randomized algorithms, where it refers to choosing the worst-possible randomization.

☐ (C) You can sort a list of n arbitrary comparable elements asymptotically faster with a randomized algorithm than with a deterministic one.

☐ (D) If $f(n) = O(g(n))$, then $\sqrt{f(n)} = O(\sqrt{g(n)})$ for all monotone increasing positive functions f, g.

☐ (E) If $f(n) = O(g(n))$, then $10^{f(n)} = O(10^{g(n)})$ for all monotone increasing positive functions f, g.

☐ None of the above.

SOLUTION:
A and D. A is true by definition. B is false, worst-case analysis applies to deterministic algorithms too. C is false; $\Omega(n \log n)$ is a lower bound even for randomized algorithms. D is true; if $f(n) \leq cg(n)$ for all $n \geq n_0$, then $\sqrt{f(n)} \leq \sqrt{c} \cdot \sqrt{g(n)}$ for all $n \geq n_0$. E is false; a counterexample is $f(n) = 2n$ and $g(n) = n$.

2. **(35 pt.) [Algorithm Design]** Let A be an array of n integers (possibly negative). For $0 \leq i < j < n$, define the *sub-array product* for i, j to be $\prod_{\ell=i}^{j} A[\ell]$.

   (a) **(25 pt.)** Design a *divide and conquer* algorithm that takes A as input and finds the *maximum* sub-array product. For example, if $A = [0, 10, -1, 2, 6]$, then the maximum sub-array product is $12 = A[3] \times A[4]$, corresponding to $i = 3, j = 4$. Your algorithm should run in time $O(n)$.

   For partial credit you may assume that all of the integers in A are non-negative. For partial credit you may give a slower (e.g. $O(n \log n)$-time) algorithm.

   [**We are expecting:** *Clear pseudocode and an English description of what your algorithm is doing. You do not need to prove that it is correct or justify the running time (for now). You may assume that n is a power of two.*]

   Note for practice exam: Depending on how you do this problem, the solution may involve more writing that we would shoot for on an actual exam; we recognize that the exam is only 60 minutes long and writing stuff down takes time. But we hope that the conceptual difficulty of this problem is representative.

SOLUTION:
English Description: We give a divide-and-conquer algorithm. We first divide the array into two halfs, L and R. In each half, we recursively run an algorithm that finds:

- The maximum sub-array product
- The maximum prefix product (aka, corresponding to indices $i = 0$ and $j \geq 0$)
- The minimum (most negative) prefix product
- The maximum suffix product (aka, corresponding to indices $i < n$ and $j = n - 1$)
- The minimum (most negative) suffix product
- The total product

The max sub-array product is then the max of:

- L's max sub-array product
- R's max sub-array product
- L's max sub-array suffix times R's max sub-array prefix
- L's *min* sub-array suffix times L's *min* sub-array prefix

The maximum prefix is the max of:

- L's max sub-array prefix
- The product of all of L, times R's max sub-array prefix

- The product of all of L, times R's min sub-array prefix

and so on.

Note: The problem is simpler if we assume the elements of A are non-negative. In that case, we don't need to keep track of the minimum sub-array products, which are there because the product of two negative numbers is a positive number.

Pseudocode: (Here we assume n is a power of 2, as the question said we are allowed to do).

```
def maxSubArrayProd(A):
    maxProd, maxPre, minPre, maxSuff, minSuff, totalProd
                                   =  maxSubArrayHelper(A)
    return maxProd

def maxSubArrayHelper(A):
    if len(A) == 1:
        return A[0], A[0], A[0], A[0], A[0], A[0]
    n = len(A)
    L = A[:n/2] # left half of the array
    R = A[n/2:] # right half of the array
    LmaxProd, LmaxPre, LminPre, LmaxSuff, LminSuff, Lprod
                                    = maxSumArrayHelper(L)
    RmaxProd, RmaxPre, RminPre, RmaxSuff, RminSuff, Rprod
                                    = maxSumArrayHelper(R)
    maxProd = max( LmaxProd, RmaxProd,
                   LmaxSuff * RmaxPre,
                   LminSuff * RminPre )
    maxPre = max( LmaxPre,
                  Lprod * RmaxPre,
                  Lprod * RminPre )
    minPre = min( LminPre,
                  Lprod * RmaxPre,
                  Lprod * RminPre )
    maxSuff = max( RmaxSuff,
                   Rprod * LmaxSuff,
                   Rprod * LminSuff )
    minSuff = min( RminSuff,
                   Rprod * LmaxSuff,
                   Rprod * LminSuff )
```

```
            totalProd = Rprod * Lprod
            return maxProd, maxPre, minPre, maxSuff, minSuff, totalProd
```

Another part on next page!

more space for previous part...

(b) **(10 pt.)** What do you think the running time of your algorithm in part (a) is? (We asked for $O(n)$, but we will give partial credit for slower running times). In a few (possibly mathematical) sentences, justify your answer.

[**We are expecting:** *A big-Oh running time and an explanation. For your explanation, writing down a recurrence relation that you algorithm satisfies and explaining its solution would be an appropriate level of detail. You do not need to give a formal proof.*]

SOLUTION:
The running time is $O(n)$. The running time satisfies the recurrence relation $T(n) = 2T(n/2) + O(1)$, because the algorithm breaks up the problem into two problems of size $n/2$, and then does $O(1)$ work (taking $O(1)$ maxima of $O(1)$ things) to combined them. By the Master Method with $a = 2, b = 2, d = 0$, the solution to this recurrence relation is $O(n)$.

3. **(20 pt.) [Proving Stuff!]** Let $g(n)$ be defined by:

$$g(n) = \begin{cases} 1 & n \leq 4 \\ g(\lfloor \sqrt{n} \rfloor) + 1 & n > 4 \end{cases}$$

Prove formally, using induction, that $g(n) = O(\log\log n)$. *Note: "$\log\log n$" means* $\log(\log(n))$, *not* $(\log n)^2$.

[**HINT:** *Use the following inductive hypothesis:*

Inductive Hypothesis (for $n \geq 2$***):*** $g(n) \leq \log_2 \log_2 n + 1$.

]

[**We are expecting:** *A formal proof by induction. Be sure to clearly state your inductive hypothesis, base case, inductive step, and conclusion.*]

SOLUTION:
We follow the hint to come up with our inductive hypothesis.

- **Inductive Hypothesis (for $n \geq 2$):** $g(n) \leq \log\log n + 1$, where the logs are base 2.
- **Base Case:** We establish the base case for $n = 2, 3, 4$, corresponding to the base cases in the recursion.
  - $g(4) = 1$ and $\log\log 4 = 1$, so $g(4) \leq \log\log 4 + 1$.
  - $g(3) = 1$ and $0 \leq \log\log 3$, so $g(3) \leq \log\log 3 + 1$.
  - $g(2) = 1$ and $0 = \log\log 2$, so $g(2) \leq \log\log 2 + 1$.
- **Inductive Step:** Let $n > 4$, and suppose that the inductive hypothesis holds for all m so that $2 \leq m < n$. (Note: we are doing strong induction here). We want to show that it holds for n, namely that $g(n) \leq \log\log n + 1$. Notice that since $n > 4$, $\lfloor \sqrt{n} \rfloor$ satisfies $2 \leq \lfloor \sqrt{n} \rfloor < n$, and thus we can apply the inductive hypothesis to $g(\lfloor \sqrt{n} \rfloor)$. Then we have:

$$\begin{aligned} g(n) &= g(\lfloor \sqrt{n} \rfloor) + 1 \\ &\leq \log\log(\lfloor \sqrt{n} \rfloor) + 2 \\ &\leq \log\log(\sqrt{n}) + 2 \\ &= \log(\frac{1}{2} \cdot \log(n)) + 2 \\ &= \log(1/2) + \log\log(n) + 2 \\ &= \log\log(n) + 1. \end{aligned}$$

  This is what we wanted to show, and this establishes the inductive hypothesis for all $n \geq 2$.

- **Conclusion**. We conclude that for all $n \geq 2$, we have $g(n) \leq \log \log n + 1$. In particular, for all $n \geq 4$, we have $g(n) \leq 2 \log \log n$ (this follows since $\log \log n \geq 1$ for $n \geq 4$. Thus, we conclude that $g(n) = O(\log \log n)$, using the definition of big-Oh with $n_0 = 4$ and $c = 2$.

This is the end of the exam!

This is the end of the exam! You can use this page for extra work on any problem. **Keep this page attached** to the exam packet (whether or not you use it), and if you want extra work on this page to be graded, clearly label which question your extra work is for.

This page is for extra work on any problem. **Keep this page attached** to the exam packet (whether or not you use it), and if you want extra work on this page to be graded, clearly label which question your extra work is for.

This page is for extra work on any problem. **Keep this page attached** to the exam packet (whether or not you use it), and if you want extra work on this page to be graded, clearly label which question your extra work is for.

This page is for extra work on any problem. **Keep this page attached** to the exam packet (whether or not you use it), and if you want extra work on this page to be graded, clearly label which question your extra work is for.