CSE548, AMS542: Analysis of Algorithms, Spring 2019 **Date:** May 1, 2019

Final In-Class Exam

(2:30 PM – 3:45 PM : 75 Minutes)

- This exam will account for either 15% or 30% of your overall grade depending on your relative performance in the midterm and the final. The higher of the two scores (midterm and final) will be worth 30% of your grade, and the lower one 15%.
- There are three (3) questions, worth 75 points in total. Please answer all of them in the spaces provided.
- There are 17 pages including four (4) blank pages and one (1) page of appendix. Please use the blank pages if you need additional space for your answers.
- The exam is *open slides* and *open notes*. But *no books* and *no computers* (no laptops, tablets, capsules, cell phones, etc.).

Good Luck!

Question	Pages	Score	Maximum
1. Splitting into Parentheses-free Statements	2–6		25
2. Stealing from Random Victims	8–10		30
3. Evaluating Polynomials	12–14		20
Total			75

Name: _____

Question 1. [25 Points] Splitting into Parentheses-free Statements. Given a valid arithmetic statement $E[1 : n]$ containing n items the function Process-Statement shown in Figure 1 rewrites E as a sequence of parentheses-free statements. We assume that $E[n] =$';' (i.e., end of statement) and each other item $E[i]$ for $1 \leq i < n$ is either a variable name or a constant or a symbol chosen from the following set.

$$+ \qquad - \qquad \times \qquad \div \qquad := \qquad (\qquad)$$

All parentheses in E are matched, and variable names r_j for $j > 0$ do not appear in E.

Figure 2 shows how Process-Statement splits the arithmetic expression

$$x := x + ((a - b) \times (c + d) + e) \div f;$$

into the following four parentheses-free statements:

$$\begin{aligned} r_1 &:= a - b; \\ r_2 &:= c + d; \\ r_3 &:= r_1 \times r_2 + e; \\ x &:= x + r_3 \div f; \end{aligned}$$

1(a) **[4 Points]** What is the worst-case cost of a single call to Process-Item assuming that the input E to Process-Statement has n items? Justify your answer.

```
PROCESS-STATEMENT( E[1 : n] )   {process the arithmetic statement stored in E[1 : n] in left to right direction}
 1. S ← ∅                       {empty global stack S which is accessible from both functions below}
 2. PUSH( S, '$' )              {$ is our empty stack marker}
 3. j ← 1                       {global variable j to keep track of output variables r_j, j = 1, 2, ...}
 4. for i ← 1 to n do
 5.     PROCESS-ITEM( E[i] )    {process the i-th item (from left) of statement E[1 : n]}
```

```
PROCESS-ITEM( z )   {process item z based on the previously read/processed items stored in global stack S}
 1. if z ≠')' and z ≠';' then   {if z is neither a closing parenthesis nor the end of statement}
 2.     PUSH( S, z )            {push z into the global stack}
 3. elif z =')' then            {else if z is a closing parenthesis}
 4.     print "r_j :="          {we will create an assignment statement for r_j}
 5.     PRINT-PARENTHESES-FREE-STATEMENT( '(' )   {create a parentheses-free statement by popping items
                                                   from stack S until hitting an opening parenthesis}
 6.     PUSH( S, r_j )          {push the new variable r_j into S}
 7.     j ← j + 1
 8. else                        {we have reached the end of the input statement}
 9.     PRINT-PARENTHESES-FREE-STATEMENT( '$' )   {create a parentheses-free statement by popping items
                                                   from stack S until hitting the bottom of S}
10. fi
```

```
PRINT-PARENTHESES-FREE-STATEMENT( q )   {create a parentheses-free statement by popping items
                                         from the global stack S until hitting symbol q}
 1. S' ← ∅                      {empty stack S' for temporary storage}
 2. y ← POP( S )
 3. while y ≠ q do              {transfer items from S to S' until hitting symbol q}
 4.     PUSH( S', y )           {push the current item into S'}
 5.     y ← POP( S )            {pop a new item from S}
 6. endwhile
 7. while S' ≠ ∅ do             {create and print a parentheses-free assignment statement from S'}
 8.     y ← POP( S' )           {pop the next item from S'}
 9.     print "y"
10. endwhile
11. print ";"                   {print end marker}
12. print newline
```

Figure 1: Given a valid arithmetic statement $E[1:n]$ as input PROCESS-STATEMENT rewrites the statement as a sequence of parentheses-free statements.

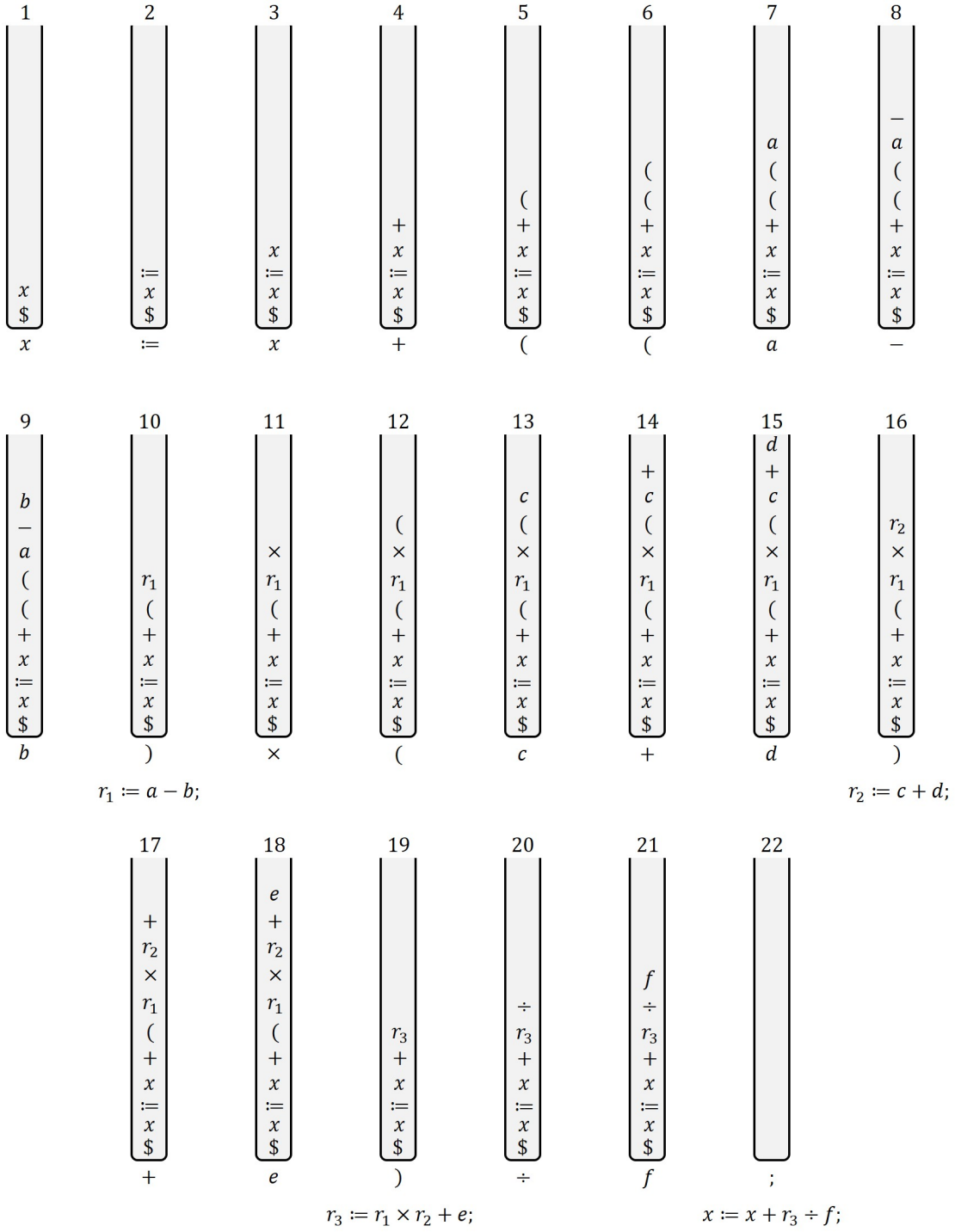

Figure 2: State of the stack S and the expression generated (if any) after each item from the following expression processed from left to right: "$x := x + ((a - b) \times (c + d) + e) \div f$;".

1(b) [**15 Points**] Use either the accounting method or the potential method to show that the amortized cost of PROCESS-ITEM is $\Theta(1)$.

1(c) [**6 Points**] What is the total worst-case running time of PROCESS-STATEMENT($E[1:n]$) based on your result from part 1(a)? What is the total worst-case running time based on your results from part 1(b)?

Use this page if you need additional space for your answers.

Question 2. [**30 Points**] **Stealing from Random Victims.** Suppose a multicore machine has p processing cores and each core is running a thread. Initially, only one thread has n units of work and all other threads are idle. But whenever a thread is idle (i.e., has no work) it becomes a thief. It chooses a thread uniformly at random (including itself) as a victim and steals half the work of the victim provided the victim has at least 2 units of work. If it is able to steal something it starts working on the stolen work and does not try stealing until it runs out of work again. If the steal attempt fails it tries again.

We assume for simplicity that in the entire system no two steal attempts happen at exactly the same time. We order all steal attempts in the entire system in increasing order of their time of occurence and group them into rounds. We assume for simplicity that each round will have exactly $\alpha p \ln p$ consecutive steal attempts for some $\alpha > 1$.

Now answer the following questions.

2(a) [**10 Points**] Prove that after the first round of stealing no thread will have more than $\frac{n}{2}$ units of work w.h.p. in p provided $\alpha > 1$.

2(b) [**10 Points**] Prove that w.h.p. in p after $\log_2 n$ rounds of stealing every steal attempt will fail provided $\alpha \geq 2 + \frac{\log \log_2 n}{\log p}$.

2(c) [**10 Points**] Show that during any given round no thread will be the victim of more than $\Theta(\ln p)$ steal attempts w.h.p. in p. You can assume $\alpha > 8$ for simplicity.

Use this page if you need additional space for your answers.

Question 3. [20 Points] Evaluating Polynomials. In this task we want to evaluate the following polynomial of degree $n-1$ at some given point $\hat{x}$, where $a_0, a_1, \ldots, a_{n-1}$ are given constants.

$$P(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_{n-1} x^{n-1}$$

Figure 3 shows a function Eval-Poly that calls another function named Rec-Eval-Poly for evaluating $P(x)$ at point $\hat{x}$ recursively. The basic idea is to split the polynomial into two halves, recursively evaluate both halves at $\hat{x}$, and combine the results from the two halves to obtain the final solution.

```
EVAL-POLY( x̂, ⟨a_0, a_1, ..., a_{n-1}⟩ )      {evaluate P(x̂) = a_0 + a_1 x̂ + a_2 (x̂)^2 + ... + a_{n-1} (x̂)^{n-1}}
 1. array A[0 : n − 1],  X[0 : n − 1],  V[0 : n − 1]
 2. for i ← 0 to n − 1 do A[i] ← a_i                {initialize A[0 : n − 1] with a_0, a_1, ..., a_{n-1}}
 3. REC-EVAL-POLY( x̂, A, 0, n − 1, X, V )          {recursively evaluate P(x̂)}
 4. return V[n − 1]                                 {V[n − 1] contains the sum of all terms of P(x̂)}
```

```
REC-EVAL-POLY( x̂, A, q, r, X, V )
 1. if q = r then                                   {base case: problem of size 1}
 2.     X[q] ← 1
 3.     V[q] ← A[q]
 4. else
 5.     m ← ⌊(q+r)/2⌋                               {take midpoint}
 6.     REC-EVAL-POLY( x̂, A, q, m, X, V )           {evaluate left part}
 7.     REC-EVAL-POLY( x̂, A, m + 1, r, X, V )       {evaluate right part}
 8.     for i ← m + 1 to r do                       {combine left and right parts}
 9.         X[i] ← x̂ · X[m] · X[i]                  {X[m] contains x̂^{m−q}}
10.         V[i] ← V[m] + x̂ · X[m] · V[i]
```

Figure 3: To evaluate polynomial $P(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_{n-1} x^{n-1}$ at a given point $\hat{x}$ call Eval-Poly($\hat{x}$, $\langle a_0, a_1, \ldots, a_{n-1} \rangle$).

3(a) [**8 Points**] Parallelize the two functions given in Figure 3 so that they still produce correct results. You do not need to write the entire pseudocode for this parallel version. Simply modify the pseudocode in Figure 3 by (clearly and unambiguously) writing ***parallel*** to the left of the line number of each ***for*** loop you want to parallelize and by writing down ***spawn*** and ***sync*** keywords at the right places.

For each ***for*** loop justify your decision (in a sentence or two), i.e., if you parallelized a ***for*** loop explain why that ***parallel for*** loop will still produce correct results, and if you chose not to parallelize a ***for*** loop explain how parallelizing it will produce incorrect results. Similarly justify your ***spawn*** and ***sync*** decisions.

3(b) [**12 Points**] Find the work, span and parallelism of your parallel version of EVAL-POLY from part 3(a). What is its parallel running time on p processing cores under a greedy scheduler?

Use this page if you need additional space for your answers.

Use this page if you need additional space for your answers.

Appendix I: Useful Tail Bounds

Markov's Inequality. Let X be a random variable that assumes only nonnegative values. Then for all $\delta > 0$, $Pr[X \geq \delta] \leq \frac{E[X]}{\delta}$.

Chebyshev's Inequality. Let X be a random variable with a finite mean $E[X]$ and a finite variance $Var[X]$. Then for any $\delta > 0$, $Pr[|X - E[X]| \geq \delta] \leq \frac{Var[X]}{\delta^2}$.

Chernoff Bounds. Let $X_1, \ldots, X_n$ be independent Poisson trials, that is, each X_i is a 0-1 random variable with $Pr[X_i = 1] = p_i$ for some p_i. Let $X = \sum_{i=1}^{n} X_i$ and $\mu = E[X]$. Following bounds hold:

Lower Tail:

- for $0 < \delta < 1$, $Pr[X \leq (1-\delta)\mu] \leq \left(\frac{e^{-\delta}}{(1-\delta)^{(1-\delta)}}\right)^{\mu}$
- for $0 < \delta < 1$, $Pr[X \leq (1-\delta)\mu] \leq e^{-\frac{\mu\delta^2}{2}}$
- for $0 < \gamma < \mu$, $Pr[X \leq \mu - \gamma] \leq e^{-\frac{\gamma^2}{2\mu}}$

Upper Tail:

- for any $\delta > 0$, $Pr[X \geq (1+\delta)\mu] \leq \left(\frac{e^{\delta}}{(1+\delta)^{(1+\delta)}}\right)^{\mu}$
- for $0 < \delta < 1$, $Pr[X \geq (1+\delta)\mu] \leq e^{-\frac{\mu\delta^2}{3}}$
- for $0 < \gamma < \mu$, $Pr[X \geq \mu + \gamma] \leq e^{-\frac{\gamma^2}{3\mu}}$

Appendix II: The Master Theorem

Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function, and let $T(n)$ be defined on the nonnegative integers by the recurrence

$$T(n) = \begin{cases} \Theta(1), & \text{if } n \leq 1, \\ aT\left(\frac{n}{b}\right) + f(n), & \text{otherwise,} \end{cases}$$

where, $\frac{n}{b}$ is interpreted to mean either $\lfloor \frac{n}{b} \rfloor$ or $\lceil \frac{n}{b} \rceil$. Then $T(n)$ has the following bounds:

Case 1: If $f(n) = \mathcal{O}\left(n^{\log_b a - \epsilon}\right)$ for some constant $\epsilon > 0$, then $T(n) = \Theta\left(n^{\log_b a}\right)$.

Case 2: If $f(n) = \Theta\left(n^{\log_b a} \log^k n\right)$ for some constant $k \geq 0$, then $T(n) = \Theta\left(n^{\log_b a} \log^{k+1} n\right)$.

Case 3: If $f(n) = \Omega\left(n^{\log_b a + \epsilon}\right)$ for some constant $\epsilon > 0$, and $af\left(\frac{n}{b}\right) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$.