Due date: Monday, March 1, 9:00am

Submission Instructions:

• Each person must submit a solution to this assignment on Moodle on the due date.

• Assignments must be typed. I suggest using Latex. If the grader (me!) cannot (easily) read an answer, she will give it a 0.

• Each group will work together, and create one solution, but each member must submit a solution to Moodle.

• Your submission must have the name of every group member clearly marked.

• All arguments should be clear, concise, and (of course) correct. You will lose points for poor writing, bad grammar, etc...

• Use of the Internet for solving these problems is strictly forbidden, and will be treated as an honor code violation.

• Inter-group work: Members of different groups may discuss the questions. However:
  ● With your submission you must provide a list of all non-group members with whom any member of your group discussed the assignment.
  ● No written material may leave the inter-group discussion. If you talk with someone about the assignment, you must throw away and written notes at the end of the discussion.
Problem 1 (20 points): Suppose I have a function $foo$ that takes a variable $n$ and runs in $\Theta(n)$ time. Give $\Theta$ bounds for each of the code snippets. (You do not need to prove your answer, but you should provide some explanation.)

- for $i \leftarrow 1$ to $n$:
  - $\text{foo}(n)$

- for $i \leftarrow 1$ to $n$:
  - for $j \leftarrow 1$ to $n$:
    - $\text{foo}(n)$

- for $i \leftarrow 1$ to $n$:
  - $\text{foo}(i)$

- $p \leftarrow 1$
  - for $i \leftarrow 1$ to $n$
    - if $i == p$:
      - $\text{foo}(i)$
      - $p \leftarrow 2p$
    - $\text{foo}(1)$
Problem 2 (20 points): Consider the following algorithm:

// Preconditions: M1 and M2 are nxn matrices
Matrix MatrixMult(Matrix M1, Matrix M2, int n)
(1) Array S; // This is pseudocode -- we won’t worry about sizing.
(2) for i <-- 0 to n-1
(3) for j <-- 0 to n-1
(4) for k <-- 0 to n-1
(5) M[i][j] = M1[i][k]*M2[k][j]
(6) return M

1. Give a tight run time bound for the MatrixMult(A1, A2, n) algorithm in terms of n.

2. MatrixMult solves the problem of multiplying two $n \times n$ matrices. Give a good lower bound (e.g. $\Omega$) for this problem.

3. Based (only) on your answers to (1) and (2), can you give a $\Theta$ bound for the problem?
Problem 3 (30 points): Let $L$ be an array of $n$ integers, and let:

$$f(i, j) = \sum_{k=i}^{j-1} L[k]$$

(That is, the sum of all the elements from $L[i]$ up to, but not including, $L[j]$.) Describe a data structure that can return the value of $f(i, j)$ (for any $0 \leq i < j < n$) in $\Theta(1)$ time and that can be created from $L$ in $O(n)$ time.
Problem 4 (20 points):

You work for a search engine company, maintaining a database that maps search terms to web addresses. Periodically this database needs to be reindexed, which takes a serious amount of computation time. To do this job, the company owns a single supercomputer, and an essentially unlimited supply of high-end Mac computers\footnote{I should really say Higher end”. With a Mac, the High end” is redundant.}.

You have broken the overall computation into $n$ distinct jobs, labeled $J_1, J_2, \ldots, J_n$, which can performed completely independently of one another. Each job consists of two stages: it needs to be preprocessed on the supercomputer, and then finished on one of the Macs. For any $i$, job $J_i$ requires $p_i$ seconds of pre-processing on the supercomputer, followed by $f_i$ seconds to finish on a Mac.

You have $n$ jobs, and at least $n$ Macs – so the jobs can be finished in parallel. However, the supercomputer can only work on a single job at a time, and thus you need to work out the correct order in which to feed the jobs into it. When the supercomputer has finished pre-processing a job, that job will be automatically handed off to an idle Mac for finishing and the supercomputer will immediately begin work on the next job. There is no lag time finishing between the completion of one pre-processing job and starting the next.

A schedule is an ordering of jobs for the supercomputer, and the completion time of the schedule is the earliest time at which all jobs have finished being processed by the Macs. The updated database cannot go on line until all jobs are completed, so clearly, the best” schedule is the one that has the earliest completion time.

Give a polynomial time algorithm that finds a best schedule (i.e. a schedule with the minimum possible completion time), and prove that your algorithm is correct.