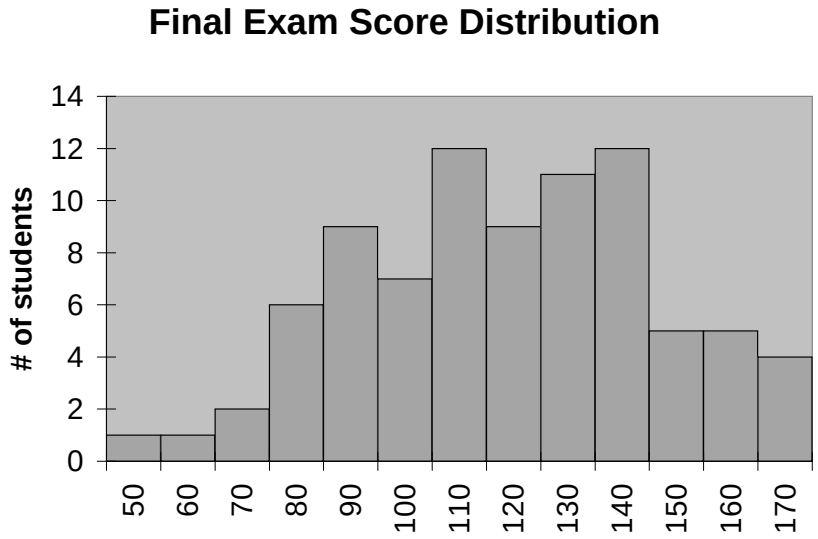
*Introduction to Algorithms* December 23, 2005
Massachusetts Institute of Technology 6.046J/18.410J
Professors Erik D. Demaine and Charles E. Leiserson Handout 36

# Final Exam Solutions



**Problem 1. Recurrences** [15 points] (3 parts)

Give a tight asymptotic upper bound ($O$ notation) on the solution to each of the following recurrences. You need not justify your answers.

**(a)** $T(n) = 2T(n/8) + \sqrt[3]{n}$.

**Solution:** $\Theta(n^{1/3} \lg n)$ by Case 2 of the Master Method.

**(b)** $T(n) = T(n/3) + T(n/4) + 5n$

**Solution:** $\Theta(n)$.

**(c)** $T(n) = \begin{cases} 8T(n/2) + \Theta(1) & \text{if } n^2 > M, \\ M & \text{if } n^2 \leq M; \end{cases}$
where $M$ is a variable independent from $n$.

**Solution:** $\Theta\left(n^3/\sqrt{M}\right)$. The recursion tree has approximately $\lg n - \lg\sqrt{M} = \lg\left(n/\sqrt{M}\right)$ levels. In the tree, every internal node has 8 children, each with cost $O(1)$. At the bottom of the tree, there are approximately $8^{\lg\left(n/\sqrt{M}\right)} = \left(n/\sqrt{M}\right)^3$ leaves. Since each leaf costs $M$, and the total cost is dominated by the leaves, the solution is $\Theta\left(M\left(n/\sqrt{M}\right)^3\right) = \Theta\left(n^3/\sqrt{M}\right)$.

**Problem 2. Algorithms and running times** [9 points]

Match each algorithm below with the tightest asymptotic upper bound for its worst-case running time by inserting one of the letters A, B, . . ., I into the corresponding box. For sorting algorithms, $n$ is the number of input elements. For matrix algorithms, the input matrix has size $n \times n$. For graph algorithms, the number of vertices is $n$, and the number of edges is $\Theta(n)$.

You need not justify your answers. Some running times may be used multiple times or not at all. Because points will be deducted for wrong answers, do not guess unless you are reasonably sure.

| | | |
|---|---|---|
| ☐ | Insertion sort | A: $O(\lg n)$ |
| ☐ | Heapsort | B: $O(n)$ |
| ☐ | BUILD-HEAP | C: $O(n \lg n)$ |
| ☐ | Strassen's | D: $O(n^2)$ |
| ☐ | Bellman-Ford | E: $O(n^2 \lg n)$ |
| ☐ | Depth-first search | F: $O(n^{2.5})$ |
| ☐ | Floyd-Warshall | G: $O(n^{\lg 7})$ |
| ☐ | Johnson's | H: $O(n^3)$ |
| ☐ | Prim's | I: $O(n^3 \lg n)$ |

**Solution:** From top to bottom: D, C, B, G, D, B, H, E, C.

**Problem 3. Substitution method** [10 points]

Use the substitution method to prove a tight asymptotic lower bound ($\Omega$-notation) on the solution to the recurrence

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

**Solution:** By the master method, we know $T(n) = \Theta(n^2 \lg n)$. Therefore, our induction hypothesis is $T(m) \geq cm^2 \lg m$ for all $m < n$.

For $m = 1$, we have the base case that $T(1) = 1 > c1^2 \lg 1$ for all $c > 0$.

For the inductive step, assume for all $m < n$, $T(m) \geq cm^2 \lg m$. The induction hypothesis yields

$$\begin{aligned} T(n) &= 4T(n/2) + n^2 \\ &\geq 4c\left(\frac{n}{2}\right)^2 \lg\left(\frac{n}{2}\right) + n^2 \\ &= cn^2 \lg n - cn^2 \lg 2 + n^2 \\ &= cn^2 \lg n + (1-c)n^2. \end{aligned}$$

For $c < 1$, this quantity is always greater than $cn^2 \lg n$. Therefore, $T(n) = \Omega(cn^2 \lg n)$.

**Problem 4. True or False, and Justify** [35 points] (7 parts)

Circle **T** or **F** for each of the following statements to indicate whether the statement is true or false, respectively. If the statement is correct, briefly state why. If the statement is wrong, explain why. The more content you provide in your justification, the higher your grade, but be brief. Your justification is worth more points than your true-or-false designation.

**T F** Let $A_1$, $A_2$, and $A_3$ be three sorted arrays of $n$ real numbers (all distinct). In the comparison model, constructing a balanced binary search tree of the set $A_1 \cup A_2 \cup A_3$ requires $\Omega(n \lg n)$ time.

**Solution:** **False.** First, merge the three arrays, $A_1$, $A_2$, and $A_3$ in $O(n)$ time. Second, construct a balanced binary search tree from the merged array: the median of the array is the root; recursively build the left subtree from the first half of the array and the right subtree from the second half of the array. The resulting running time is $T(n) = 2T(n/2) + O(1) = O(n)$.

**T F** Let $T$ be a complete binary tree with $n$ nodes. Finding a path from the root of $T$ to a given vertex $v \in T$ using breadth-first search takes $O(\lg n)$ time.

**Solution:** **False.** Breadth-first search requires $\Omega(n)$ time. Breadth-first search examines each node in the tree in breadth-first order. The vertex $v$ could well be the last vertex explored. (Also, notice that $T$ is not necessarily sorted.)

**T F** Given an unsorted array $A[1 \,..\, n]$ of $n$ integers, building a max-heap out of the elements of $A$ can be performed asymptotically faster than building a red-black tree out of the elements of $A$.

**Solution:** **True.** Building a heap takes $O(n)$ time, as described in CLRS. On the other hand, building a red-black tree takes $\Omega(n \lg n)$ time, since it is possible to produce a sorted list of elements from a red-black tree in $O(n)$ time by doing an in-order tree walk (and sorting requires $\Omega(n \lg n)$ time in a comparison model).

**T F** Suppose we use a hash function $h$ to hash $n$ distinct keys into an array $T$ of length $m$. Assuming simple uniform hashing, the expected number of colliding pairs of elements is $\Theta(n^2/m)$.

**Solution:** **True.** Let $X_{i,j}$ be an indicator random variable equal to 1 if elements $i$ and $j$ collide, and equal to 0 otherwise. Simple uniform hashing means that the probability of element $i$ hashing to slot $k$ is $1/m$. Therefore, the probability that $i$ and $j$ both hash to the same slot $\Pr(X_{i,j}) = 1/m$. Hence, $\mathrm{E}[X_{i,j}] = 1/m$. We now use linearity of expectation to sum over all possible pairs $i$ and $j$:

$$\begin{aligned}
\mathrm{E}[\text{number of colliding pairs}] &= \mathrm{E}\left[\sum_{i=1}^{n}\sum_{j=i+1}^{n} X_{i,j}\right] \\
&= \sum_{i=1}^{n}\sum_{j=i+1}^{n} \mathrm{E}[X_{i,j}] \\
&= \sum_{i=1}^{n}\sum_{j=i+1}^{n} 1/m \\
&= \frac{n(n+1)}{2m} \\
&= \Theta(n^2/m)
\end{aligned}$$

**T F** Every sorting network with $n$ inputs has depth $\Omega(\lg n)$.

**Solution:** **True.** Let $d$ be the depth of the network. Since there are $n$ inputs to the network, there can be at most $nd$ comparators. (One way of seeing this is by the pigeonhole principle: if there are $n$ wires and more than $nd$ comparators, then some wire must traverse more than $d$ comparators, resulting in a depth $> d$.)

In the comparison-based model, it is possible to simulate the sorting network one comparator at a time. The running time is equal to the number of comparators (times a constant factor overhead). Therefore, every sorting network must have at least $\Omega(n \lg n)$ comparators, by the lower-bound for sorting in the comparison-based model.

Therefore, $nd > \Omega(n \lg n)$, implying that the depth $d$ is $\Omega(\lg n)$.

**T F** If a dynamic-programming problem satisfies the optimal-substructure property, then a locally optimal solution is globally optimal.

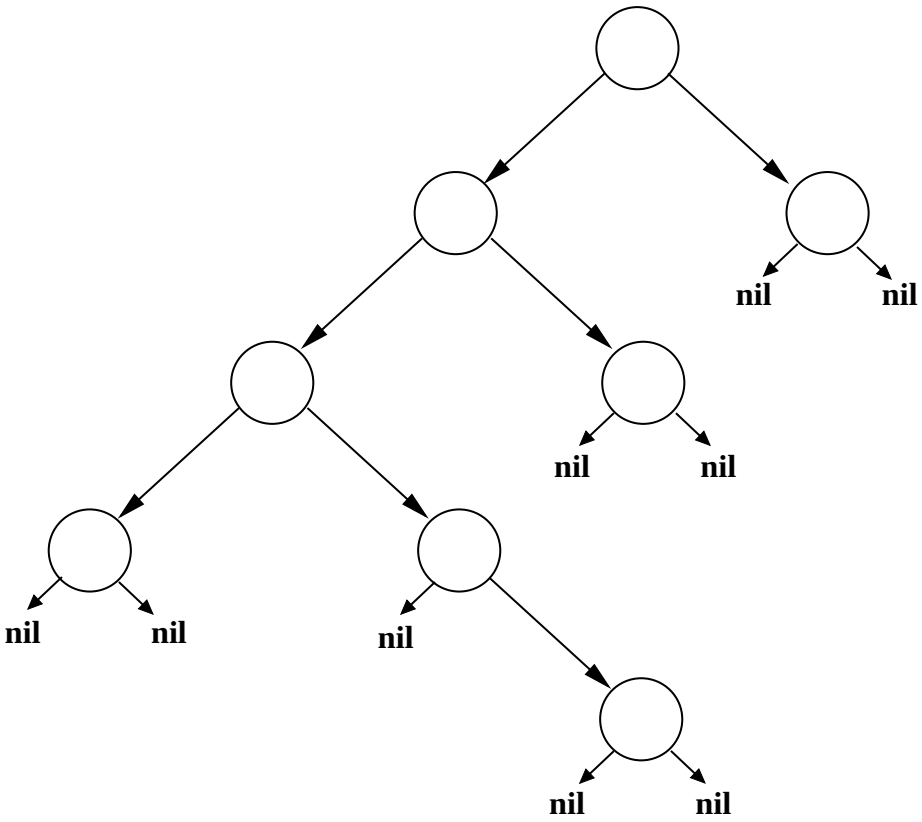
**Solution:** **False.** The property which implies that locally optimal solutions are globally optimal is the *greedy-choice property*. Consider as a counterexample the edit distance problem. This problem has optimal substructure, as was shown on the problem set, however a locally optimal solution—making the best edit next—does not result in a globally optimal solution.

**T F** Let $G = (V, E)$ be a directed graph with negative-weight edges, but no negative-weight cycles. Then, one can compute all shortest paths from a source $s \in V$ to all $v \in V$ faster than Bellman-Ford using the technique of reweighting.

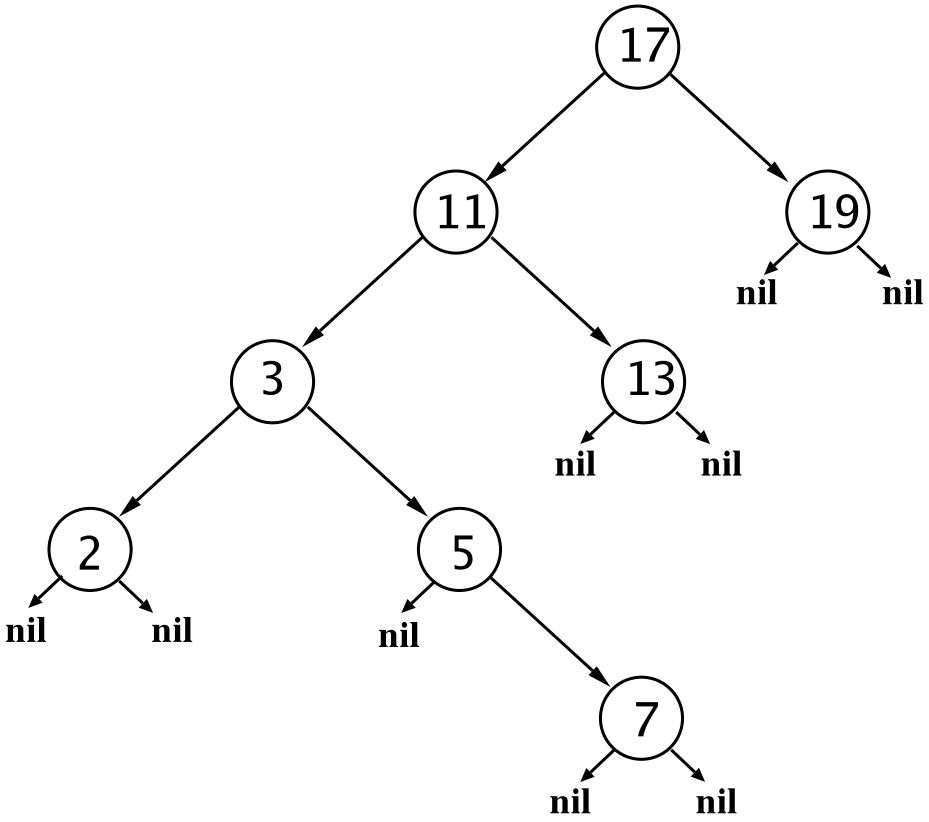
**Solution:** **False.** The technique of reweighting preserves the shortest path by assigning a value $h(v)$ to each vertex $v \in V$, and using this to calculate new weights for the edges: $\tilde{w}(u, v) = w(u, v) + h(u) - h(v)$. However, to determine values for $h(v)$ such that the edge weights are all non-negative, we use Bellman-Ford to solve the resulting system of difference constraints. Since the technique of reweighting relies on Bellman-Ford, it cannot run faster than Bellman-Ford.

**Problem 5. Red-black trees** [15 points] (3 parts)

**(a)** Assign the keys $2, 3, 5, 7, 11, 13, 17, 19$ to the nodes of the binary search tree below so that they satisfy the binary-search-tree property.




**Solution:** 5 points for the correct answer:



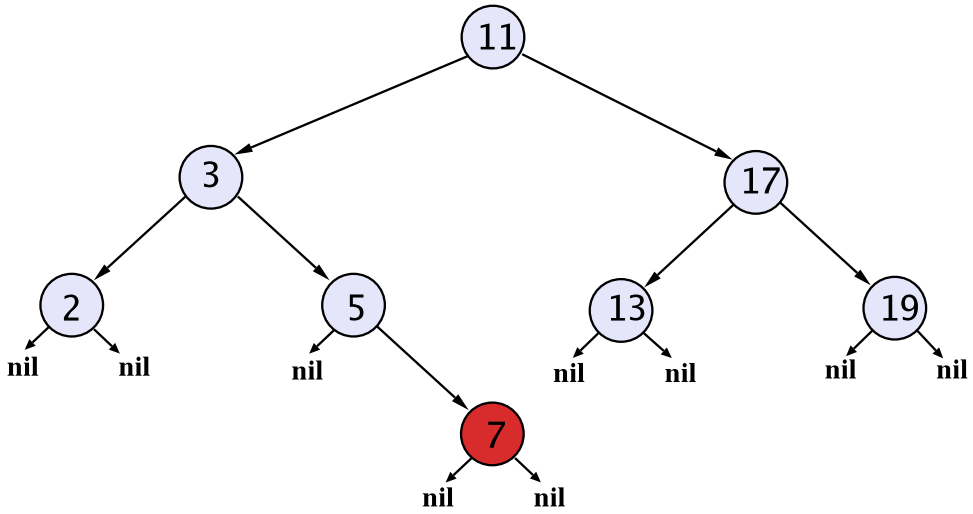

**(b)** Explain why this binary search tree cannot be colored to form a legal red-black tree.

**Solution:** We prove this by contradiction. Suppose that a valid coloring exists. In a red-black tree, all paths from a node to descendant leaves contain the same number of black nodes. The path $(17, 19, \text{NIL})$ can contain at most three black nodes. Therefore the path $(17, 11, 3, 5, 7, \text{NIL})$ must also contain at most three black nodes and at least three red nodes. By the red-black tree properties, the root 17 must be black and the NIL node must also be black. This means that there must be three red nodes in the path $(11, 3, 5, 7)$, but this would mean there are two consecutive red nodes, which violates the red-black tree properties. This is a contradiction, therefore the tree cannot be colored to form a legal red-black tree.

5 points for correct answer. 2-4 points for stating the red-black tree rules, but not proving why the tree does not satisfy the rules. 1 point for stating that the tree is unbalanced.

**(c)** The binary search tree can be transformed into a red-black tree by performing a single rotation. Draw the red-black tree that results, labeling each node with "red" or "black." Include the keys from part (a).

**Solution:** Rotate right around the root. There are several valid ways to color the resulting tree. Here is one possible answer (all nodes are colored black except for 7).




5 points for the correct answer. 2 points for the correct rotation but incorrect coloring.

**Problem 6. Wiggly arrays** [10 points]

An array $A[1\,..\,2n+1]$ is ***wiggly*** if $A[1] \leq A[2] \geq A[3] \leq A[4] \geq \ldots \leq A[2n] \geq A[2n+1]$. Given an unsorted array $B[1\,..\,2n+1]$ of real numbers, describe an efficient algorithm that outputs a permutation $A[1\,..\,2n+1]$ of $B$ such that $A$ is a wiggly array.

**Solution:** There are several ways to solve this problem in $\Theta(n)$ time. You can find the median $k$ of $B$ using the deterministic $\Theta(n)$ select algorithm and partition $B$ around the median $k$ into two equal sized sets $B_{\mathit{low}}$ and $B_{\mathit{high}}$. Assign $A[1] \leftarrow k$. Then for each $i > 1$, if $i$ is even, assign an element from $B_{\mathit{high}}$ to $A[i]$, otherwise assign an element from $B_{\mathit{low}}$ to $A[i]$. Since all elements in $B_{\mathit{high}}$ are greater than or equal to all elements in $B_{\mathit{low}}$ and the median is less than or equal to all elements in $B_{\mathit{high}}$, the array is wiggly. The overall running time is $\Theta(n)$.

10 points for correct $\Theta(n)$ solution and correct analysis. 8-9 points for correct solution but missing a minor step in the analysis.

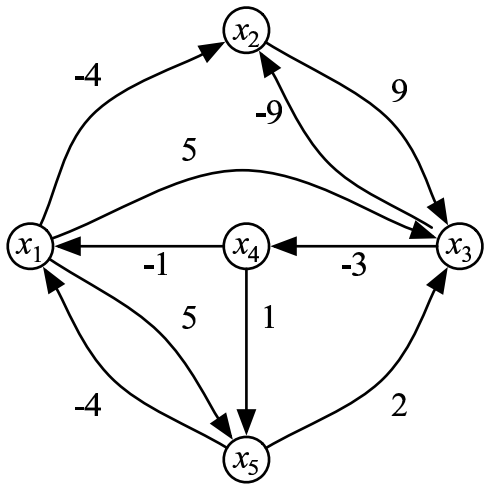
5 points for correct $\Theta(n \lg n)$ solution and correct analysis. 2-4 points for correct solution but incomplete analysis.

**Problem 7. Difference constraints** [12 points] (2 parts)

Consider the following linear-programming system of difference constraints (note that one constraint is an equality):

$$\begin{aligned}
x_1 - x_4 &\leq -1 \\
x_1 - x_5 &\leq -4 \\
x_2 - x_1 &\leq -4 \\
x_2 - x_3 &= -9 \\
x_3 - x_1 &\leq 5 \\
x_3 - x_5 &\leq 2 \\
x_4 - x_3 &\leq -3 \\
x_5 - x_1 &\leq 5 \\
x_5 - x_4 &\leq 1
\end{aligned}$$

**(a)** Draw the constraint graph for these constraints.




**Solution:** The equality constraint can be written as two inequality constraints, $x_2 - x_3 \leq -9$, and $x_3 - x_2 \leq 9$. A completely correct constraint graph received 6 points. Solutions that had edges in the wrong direction received only 4 points, and solutions that did not handle the equality constraint received 3 points.

**(b)** Solve for the unknowns $x_1$, $x_2$, $x_3$, $x_4$, and $x_5$, or explain why no solution exists.

**Solution:** No solution to this system exists because the constraint graph has a negative-weight cycle. For example, $x_1 \rightsquigarrow x_3 \rightsquigarrow x_4 \rightsquigarrow x_5 \rightsquigarrow x_1$ has weight $5-3+1-4 = -1$. A full-credit solution (6 points) must exhibit the negative weight cycle.

**Problem 8. Amortized increment** [12 points]

An array $A[0\,..\,k-1]$ of bits (each array element is 0 or 1) stores a binary number $x = \sum_{i=0}^{k-1} A[i] \cdot 2^i$.

To add 1 (modulo $2^k$) to $x$, we use the following procedure:

```
INCREMENT(A, k)
1  i ← 0
2  while i < k and A[i] = 1
3        do A[i] ← 0
4           i ← i + 1
5  if i < k
6     then A[i] ← 1
```

Given a number $x$, define the potential $\Phi(x)$ of $x$ to be the number of 1's in the binary representation of $x$. For example, $\Phi(19) = 3$, because $19 = 10011_2$. Use a potential-function argument to prove that the amortized cost of an increment is $O(1)$, where the initial value in the counter is $x = 0$.

**Solution:** $\Phi(x)$ is a valid potential function the number of 1's in the binary representation of $x$ is always nonnegative, i.e., $\Phi(x) \geq 0$ for all $x$. Since the initial value of the counter is 0, $\Phi_0 = 0$.

Let $c_k$ be the real cost of the operation INCREMENT$(A, k)$, and let $d_k$ be the number of times the while loop body in Lines 3 and 4 execute (i.e., $d_k$ is the number of consecutive 1's counting from the least-significant bit of the binary representation of $x$). If we assume that executing all of Lines 1, 2, 5, and 6 require unit cost, and executing the body of the while loop requires unit cost, then the real cost is $c_k = 1 + d_k$,

The potential decreases by one every time the while loop is executed. Therefore, the change in potential, $\Delta\Phi = \Phi_k - \Phi_{k-1}$ is at most $1 - d_k$. More specifically, $\Delta\Phi = 1 - d_k$ if we execute Line 6 , and $\Delta\Phi = -d_k$ if we do not.

Thus, using the formula for amortized cost, we get

$$\begin{aligned} \hat{c_k} &= c_k + \Delta\Phi \\ &= 1 + d_k + \Delta\Phi \\ &\leq 1 + d_k + 1 - d_k \\ &= 2. \end{aligned}$$

Therefore, the amortized cost of an increment is $O(1)$.

Only solutions that explicitly calculate/explain the real cost $c_k$ of a single increment received full credit. Solutions that give an aggregate analysis or any other method that did not use the potential function received at most 6 points. One or two points were deducted from correct solutions that did not explain why the potential function is valid or calculate $\Phi_0$.

**Problem 9. Minimum spanning trees** [12 points]

Let $G = (V, E)$ be a connected, undirected graph with edge-weight function $w : E \to \mathbb{R}$, and assume all edge weights are distinct. Consider a cycle $\langle v_1, v_2, \ldots, v_k, v_{k+1} \rangle$ in $G$, where $v_{k+1} = v_1$, and let $(v_i, v_{i+1})$ be the edge in the cycle with the largest edge weight. Prove that $(v_i, v_{i+1})$ does *not* belong to the minimum spanning tree $T$ of $G$.

**Solution:** Proof by contradiction. Assume for the sake of contradiction that $(v_i, v_{i+1})$ does belong to the minimum spanning tree $T$. Removing $(v_i, v_{i+1})$ from $T$ divides $T$ into two connected components $P$ and $Q$, where some nodes of the given cycle are in $P$ and some are in $Q$. For any cycle, at least two edges must cross this cut, and therefore there is some other edge $(v_j, v_{j+1})$ on the cycle, such that adding this edge connects $P$ and $Q$ again and creates another spanning tree $T'$. Since the weight of $(v_j, v_{j+1})$ is less than $(v_i, v_{i+1})$, the weight of $T'$ is less than $T$ and $T$ cannot be a minimum spanning tree. Contradiction.

**Problem 10. Multithreaded scheduling** [10 points]

A greedy scheduler runs a multithreaded computation in $260$ seconds on $4$ processors and in $90$ seconds on $32$ processors. Is it possible that the computation has work $T_1 = 1024$ and critical-path length $T_\infty = 64$? Justify your answer.

**Solution:** For greedy schedulers, we know that $\min\{T_1/P, T_\infty\} \leq T_P \leq T_1/P + T_\infty$. The numbers given above satisfy all of the above inequalities for both values of $P$. Therefore, it is possible that the work and critical path are correct.

**Problem 11. Transposing a matrix** [40 points] (4 parts)

Let $X$ be an $N \times N$ matrix, where $N$ is an exact power of 2. The following code computes $Y = X^{\mathsf{T}}$:

```
TRANS(X, Y, N)
1  for i ← 1 to N
2      do for j ← 1 to N
3             do Y[j, i] ← X[i, j]
```

Consider the cache-oblivious two-level memory model with a cache of $M$ elements and blocks of $B$ elements. Assume that both matrices $X$ and $Y$ are stored in row-major order, that is, the linear order of $X$ in memory is $X[1,1], X[1,2], \ldots, X[1,N], X[2,1], X[2,2], \ldots, X[2,N], \ldots, X[N,1], X[N,2], \ldots, X[N,N]$, and similarly for $Y$.

**(a)** Analyze the number $MT(N)$ of memory transfers incurred by TRANS when $N \gg M$.

**Solution:** Since the loop scans through matrix $X$ one row at a time, the number of memory transfers required for accessing $X$ is $O(N^2/B)$. We incur $O(N^2)$ memory transfers to access $Y$, however, because $Y$ is accessed one column at a time. When we access the block containing $Y[j,i]$, since $N \gg M$ and we scan through all of column $i$ before accessing $Y[j+1,i]$, each access to $Y$ may incur a memory transfer.
Therefore, $MT(N) = O(N^2)$.

Now, consider the following divide-and-conquer algorithm for computing the transpose:

```
R-TRANS(X, Y, N)
 1  if N = 1
 2     then Y[1,1] ← X[1,1]
 3     else Partition X into four (N/2) × (N/2) submatrices X11, X12, X21, and X22.
 4          Partition Y into four (N/2) × (N/2) submatrices Y11, Y12, Y21, and Y22.
 5          R-TRANS(X11, Y11, N/2)
 6          R-TRANS(X12, Y21, N/2)
 7          R-TRANS(X21, Y12, N/2)
 8          R-TRANS(X22, Y22, N/2)
```

Assume that the cost of partitioning is $O(1)$.

**(b)** Give and solve a recurrence for the number $MT(N)$ of memory transfers incurred by R-TRANS when $N \gg M$.

**Solution:** If we make either the tall-cache assumption (i.e., $M = \Omega(B^2)$), or we assume that the matrix is stored in the recursive block layout, then

$$MT(n) = \begin{cases} 4T(n/2) + \Theta(1) & \text{if } cn^2 > M, \\ M/B & \text{if } cn^2 \le M \end{cases} .$$

The recursion tree has approximately $\lg N - \lg(\sqrt{M/c})$ levels. Since every level has 4 times as many nodes as the previous level, the solution to the recurrence is dominated by the cost of the leaves. The tree has $4^{\lg(N\sqrt{c}/\sqrt{M})}$ leaves, each of cost $M/B$. Therefore, $MT(N)$ is

$$\begin{aligned} MT(N) &= \frac{M}{B}\left(\frac{N\sqrt{c}}{\sqrt{M}}\right)^2 \\ &= \frac{cN^2}{B} \\ &= O\left(\frac{N^2}{B}\right). \end{aligned}$$

The following multithreaded algorithm computes the transpose in parallel:

```
P-TRANS(X, Y, N)
 1  if N = 1
 2     then Y[1,1] ← X[1,1]
 3     else Partition X into four (N/2) × (N/2) submatrices X11, X12, X21, and X22.
 4          Partition Y into four (N/2) × (N/2) submatrices Y11, Y12, Y21, and Y22.
 5          spawn P-TRANS(X11, Y11, N/2)
 6          spawn P-TRANS(X12, Y21, N/2)
 7          spawn P-TRANS(X21, Y12, N/2)
 8          spawn P-TRANS(X22, Y22, N/2)
 9          sync
```

**(c)** Give and solve recurrences describing the work $T_1(N)$ and critical-path length $T_\infty(N)$ of P-TRANS. What is the asymptotic parallelism of the algorithm?

**Solution:**

$$\begin{aligned} T_1(N) &= 4T_1\left(\frac{N}{2}\right) + O(1) \\ &= \Theta(N^2). \\ T_\infty(N) &= T_\infty\left(\frac{N}{2}\right) + O(1) \\ &= \Theta(\lg N). \end{aligned}$$

The parallelism of the algoritm is $T_1/T_\infty$, or $\Theta(N^2/\lg N)$.

Professor Kellogg inadvertantly places two additional **sync** statements into his code as follows:

```
K-TRANS(X, Y, N)
 1  if N = 1
 2     then Y[1,1] ← X[1,1]
 3     else Partition X into four (N/2) × (N/2) submatrices X11, X12, X21, and X22.
 4          Partition Y into four (N/2) × (N/2) submatrices Y11, Y12, Y21, and Y22.
 5          spawn K-TRANS(X11, Y11, N/2)
 6          sync
 7          spawn K-TRANS(X12, Y21, N/2)
 8          sync
 9          spawn K-TRANS(X21, Y12, N/2)
10          spawn K-TRANS(X22, Y22, N/2)
11          sync
```

**(d)** Give and solve recurrences describing the work $T_1(N)$ and critical-path length $T_\infty(N)$ of K-TRANS. What is the asymptotic parallelism of this algorithm?

**Solution:** The work for the algorithm remains the same.

$$\begin{aligned} T_1(N) &= 4T_1\left(\frac{N}{2}\right) + O(1) \\ &= \Theta(N^2). \end{aligned}$$

The critical path increases, however, because we only solve one of the subproblems in parallel.

$$\begin{aligned} T_\infty(N) &= 3T_\infty\left(\frac{N}{2}\right) + O(1) \\ &= \Theta(N^{\lg 3}). \end{aligned}$$

The parallelism of the algorithm is $T_1/T_\infty$, or $\Theta(N^{2-\lg 3})$.

SCRATCH PAPER — Please detach this page before handing in your exam.

SCRATCH PAPER — Please detach this page before handing in your exam.