1. In our linear-time selection algorithm, the inputs are divided into groups of 5. What if you used groups of 3 instead? What if used groups of 7 or larger (odd integers)?

**Solution:** It does not work for groups of 3 because we get the following recursion:

\[
T(n) \leq T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3} + 4\right) + O(n).
\]

Indeed, if we try similarly to prove \( T(n) \leq cn \) with substitution, it doesn’t work for any constant \( c \):

\[
T(n) \leq \frac{cn}{3} + \frac{2cn}{3} + 4c + an = cn + 4c + an \not\leq cn
\]

However, it works for groups of 7 or larger with a slightly worse constant. Say we use groups of size \( 2k + 1 \) with an arbitrary integer \( k \geq 3 \). Then we get the following recursion which has a linear solution for each fixed \( k \):

\[
T(n) \leq T\left(\frac{n}{2k + 1}\right) + T\left(\frac{(3k + 1)n}{2(2k + 1)} + 2(k + 1)\right) + O(n).
\]

Indeed, as for groups of 5 we can prove \( T(n) \leq cn \) by substitution if \( c \) is large enough compared to \( k \). We get

\[
T(n) \leq cn - \frac{c(k - 1)n}{2(2k + 1)} + 2c(k + 1) + an \leq cn,
\]

if \( c \) is large enough compared to \( k \) and \( a \). (15 points)
2. Not every greedy strategy works. Show that if in the activity-selection problem we always select the activity of least duration (another greedy strategy) from among those that are compatible with the previously selected activities we might not get an optimal solution.

Solution: Assume we have the following three activities: \( a_1 \) with \( s_1 = 0, f_1 = 3 \), \( a_2 \) with \( s_2 = 2, f_2 = 4 \) and \( a_3 \) with \( s_3 = 3, f_3 = 6 \). The suggested greedy approach selects just \( \{a_2\} \), but the optimal solution selects \( \{a_1, a_3\} \). (15 points)

3. Give an \( O(n) \)-time dynamic programming algorithm for the maximum subarray problem. That is, given an array of real numbers \( a_1, a_2, \ldots, a_n \), your algorithm must compute the maximum sum \( \sum_{i=j}^{k} a_i \), where \( 1 \leq j \leq k \leq n \). Draw the subproblem graph. How many vertices and edges are in the graph?

Solution: Here is the algorithm:

Maximum-subarray(n)

let \( M[1..n] \) be a new array

\( M[1] = a_1 \)

for \( i = 2 \) to \( n \)

\( M[i] = \max(M[i-1] + a_i, a_i) \)

we find the maximum \( M \) of \( M[i], 1 \leq i \leq n \)

return \( M \)

Each number in the sequence is computed from the previous number in the sequence. The running time is clearly \( O(n) \). The subproblem graph consists of \( n \) vertices, \( v_1, \ldots, v_n \). For \( i = 2, \ldots, n \), vertex \( v_i \) has one leaving edge: to vertex \( v_{i-1} \). No edge leaves vertex \( v_1 \). Thus, the subproblem graph has \( (n - 1) \) edges. (15 points)

4. Suppose we perform a sequence of \( n \) operations on a data structure in which the \( i \)-th operation costs \( i \) if \( i \) is an exact power of 2, and 1 otherwise. Use the accounting method to determine the amortized cost per operation and use this to get an upper bound on the actual cost of the sequence of \( n \) operations.
Solution: Let \( c_i = \) actual cost of the \( i \)-th operation, i.e.

\[
c_i = \begin{cases} 
  i & \text{if } i \text{ is an exact power of 2,} \\
  1 & \text{otherwise.}
\end{cases}
\]

Then let \( \overline{c_i} = \) amortized cost of the \( i \)-th operation = 3 for every \( 1 \leq i \leq n \). Thus we have \( \sum_{i=1}^{n} \overline{c_i} = 3n \). Furthermore, we have

\[
\sum_{i=1}^{n} c_i \leq n + \sum_{j=0}^{\lfloor \log_2 n \rfloor} 2^j \leq n + (2n - 1) < 3n,
\]

thus indeed \( \sum_{i=1}^{n} c_i \leq \sum_{i=1}^{n} \overline{c_i} = 3n \). (20 points)

5. Consider the same set-up as in the previous problem but now use the potential method to determine the amortized cost per operation and use this to get an upper bound on the actual cost of the sequence of \( n \) operations.

Solution: We define the potential function \( \Phi \) as twice the distance of \( i \) from the largest power of 2 that is at most \( i \), i.e.

\[
\Phi(D_i) = 2 \left( i - 2^{\lfloor \log_2 i \rfloor} \right).
\]

Then we have \( \Phi(D_i) \geq 0 = \Phi(D_0) \), and thus \( \sum_{i=1}^{n} c_i \leq \sum_{i=1}^{n} \overline{c_i} \). We just have to estimate \( \overline{c_i} \). If \( i \neq 2^k \), then

\[
\overline{c_i} = c_i + \Phi(D_i) - \Phi(D_{i-1}) = 1 + 2 = 3.
\]

If \( i = 2^k \), then

\[
\overline{c_i} = c_i + \Phi(D_i) - \Phi(D_{i-1}) = i + 0 - 2(2^k - 1 - 2^{k-1}) = 2^k - 2^k + 2 \leq 3.
\]

Thus \( \sum_{i=1}^{n} c_i \leq \sum_{i=1}^{n} \overline{c_i} \leq 3n \). (20 points)

6. Describe an algorithm for finding a spanning tree with minimal weight containing a specified acyclic (cycle-free) set of edges \( A \) in a connected weighted undirected graph.

Solution: We just run Kruskal’s algorithm, but instead of starting with an empty set of edges, we start with the edge set \( A \) given. (15 points)