# Homework 5 – Due Thursday, October 6, 2016 on Canvas

Please refer to HW guidelines from HW1, course syllabus, and collaboration policy.

**Exercises** These should not be handed in, but the material they cover may appear on exams:

1. Using the PARTITION subroutine and the linear time algorithm for finding the median of list of numbers, design a divide-and-conquer sorting algorithm that runs in $O(n \log n)$ time.
2. Show how to multiply the complex numbers $a + bi$ and $c + di$ using only three real multiplications. The algorithm should take $a, b, c$, and $d$ as input and produce the real component $ac - bd$ and the imaginary component $ad + bc$ separately.
3. What is the largest $k$ such that if you can multiply $3 \times 3$ matrices using $k$ multiplications, then you can multiply $n \times n$ matrices in time $o(n^{\log 7})$, i.e., asymptotically faster than in time $\Theta(n^{\log 7})$? What would the running time of this algorithm be?
4. There is a way of multiplying $68 \times 68$ matrices using $132,464$ multiplications, a way of multiplying $70 \times 70$ matrices using $143,640$ multiplications, and a way of multiplying $72 \times 72$ matrices using $155,424$ multiplications. Which method yields the best asymptotic running time when used in a divide-and-conquer matrix-multiplication algorithm? How does it compare to Strassen's algorithm?
5. Determine an LCS of 1,0,0,1,0,1,0,1 and 0,1,0,1,1,0,1,1,0
6. Write pseudocode for the procedure that can reconstruct an LCS from the table of optimal lengths, discussed in class, in time $O(m + n)$.
7. Give a memoized version of the LCS algorithm discussed in class.
8. Explain how to modify the LCS algorithm discussed in class, so that it computes the length of a LCS (but not the subsequence itself) using $O(\min(m, n))$ space. The running time should still be $O(mn)$.
9. Chapter 6, problem 1.

**General hint for designing divide-and-conquer algorithms:** When the target running time is given, think which recurrence would yield this running time. Try to divide your input accordingly and see if you can combine the results of the recursive calls in the required time.

**Problems to be handed in** (Don't forget to prove correctness and analyze time/space requirements of your algorithm.)

1. (**Median** 10 points, 2-page limit)
   (a) The *majority element* of an array $A[1..n]$, containing $n$ integers, is the element that appears more than $n/2$ times in the array. Design an efficient algorithm that, given an integer array, outputs its majority element if it exists. Your algorithm should use the Order Statistics algorithm we covered in class as a subroutine. Briefly describe your algorithm (at most 3 sentences), and state its running time. You do not have to write down the proof of correctness.

(b) Chapter 5, problem 1. Give a divide-and-conquer algorithm for this problem. (Don't forget that a recursive algorithm needs an initial call.) Briefly argue correctness: assuming that the results of recursive calls are correct, your algorithm should perform the "combine" step correctly. (Do not forget to take care of the base case.) Don't forget to analyze the running time and the space complexity of *your* algorithm.

2. (**Divide and conquer**, 20 points, 3-page limit)

A *linear k-spanner* is a directed graph with vertex set $V = \{1, \ldots, n\}$ that contains a directed path of length at most $k$ from vertex $i$ to vertex $j$ for all $1 \leq i < j \leq n$. Moreover, a linear $k$-spanner contains no edges that go "backwards", that is, all edges $(i, j) \in E$ satisfy $i < j$. Intuitively, this graph is a directed path $(1, 2), (2, 3), \ldots, (n-1, n)$ augmented with additional edges to ensure that each vertex $j$ is reachable from each vertex $i$ by traversing at most $k$ edges in the graph.

(-) (**Do not hand this part in**) Using $\Theta$-notation, how many edges are there in a linear 1-spanner on $n$ vertices?

(a) (**10 points**) Design an efficient algorithm that, given $n$, constructs a linear 2-spanner on $n$ vertices with $O(n \log n)$ edges. Prove correctness (i.e., that you indeed output a linear 2-spanner and that it has sufficiently few edges).

(b) (**10 points**) Consider the following algorithm for computing the edge set $E$ of a linear 3-spanner.

```
On input $n$,
  • If $n \leq 3$, output $E = \{(i, j) \mid 1 \leq i < j \leq n\}$.
  • Else
      – designate vertices $\lfloor\sqrt{n}\rfloor, \lfloor 2\sqrt{n}\rfloor, \ldots, \lfloor(\sqrt{n}-1)\sqrt{n}\rfloor$ as **hubs**.
      – For each pair of hubs $h_1 < h_2$, add an edge $(h_1, h_2)$ to $E$.
      – For each vertex $i \in \{1, ..., n\}$, let $\ell(i)$ be the hub immediately before $i$ and $r(i)$ be
        the immediately after $i$. Add $(\ell(i), i)$ and $(i, r(i))$ to $E$.
      – For each $k$ from 1 to $\sqrt{n}$, recursively construct a linear 3-spanner for segment of
        the graph between hubs $k-1$ and $k$ (that is, between nodes $(k-1)\sqrt{n}$ and $k\sqrt{n}$)
        and add the resulting edges to $E$.
  • Return $E$.
```

i. Prove by induction that the above algorithm correctly constructs a linear 3-spanner.

ii. Let $S(n)$ be the number of edges in $E$ output by the algorithm. Give a recurrence for $S(n)$. You may assume, for simplicity, that $\sqrt{n}$ is always an integer.

iii. Find the size of a linear 3-spanner returned by the algorithm by solving your recurrence.

3* (**Optional, no collaboration, 2-page limit**) Explain how to construct a linear 4-spanner on $n$ nodes (defined in the previous problem) with as few edges as you can. Only hand in your solution if the number of edges in your linear 4-spanner is asymptotically smaller than in the linear 3-spanner returned by the algorithm above. (Don't forget to prove correctness and analyze the number of edges).

You might need the following definition. The *iterated logarithm* of $n$, written $\log^* n$ (pronounced "log star"), is the number of times the logarithm function must be iteratively applied before the result is less than or equal to 1. Formally,

$$\log^* n = \begin{cases} 0 & \text{if } n \leq 1; \\ 1 + \log^*(\log n) & \text{if } n > 1. \end{cases}$$