COMP 3711H – Fall 2016
Tutorial 4

Note. The answers given below might not be complete. Some are sketches or hints, sometimes missing details.

1. **Using Black-box median algorithms (modified from CLRS)**
   For this problem, you assume that you are given a black-box ($O(n)$ time algorithm for finding the median ($\lceil n/2 \rceil$nd) item in a size n array. This means that you can call the algorithm and use its result but can't peer inside of it.

   (a) Show how *Quicksort* can be modified to run in $O(n \log n)$ *worst case* time.
   Before every partition step in Quicksort use the $O(n)$ black box algorithm to find the median of the current subarray. Then use that median as the pivot. This splits the subarray into two (almost) equal parts so the running time of the full algorithm satisfies

   $$T(n) = 2T(n/2) + O(n)$$

   implying $T(n) = O(n \log n)$.

   (b) Give a simple linear-time algorithm that solves the selection problem for an arbitrary order statistic. That is, given k, your algorithm should find the k smallest item.
   Just modify the randomized selection algorithm so that,at every step, instead of choosing the pivot at random from the current subarray. it uses the $O(n)$ median finding algorithm to find the median and then uses the median as pivot. At each step, the algorithm will reduce the size of the array in which it is searching by $1/2$ so the running time satsifies

   $$T(n) \leq T(n/2) + O(n)$$

   which yields $T(n) = O(n)$.

   (c) For n distinct elements $x_1, x_2, \ldots, x_n$ with associated positive weights $w_1, w_2, \ldots, w_n$ such that $\sum_{i=1}^{n} w_i = 1$, the **weighted (lower) medium** is the element x_k satisfying

   $$\sum_{x_i < x_k} w_i < \frac{1}{2} \quad \text{and} \quad \sum_{x_i > x_k} w_i \leq \frac{1}{2}.$$

   If the x_i are sorted, then it is easy to solve this problem in $O(n)$ time by just summing up the weights from left to right and walking through the sums until k is found. Show that if the items are *not* sorted you can still solve the problem in linear time using the black box median finding algorithm.
   If there are 2 items we can solve the problem in $O(1)$ time.
   Otherwise, in $O(n)$ time find the median x_m of the x_i. In another $O(n)$ time calculate $W_L = \sum_{x_i \leq x_m} w_i$. Note that $W_R = \sum_{x_i < x_m} w_i = 1 - W_L$.
   If $W_R \leq 1/2$ then we know that $k \leq m$. Now add $1 - W_L$ to x_m. Note that, for the new weights, $\sum_{i=1}^{m} x_i = 1$ and the weighted median of $x_1, \ldots, x_m$ is the same as the weighted median of the original set so recurse to find the weighted median of $x_1, \ldots, x_m$.

If $W_R > 1/2$ then we know that $k > m$. Add W_L to x_m and note that the weighted median of $x_m, \ldots, x_n$ is the same as the weighted median of the original set so recurse to find the weighted median of $x_m, \ldots, x_n$.
Each recursion splits the problem by 1/2 so

$$T(n) = T(n/2) + O(n)$$

and $T(n) = O(n)$.

2. **Polynomial Evaluation** The input to this problem is a set of $n+1$ coefficients $a_0, a_1, \ldots, a_n$. Define $A(x) = \sum_{i=0}^{n} a_i x_i$

   (a) Given value x, how can you evaluate $A(x)$ using $O(n)$ multiplications and $O(n)$ additions?
   Can you evaluate $A(x)$ using at most n multiplications and n additions?
   Note that

$$\sum_{i=0}^{n} a_i x_i = a_0 + x(a_1 + x(a_2 + x(\cdots(a_{n-2} + x(a_{n-1} + xa_n)\cdots).$$

   This will permit evaluating the polnomial using n additions and n multiplications. This method is known as Horner's rule.

   (b) Now suppose that $A(x)$ has at most k non-zero terms. How can you evaluate $A(x)$ using $O(k \log n)$ operations.
   Hint. How can you evaluate x^n using $O(\log n)$ operations.

   In $\log n$ time precompute and store the values

$$x, x^2, x^4 = (x^2)^2, x^8 = (x^4)^2, \ldots, x^{2^j} = \left(x^{2^{j-1}}\right)^2$$

   where $j = \lfloor \log n \rfloor$ Note that if $m < n$ then we can write m in binary as $m = \sum_{i=0}^{j} \epsilon_1 2^i$ where $\epsilon_i \in \{0, 1\}$ and the ϵ_i can be calculated in $O(j) = O(\log n)$ time. Thus

$$x^m = \prod_{i=0}^{j} \epsilon_i x^{2^i}$$

   can be calculated in $O(\log n)$ time using the precomputed squares. This can then be used to evaluate the polynomial in total time $O(k \log n)$ (including the precomputation time).

3. **Interpolating Polynomials** The values $A(x_0), A(x_1), \ldots, A(x_n)$, define a unique degree n polynomial having those values. In class, we saw the Langragian interpolation formula for finding the coefficients $a_0, a_1, \ldots, a_n$ of $A(x)$. This worked by first setting

$$I_i(x) = \prod_{0 \le j \le n,\, j \ne i} \frac{x - x_j}{x_i - x_j}$$

   and then defining

$$A(x) = \sum_i A(x_i) I_i(x).$$

Show how to use the formula to evaluate the coefficients of $A(x)$ in $O(n^2)$ time.

Hint How long does it take to divide a degree n polynomial by a degree one polynomial? You can use this procedure as a subroutine.

First calculate the values of the coefficients of $P(x) = \prod_{0 \le j \le n} x_- x_j$. *This can be done by iterativel evaluating*

$$P_i(x) = \prod_{0 \le j \le i} x_- x_j = (x - x_i) P_{i-1}(x).$$

Since $P_{i-1}(x)$ *is a degree i poynomial and a degree 1 polynomial can be multiplied by a degree i polynomial in* $O(i)$ *time, finding* $P(x) = P_n((x)$ *takes* $O(\sum_i i) = O(n^2)$ *time.*

Next note that

$$\prod_{0 \le j \le n,\, j \ne i} x - x_j = \frac{P(x)}{x - x_i}$$

Division of a degree $n+1$ *polynomial by a degree one polynomai can be done (basic long division) in* $O(n)$ *time so, knowing* $P(x)$, *we can calculate* $\prod_{0 \le j \le n,\, j \ne i} x - x_j = \frac{P(x)}{x - x_i}$ *in* $O(n)$ *time for each i. Doing this for all i then needs* $O(n^2)$ *time.*

Note that, for each i calculating $\prod_{0 \le j \le n,\, j \ne i} = \frac{P(x)}{x_j - x_i}$ *takes* $O(n)$ *time (simple multiplcation) so doing this for each i takes* $O(n^2)$ *time.*

Also, for fixed i, calculating the scalar value $\prod_{0 \le j \le n,\, j \ne i} x_i - x_j$ *takes* $O(n)$ *time. Do this foir all i, using* $O(n^2)$ *time in total.*

Finally, recall that

$$I_i(x) = \prod_{0 \le j \le n,\, j \ne i} \frac{x - x_j}{x_i - x_j} = \frac{\prod_{0 \le j \le n,\, j \ne i} x - x_i}{\prod_{0 \le j \le n,\, j \ne i} x_j - x_i}.$$

Since we have already created the (polynomial) numerator and (scalar) denominator of these terms we can calculate each $I_i(x)$ *in* $O(n)$ *time and all of them in* $O(n^2)$ *time.*

We can then find the coefficients of each $A(x_i)I_i(x)$ *in* $O(n)$ *time or* $O(n^2)$ *in total. Adding the coeeficients together to get the coefficients of the full solution would require a further* $O(n^2)$ *time.*

Combining all of the pieces gives an $O(n^2)$ *algorithm.*

4. **More Median of Medians** For this problem you can assume the following fact: $\alpha, \beta \ge 0$, N is a non-negative integer and c, D constants (possibly negative). For $n > N$, if

$$T(n) \le T(\alpha n + c) + T(\beta n + d) + \Theta(n)$$

then

$$T(n) = \begin{cases} O(n) & \text{if } \alpha + \beta < 1 \\ O(n \log n) & \text{if } \alpha + \beta = 1 \\ \Omega(n \log n) & \text{if } \alpha + \beta > 1 \end{cases}.$$

Recall that our determinsitic selection algorithm yielded the recurrence

$$T(n) = T(n/5) + T(7n/10 + 6) + \gamma n$$

for some constant γ. The formula above implies $T(n) = O(n)$.

Our algorithm (i) splits the items into sets of 5 elements, (ii) found the median of each set and then (iii) found x, the median of those medians. It then ran *partition* with x as a pivot and recursed on the appropriate subset. From the definition of x we were able to prove that the subarrays created by partition both had at most $7n/10 + 6$ elements, leading to the recurrence relation and hence $O(n)$ running time.

Now suppose that instead of splitting the items into sets of size 5, we split them into sets of size 3 and then ran the algorithm the same way. Would we still get an $O(n)$ time algorithm?

What about if we split into sets of size 7?

If we split into groups of 3 we would have to find the median of $n/3$ items.

The same type of argument that we developed in class would show that after pivoting on that median, recursion throws away at least $\frac{1}{2} \times \frac{2}{3}n - c = \frac{1}{3}n - c$ items (c a small constant) and could retain $\frac{2}{3}n + c$ items.

So the recurrence would be

$$T(n) = T(n/3) + T(2n/3 + c) + O(n)$$

which gives $O(n \log n)$ and not $O(n)$.

If we split into groups of size 7 then we would have to first find the median of $n/7$ items. After using that as the pivot we would throw away $\frac{1}{2} \times \frac{4}{7}n - c' = \frac{2}{7}n - c'$ items; the recursion would retain at most $\frac{5}{7}n + c$ items. So the recurrence would be

$$T(n) = T(n/7) + T(5n/7 + c') + O(n)$$

which only gives $O(n)$.

5. **Extra Problem: Finding defective coins**
You have just been hired as the quality-control engineer for a company that makes coins. The coins must all have identical weight. You are given a set of n coins and are told that *at most one* (possibly none) of the n coins is lighter than the others. Your task is to develop an efficient test procedure to determine which of the n coins is defective or report that none is defective. To do this test you have a scale. For each measurement you place some of the coins on the left side of the scale and some of the coins on the right side. The scale indicates either that (1) the left side is heavier, (2) the right side is heavier or (3) both subsets have the same weight. It does not indicate how much heavier or lighter.

   (A) Design an algorithm for solving this problem that works in $\log_2 n + c$ time, for some constant c (try to make c as small as possible).
   Hint: try a divide and conquer approach.

   (B) Design an algorithm for solving this problem that works in $\log_3 n + c$ time, for some constant c (try to make c as small as possible).

   (C) The problem now changes in that the defective coin, if it exists, may be lighter or *heavier* than the other coins. Modify the algorithm from part (A) or (B) to work in this case. How fast is your algorithm?