CS 580 HW 2

Ankur Dhoot

January 28, 2017

Q1

We'll give an algorithm that runs in O(nlogr) time.

Let S be the set of n numbers. Let K = k_1, k_2, ... k_r where 1 $\leq k_1 <$ $k_2 <$ ... $< k_r \leq$ n. Let A[1...r] be an array where A[1] will store the $k_1 st$ smallest number ... A[r] will store the $k_r th$ smallest number.

We begin by finding the median element, x, of S and partitioning on x. This can be done in O(n) time as seen in CLRS 9.3. We denote the rank of the median as m. We then form the sets K1 and K2, where K1 contains the k_i with $k_i <$ m, and K2 contains the k_i with $k_i >$ m, except we subtract m from these k_i since we'll be searching for the corresponding rank only in the right subarray of S.

We then recursively call SELECT(S, lo, m-1, K1) to search in the left subarray for the elements whose ranks are in K1. Then, we call SELECT(S, m+1, hi, K2) to search in the right subarray for the elements whose ranks are in K2. In between, we check if the median element, x (with rank m), corresponded to a rank in K. If so, we append S[m] to A.

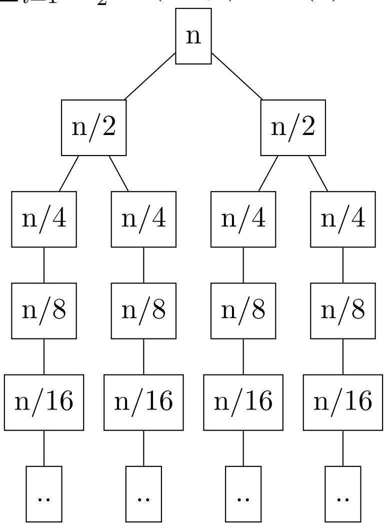
Note that the order of these calls insures that A has the property of being sorted with A[i] storing the $k_i th$ ranked element in S.

Runtime: If we think about the recursion tree, the maximum number of nodes in a level is r which will occur at a depth of $\lceil logr \rceil$. Reasoning: Take for example, r = 4. The maximum number of subproblems we can solve is r. Suppose, at some level, we are solving more than r subproblems. We only recurse on a subproblem if there remain ranks to be found in that subarray (i.e size(K1) $\neq$ 0 or size(K2) $\neq$ 0). Since K only has 4 ranks to be found, we can't be solving more than 4 subproblems at any level of the tree. Since we recurse on at most 2 subarrays, the maximum number of nodes will occur at $\lceil logr \rceil$ depth. Thereafter, there will be a maximum of 4 nodes in a level, and we continually decrease each subproblem size by 2. The recursion tree for r = 4, in the worst case would look like:

Algorithm 1 Select the k1, k2, .. kr(th) smallest elements in S

```
Let A[1...r] be a global array to hold the elements
function SELECT(S, lo, hi, K)
    if lo == hi then
        Append S[lo] to A
    end if
    x = FIND-MEDIAN(S, lo, hi)        ▷ returns the index of the median
    exchange S[hi] with S[x]           ▷ put median as partition element
    PARTITION(S, lo, hi)               ▷ same method as in CLRS 7.1
    m = ⌈(hi−lo+1)/2⌉                  ▷ rank of median element in A[lo...hi]
    K1 = {k | k ∈ K & k < m}
    K2 = {(k - m) | k ∈ K & k > m}
    if size(K1) ≠ 0 then
        SELECT(S, lo, m-1, K1)
    end if
    if m ∈ K then
        Append S[m] to A
    end if
    if size(K2) ≠ 0 then
        SELECT(S, m+1, hi, K2)
    end if
end function
```

Since the first $\lceil logr \rceil$ levels of the recursion tree each cost O(n) time independent of the subproblems, T(n) $\leq$ O(nlogr) + $\sum_{i=1}^{logn/r} \frac{n}{2^i r} r =$ O(nlogr) + $\sum_{i=1}^{logn/r} \frac{n}{2^i}$ O(nlogr) + O(n) = O(nlogr).



Q2

The algorithm for computing the minimum tri-distance amongst three points in P will be very similar to the algorithm given in CLRS 33.4, so we'll use the same notation. We presort X and Y as in CLRS 33.4 to avoid sorting the arrays in every subproblem.

We partition the points into two sets P_L and P_R using the median x value. The three points with minimum tri-distance then all reside in P_L, all reside in P_R, or cross the partition. We recursively compute the tri-distance, δ_1, among the points in P_L and the tri-distance, δ_2 among the points in P_R. We let $\delta = \min(\delta_1, \delta_2)$.

We then create the array Y', which contains all points in the 2δ wide vertical strip centered on the median x value, same as in CLRS 33.4. We now compute the minimum tri-distance in the case that the points cross the partition.

If the three points, (p_i, p_j, p_k), that comprise the minimum tri-distance cross the partition, it's clear that they must reside in a δ high rectangle in Y' (since $\mathrm{trd}(p_i, p_j, p_k) < \delta$). The key insight is that for each point p_i in Y', we need only check a constant number of (p_j, p_k) inside Y' to find the minimum tri-distance.

If we divide a 2δ wide by δ high rectangle in Y' into 32 subrectangles of size $\delta/4$ x $\delta/4$, there can be at most 2 points in any subrectangle. Why? If we place three points as far as possible in any subrectangle (as shown below), then the tri-distance is $\delta/4 + \delta/4 + \sqrt{(\delta/4)^2 + (\delta/4)^2} = \delta(1/2 + \sqrt{(2)}/4) < \delta$, a contradiction to how we defined δ. Thus, there can be at most 2 points in any subrectangle. Thus, there can be at most 64 points in any 2δ x δ rectangle.

Without loss of generality, suppose that p_i is the lowest point of (p_i, p_j, p_k) (i.e p_i precedes p_j and p_k in Y'). Then p_j and p_k must be among the 63 points following p_i. Thus, we need only check all $\binom{63}{2}$ possible pairs of points among the 63 points following p_i.

Since $\binom{63}{2}$ is a constant, the recurrence relation is the same as in CLRS 33.4. That is, assuming we presort X and Y in O(nlogn) time, the recurrence is then T(n) = 2T(n/2) + O(n). At each step, we solve 2 subproblems of size n/2. Forming P_L, P_R, X_L, X_R, Y_L, Y_R, and Y' takes linear time as described in CLRS 33.4. For each point in Y', of which there are at most n, we need only do constant work to check $\binom{63}{2}$ tri-distances. Thus lines 10-12 run in O(n). Thus the recurrence gives us T(n) = O(nlogn).

Algorithm 2 Find the three points whose pairwise distance sum is minimum among all sets of three points in P; X and Y are sorted by x and y coordinate, respectively

function TRIDISTANCE(P, X, Y)
  if |P| < 6 **then**
    Brute force all $\binom{P}{3}$ possibilities and **return** the min
  end if
  Form P_L, P_R, X_L, X_R, Y_L, Y_R as in CLRS 33.4
  $\delta_1 =$ TRIDISTANCE(P_L, X_L, Y_L)
  $\delta_2 =$ TRIDISTANCE(P_R, X_R, Y_R)
  $\delta = \min(\delta_1, \delta_2)$
  Form array Y', the 2δ width strip as in CLRS 33.4
  For each point p_i in Y', consider the next 63 points in Y'
  For each (p_j, p_k) in the $\binom{63}{2}$ pairs possible, compute trd(p_i, p_j, p_k)
  Save the minimum as δ'
  return $\min(\delta, \delta')$
end function