Problem sheet 4

Saturday, 23 August 2025 8:00 PM

Q1) Determining if an object is in the majority.

Claim If O_i is in the majority among a set A of n objects then for any way of partitioning A into A_1, A_2 s.t. $|A_1| = |A_2| = n/2$, O_i is in the majority in at least one of A_1, A_2.

Proof: If O_i is not in the majority in either A_1, A_2 then # of occurences of $O_i < \frac{|A_1|}{2} + \frac{|A_2|}{2} = \frac{|A|}{2}$ which implies O_i is not in the majority in A.

Algorithm: 1) Divide array A into 2 equal halves A_1, A_2
2) Recursively find the object in the majority in A_1, A_2 — let these be O_1, O_2. Note O_1, O_2 might be null
3) By comparing O_1, O_2 with each object in A determine their number of occurences & thus determine which of these is in the majority
4) Return majority object & null if there is none.

Analysis: Let $T(n)$ be the number of probes required to determine a majority object in a set of n objects. Then

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

which is the mergesort recurrence & evaluates to $T(n) = O(n \log n)$

Q2) We first determine n by a doubling procedure

$i = 1$
while $A[i] \neq \infty$ do $i = 2i$

If there are n numbers which are finite then the above procedure stops with $i = m$ where m is the smallest integer larger than n which is a power of 2. Note $m < 2n$.

Now we do a binary search on $A[1..m]$ to find x. This takes time $O(\log_2 m) = O(\log_2 n)$

Q.3) Renumber lines so that $i < j \Rightarrow a_i < a_j$. To determine the upper envelope (subset of lines visible from $y = \infty$) of the set $L_1 \ldots L_n$ we determine the upper envelope of sets

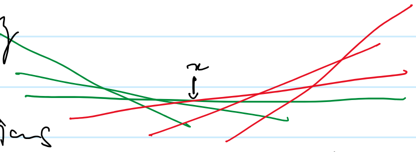
envelope (subset of lines visible from $y=\infty$)

of the set $L_1 \ldots L_n$ we determine the upper envelope of sets $L_1 \ldots L_{n/2}$ & $L_{n/2+1} \ldots L_n$ recursively & then combine these.

Let $i_1, i_2 \ldots i_k$ (resp $j_1, j_2 \ldots j_l$) be indices of the lines in the upper envelope of $L_1 \ldots L_{n/2}$ (resp $L_{n/2+1}, \ldots L_n$) say f_l, f_r

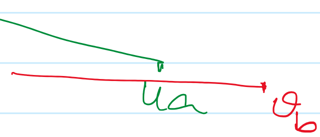
The upper envelopes are convex functions & intersect at a unique point, say x. The combine step involves determining i_r, j_s where intersection x is the intersection of these functions. The upper envelope of $L_1 \ldots L_n$ is then $i_1, i_2 \ldots i_r, j_s, j_{s+1} \ldots j_l$

Let u_i be the i^{th} point of discontinuity in f_l & v_i be the i^{th} point of discontinuity in f_r (both these functions are piecewise linear & we order the points of discontinuity in increasing order).



To determine x we consider lists $i_1, i_2 \ldots i_r$ & $j_1, j_2 \ldots j_l$ & perform a merge like operation. Suppose i_a, j_b are the lines under consideration; initially $a=b=1$

Let x be the intersection of these lines.



If $x < u_a$ & $x < v_b$ then we stop & return x.

If $u_a < x < v_b$ then $a{+}{+}$

If $v_b < x < u_a$ then $b{+}{+}$

If $x > u_a$ & $x > v_b$ then $a{+}{+}$, $b{+}{+}$

Analysis: The combine step takes linear time. Hence we get a recurrence $T(n) = 2T(n/2) + n$ which evaluates to $O(n \log n)$

Q4) We start with the root & if its value is less than both its children we have found a local minima. Else we move to the smaller child. At any node v, if both children are larger than v then v is the local minima (since parent of v was larger than v). Else we move to the smaller child. The process always terminates because when we reach a leaf we STOP.

Q6) Let A be an array such that $A[i] = a_i$. Define a procedure $X(i,j)$ which returns

1) $i \le l \le j$ where $l = \arg\max_r \sum_{k=i}^{r} A[k]$ & $L = \max_r \sum_{k=i}^{r} A[k]$

2) $i \le r \le j$ where $r = \arg\max$ [illegible]

2) $i \le r \le j$ where $r = \arg\max_l \sum_{k=l}^{j} A[k]$ & $R = \max_l \sum_{k=l}^{j} A[k]$

3) $i \le a \le b \le j$ where a, b are such that

$$S = \sum_{k=a}^{b} A[k] \text{ is maximum}$$

4) $T = \sum_{k=i}^{j} A[k]$

We compute these because the best subsequence may lie completely within one of the 2-halves or straddle the midpoint.

We divide the array A into 2 equal halves & compute these quantities on each halves. We use subscripts 1, 2 to denote the values on the left & right half respectively.

Then

1) If $T_1 > 0$ then $L = T_1 + L_2$ & $l = l_2$
else $L = L_1$ & $l = l_1$

2) If $T_2 > 0$ then $R = T_2 + R_1$ & $r = r_1$
else $R = R_2$ & $r = r_2$

3) $T = T_1 + T_2$

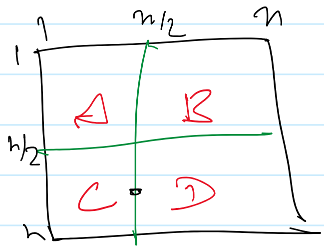
4) $S = \max(S_1, S_2, R_1 + L_2)$ &

$$\begin{aligned}(a, b) &= (a_1, b_1) \text{ if } S = S_1 \\ &= (a_2, b_2) \text{ if } S = S_2 \\ &= (r_1, l_2) \text{ if } S = R_1 + L_1\end{aligned}$$

The combine step requires constant time & hence we obtain the recurrence $T(n) = 2T(n/2) + c$ which evaluates to

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

25) Let l be the minimum value on the boundary of the square. This can be computed with $4n$ comparisons & if this is a local minima we are done. Else we proceed with divide & conquer



We update l by including the vertical line $n/2$ & horizontal line $n/2$. Let (i, j) be the point at which this minimum is attained. Suppose $j = n/2$ & $i > n/2$. This point is less than both its vertical neighbours. If it is less than both its horizontal neighbors we are done. Suppose

Suppose $y = n/2$ & $x = n/2$. This point is less than both its vertical neighbors.
If it is less than both its horizontal neighbors we are done. Suppose

$\vartheta(i, n/2) > \vartheta(i, \frac{n}{2}+1)$.

Claim: There is a local minima in the bottom right square labelled D.

Proof: All points on the boundary of D have value atleast l & $\vartheta(i, \frac{n}{2}+1) < l$. Starting from $(i, \frac{n}{2}+1)$ if we keep moving to a neighbor with lower value we will never leave square D (since the boundary points have larger value). We cannot cycle either & hence we will end up with a local minima.

Hence we recurse over square D. At the start of each recursion we have to update l by computing minimum over $2k$ points where k is the size of the square on which we recurse. This gives the recurrence

$$T(n) = T\left(\frac{n}{2}\right) + 2n$$

which evaluates to $T(n) = 4n$.