Randomized Methods in Computational Geometry

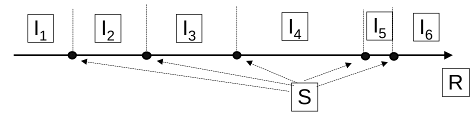
(Marco Pellegrini)

Lecture 1: A geometric interpretation of Quicksort

1

Sorting and Geometry

- Given a set S of n real numbers, sort them in increasing order.
- Given a set S if n points on the real line R find the partition H(S) and the incidence graph G(S) induced by S on R.



$$H(S) = S \cup \{I_1, I_2, I_3, I_4, I_5, I_6\}$$
$$G(S) = \{(a,b) \in H(S) \times H(S), a \neq b, a \cap b \neq \emptyset\}$$

2

Searching and Geometry

- Given a set S of n real numbers, build a data structure D(S) such that for a query number q, we can find efficiently the largest number in S smaller than q.
- Given a set S of n points on the real line, build a data structure D(S) such that for a query point q, we can find efficiently the element of H(S) containing q.
- The two problems are equivalent.
- Standard sorting and searching problems on real numbers can be seen as GEOMETRIC problems in dimension 1 (the real line).

3

Quick-sort

- Quick-sort is the canonical randomized method for sorting n numbers. We shall start by giving a geometric interpretation of quick-sort.
- Quick-sort(S)

```
 IF |S| = 1 THEN RETURN S
   ELSE
     randomly pick q in S
     S1 = { x in S | x < q}
     S2 = { x in S | x > q}
     RETURN [Quick-sort(S1),q,Quick-sort(S2)]
 END IF
```

- The efficiency of the method derives from the intuition that we expect the size of S1 and S2 to be roughly n/2, hence the depth of the recursion in O(log n), the total work is O(n log n).

4

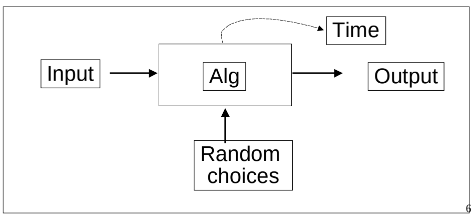
A geometric incremental view of quick-sort

- We start with $S_0 = \emptyset$
- We pick a point p(i) from S/S_i, we add it to the set S_i, thus generating S_{i+1}.
- During the process we build also the sequence: $H(S_0)$, $H(S_1)$, $H(S_2)$,...$H(S_n)$=H(S).
- We need auxiliary functions:
  $f_i(p)$ = the interval A of $H(S_i)$ containing p.
  $L_i(A)$ = the set of elements in S/S_i contained in A, for each A in $H(S_i)$. Called a conflict list.
- Call A' and A" the 2 intervals in $H(S_{i+1})$ adjacent to the added point p. The total update cost is: $O(|L_{i+1}(A')| + |L_{i+1}(A'')|) = O(|L_i(A)|)$.

5

Expected running time.

- Our objective is to bound the expected running time of the algorithm.
- The expectation is with respect to the random choices of the algorithm, not the choice of the input!



6

Expected running time 2.

- The sequence of choices of elements p of the algorithm for a permutation of the input data.
- If we fix a permutation, we obtain a deterministic algorithm, for which we have a well defined running time.
- If we assume each permutation to be equally likely we can define an expected running time:

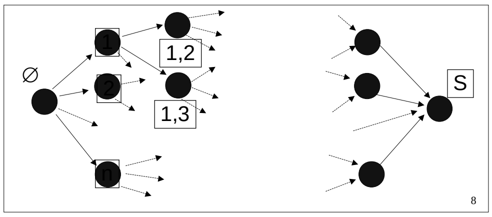
$$E[T(S)] = \sum_{P\in Perm(S)} \left(\frac{1}{n!}\right) T(S,P)$$

7

Random Permutations

forward and backward

- Consider a directed graph G=(X,E), where X=Pow(S), (A,B) in E iff exists a in S \ A st A ∪{a} = B.
- Such graph is a DAG with a source, the empty set and a sink S.



8

Random permutations 2

- Every path from the source to the sink represents a different permutation.
- If we take an outgoing edge from a node with probability 1/(outdegree) then a random walk from source to the sink generates a random permutation. Forward case.
- We can define the inverse graph G'=(X,E') where (A,B) in E' iff exists b in A st A-{b} = B.
- Clearly G' is G with the direction of arrows reversed.
- If we take a random walk on G' selecting an edge with probability 1/(outdegree) we obtain a permutation of S with a uniform random distribution. Backward case.
- It turns out that in the backward case it is much easier to make tight estimates of the computational costs, although the costs are incurred in a simulation of the forward case.

9

Expected analysis of quicksort

- Quick-sort works in n phases, in phase i we build $H(S_{i+1})$ from $H(S_i)$.
- By the linearity of expectation we obtain the expected time by summing the expected cost of each phase.
- Looking backward, each element of S_{i+1} could be chosen with probability 1/(i+1). The expectd cost of phase i is:

$$\sum_{p\in S_{i+1}} \left(\frac{1}{i+1}\right) \sum_{\substack{I\cap p\neq\emptyset \\ I\in H(S_{i+1})}} |L(I)| \le \frac{2n}{i+1}$$

10

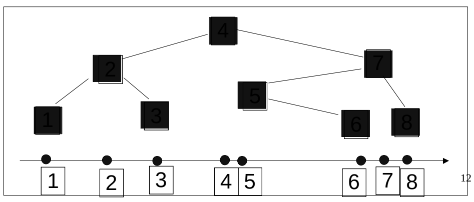
Expected cost of quick-sort 2

- Finally we sum up all phases obtaining the n-th harmonic number. The total expected time is thus O(n log n).
- The main advantage of backward analysis is that for the element "deleted" while in a given state we have additional geometric information which is lost if the element does not belong to the state.
- We basically only us the fact that a point in S_{i+1} is adjacent to two intervals of $H(S_{i+1})$.
- Also notice that we switch from order relations <,>,= to membership relations (p ∈I), or more generally to intersection relations.

11

Searching

- Quick-sort suggests how to build a data structure D(S) to search for a query number q.
- We can define D(S) recursively as a binary tree. At root we place the chosen element p. We recursively build the leftside subtree on S1, the rightside subtree on S2.



12

Searching 2

- Searching algorithm: compare q with the value at the root, if q < Val(root) search in the left subtree, otherwise search in the right subtree. Return the largest value smaller than q to which q has been compared.
- This searching algorithm mimics quicksort as a divide and conquer algorithm.
- If we consider the incremental view, we obtain D(H) from the sequence $H(S_0)$, $H(S_1)$, $H(S_2)$,...,$H(S_n)$ by storing the sequence compactly as a tree.
- The leaves of the tree $D(S_i)$ are the intervals of $H(S_i)$. If we obtain $H(S_{i+1})$ by splitting A into A' and A", we attach A' and A" as children of node A, thus obtaining $D(S_{i+1})$. Our final data structure is $D(S_n)$=D(S).

Expected time analysis.

- D(S) stores intermediate results of the algorithm to build H(S), thus the expected time is the same O(n log n).
- Now we fix a query point q, and we want to bound the expected time needed to answer query(q), where the expectations is with respect to the random choices of the algorithm to build D(S).
- We use the backward analysis on D(S). Going backward, the length of the search path is the number of times the interval containing q is destroyed. The tree $D(S_i)$ has i+1 leaves, and each leaf is destroyed when one of its 2 endpoints is selected. Thus the probability of destroying the leaf holding q is 2/(i+1).
- The sum for all i, is the Harmonic number, O(log n).

Application of Chernoff Bound

We have proved an expected bound for a fixed point q $\in$ R.

Chernoff bound argument are used to show that the probabiliy of exceding the expectation by a large factor is exponentially small

The points in R are partitioned into a polynomial number of classes for which separately the anlaysis holds.

The union bound is used to conclude that the probability of exceeding the expectation by a large factor in any class is exponentially small.