Randomized Algorithms: QuickSort

1 Quicksort

Algorithm: Quicksort

```
fun quicksort(S : sequence<T>) -> sequence<T>:
  if length(S) == 0:
    return S
  p = a uniformly random element of S
  L = filter(fn x => (x < p), S)
  M = filter(fn x => (x = p), S)
  R = filter(fn x => (x > p), S)
  SortedL, SortedR = parallel (quicksort(L), quicksort(R))
  return SortedL + M + sortedR
```

Notice that quicksort is almost identical to quickselect. Both algorithms begin by picking a random pivot element, and partitioning the input sequence using the pivot. (These partitioning steps can be done in parallel, although this does not affect the work and span bounds.) Unlike quickselect, quicksort makes recursive calls into *both* the left and right parts. It then puts the pieces back together into a single sorted list. The base case is the empty sequence. Every element is eventually picked as a pivot.

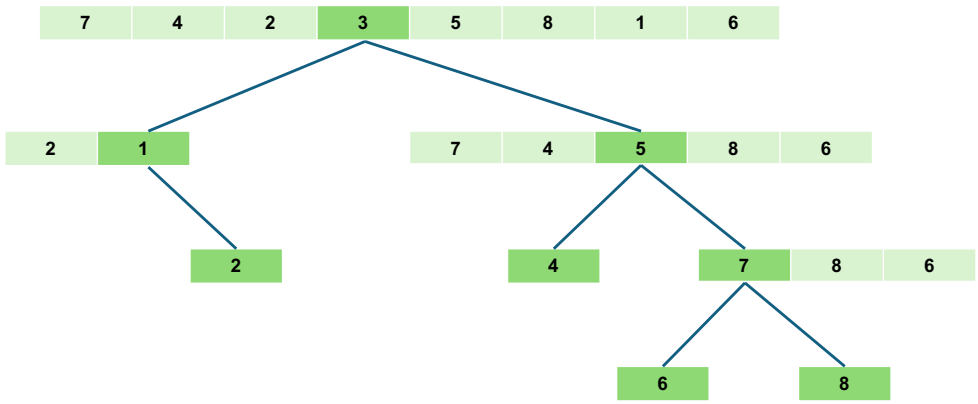
We are, of course, interested in analyzing the work and span of quicksort, as a function of n, the length of the sequence being sorted. Throughout this lecture we will assume that the comparison function among elements of S has $O(1)$ work and span. In this lecture we will show that the work is $O(n)$ in both expectation and w.h.p., and that the span is $O(n \log^2 n)$ both in expectation and w.h.p.

2 Pivot Trees

It's going to be useful to change the way randomness is deployed in the quicksort algorithm. So instead of picking a pivot at random at each level, we will (at the beginning) assign each element of the sequence a random and distinct **priority**. This priority will be used at each level to pick the pivot. The pivot will be the element among those in the current sequence with the highest priority. Viewing these priorities as part of the input converts the quicksort algorithm from randomized to deterministic.

A **pivot tree** is a type of binary search tree which models a particular run of quicksort on a sequence. Just as quicksort is a recursive function, pivot trees can be defined recursively. Say we quicksort a sequence S. The first pivot p that we pick from S is stored root node. The left

subtree of this node is the pivot tree corresponding to the recursive call of quicksort L, and the right subtree is the pivot tree corresponding to the quicksort R call. Every node eventually occurs as a pivot, and therefore appears in the pivot tree. The base case of an empty sequence corresponds to a tree with no nodes. This is illustrated in the following diagram.



The diagram above shows a run of quicksort. It begins with a sequence $S = [7,4,2,3,5,8,1,6]$. It picks the pivot 3 in S. Then the filter forms in $L = [2,1]$ and $R = [7,4,5,8,6]$. It picks a pivot 1 in L, and a pivot 5 in R, and so on. The pivots are shown in dark green, and edges are drawn from the pivot in S to the pivots chosen in L and R. The edges connecting the pivots form a binary tree called the **pivot tree**. Every node in in the input sequence is a node in this tree. The depth of the tree is 4, which is the number of nodes on the longest path in the pivot tree.

From now on we will only be considering runs of quicksort where the input sequence has no duplicate elements. This implies that the pivot tree is well defined, and has a node for each element of the input sequence.

Pivot trees help us to analyze the span of quicksort. The length of the path from the root to a node represents the number of recursive calls needed to place that node in the correct spot. The maximum depth of the tree corresponds to the number of calls that are made. We will use this pivot tree when analyzing the span.

More specifically:

- The work of quicksort is $O(n \cdot (\text{depth of the tree}))$.
- The span of quicksort is $O((\log n) \cdot (\text{depth of the tree}))$.

The $\log n$ term in the span comes from filter, and the n in the work bound comes from the fact that at most $O(n)$ work being at each level of the tree. So our focus now is on analyzing the depth of the pivot tree.

3 Depth of the Pivot Tree

To make things more precise, the depth of the pivot tree is the maximum over all elements of the sequence of the number of nodes on the path from the root to the element in the pivot tree. So in the example above, the depth of the pivot tree is 4.

We now define a random variable $D(n)$ which is the depth of the pivot tree for quicksort being run on a sequence of n distinct elements with random priorities. And we will prove the following theorem.

Theorem: Pivot Tree Depth Theorem

Let the random variable $D(n)$ be the depth of the pivot tree of a run of quicksort on a sequence of n distinct elements. Then $D(n) \in O(\log n)$ w.h.p. Or, more specifically, $D(n) \leq 10 \log_2 n$ w.h.p.

Proof. The key idea of the proof is that there is a perfect correspondence between the pivot tree and running quickselect on all of the elements, and taking the maximum depth.

Say we have a sequence of elements with (no duplicates). Let $R(n,k)$ (where $0 \leq k \leq n-1$) be the random variable which is the recursion depth quickselect uses to find the element of the input sequence of rank k. $R(n,k)$ is also equal to the depth of that element in the pivot tree.

We are now going to make use of a theorem we proved in the last lecture. Here is that theorem:

Theorem: Quickselect Recursion Depth

On a sequence of length n let $R(n)$ be the random variable which is the recursion depth of quickselect running on S. $R(n) \in 5 \log_2 n$ w.h.p.

In proving the QuickSelect Recursion Depth theorem we did not consider the role of k on the depth. We avoided it by making the pessimistic assumption that the algorithm always chooses the larger half to recurse into. Therefore the $R(n)$ has its probability mass "pushed to the right" compared to the random variable $R(n,k)$ defined above. This is called **stochastic dominance**.

Definition: Stochastic Dominance

A random variable A is stochastically dominant over a random variable B if for all x,

$$\Pr(B \geq x) \leq \Pr(A \geq x).$$

Notice that $R(n)$ is stochastically dominant over $R(n,k)$. This is because quickselect is monotonic. That is, if we give it a longer sequence as input the value of the resulting recursion depth cannot decrease. Therefore by making the assumption that we always recurse into the larger part we obtain stochastic dominance over the actual depth that would occur. (More details of this analysis can be found in the lecture on quickselect.)

We have shown that $R(n) \leq 5 \log_2 n$ w.h.p. It follows immediately by stochastic dominance that $X(n,k) \leq 5 \log_2 n$ w.h.p.

So to finish the proof, we can give this definition of $D(n)$ in terms of the $R(n,k)$ random variables:

$$D(n) = \max(R(n,0), R(n,1), ..., R(n,n-1))$$

We can now apply the Max Preserves w.h.p. theorem. Since for each $R(n,k) \leq 5\log_2(n)$ w.h.p. it follows that $D(n) \leq 10\log_2(n)$ w.h.p.

Notice that the $R(n,k)$ are not independent – two elements near each other in the pivot tree have many random choices in common (they have many ancestor pivots in common). But the theorem does not require independence. □

4 Bounding the Expectation

We've shown that $D(n) \in O(\log n)$ w.h.p. Now we prove that its expected value is also $O(\log n)$. It turns out to be quite easy because the maximum depth of quicksort is bounded by n.

Theorem: Expected Depth of Quicksort

The expected depth of quicksort is $O(\log n)$.

Proof. First we make the observation that $D(n) \leq n$. I.e. quicksort always terminates by level n. We have also established the fact that $D(n) \leq 10\log_2(n)$ w.h.p.

If $n \leq 10\log_2(n)$ then $D(n) \leq 10\log_2(n) = O(\log n)$.

If $n > 10\log_2(n)$, then in this case we have:

$$\Pr(D(n) \geq n) \leq \Pr(D(n) > 10\log_2(n)) < 1/n$$

So the contribution to $\mathbb{E}[D(n)]$ in the range where $D(n) \leq 10\log_2(n)$ is at most $10\log_2(n) \in O(\log n)$. And the contribution to in the range where $D(n) > 10\log_2(n)$ is at most $n \cdot (1/n) = 1$.

We conclude that $\mathbb{E}[D(n)] = O(\log n)$. □

Finally we conclude with the following theorem:

Theorem: Quicksort Bounds

The span of quicksort is $O(\log^2 n)$ w.h.p. and in expectation. The work of quicksort is $O(n \log n)$ w.h.p. and in expectation.

Proof. These follow immediately from the fact filter has $O(\log n)$ span and $O(n)$ work, and that the depth of quicksort is $O(\log n)$ w.h.p. and in expectation. □