CS124	Lecture 9	Spring 2022

Selection

Given an unsorted array L of length n and a value x, how quickly can we compute $RANK(L,x)$, the number of elements of L that are at most x? We can do so in time $O(n)$ by iterating through the elements of L, keeping count of how many are at most x, and this linear-time algorithm is the best possible.

A converse to that problem is selection: given an unsorted array L of length n, how quickly can we find the kth-smallest element, $SELECT(L,k)$? One approach is to sort the array and return its kth element, but the time to sort the array (e.g. $O(n \log n)$ by merge sort) is slower than optimal. Another approach is to find the smallest element in $O(n)$ time, remove it, and repeat k times. If k is constant, this is an $O(n)$ algorithm, but the $O(kn)$ runtime is again slower than optimal if k is nonconstant.

For a faster algorithm, we make use of the following observation: if we have the median element of L, and $k < \frac{n}{2}$, then we only need to look at elements of L less than the median; we can make a new list containing only those, and reduce the problem to one of half the size. Similarly, if we have the median element of L and $k > \frac{n}{2}$, we only need the larger half of the list. If we have not the median element but an element x in the middle, say, 40% of L, then we can throw out at least the top or bottom 30% of L. If we can find such an element in time $S(n)$, then we can compute $SELECT(L,k)$ in time $T(n)$ given by $T(n) \leq S(n) + T(.7n) + O(n)$. By the Master Theorem, if $S(n) < T(.29n) + O(n)$, then we'd have a linear-time algorithm for SELECT.

We'll consider two approaches for finding such a nearly-median element x: a deterministic median-of-medians approach, and randomness.

Divide L into $\frac{n}{5}$ groups of 5 elements and, for each of those $\frac{n}{5}$ groups, find the median (e.g. by sorting). Make a new array L' of those $\frac{n}{5}$ medians, and recursively find the median x' of L'. In each of the $\frac{n}{10}$ groups whose median is *less than* x', x' and the two smaller elements of the group all have rank less than $n/2$, so those $\frac{3n}{10}$ elements of L are less than x'. Similarly, $\frac{3n}{10}$ elements of L are greater than x', so x' is the desired element in the middle 40% of L.

Putting it all together, we have the following algorithm (ignoring the issues of lists whose length is not a multiple of 5, and of equal elements):

```
procedure RANK(L,x)
    s = 0
```

```
    for ℓ in L:
        if ℓ ≤ x:
            s = s + 1
    return s

procedure SELECT(L,k)
    L' = empty list
    for i from 0 to (size(L)/5) - 1:
        m = median(L[5i],L[5i+1],L[5i+2],L[5i+3],L[5i+4])
        L'.append(m)
    x' = SELECT(L', n/10)
    L" = empty list
    if RANK(L, x') > k:
        for ℓ in L, ℓ ≤ x':
            L".append(x)
        return SELECT(L",k)
    else:
        for ℓ in L, ℓ ≥ x':
            L".append(x)
        return SELECT(L",k + size(L") - size(L))
```

If $T(n)$ is the runtime for this algorithm on a list of length n, then $T(n) \leq T(\frac{n}{5}) + T(\frac{7n}{10}) + O(n)$, whose solution has $T(n) = O(n)$.

This algorithm runs in time $O(n)$, which is asymptotically the best possible: any algorithm with an $o(n)$ runtime couldn't look at every element of L. However, the constant factor hidden by asymptotic notation is rather large, and may be impractical.

We can instead find a nearly-median element x by choosing a random element of L, calculating its rank, and throwing it out and repeating if x isn't in the middle 40%. In expectation, we only need to repeat 2.5 times to find an appropriate x.

Alternately, rather than throwing out an x too far from the median, we can go through the process of throwing out elements on the other side of x from $SELECT(L,k)$ even if there are not many of them, and get an algorithm called quickselect:

```
procedure QUICKSELECT(L,k)
    x' = random element of L
    L" = empty list
    if RANK(x') > k:
        for ℓ in L, ℓ ≤ x':
            L".append(x)
        return SELECT(L",k)
```

```
else:
    for ℓ in L, ℓ ≥ x':
        L".append(x)
    return SELECT(L",k + size(L") - size(L))
```

Quickselect's expected runtime is $O(n)$, and is often faster than the *SELECT* algorithm given above.