CS 240 — Lecture 6

Bartosz Antczak *Instructor: Eric Schost* *January 19, 2017*

Today's Plan

Today we'll observe the average case analysis of two algorithms, **quick-select** and **quick-sort**. Note that this will probably be the hardest algorithm analysis in this course (according to Prof. Schost).

Recall

The **selection problem** states

Given an array A of n numbers, and $0 \leq k < n$, find the element in position k of the sorted array

We'll look at the two previously mentioned algorithms to solve this problem

6.1 Solving the Selection Problem

6.1.1 Partition Algorithm

Algorithm Description

The **partition** algorithm takes an input of one array `A` and an arbitrary integer `p`. It rearranges `A` such that all of the elements smaller than the element at index `p` are to the left of it and the elements larger than it are to the right.
The algorithm is structured as:

```
// A - array of size n
// p - arbitrary integer that represents our pivot
partition(A, p) {
  swap(A[0], A[p])
  i = 1
  j = n-1
  loop
    while i < n and A[i] <= A[0] do
      i = i + 1
    while j >= 1 and A[j] > A[0] do
      j = j - 1
    if (j < i) then break
    else swap(A[i], A[j])
  end loop
  swap(A[0], A[j])
  return j // the new position of the pivot
}
```

Let's analyse this algorithm through an example. Let $A = [7, 3, 6, 9, 2, 1, 5]$ and $p = 0$. After each iteration of the big loop, we have

1. $[7, 3, 5, 9, 2, 1, 6]$, $i = 2, j = 6$
2. $[7, 3, 5, 6, 2, 1, 9]$, $i = 3, j = 6$

3. $[7, 3, 5, 9, 1, 2, 6]$, $i = 4, j = 5$
4. $[7, 3, 5, 6, 1, 2, 9]$, $i = 5, j = 5$
5. $[2, 3, 5, 6, 1, 7, 9]$, $i = 6, j = 5$ and then exit and return the value of j, 5

Observe that the value 7 was our pivot. After the algorithm, 7 is now in its correct position. The returned value (5) is the position of the pivot after partitioning.

6.1.2 QuickSelect Algorithm

Algorithm Description

The **quick select** algorithm returns the kth smallest element an array. The structure is as follows:

```
// A - array of size n
// k - arbitrary integer such that 0 <= k < n
quick-select1(A, k) {
  p = choosePivot(A)
  i = partition(A, p)
  if (i = k) then
    return A[i]
  else if i > k then
    return quick-select1(A[0, 1, ..., i-1], k)
  else if i < k then
    return quick-select1(A[i+1, i+2, ..., n-1], k-i-1)
}
```

A few words about correctness. After line `i = partition(A, p)`, i contains the position of the pivot in the partitioned array, which is also the position it would have in the sorted array. So if $k = i$, we are done, we return the pivot. If $k < i$, we have to look on the left side of the array (so we restrict the search to $A[0, 1, ..., i-1]$ and we don't change k). If $k > i$, we have to look on the right side of the array, so we restrict the search to $A[i+1, 1, ..., n-1]$, but we need to subtract from k the number of elements we skipped, which is $i + 1$.

Algorithm Analysis

The recurrence relation for this algorithm is:

$$T(n) = \begin{cases} T(n-1) + cn & n \geq 2 \\ d & n = 1 \end{cases}$$

(cn represents the time to partition the array, which takes linear time)
The **worst-case** analysis will have a recursive call that will always have size $n - 1$. We calculate it to be

$$T(n) = cn + c(n-1) + \cdots + c \cdot 2 + d \in \Theta(n^2)$$

Proof

Simply use the recurrence relation from n down to 2:

$$T(n) = cn + T(n-1) = cn + c(n-1) + T(n-2) = \cdots = cn + c(n-1) + \cdots + c \cdot 2 + T(1).$$

The **best-case** analysis will have the first chosen pivot being the kth element. This means there are no recursive calls and the runtime is

$$\Theta(n)$$

.

Average Case Analysis

Warning: this proof is difficult. I (É.S.) wish I knew how to simplify it, but I don't. What I would expect you to remember from this: the result itself (that the expected runtime is linear) and possibly Lemma **??** below.

A priori, it is not straightforward to speak of the average case running time of quick-select: for a given input length n, there are infinitely many choices for $A[0], \ldots, A_[n-1]$ (so we would not be dealing with a finite sum to compute our average).

To simplify things a bit, we make a first assumption: **all entries in A are pairwise distinct** (no repetitions are allowed). Then, we are going to use the following remark: the behavior of the algorithm does not depend on the actual values in A; rather, it depends on their *order*. For instance $[2, 3, 10, 5]$ and $[1, 2, 4, 3]$ are identical for the quick-select algorithm (i.e., the array is sorted up to the second last index).

For an input consisting of array A and integer k, let us call $T(A, k)$ the runtime of quick-select on input A, k. The remark in the previous paragraph means that $T([2, 3, 10, 5], k) = T([1, 2, 4, 3], k)$ (for any k); we also equalities such as $T([3, 4], 1) = T([1, 2], 1)$ and $T([4, 3], 1) = T([4, 3], 1)$. Since we are assuming that there are no repetitions, for an input size n, we can do as if the entries in A were a permutation of $1, 2, \ldots, n$. Then, the average we want to compute is

$$T(n, k) = \frac{1}{n!} \sum_{A \text{ permutation of } [1,\ldots,n]} T(A, k).$$

A few examples:

- For $n = 1$, we can only have $k = 0$ (since $0 \leq k \leq n-1$), and we have

  $$T(1, 0) = T([1], 0),$$

  since there is only one permutation of $[1]$.

- For $n = 2$, we can only have $k = 0$ or $k = 1$, and we have

  $$T(2, k) = \frac{1}{2}\left(T([1, 2], k) + T([2, 1], k)\right).$$

- For $n = 3$, we can have $k = 0, 1, 2$, and we have

$$T(3,k) = \frac{1}{6}\left(T([1,2,3],k) + T([1,3,2],k) + (T([2,1,3],k) + T([2,3,1],k) + (T([3,1,2],k) + T([3,2,1],k)\right). \tag{6.1}$$

We want to prove that $T(n,k) \in O(n)$, for any given k. Let us define $T(n) = \max_{0 \le k \le n-1} T(n,k)$; we will actually prove that $T(n)$ is in $O(n)$. Since $T(n,k) \le T(n)$ holds for all k, this will prove what we want.

We will rely on the following result.

Lemma 1. *If a function $F(n)$ satisfies a "sloppy" recurrence $F(n) \le kn + F(\alpha n)$, for some constants k and α, with $\alpha < 1$, then $F(n) \in O(n)$.*

Proof

We only do the proof for $\alpha = 1/2$ and n being a power of 2 (but the result is true in general).

$$\begin{aligned} F(n) &\le kn + F\left(\frac{1}{2}n\right) \\ &\le kn + k\frac{n}{2} + F\left(\frac{1}{4}n\right) \\ &\le kn + kn/2 + kn/4 + \cdots + d \\ &\le kn(1 + 1/2 + 1/4 + 1/8 \cdots) + d. \end{aligned}$$

Since $1 + 1/2 + 1/4 + 1/8 \cdots \le 2$, $F(n) \in O(n)$.

To prove that $T(n)$ is in $O(n)$, we are going to prove that $T(n)$ satisfies an inequality of the form $T(n) \le kn + T(3n/4)$, for some constant k; then, the lemma above will imply that $T(n)$ is indeed in $O(n)$. To start with, we let c be a constant such that for any input of size n, choosing the pivot and partitioning take time cn. Then, the runtime of quickselect on input $A = [A[0], \ldots, A[n-1]]$ and k looks like

$$T(A,k) = \begin{cases} cn & k = i \\ cn + T([A'[0], \ldots, A'[i-1], k) & k < i \\ cn + T([A'[i+1], \ldots, A'[n-i-1], k-i-1) & k > i, \end{cases} \tag{6.2}$$

where i is the position of the pivot, found in the first step, and A' represents the array A *after partition.*

We make here a second assumption: **after the partition, the elements less than the pivot are in the same order as before the partition, and similarly for the elements greater than the pivot**. For instance, consider $A = [2, 5, 3, 1, 6, 4]$. If we let the pivot index p equal 2 ($A[p] = 3$), then we expect the array after the partition to look like this:

$$A = [2, 1, \mathbf{3}, 5, 6, 4],$$

with the pivot written in bold. This assumption may not be satisfied for our partition function, but it is not hard to write a partition function that ensures this, and still takes linear time.

This assumption will be handy to understand how the recursive calls look like. For instance, suppose our input it $A = [3, 2, 1, 4]$, with pivot index $p = 0$, $A[p] = 3$, and $i = 2$ (the position of the pivot after partition); then, the array after partition must be $[2, 1, \mathbf{3}, 4]$. Then, $T([3, 2, 1, 4], 1) = 4c + T([2, 1], 1)$, since the recursive call is done on the left, with inputs $[2, 1]$ and 1.

We see that if we want to compute the average of all $T(A, k)$'s, we have to take the index i into account. One way to do this is to subdivide our set of $n!$ inputs. Among the $n!$ permutations of $[1, \ldots, n]$, there are $(n-1)!$ for which index i above (the position of the pivot after partition) is 0, $(n-1)!$ for which index i is 1, $\ldots$, $(n-1)!$ for which index i is $n-1$. For instance, for $n = 3$, we have $[1, 2, 3]$ and $[1, 3, 2]$, for which i will be 0, $[2, 1, 3]$ and $[2, 3, 1]$, for which i will be 1, and $[3, 1, 2]$ and $[3, 2, 1]$, for which i will be 2. As a result, we can write

$$T(n, k) = \frac{1}{n} \sum_{i=0}^{n-1} T(n, k, i), \tag{6.3}$$

where $T(n, k, i)$ is the average run-time of quick-select over all A's that are permutations of $[1, \ldots, n]$ and for which the position of the pivot, computed at the first step, is i (we saw that there are $(n-1)!$ of those). For instance, with $n = 3$, we have

$$\begin{aligned} T(n, k, 0) &= \frac{1}{2}(T([1, 2, 3], k) + T([1, 3, 2], k)), \\ T(n, k, 1) &= \frac{1}{2}(T([2, 1, 3], k) + T([2, 3, 1], k)), \\ T(n, k, 2) &= \frac{1}{2}(T([3, 1, 2], k) + T([3, 2, 1], k)), \end{aligned}$$

and plugging these into (**??**) gives us the sum in (**??**).

Let us plug the recursive formula (**??**) into the definition of $T(n, k, i)$. Let us deal with the case $k < i$, for instance.

$$\begin{aligned} T(n, k, i) &= \frac{1}{(n-1)!} \sum_{A \text{ permutation of } [1, \ldots, n] \text{ with pivot position } i} T(A, k) \\ &= \frac{1}{(n-1)!} \sum_{A \text{ permutation of } [1, \ldots, n] \text{ with pivot position } i} cn + T(A'[0], \ldots, A'[i-1], k) \\ &= cn + \frac{1}{(n-1)!} \sum_{A \text{ permutation of } [1, \ldots, n] \text{ with pivot position } i} T(A'[0], \ldots, A'[i-1], k). \end{aligned}$$

For example, look at $n = 3$, $k = 1$ and $i = 2$. We saw that there the inputs having $i = 2$ are $[3, 1, 2]$ and $[3, 2, 1]$, which become $[1, 2, \mathbf{3}]$ and $[2, 1, \mathbf{3}]$ after partition (I am using my second assumption here). With $k = 1$, the costs are respectively $T([3, 1, 2], 1) = 3c + T([1, 2], 1)$ and $T([3, 2, 1], 1) = 3c + T([2, 1], 1)$. Finally

$$T(3, 1, 2) = 3c + \frac{1}{2}(T([1, 2], 1) + T([2, 1], 1)),$$

and we saw previously that the second term is $T(2, 1)$. So finally, $T(3, 1, 2) = 3c + T(2, 1)$. Since $T(2, 1) \leq T(2)$ (by definition of T!), we have

$$T(3, 1, 2) \leq 3c + T(2).$$

This fact is true in general: from the expression above giving $T(n, k, i)$, and using our assumption on the partition function, we can prove the following:

Lemma 2.

$$T(n,k,i) \le cn + \begin{cases} 0 & k = i \\ T(i) & k < i \\ T(n-i-1) & k > i. \end{cases}$$

From now on, I am going to pretend that T is a non-decreasing function, and I will finish the proof under this assumption. I could do without it, but it would make everything even more complicated. This assumption gives us in particular

$$T(n,k,i) \le cn + T(\max(i, n-i-1)).$$

Let us plug this in (**??**); we get

$$T(n,k) \le cn + \frac{1}{n}\sum_{i=0}^{n-1} T(\max(i, n-i-1)).$$

The right-hand side does not depend on k, so I can take the max over all k's on the left, and I get

$$T(n) \le \frac{1}{n}\sum_{i=0}^{n-1} T(\max(i, n-i-1)). \tag{6.4}$$

We finish by doing a case discussion on i (and we pretend that n is a multiple of 4)

- If $i < n/4$, then I will simply using the fact that both i and $n-i-1$ are at most n, so (because we assume that T is non-decreasing)

  $$T(\max(i, n-i-1)) \le T(n).$$

  Same thing with $i \ge 3n/4$; altogether, there are $n/2$ such values of i.

- The sweet spot is when $n/4 \le i < 3n/4$. In that case, we have both $i < 3n/4$ and $n-i-1 < 3n/4$, so their max is less than $3n/4$, and (because we assume that T is non-decreasing)

  $$T(\max(i, n-i-1)) \le T(3n/4).$$

  These account for the other $n/2$ values of i.

Bottom line, for $n/2$ values of i, we plainly have $T(\max(i, n-i-1)) \le T(n)$, and for the other $n/2$ we have $T(\max(i, n-i-1)) \le T(3n/4)$. Plugging this into (**??**), we obtain

$$T(n) \le cn + \frac{1}{2}T(n) + \frac{1}{2}T(3n/4),$$

which is equivalent to

$$T(n) \le 2cn + T(3n/4).$$

As promised we can then use Lemma **??** to conclude that $T(n)$ is in $O(n)$.